ENSAE

Ecole Nationale de la Statistique et de l'Administration Economique

# Initiation à la programmation

# avec les langages *Python*

# et *Visual Basic for Application (VBA)*

Xavier Dupré

http ://www.xavierdupre.fr/

# Table des matières

*1. Avant-propos* . . . . . 8

*partie I Le langage Python* 10

*2. Introduction* . . . . . 11
2.1 Ordinateur et langages . . . . . 11
2.2 Présentation du langage *Python* . . . . . 12
2.3 Installation du langage *Python* . . . . . 14
2.4 Installation d'un éditeur de texte . . . . . 17
2.5 Premier programme . . . . . 20
2.6 Installation d'extensions (ou modules externes) . . . . . 22
2.7 Outils connexes . . . . . 26

*3. Types et variables du langage Python* . . . . . 29
3.1 Variables . . . . . 29
3.2 Types immuables (ou *immutable*) . . . . . 30
3.3 Types modifiables (ou *mutable*) . . . . . 38
3.4 Extensions . . . . . 48

*4. Syntaxe du langage Python (boucles, tests, fonctions)* . . . . . 51
4.1 Les trois concepts des algorithmes . . . . . 51
4.2 Tests . . . . . 53
4.3 Boucles . . . . . 57
4.4 Fonctions . . . . . 63
4.5 Indentation . . . . . 76
4.6 Fonctions usuelles . . . . . 76
4.7 Constructions classiques . . . . . 77

*5. Classes* . . . . . 83
5.1 Présentation des classes : méthodes et attributs . . . . . 83
5.2 Constructeur . . . . . 87
5.3 Apport du langage *Python* . . . . . 88
5.4 Opérateurs, itérateurs . . . . . 91
5.5 Méthodes, attributs statiques et ajout de méthodes . . . . . 96
5.6 Copie d'instances . . . . . 101

5.7 Attributs figés . . . 107
5.8 Héritage . . . 108
5.9 Compilation de classes . . . 117
5.10 Constructions classiques . . . 118

*6. Exceptions* . . . 122
6.1 Principe des exceptions . . . 122
6.2 Définir ses propres exceptions . . . 129
6.3 Exemples d'utilisation des exceptions . . . 130

*7. Modules* . . . 133
7.1 Modules et fichiers . . . 133
7.2 Modules internes . . . 138
7.3 Modules externes . . . 139
7.4 *Python* et les autres langages . . . 140
7.5 Ecrire un module en langage *C++* . . . 141
7.6 *Boost Python* . . . 144

*8. Fichiers, expressions régulières, dates* . . . 153
8.1 Format texte . . . 153
8.2 Fichiers zip . . . 157
8.3 Manipulation de fichiers . . . 158
8.4 Format binaire . . . 161
8.5 Paramètres en ligne de commande . . . 165
8.6 Expressions régulières . . . 167
8.7 Dates . . . 173
8.8 Problème de jeux de caractères . . . 173

*9. Interface graphique* . . . 175
9.1 Introduction . . . 175
9.2 Les objets . . . 176
9.3 Disposition des objets dans une fenêtre . . . 185
9.4 Evénements . . . 188
9.5 D'autres fenêtres . . . 194
9.6 Constructions classiques . . . 196

*10. Threads* . . . 202
10.1 Premier thread . . . 202
10.2 Synchronisation . . . 204
10.3 Interface graphique . . . 207
10.4 Files de messages . . . 209

*partie II Enoncés pratiques, exercices* 213

*11. Exercices pratiques pour s'entraîner* . . . 214
11.1 Montant numérique, montant littéral . . . 214
11.2 Représentation des données, partie de dames . . . 217
11.3 Reconnaître la langue d'un texte . . . 220
11.4 Carrés magiques . . . 228
11.5 Tri rapide ou quicksort . . . 234

*12. Exercices pratiques pour s'évaluer* . . . 242
12.1 Recherche dichotomique (2005-2006) . . . 242
12.2 Ajouter un jour férié (2006-2007) . . . 244
12.3 Fréquentation d'un site Internet (2007-2008) . . . 247
12.4 Ajout d'une pièce de monnaie (2008-2009) . . . 250
12.5 Fusion de fichiers (2008-2009) . . . 255

*13. Exercices écrits* . . . 259
13.1 Premier énoncé (2005-2006) . . . 259
13.2 Second énoncé (2006-2007) . . . 271
13.3 Troisème énoncé (2007-2008) . . . 279
13.4 Quatrième énoncé (2007-2008) . . . 289
13.5 Cinquième énoncé (2008-2009) . . . 295
13.6 Exercices supplémentaires . . . 305

*partie III Résumé partique, outils d'aide, algorithmes à connaître* 309

*14. Instructions fréquentes en Python* . . . 311
14.1 Le langage . . . 311
14.2 Les variables . . . 311
14.3 Tests et boucles . . . 315
14.4 Fonctions . . . 318
14.5 Classes . . . 318
14.6 Fichiers . . . 321
14.7 Modules . . . 323
14.8 Exceptions . . . 324
14.9 Erreurs, confusions fréquentes . . . 326

*15. Outils d'aide au développement* . . . 328
15.1 Modules d'aide au développement . . . 328
15.2 Tortoise SVN : suivi de version . . . 336
15.3 HTML Help Workshop : construction d'un fichier d'aide . . . 342
15.4 InnoSetup : installateur . . . 349
15.5 MoinMoin : wiki . . . 356

15.6 SQL . . . 367
15.7 Modules divers . . . 378
15.8 Graphisme . . . 380
15.9 Apache : serveur . . . 383

*16. Algorithmes usuels* . . . 387
16.1 Calcul d'une intégrale . . . 387
16.2 Tri simple et tri fusion . . . 388

*partie IV Visual Basic For Applications (VBA)* 398

*17. Introduction* . . . 399
17.1 Visual Basic et macros . . . 399
17.2 Première procédure . . . 399
17.3 Macro liée à une feuille Excel . . . 400
17.4 Erreurs . . . 401
17.5 Autres possibillités . . . 402

*18. Le langage VBA* . . . 404
18.1 Préliminaires . . . 404
18.2 Variables et opérateurs . . . 404
18.3 Affichage . . . 408
18.4 Tests . . . 409
18.5 Boucles . . . 410
18.6 Fonctions et procédures . . . 411
18.7 Erreurs . . . 412
18.8 Fichiers . . . 413

*19. Exercices autour de l'automatisation de tâches quotidiennes* . . . 415
19.1 Création de PDF puis envoi automatique de mails . . . 415
19.2 Création d'une présentation à partir de photos de vacances . . . 424
19.3 Résolution d'un sudoku . . . 432

*20. VBA et Python* . . . 443
20.1 *Excel* et *Python* . . . 443
20.2 *Python*, valeurs propres . . . 446
20.3 Des graphes complexes avec *R* sous *Excel* . . . 449

*partie V Exemples variés* 455

*21. Périphériques* . . . 457
21.1 Copie de fichiers vers une clé USB . . . 457
21.2 Recherche sur Internet . . . 461

21.3 Python, Excel, Latex . . . 471
21.4 Fond d'écran . . . 471

*22. Logique et jeux* . . . 474
22.1 Résolution d'un sudoku . . . 474
22.2 Résolution d'un puzzle . . . 479
22.3 Résolution d'une énigme . . . 494
22.4 Simulation informatique . . . 501

*23. Mathématique, recherche opérationnelle* . . . 513
23.1 Plus court chemin dans un graphe . . . 513
23.2 Problème du voyageur de commerce . . . 527
23.3 Classification à l'aide de plus proches voisins . . . 557
23.4 Optimisation numérique du second ordre . . . 577
23.5 Optimiser de coûts en environnement aléatoire . . . 599
23.6 Stratégie automatique de trading en finance . . . 607

*partie VI Exercices variés* 631

*24. Sujets informatiques* . . . 632
24.1 Dictionnaires (1) . . . 632
24.2 Dictionnaires (2) . . . 634
24.3 Popularité selon différents moteurs de recherce . . . 635
24.4 Tri simple et tri fusion . . . 641
24.5 Equation différentielle . . . 643

*25. Sujets autour des interfaces graphiques* . . . 647
25.1 Interface graphique . . . 647
25.2 Jeu avec *PyGame* . . . 653

*26. Sujets mathématiques, statistiques* . . . 660
26.1 Casser le code de Vigenère . . . 660
26.2 Disparition des gens aux yeux bleus . . . 666
26.3 Calculer des espérances de vie . . . 670
26.4 Trouver une personne ayant les mêmes goûts que vous . . . 679
26.5 Calcul d'un histogramme . . . 683

*partie VII Domaine hors sujets* 691

*27. Sujets mathématiques* . . . 692
27.1 File d'attente . . . 692
27.2 Simulation génétique . . . 699
27.3 Tracer une ligne sans multiplication . . . 701

27.4 Simulation d'une corde . . . . . 710

*28. Inclassables* . . . . . 719
28.1 Filtrage du signal sonore . . . . . 719
28.2 Images de synthèse, modèle d'illumination . . . . . 724

*partie VIII Exercices autour de sujets déjà abordés* 749

*29. Exercices plutôt mathématiques* . . . . . 750
29.1 Tracé d'une ligne sans multiplication . . . . . 750
29.2 Un portrait lointain à l'aide du TSP . . . . . 757
29.3 Image de synthèse . . . . . 775

*30. Exercices plutôt informatiques* . . . . . 792
30.1 Conserver ses fichiers sur une clé USB avec une interface graphique . . . . . 792
30.2 Conserver ses fichiers sur une clé USB avec une interface graphique (2) . . . . . 800

*partie IX Bibliographie, index, remerciements* 804

*Index* 808

# Chapitre 1

# Avant-propos

Quelques années passées à initier les élèves à l'informatique avec différents langages me poussent à dire que le langage *Python* est sans doute un langage qui s'apprend plus vite sans pour autant être limité. Les ordinateurs ont progressé et il n'est plus souvent nécessaire d'utiliser des langages tels que le *C* certes plus rapide et plus connu mais aussi plus difficile à maîtriser. A partir de 2004, le langage *Python* a été utilisé à l'ENSAE [1] pour initier les élèves à la programmation et, depuis cette date, j'ai pu observer un regain d'intérêt des élèves pour cette matière qui s'appuyait auparavant sur le langage *C*. Certains étudiants avouent même s'être amusés ; leur projet de programmation n'est plus centré sur un langage mais sur les résultats qu'il permet d'obtenir.

Le langage *Python* suffit pour la plupart des besoins d'un ingénieur et permet de réaliser beaucoup plus rapidement des programmes informatiques qui fonctionnent sur les principaux systèmes d'exploitation. Internet fournit également un panel très large de librairies pour faire du calcul scientifique, des jeux ou encore des sites Internet. Il suffit parfois de quelques minutes pour télécharger une extension et s'en servir. Malgré ces avantages, il existe peu de livres de langue française sur ce langage, beaucoup moins que ceux écrits en langue anglaise dont quelques uns seulement sont traduits.

Quelques d'ouvrages décrivent minutieusement le langage et ses extensions ([Swinnen2004], [Martelli2004], [Pilgrim2004], [Ziadé2006]). L'ouvrage [Swinnen2004] part d'exemples pour expliquer le langage. Les trois autres sont plus classiques dans leur présentation : les exemples sont moins présents et leur plan suit plus les différentes fonctionnalités du langage. Un autre ouvrage ([Dayley2007]) prend une direction différente : des explications courtes, beaucoup de courts exemples qui répondent à des besoins précis. Un livre ([Brucher2008]) s'intéresse à l'utilisation du langage que pourrait en faire un ingénieur : les fonctionnalités liées au calcul numérique sont mieux représentées. Un dernier livre ([Bailly2008]) propose des exemples de programmes équivalents rédigés en *Python* et *C++* sans présenter de cas associant ces deux langages. J'ai apprécié [Dayley2007] pour ses exemples concis, [Martelli2004] pour son exhaustivité. [Ziadé2006] est une bonne référence pour ceux qui souhaitent prendre de bonnes habitudes de programmation.

Ce livre n'aborde pas des sujets comme la programmation web ou l'utilisation conjointe des langages *Python* et *SQL*. La description du langage couvre les besoins qui sont ceux d'un ingénieur, c'est-à-dire une personne dont la programmation ne constitue pas l'essentiel de son travail. Chaque notion est illustrée par des exemples [2]. Ceux-ci sont particulièrement présents lors de l'introduction de notions pas toujours évidentes à comprendre comme les threads, les interfaces graphiques et l'utilisation de *C++*. Le livre se ferme sur plusieurs chapitres ne contenant que des exercices et leurs corrections. Ces exercices sont de deux types : des travaux dirigés et des problèmes courts. Ces derniers ont été posés lors des examens à l'ENSAE et ont été conçus pour préparer les élèves à répondre aux questions techniques lors des entretiens d'embauche.

Le langage *Python* évolue assez vite, il est de plus en plus utilisé ce qui explique la multitude d'informations qu'on peut trouver sur Internet. Le site officiel [3] recense la plupart des extensions (plus d'une centaine) accessibles gratuitement. Il existe également de nombreux sites d'aide comme *WikiPython* [4] qui recense des exemples de programmes sur des sujets variés. L'anglais est la langue d'échange par excellence dans le monde informatique et le langage *Python* n'y échappe pas même si les moteurs de recherches retournent

1. Ecole Nationale de la Statistique et de l'Administration Economique
2. L'ensemble de ces exemples est accessible depuis le site http://www.xavierdupre.fr.
3. http://www.python.org/
4. http://wikipython.flibuste.net/

presque toujours des résultats en français pour des requêtes concernant *Python*, à condition toutefois d'y insérer un mot de syntaxe exclusivement française.

Il n'est pas utile de lire tous les chapitres pour savoir programmer. Le chapitre 2 décrit l'installation du langage *Python* et son utilisation via des éditeurs de texte. Les chapitres 3 et 4 introduisent les premiers concepts, à partir de là, il est déjà possible d'écrire des programmes conséquents et notamment de lire les chapitres 12 à 13 pour s'exercer.

Les chapitres 5 et 6 s'intéressent à la programmation objet, qui n'est pas indispensable dans un premier temps. La programmation est utilisée en tant qu'outil pour atteindre un objectif, réaliser une étude ou tester un modèle. Nombreux sont les élèves qui choisissent de contourner les classes et les objets sans pour autant nuire à la qualité de leur travail ; les programmes qu'ils conçoivent sont sans doute moins lisibles mais ils parviennent à l'objectif fixé.

Le chapitre 7 présente comment créer ses propres extensions. Les derniers paragraphes ne concernent que des utilisateurs experts puisqu'ils parlent de l'utilisation conjointe des langages *Python* et *C++* à partir d'un exemple conçu à cet effet et utilisable en peu de temps [5]. Cette technique est intéressante dans des domaines où les calculs doivent être impérativement très rapides comme en finance, matière à laquelle sont formés les étudiants de l'ENSAE.

Le chapitre 8 introduit les fichiers où comment conservées des résultats sur disque dur. Le chapitre 9 présente les interfaces graphiques et le chapitre 10 les threads. Les chapitres 11 à 13 ne contiennent que des exercices et problèmes sur lesquels plusieurs promotions de l'ENSAE ont été évaluées.

**Visual Basic for Application (VBA)**

Lorsqu'une application comme *Excel* est aussi couramment utilisée, il est parfois utile de savoir programmer dans le langage associé à ce logiciel. Même s'il n'est pas toujours aussi convivial que *Python*, on y retrouve les mêmes éléments (boucles, tests, fonctions). Néanmoins, comme *Python*, VBA est un langage assez lent et il est aussi difficile de trouver des extensions de calcul numérique. C'est pourquoi, il est parfois utile de lier les deux langages afin de pouvoir utiliser les nombreux modules disponibles pour le langage *Python* tels que `scipy` ou encore `BioPython`. Le premier regroupe aborde le traitement du signal, l'estimation de densité, la classification, l'optimisation... Le second module aborde des techniques spécifiques au domaine biologique et plus spécifique les séquences ADN.

5. Concernant ce point, cet exemple téléchargeable permet de construire entièrement un module *Python* écrit en *C++*, l'ajout d'une fonctionnalité ne prenant que quelques minutes. Cet exemple fonctionne sous *Microsoft Windows* avec *Microsoft Visual C++*, il utilise la librairie *Boost Python*. *Microsoft Windows* est le système d'exploitation le plus courant dans les sociétés.

Première partie

# LE LANGAGE PYTHON

Les programmes informatiques sont souvent l'aboutissement d'un raisonnement, d'une construction, d'une idée, parfois imprécise mais dont le principe général est compris. Et pour valider cette idée, on utilise un langage de programmation qui ne tolère jamais l'imprécision, qui refuse de comprendre au moindre signe de ponctuation oublié, qui est d'une syntaxe si rigide.

On reste parfois indécis devant une erreur qui se produit avec un exemple qu'on a pourtant recopié sans ajout. On compare les deux versions sans faire attention aux petits détails qui ne changent rien pour le concepteur et beaucoup pour le langage. C'est un espace en plus ou en moins, un symbole : oublié... Il est fréquent de ne pas vérifier ces petits détails lorsqu'on commence à programmer. Et ils découragent souvent. On s'habitue peu à peu au fait qu'il ne faut pas confondre parenthèses et crochets, qu'il manque des guillemets, qu'il y a une lettre en plus, ou en moins.

Lorsque sa syntaxe n'est pas respectée, le langage de programmation ne cherche jamais à comprendre l'intention du programmeur. Il faut penser à la vérifier en premier lorsqu'une erreur se produit. Les messages d'erreur, obscurs au premier abord, donnent néanmoins un bon indice pour corriger un programme. Et si cela ne suffisait pas, il ne faut pas hésiter à recopier ce message dans un moteur de recherche sur Internet pour y trouver une indication dès les premiers résultats, et le plus souvent en français.

Cette première partie est consacrée à la description du langage *Python*. Il est parfois utile de reproduire un des exemples, de le faire fonctionner, de le modifier, d'y introduire des erreurs puis de les corriger. Plus tard, dans de plus longs programmes, les mêmes erreurs seront plus difficiles à déceler et l'expérience montre qu'on fait presque toujours les mêmes erreurs.

# Chapitre 2

# Introduction

---

Ce chapitre s'intéresse tout d'abord à l'installation du langage *Python* et à la réalisation d'un premier programme avec des instructions dont le sens est intuitif. Les derniers paragraphes présentent de nombreuses extensions disponibles sur Internet. Elles rendent le langage *Python* très attractif dans des domaines variés. Ces extensions témoignent que ce langage emporte l'adhésion de nombreux informaticiens qui en retour assurent sa pérennité. Il permet de relier facilement différents éléments, différentes applications. C'est une des raisons de son succès.

## 2.1 Ordinateur et langages

Il est rare aujourd'hui de ne pas avoir déjà entendu ou lu les termes informatiques définis ci-dessous. C'est un rapide rappel.

### 2.1.1 L'ordinateur

On peut considérer simplement qu'un ordinateur est composé de trois ensembles : le microprocesseur, la mémoire, les périphériques [1]. Cette description n'a pas varié en cinquante ans depuis qu'un scientifique du nom de von Neumann l'a imaginée.

Le microprocesseur est le cœur de l'ordinateur, il suit les instructions qu'on lui donne et ne peut travailler qu'avec un très petit nombre d'informations. Sa vitesse se mesure en GigaHertz (GHz) qui correspondent au nombre d'opérations qu'il est capable d'effectuer en une seconde et en nombre de cœurs qui détermine le nombre d'opérations en parallèle qu'il est capable d'exécuter. On lui adjoint une mémoire avec laquelle il échange sans arrêt des données. Sa capacité se mesure en octets (kilo-octets, mégaoctets, gigaoctets ou leurs abréviations Ko, Mo, Go [2]). Ces échanges entre processeur et mémoire sont rapides.

Les périphériques regroupent tout le reste (écran, clavier, souris, disque dur, imprimante...). Ils sont principalement de deux types : les périphériques de stockages (disque dur, DVD) et ceux qui nous permettent de dialoguer avec l'ordinateur, que ce soit pour afficher (écran, enceintes) ou pour recevoir (souris, clavier, webcam, micro...).

### 2.1.2 Termes informatiques

Certains termes reviendront fréquemment dans le livre. Ils sont souvent utilisés comme si tout le monde les connaissait comme la notion d'algorithme qui paraît plutôt abstraite si on se réfère à sa définition. Dans la pratique, on confond souvent algorithme avec programme informatique.

> **Définition 2.1 : algorithme**
> Un algorithme est une suite finie de règles à appliquer dans un ordre déterminé à un nombre fini de données pour arriver avec certitude, en un nombre fini d'étapes, à un certain résultat et cela, indépendamment des données.

1. écran, imprimante, disque dur, ...
2. Un kilo-octets équivaut à $1024 = 2^{10}$ octets, un Mo à 1024 Ko et un Go à 1024 Mo.

Leur écriture est indépendante du langage choisi, qu'il soit écrit en *Basic*, en *Pascal*, en *C*, en *Perl*, en *PHP*, en *Python*, en français, un algorithme reste le même. Pour les algorithmes simples (un tri par exemple), le passage d'un langage à l'autre consiste souvent à traduire mot-à-mot, ce qui favorise l'apprentissage d'un nouveau langage informatique lorsqu'on en connaît déjà un. Les différences entre ces langages résident dans leur spécialisation, le *Visual Basic* est réservé aux applications *Microsoft Office*, le *PHP* à la programmation Internet. Le langage *Python* ne l'est pas. Il est parfois moins efficace que chaque langage appliqué à son domaine de prédilection mais il évite l'apprentissage d'un syntaxe différente.

**Définition 2.2 : programme**
Un programme informatique est une suite d'instructions ou séquence d'instructions. C'est la réalisation informatique d'un ou plusieurs algorithmes. Il dépend du langage.

**Définition 2.3 : compilateur et compilation**
Le compilateur est un programme qui traduit un code écrit dans un langage de programmation en langage dit "machine", compréhensible par l'ordinateur. La compilation est le fait de traduire un programme afin que l'ordinateur le comprenne.

**Définition 2.4 : langage interprété**
Un langage interprété est converti en instructions propres à la machine au fur et à mesure de son exécution.

Le langage *Python* n'est pas un langage compilé car un programme *Python* n'est pas traduit en langage machine, il est un langage interprété. Entre son écriture et son exécution, il n'y a pas d'étape intermédiaire telle que la compilation et on peut ainsi tester un programme plus rapidement.

**Définition 2.5 : mot-clé**
Un mot-clé est une composante du langage et fait partie de sa grammaire qui comprend également les opérateurs numériques.

La table 4.1 (page 52) regroupe les mots-clés du langage *Python*. Elle contient peu de mots-clés : son concepteur s'est attaché à créer un langage objet avec la grammaire la plus simple possible.

**Définition 2.6 : instruction**
Ce terme est assez vague et dépend en général du langage. On peut considérer qu'une instruction est une expression syntaxiquement correcte pour un langage donné.

Les instructions sont très courtes et tiennent sur une ligne mais si on n'en prend qu'une partie, elle perd son sens. Ce serait comme considérer une phrase avec un verbe transitif mais sans son complément d'objet direct.

## 2.2 Présentation du langage *Python*

### 2.2.1 Histoire résumée

*Python* est un langage objet interprété de haut niveau, il a été créé au début des années quatre-vingt-dix par Guido Van Rossum. Entre 1995 et 2001, Rossum a changé plusieurs fois de travail tout en continuant l'élaboration du langage *Python*. En 2001, la PSF (Python Software Foundation) est créée. Il s'agit d'une organisation à but non lucratif détenant les droits de propriété intellectuelle de *Python*. Il est depuis distribué sous forme de logiciel libre. *Python* est couvert par sa propre licence [3]. Toutes les versions depuis

3. Voir le site http://www.python.org/psf/license/, cette licence autorise les usages commerciaux.

la 2.0.1 sont compatibles avec la licence GPL [4]. Selon Wikipedia [5], Rossum travaillerait maintenant pour l'entreprise *Google* qui propose une plate-forme de développement de sites web en *Python* [6] et qui héberge de nombreux projets en langage *Python* via son projet *Google Code* [7].

### 2.2.2 Le langage en quelques points

On distingue plusieurs classes parmi les langages informatiques selon la syntaxe qu'ils proposent ou les possibilités qu'ils offrent. *Python* est un langage : interprété, orienté objet, de haut niveau, modulaire, à syntaxe positionnelle, au typage dynamique.

Le langage *Python* est dit *interprété* car il est directement exécuté sans passer par une phase de compilation qui traduit le programme en langage machine, comme c'est le cas pour le langage *C*. En quelque sorte, il fonctionne autant comme une calculatrice que comme un langage de programmation. Afin d'accélérer l'exécution d'un programme *Python*, il est traduit dans un langage intermédiaire [8] qui est ensuite interprété par une machine virtuelle *Python*. Ce mécanisme est semblable à celui propre au langage *Java*.

Le langage est orienté objet car il intègre le concept de classe ou d'objet. Un objet regroupe un ensemble de données et un ensemble de fonctionnalités attachées à ces données. Ce concept relie la description des données et les algorithmes qui leur sont appliqués et les rend indissociables. Un objet est en quelque sorte une entité autonome avec laquelle on peut communiquer via une interface.

On considère que le langage *Python* est de haut niveau car il propose des fonctionnalités avancées et automatiques comme le *garbage collecting*. Cette tâche correspond à la destruction automatique des objets lorsqu'ils ne sont plus utilisés. Cette fonctionnalité est proposée par la plupart des langages de script ou encore *Java* mais pas *C++*. Il propose également des structures de données complexes telles que des dictionnaires, éloignées des types numériques standards.

Le langage *Python* est modulaire. La définition du langage est très succincte et autour de ce noyau concis, de nombreuses librairies ou modules ont été développées. *Python* est assez intuitif. Etre à l'aise avec ce langage revient à connaître tout autant sa syntaxe que les nombreux modules disponibles. Comme il est assez simple de construire un pont entre *Python* et *C++*, de nombreux projets open source antérieurs à *Python* sont maintenant accessibles dans ce langage et sous toutes les plateformes.

Le langage *Python* est à syntaxe positionnelle en ce sens que l'indentation fait partie du langage. Le point virgule permet de séparer les instructions en langage *C*, l'accolade permet de commencer un bloc d'instructions. En *Python*, seule l'indentation permet de marquer le début et la fin d'un tel bloc, ce procédé consiste à décaler les lignes vers la droite pour signifier qu'elles appartiennent au même bloc d'instructions et qu'elles doivent être exécutées ensemble. Cette contrainte rend les programmes *Python* souvent plus faciles à lire.

Le langage *Python* est à typage dynamique. Le type d'une variable, d'une donnée est définie lors de l'exécution du programme. Chaque information est désigné par un identificateur qui est manipulé tout au long du programme. Le fait que telle ou telle opération soit valide est vérifiée au moment où elle doit être réalisée et non au moment où le programme est traduit en langage machine.

### 2.2.3 Avantages et inconvénients du langage *Python*

Alors qu'il y a quelques années, le langage *C* puis *C++* s'imposaient souvent comme langage de programmation, il existe dorénavant une profusion de langages (*Java*, *JavaScript*, *PHP*, *Visual Basic*, *Perl*, *PHP*, ...). Il est souvent possible de transposer les mêmes algorithmes d'un langage à un autre. Le choix approprié est bien souvent celui qui offre la plus grande simplicité lors de la mise en œuvre d'un programme

4. Ou GNU Public Licence (voir le site http://www.gnu.org/copyleft/gpl.html).
5. httpstylehttp ://fr.wikipedia.org/wiki/Python_(langage)
6. http://code.google.com/appengine/
7. http://code.google.com/
8. Celui-ci est appelé *bytecode*. Ce mot ne signifie pas grand chose excepté que ce *bytecode* est seulement compréhensible par la machine virtuelle *Python*.

même si cette simplicité s'acquiert au détriment de la vitesse d'exécution. Le langage *PHP* est par exemple très utilisé pour la conception de sites Internet car il propose beaucoup de fonctions très utiles dans ce contexte bien que, dans ce domaine, le langage *Python* soit une solution alternative très intéressante.

Comme la plupart des langages, le langage *Python* est tout d'abord portable puisqu'un même programme peut être exécuté sur un grand nombre de systèmes d'exploitation comme *Linux*, *Microsoft Windows*, *Mac OS X*... *Python* possède également l'avantage d'être entièrement gratuit tout en proposant la possibilité de pouvoir réaliser des applications commerciales à l'aide de ce langage.

Si le langage *C* reste le langage de prédilection pour l'implémentation d'algorithmes complexes et gourmands en temps de calcul ou en capacités de stockage, un langage tel que *Python* suffit dans la plupart des cas. De plus, lorsque ce dernier ne convient pas, il offre toujours la possibilité, pour une grande exigence de rapidité, d'intégrer un code écrit dans un autre langage tel que le *C/C++* [9] ou *Java*, et ce, d'une manière assez simple. La question du choix du langage pour l'ensemble d'un projet est souvent superflue, il est courant aujourd'hui d'utiliser plusieurs langages et de les assembler. Le langage *C* reste incontournable pour concevoir des applications rapides, en revanche, il est de plus en plus fréquent d'"habiller" un programme avec une interface graphique programmée dans un langage tel que *Python*.

En résumé, l'utilisation du langage *Python* n'est pas restreinte à un domaine : elle comprend le calcul scientifique, les interfaces graphiques, la programmation Internet. Sa gratuité, la richesse des extensions disponibles sur Internet le rendent séduisant dans des environnements universitaire et professionnel. Le langage est vivant et l'intérêt qu'on lui porte ne décroît pas. C'est un critère important lorsqu'on choisit un outil parmi la myriade de projets open source. De plus, il existe différentes solutions pour améliorer son principal point faible qui est la vitesse d'exécution.

## 2.3 Installation du langage *Python*

### 2.3.1 Installation du langage *Python*

*Python* a l'avantage d'être disponible sur de nombreuses plates-formes comme *Microsoft Windows*, *Linux* ou *Mac OS X*. L'installation sous *Windows* est simple. Le langage *Python* est déjà intégré aux systèmes d'exploitation *Linux* et *Mac OS X*.

Sous *Microsoft Windows*, il suffit d'exécuter le fichier *Python-2.5.2.msi* [10] ou tout autre version plus récente. En règle générale, il est conseillé de télécharger la dernière version stable du langage, celle avec laquelle le plus grande nombre d'extensions seront compatibles. Les options d'installation choisies sont celles par défaut, le répertoire d'installation est par défaut C:/Python25. A la fin de cette installation apparaît un menu supplémentaire dans le menu *Démarrer* (ou *Start*) de *Microsoft Windows* comme le montre la figure 2.1. Ce menu contient les intitulés suivant :

| | |
|---|---|
| IDLE (Python GUI) | éditeur de texte, pour programmer |
| Module Docs | pour rechercher des informations dans la documentation |
| Python (command line) | ligne de commande *Python* |
| Python Manuals | documentation à propos du langage *Python* |
| Uninstall Python | pour désinstaller *Python* |

La documentation (de langue anglaise [11]) décrit en détail le langage *Python*, elle inclut également un tutoriel [12] qui permet de le découvrir. La ligne de commande (voir figure 2.2) permet d'exécuter des

9. Ce point est d'ailleurs abordé au paragraphe 7.6, page 144.
10. Cette version est disponible à l'adresse http://www.python.org/download/ depuis février 2008, la version 2.6 est disponible depuis octobre 2008, la documentation associée est fournie à l'adresse http://docs.python.org/download.html. C'est la version utilisée pour développer les exemples présents dans ce livre. Il est conseillé d'attendre quelques mois avant d'installer une nouvelle version afin que les extensions aient eu le temps d'être mises à jour.
11. http://www.python.org/doc/
12. La requête *python français* de nombreux site de documentation sur *Python* en français.

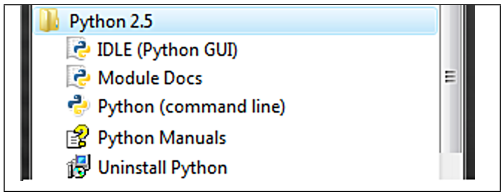



**Figure 2.1 :** *Menu ajouté à Microsoft Windows.*

instructions en langage *Python*. Elle est pratique pour effectuer des calculs mais il est nécessaire d'utiliser un éditeur de texte pour écrire un programme, de le sauvegarder, et de ne l'exécuter qu'une fois terminé au lieu que chaque ligne de celui-ci ne soit interprétée immédiatement après qu'elle a été écrite comme c'est le cas pour une ligne de commande.

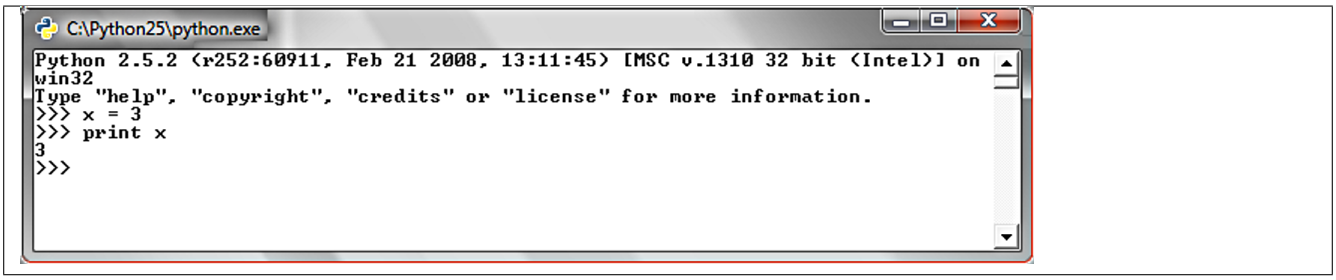



**Figure 2.2 :** *Ligne de commande, la première ligne affecte la valeur 3 à la variable* `x`*, la seconde ligne l'affiche.*

### 2.3.2 Particularité de *Mac OS X* et *Linux*

Le langage *Python* est déjà présent sur les ordinateurs d'*Apple* si ceux-ci sont équipés du système d'exploitation *Mac OS X*. Il est également présent dans les distributions *Linux*. L'installation de *Python* est parfois rendue nécessaire si on désire utiliser la dernière version du langage. L'installation des extensions du langage est similaire à celle d'autres extensions *Linux*.

Il ne faut pas oublier de vérifier la version du langage *Python* installée sur l'ordinateur *Linux* et *Mac OS X*. Il suffit pour cela de taper la ligne `help()` pour en prendre connaissance. Cela signifie que toutes les extensions qui doivent être installées doivent l'être pour cette version qu'il est néanmoins possible de mettre à jour cette version [13].

### 2.3.3 Utilisation de l'éditeur de texte

La figure 2.1 montre le menu installé par *Python* dans le menu "Démarrer" de *Microsoft Windows*. En choisissant l'intitulé "IDLE (Python GUI)", on active la fenêtre de commande de *Python*(voir figure 2.3). Les instructions sont interprétées au fur et à mesure qu'elles sont tapées au clavier. Après chaque ligne, cette fenêtre de commande conserve la mémoire de tout ce qui a été exécuté. Par exemple :

```
>>> x = 3
>>> y = 6
>>> z = x * y
>>> print z
18
>>>
```

13. voir la page http://www.python.org/download/mac/ ou http://www.python.org/download/linux/

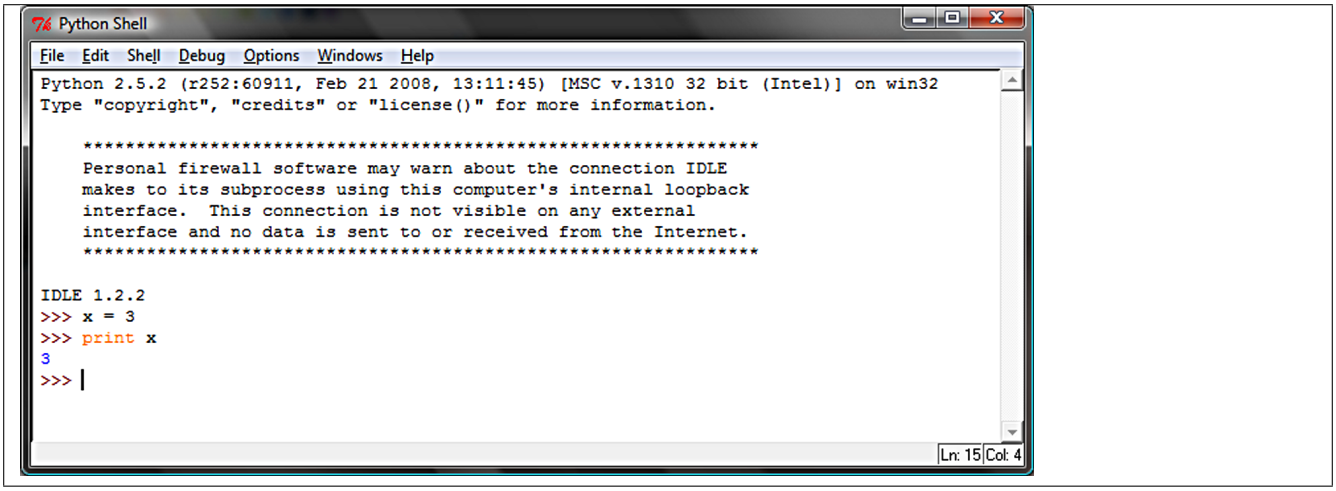



**Figure 2.3 :** *Fenêtre de commande fournie avec le langage Python.*

Après l'exécution de ces quelques lignes, les variables [14] `x`, `y`, `z` existent toujours. La ligne de commande ressemble à une calculatrice améliorée. Pour effacer toutes les variables créées, il suffit de redémarrer l'interpréteur par l'intermédiaire du menu `Shell`–>`Restart Shell`. Les trois variables précédentes auront disparu.

Il n'est pas possible de conserver le texte qui a été saisi au clavier, il est seulement possible de rappeler une instruction déjà exécutée par l'intermédiaire de la combinaison de touches `ALT` + `p`, pressée une ou plusieurs fois. La combinaison `ALT` + `n` permet de revenir à l'instruction suivante. Pour écrire un programme et ainsi conserver toutes les instructions, il faut d'actionner le menu `File`–>`New Window` qui ouvre une seconde fenêtre qui fonctionne comme un éditeur de texte (voir figure 2.4).

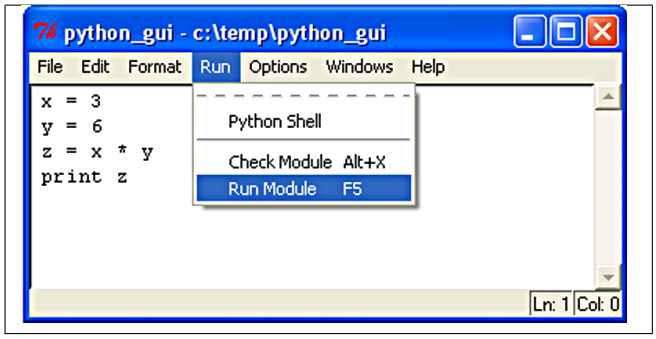



**Figure 2.4 :** *Fenêtre de programme, le menu* `Run`–>`Run Module` *permet de lancer l'exécution du programme. L'interpréteur Python est réinitialisé au début de l'exécution et ne conserve aucune trace des travaux précédents.*

Après que le programme a été saisi, le menu `Run`–>`Run Module` exécute le programme. Il demande au préalable s'il faut enregistrer puis réinitialise l'interpréteur *Python* pour effacer les traces des exécutions précédentes. Le résultat apparaît dans la première fenêtre (celle de la figure 2.3). La pression des touches "Ctrl + C" permet d'arrêter le programme avant qu'il n'arrive à sa fin.

**Remarque 2.7 : fenêtre intempestive**

Si on veut se débarrasser de cette fenêtre intempestive qui demande confirmation pour enregistrer les dernières modifications, il suffit d'aller dans le menu `Options`–>`Configure IDLE` et de choisir `No prompt` sur la quatrième ligne dans la rubrique `General` (voir figure 2.5). Ce désagrément apparaît sur la plupart des éditeurs de texte et se résout de la même manière.

14. Les variables sont définies au chapitre 3. En résumé, on peut considérer une variable comme une lettre, un mot qui désigne une information. Dans cet exemple, les trois variables désignent des informations numériques qu'on peut manipuler comme si elles étaient effectivement des nombres. Les variables supprimées, les informations qu'elles désignaient ne plus accessibles et sont perdues également.

Cette description succincte permet néanmoins de réaliser puis d'exécuter des programmes. Les autres fonctionnalités sont celles d'un éditeur de texte classique, notamment la touche `F1` qui débouche sur l'aide associée au langage *Python*. Il est possible d'ouvrir autant de fenêtres qu'il y a de fichiers à modifier simultanément.

Néanmoins, il est préférable d'utiliser d'autres éditeurs plus riches comme celui proposé au paragraphe 2.4.1. Ils proposent des fonctions d'édition plus évoluées que l'éditeur installé avec *Python*.

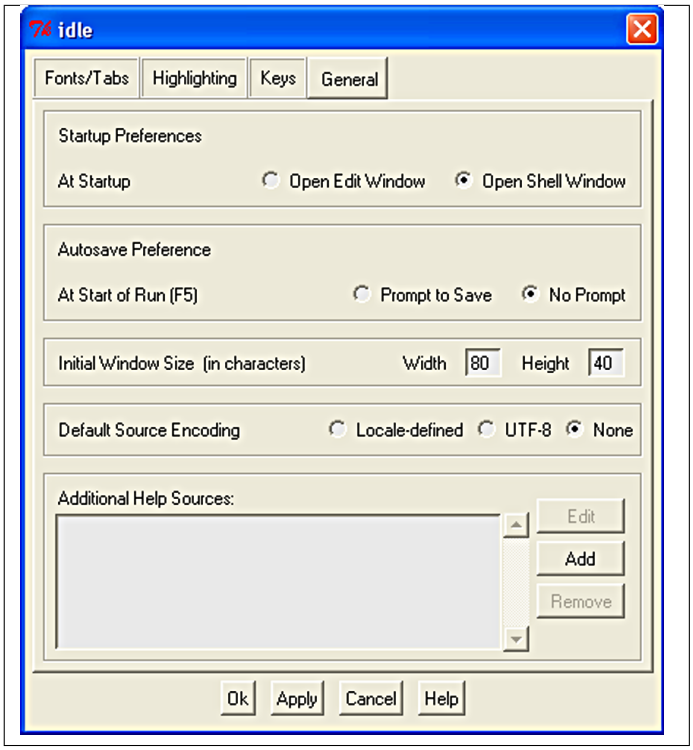



**Figure 2.5 :** *Cliquer sur* `General` *puis sur* `Noprompt` *pour se débarrasser de la fenêtre intempestive qui demande à l'utilisateur de confirmer la sauvegarde des dernières modifications. D'autres éditeurs de texte se comportent de la même manière et disposent d'une option similaire.*

## 2.4 Installation d'un éditeur de texte

### 2.4.1 Editeur SciTe

Il existe de nombreux éditeurs de texte, payants ou gratuits. Le premier proposé dans ce livre est *SciTe*, il est gratuit et fonctionne avec de nombreux compilateurs de langages différents (voir figure 2.6). L'éditeur *SciTe*[15] est léger et son utilisation ne nécessite pas de droits spécifiques[16]. La gestion des erreurs est également plus pratique puisqu'il est possible de cliquer sur une erreur pour que l'éditeur positionne automatiquement le curseur sur la ligne ayant généré cette erreur.

Une fois installé, il faut configurer l'éditeur de manière à ce qu'il puisse utiliser le compilateur *Python*. Il faut cliquer sur le menu `Option` et choisir la rubrique `Open python.properties` (voir figure 2.6).

Après avoir sélectionné ce menu, un fichier texte s'ouvre. Vers la fin, on trouve les lignes suivantes qui contiennent la ligne de commande qui permet d'exécuter un programme *Python* :

```
if PLAT_WIN
    command.go.*.py=python -u "$(FileNameExt)"
    command.go.subsystem.*.py=0
```

15. Cet éditeur est accessible depuis l'adresse http://www.scintilla.org/SciTE.html et téléchargeable depuis l'adresse http://scintilla.sourceforge.net/SciTEDownload.html.

16. Il arrive fréquemment que des universités, des sociétés interdisent à leurs utilisateurs l'installation de nouveaux logiciels sur leurs machines pour des raisons de sécurité. Il faut des droits spécifiques pour contourner cette interdiction.

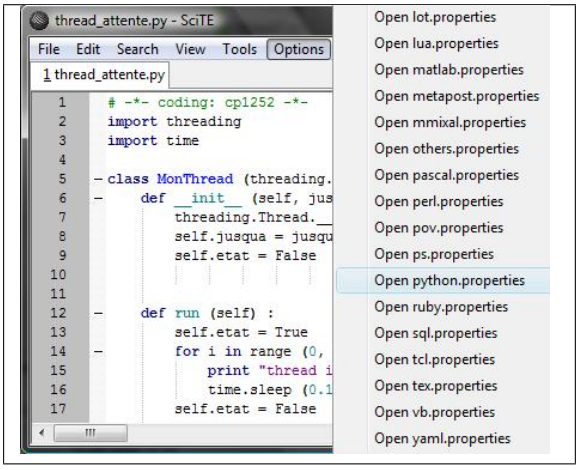



**Figure 2.6 :** *Configuration de l'éditeur de texte SciTe. Il faut cliquer sur le menu* `Option`*, puis sur* `Open python.properties` *pour ouvrir les paramètres de configuration associées au langage Python.*

```
    command.go.*.pyw=pythonw -u "$(FileNameExt)"
    command.go.subsystem.*.pyw=1
```

Il suffit de préciser le répertoire où se trouve l'interpréteur qui devrait être `c : /python25` sur *Microsoft Windows* si le répertoire d'installation par défaut du langage *Python* n'a pas été modifié.

```
if PLAT_WIN
    command.go.*.py=c:\python25\python -u "$(FileNameExt)"
    command.go.subsystem.*.py=0
    command.go.*.pyw=c:\python25\pythonw -u "$(FileNameExt)"
    command.go.subsystem.*.pyw=1
```

Une fois ce fichier enregistré, il ne reste plus qu'à écrire un programme *Python*. Une fois terminé, la touche `F5` lance son exécution. L'exemple de la figure 2.7 montre un cas où l'exécution a été interrompue par une erreur.

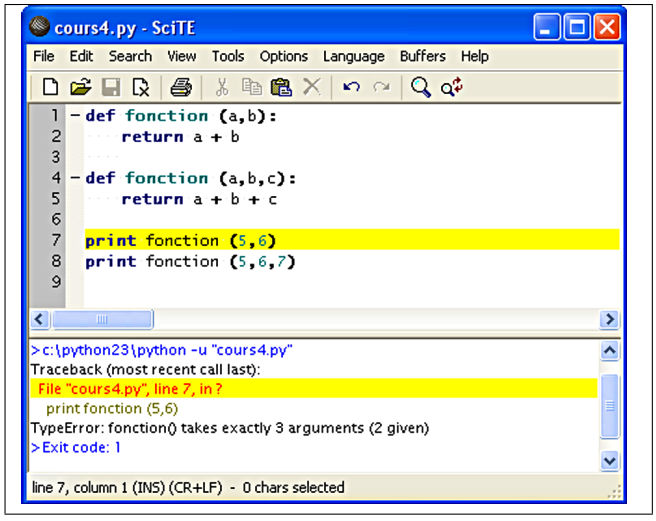



**Figure 2.7 :** *Exécution d'un programme interrompue par une erreur. Pour cet éditeur, il suffit de cliquer sur le message d'erreur pour placer le curseur sur la ligne erronée.*

Le site Internet de *SciTe* propose quelques trucs et astuces pour configurer l'éditeur à votre convenance. Comme à peu près tous les logiciels, il dispose d'une rubrique FAQ [17] (Frequently Asked Questions) qui recense les questions et les réponses les plus couramment posées. Cette rubrique existe pour la plupart des éditeurs et plus généralement pour la plupart des applications disponibles sur Internet.

17. http://scintilla.sourceforge.net/SciTEFAQ.html

### 2.4.2 Python Scripter

Le second éditeur proposé dans ce livre est *Python Scripter*[18] (voir figure 2.8). Il est plus lourd mais offre plus de fonctionnalités notamment un débugger qui permet d'exécuter pas à pas un programme afin de vérifier chaque ligne de son exécution. Lorsqu'un programme produit de mauvais résultats, la première solution consiste à afficher de nombreux résultats intermédiaires mais cela nécessite d'ajouter des instructions un peu partout dans le programme. La seconde méthode est l'utilisation du débugger. Sans modifier le code informatique, il est possible d'arrêter temporairement l'exécution à une ligne donnée pour inspecter le contenu des variables à cet instant de l'exécution. Ce logiciel fonctionne sans autre manipulation supplémentaire autre que son installation.

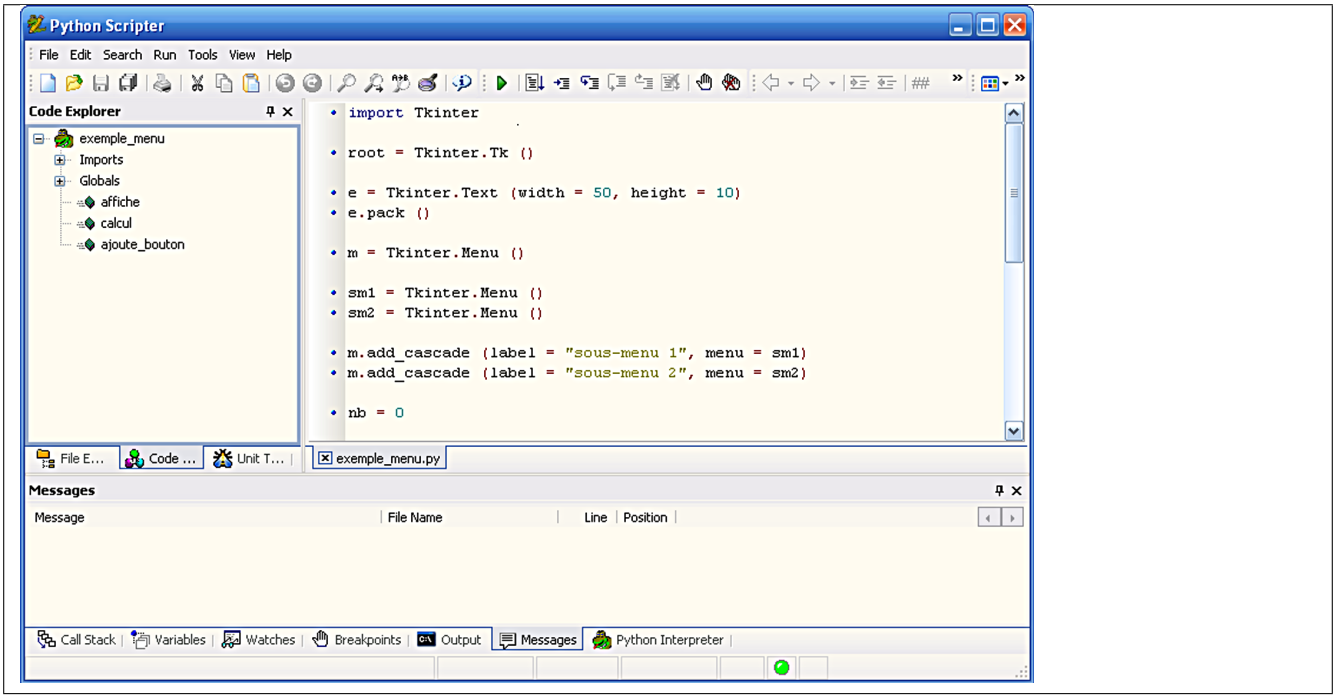



**Figure 2.8 :** *Copie d'écran de l'éditeur Python Scripter.*

Cet éditeur, contrairement à *SciTe*, ne fonctionne qu'avec *Python* et ne peut être utilisé avec d'autres langages. Son apparence est adaptée au développement. Il permet également de voir la liste des variables, fonctions, classes, présentes dans un programme.

### 2.4.3 Eclipse et PyDev

*Eclipse*[19] est un environnement complet de développement adapté à une multitude de langages dont *Python* via une extension appelé *PyDev*[20]. Cette solution s'adresse plus à des gens ayant déjà une expérience en informatique ou souhaitant construire un projet conséquent. En constante évolution, *Eclipse* est un éditeur très complet mais son utilisation nécessite un apprentissage non négligeable qui n'est pas indispensable pour des projets de petite taille.

18. http://mmm-experts.com/
19. http://www.eclipse.org/
20. http://pydev.sourceforge.net/

### 2.4.4 Côté *Mac OS X* : *TextWrangler*

Un éditeur possible est *TextWrangler*[21] qui est gratuit. Lorsqu'on programme à la fois sous *Microsoft Windows* et *Linux* ou *Mac OS X*, les fichiers contiennent des codes de fins de lignes différents et l'éditeur vous le fait remarquer à chaque exécution. Pour éviter ce désagrément, il convient d'écraser le fichier contenant le programme en le sauvegardant à nouveau en prenant soin de choisir des fins de lignes identiques à celles du monde *Linux*. *Linux* fait également la différence entre les minuscules et les majuscules en ce qui concerne les noms de fichiers contrairement à *Microsoft Windows*.

### 2.4.5 Autres éditeurs

Certains éditeurs de texte sont eux-mêmes écrits en *Python*, ce qui leur permet de fonctionner sur toutes les plates-formes. Trois sont assez connues, *DrPython*[22], *Boa Constructor*[23], *The Eric Python IDE*[24]. Ce dernier s'appuie sur une interface graphique de type `pyQt`[25], son interface est riche, presque trop lorsqu'on commence à programmer. *Boa Constructor* s'appuie sur `wxPython` et offre des outils permettant de dessiner des interfaces graphiques. *DrPython* est le plus simple des trois.

Certains éditeurs connus du monde *Linux* ont été déclinés pour le monde *Windows*, c'est le cas de *Emacs*[26]. *SciTe* est souvent utilisé car c'est le plus simple. Les autres offrent des fonctionnalités variées dont l'utilité n'apparaît que pour des programmes conséquents. Le site officiel[27] répertorie d'autres éditeurs de texte dont certains sont de véritables environnements de développements. Il n'est pas aberrant d'en utiliser plus d'un car chacun d'eux est plus approprié dans telle situation ou pour tel type de programme.

### 2.4.6 Tabulations

De nombreux éditeurs utilisent les tabulations pour mettre en forme un programme et tenir compte des décalages (indentation) d'un bloc par rapport à un autre. Il est conseillé de remplacer ces tabulations par des espaces, cela évite les confusions et les mauvaises interprétations lorsque tabulations et espaces sont mélangés. La taille d'une tabulation peut varier d'un éditeur à l'autre contrairement aux espaces.

## 2.5 Premier programme

Après avoir installé le langage *Python* ainsi qu'un éditeur, il suffit d'écrire quelques lignes pour vérifier que cette première étape s'est achevée correctement. Tous les programmes *Python* porte l'extension `.py`.

### 2.5.1 Afficher le résultat d'un calcul

Le programme suivant, qu'on peut appeler `premier.py`, affiche le message `premier message` une fois qu'il est exécuté. Le mot-clé `print` précédant le message - toujours entre guillemets ou entre apostrophes sauf si ce sont des nombres - stipule qu'il faut afficher quelque chose.

```
print "premier message"
```

21. http://www.barebones.com/products/textwrangler/
22. http://drpython.sourceforge.net/
23. http://boa-constructor.sourceforge.net/
24. http://www.die-offenbachs.de/eric/index.html
25. Voir le paragraphe 2.6.5 sur les interfaces graphiques.
26. http://www.gnu.org/software/emacs/windows/
27. plus précisément à l'adresse http://wiki.python.org/moin/PythonEditors ou encore à l'adresse http://wiki.python.org/moin/IntegratedDevelopmentEnvironments

Un programme ne barbouille pas l'écran de texte indésirable, il n'affiche que ce que le programmeur lui demande d'afficher et cette intention se matérialise grâce à l'instruction `print`. On peut ensuite écrire quelques lignes qu'on pourrait également taper sur une calculatrice :

```
print 3.141592 * 5*5
print 3.141592 * 5**2
```

Ces deux lignes affichent 78.5398 qui est la surface d'un cercle de 5 mètres de côté. Le symbole $**$ correspond à l'opération puissance. Il existe une constante PI ou $\pi$ mais qui ne fait pas partie du langage lui-même : elle est définie dans une extension interne [28] du langage qui contient bon nombre de fonctions mathématiques telles que les cosinus, sinus...

```
from math import *  # permet d'utiliser les extensions mathématiques de Python
print pi * 5**2
```

### 2.5.2 Les accents

Le langage *Python* est conçu pour le monde anglophone et l'utilisation des accents ne va pas de soi. Le programme suivant qui demande d'afficher un message contenant un accent provoque l'apparition d'une erreur :

```
print "accentué"
```

L'erreur est la suivante :

```
  File "essai.py", line 1
SyntaxError: Non-ASCII character '\xe9' in file i.py on line 1,
             but no encoding declared;
             see http://www.python.org/peps/pep-0263.html for details
```

Dans ce cas, il faut ajouter une ligne placée en première position qui précise que des accents pourront être utilisés. L'exécution du programme qui suit ne soulève aucune erreur.

```
# -*- coding: latin-1 -*-
print "accentué"
```

Cette ligne précise en fait que l'interpréteur *Python* doit utiliser un jeu de caractères spécial. Ces jeux de caractères ont parfois leur importance, les navigateurs n'aiment pas trop les accents au sein des adresses Internet : il est parfois préférable de ne pas utiliser d'accents sur les sites de réservations d'hôtels, de trains ou d'avions.

Par défaut, l'interpréteur *Python* utilise le jeu de caractères le plus réduit, il faut lui préciser qu'il doit en utiliser un plus étendu incluant les accents. On trouve également sous *Microsoft Windows* la syntaxe équivalente suivante :

```
# -*- coding: cp1252 -*-
print "accentué"
```

Enfin, s'il fallait utiliser des caractères autres que latins, il faudrait alors choisir le jeu de caractères `utf − 8` qui s'appliquent aussi aux langues orientales.

```
# -*- coding: utf-8 -*-
print "accentué"
```

28. Cette extension est installée automatiquement avec *Python* contrairement à une extension externe.

A cause des accents, la plupart des exemples cités dans ce livre ne fonctionnent pas sans cette première ligne qui a parfois été enlevée lors de l'impression pour des questions de lisibilité. Il faut penser à l'ajouter pour reproduire les exemples. Le paragraphe 8.8 page 173 est plus complet à ce sujet.

**Remarque 2.8 :** #!/usr/local/bin/python
Sur Internet, on trouve de nombreux exemples commençant par la ligne suivante :

```
#!/usr/local/bin/python
```

On rencontre souvent cette ligne pour un programme écrit sur une machine *Linux*, elle indique l'emplacement de l'interpréteur *Python* à utiliser.

## 2.6 Installation d'extensions (ou modules externes)

Les extensions ou modules ou packages sont des bibliothèques de fonctions non standard. Ce sont souvent des programmes existant conçus dans un autre langage tel que *C++* et qui ont été "coiffés" d'une interface permettant de les utiliser sous *Python*. Les paragraphes qui suivent décrivent quelques extensions couramment utilisées. Certains besoins reviennent fréquemment comme dessiner un graphe, charger une image, faire du calcul matriciel et *Python* seul ne permet pas de répondre à ce problème.

La première information à connaître avant d'installer un package est la version du langage *Python* installée sur votre ordinateur. Il faut impérativement utiliser la version du module élaborée pour la même version [29]. Il est nécessaire de connaître également le système d'exploitation de votre ordinateur.

Par conséquent, il ne faut pas installer le module `scipy` (calcul scientifique) élaboré pour *Python* 2.5 alors que sur votre ordinateur est installée la version 2.4, il y a de fortes chances que cela ne fonctionne pas. Certains modules dépendent d'autres, lors de leur installation, il faut vérifier que les modules dont ils dépendent ont été préalablement installés.

Le site officiel [30] recense une longue liste de modules existants (plusieurs centaines). Il recense également les dernières mises à jour. Un autre site fait de même, *Vault Parnassus* [31], avec une présentation différente. Avant de commencer à programmer, il est conseillé d'aller chercher sur ces sites si un module n'aurait pas déjà fait une partie du travail.

Il est parfois intéressant de consulter des forums d'échanges entre programmeurs autour de *Python*. Il est parfois plus intéressant de jeter un œil sur de tels sites [32] avant de se lancer dans la recherche d'un module spécifique. Internet fournit également de nombreux exemples de programmes dont on peut s'inspirer.

Les modules cités dans ce paragraphe ne sont pas installés automatiquement avec le langage *Python* mais sont parfois très utiles dans le cadre d'un travail d'ingénieur. Ils existent depuis plusieurs années et sont pour la plupart régulièrement maintenus à jour. Ils sont tous facilement accessibles depuis un moteur de recherche.

### 2.6.1 Graphisme

`matplotlib` [33] : ce module reprend la même logique que le logiciel *MatLab* ou *Octave*. Pour ceux que la finance intéresse, il comporte également des fonctions qui permettent de récupérer des historiques quotidiens de cours d'actions depuis le site *Yahoo ! Finance* [34].

`biggles` [35] : ce module permet de dessiner des courbes (2D, 3D) et de convertir des graphiques sous

29. L'instruction `help`() saisie sur la ligne de commande *Python* retourne entre autres le numéro de version.
30. http://pypi.python.org/pypi
31. http://www.vex.net/parnassus/
32. On peut citer http://wikipython.flibuste.net/ ou http://www.afpy.org/, deux sites français qui regroupent des informations intéressantes sur de nouveaux modules ou des exemples d'utilisation de modules courants.
33. http://matplotlib.sourceforge.net/
34. Avec la fonction `fetch_historical_yahoo`.
35. http://biggles.sourceforge.net/

forme d'images de tout format. Ce module nécessite l'installation de fichiers supplémentaires (libpng, zlib, plotutils) [36].

`chartdirector` [37] : beaucoup plus rapide que MatPlotLib, ce module est gratuit à condition d'accepter un bandeau jaune en bas de chaque graphique.

`pydot` [38] : ce module propose le dessin de graphes dans une image (arbre généalogique, réseau de distribution, ...). Le module `pydot` permet de dessiner tout type de graphe. Ce module nécessite un autre module `pyparsing` [39] et une application *Graphviz* [40]. Avec cet outil, on peut réaliser des graphes comme celui de la figure 2.9. Ce module est utilisé lors de l'exercice 11.5.1 page 234.

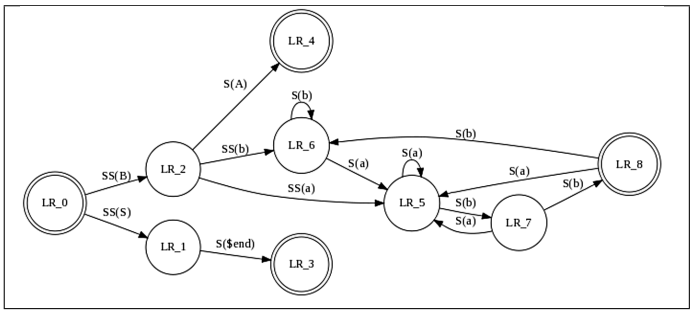



**Figure 2.9 :** *Image d'un graphe, extraite du site http ://www.graphviz.org/ où est disponible l'outil Graphviz. Il permet de représenter des graphes sur une page en minimisant les croisements d'arcs ce qui est un problème complexe.*

`pyx` [41] : ce module est aussi une librairie qui permet de tracer des graphes avec la possibilité de convertir directement ceux-ci en fichiers PDF ou EPS.

Sans utiliser nécessairement une extension, il existe des outils comme *GNUPlot* [42] venu du monde *Linux* qui s'exécutent facilement en ligne de commande. Cet outil propose son propre langage de programmation dédié à la création de graphiques. Une fois ce langage maîtrisé, il est facile de construire un programme qui permet de l'exécuter en ligne de commande pour récupérer l'image du graphique.

**Remarque 2.9 : sortie texte et graphique**
Un des moyens simples de construire un rapport mélangeant résultats numériques et graphiques est de construire un fichier au format HTML [43] puis de l'afficher grâce à un navigateur Internet. Il est même possible de construire un fichier au format *PDF* en utilisant l'outil *Latex* [44].

### 2.6.2 Calculs scientifiques

`scipy` [45] : ce module propose de nombreux algorithmes, algorithmes de classification, intégration, optimisation, génération de nombres aléatoires, traitement du signal, interpolation, algorithmes génétiques, algèbre linéaire, transformée de Fourier, dessin de courbes 2D, 3D. Son installation requiert d'abord celle du module `numpy`.

`numpy` [46] : ce module manipule efficacement les tableaux de grandes dimensions, intègre le calcul matriciel, la transformée de Fourier.

36. Accessibles depuis l'adresse http://www.gnu.org/software/plotutils/, ces informations sont disponibles depuis la page de téléchargement de `biggles`.
37. http://www.advsofteng.com/
38. http://code.google.com/p/pydot/
39. http://pyparsing.wikispaces.com/
40. http://www.graphviz.org/
41. http://pyx.sourceforge.net/
42. http://www.gnuplot.info/
43. Il est facile de trouver des pages Internet décrivant cette syntaxe via un moteur de recherche. L'exercice 7, page 239 en dira plus à ce sujet.
44. voir l'exercice 7, page 240
45. http://www.scipy.org/
46. http://numpy.scipy.org/

`orange`[47] : c'est un module qui permet de faire ce qu'on appelle du *datamining* ou fouille de données. Il propose de nombreux algorithmes de classifications, d'analyse des données qui ont trait au *machine learning*.

`mdp`[48] : ce module offre quelques fonctionnalités relatives à l'analyse des données tels que l'analyse en composante principale (ACP).

`Rpy`[49] : le langage *R*[50] est un langage dédié aux calculs statistiques. Il est optimisé pour la manipulation de grandes matrices et des calculs statistiques et matriciels rapides. Il dispose également de nombreux graphiques. Il faut au préalable installer *Python* et *R* puis le module `Rpy` fera la liaison entre les deux.

### 2.6.3 Images

`PythonImageLibrary(PIL)`[51] : ce module permet d'utiliser les images, quelque soit leur format. Elle propose également quelques traitements d'images simples, rotation, zoom, histogrammes.

### 2.6.4 Base de données

**SQLObject**[52] : ce module offre la possibilité de se connecter à la plupart des serveurs de bases de données et d'y accéder via une syntaxe objet plutôt qu'au travers de requêtes *SQL*.

**SQLAlchemy**[53] : ce module propose les mêmes fonctionnalités que le précédent. C'est le plus riche des deux. Il est souvent utilisé en association avec le module `elixir`[54].

### 2.6.5 Interfaces graphiques

Le langage *Python* propose un module interne permettant de réaliser des interfaces graphiques : `Tkinter`. Il suffit pour des applications simples qui ne nécessitent que quelques fenêtres. Pour des applications plus ambitieuses qui doivent afficher des données sous forme de matrices par exemple ou pour obtenir un aspect plus professionnel, il est impératif d'utiliser un module externe. Il est néanmoins conseillé de savoir utiliser `Tkinter` et de comprendre la programmation événementielle avant de se lancer dans l'apprentissage de `wxPython`, `pyQt` ou `pyGTK`. Ces trois modules proposent des fonctionnalités équivalentes, le choix porte surtout sur des besoins précis qu'un module couvre et sur des questions d'esthétisme.

`wxPython`[55] : il propose des interfaces graphiques plus complètes que le module standard `Tkinter` comme une fenêtre tableur. Utilisé par de nombreuses applications, il est régulièrement mis à jour. Il est entièrement gratuit et peut servir à des applications commerciales. Il s'inspire des *MFC* ou *Microsoft Fundation Class* issues de *Microsoft* mais a l'avantage d'être portable. Aujourd'hui, c'est avant tout un projet open source en constante évolution.

`pyQt`[56] : il est moins répandu que `wxPython` sans doute car il est produit par une société et n'est pas open source. Il est réputé pour être mieux structuré. Il existe des restrictions pour la conception d'applications commerciales.

`pyGTK`[57] : il est plus proche de `Tkinter` que les deux autres modules. Venu du monde *Linux*, l'aspect de l'interface lui est plus ressemblant qu'à celui du monde *Windows*.

47. http://www.ailab.si/orange/
48. http://mdp-toolkit.sourceforge.net/
49. http://rpy.sourceforge.net/
50. http://www.r-project.org/
51. http://www.pythonware.com/products/pil/
52. http://www.sqlobject.org/
53. http://www.sqlalchemy.org/
54. http://elixir.ematia.de/trac/
55. http://www.wxpython.org/
56. http://www.riverbankcomputing.co.uk/software/pyqt/intro
57. http://www.pygtk.org/

Lorsqu'on souhaite réaliser un jeu, il est nécessaire d'avoir une interface graphique plus conciliante avec la gestion des images, des dessins, des sons voire des vidéos. C'est le cas du module `pygame` qui fonctionne toujours selon la programmation événementielle. Il n'est pas prévu pour réaliser des jeux en 3D mais suffit pour bon nombre de jeux de plateau, jeux de réflexion.

`pygame` [58] : ce module est prévu pour concevoir des jeux en *Python*. Il permet entre autres d'écouter les morceaux enregistrés sur un CD, d'afficher un film et également d'afficher des images, de récupérer les touches pressées au clavier, les mouvements de la souris, ceux de la manette de jeux. Il suffit d'une heure ou deux pour écrire ses premiers programmes avec `pygame`.

Pour la réalisation d'images et d'animations 3D, on peut utiliser le module `pyOpenGL` [59] qui s'interface avec `pygame`, `wxPython` ou `pyQt`. C'est un module dont la maîtrise ne se compte plus en heures mais en dizaine d'heures tout comme le module `directpython` [60] qui est l'équivalent de `pyOpenGL` mais uniquement sous *Microsoft Windows*.

### 2.6.6 *Microsoft Windows*, *Microsoft Excel*, PDF

`win32com` [61] : ce module permet de communiquer avec les applications *Microsoft Windows* tels que *Microsoft Excel*. Il est ainsi possible de créer une feuille *Excel* depuis un programme *Python*, de l'imprimer, de la sauvegarder sans avoir à ouvrir *Excel*. Un moteur de recherche permettra de trouver de nombreux petits exemples utilisant ce module [62].

`pypdf` [63] : ce module permet de manipuler les fichiers PDF, autant pour lire les informations qu'il contient que pour les modifier.

### 2.6.7 Accélération de *Python*

*Python* est un langage interprété et pour cette raison, il est lent ce qui ne veut pas dire qu'il n'existe pas des solutions pour contourner ce problème.

`psyco` : ce module permet de multiplier la vitesse d'exécution d'un programme par trois ou quatre. Il suffit d'ajouter au programme les instructions suivantes :

```
import psyco
psyco.full ()
```

Dans les faits, le programme s'exécute plus vite mais pas uniformément plus vite : les calculs sont le plus favorisés. Toutefois, même avec ce module, le langage *Python* reste très lent par rapport à un langage compilé comme le *C++*. L'outil *Pyrex* [64] propose une solution en permettant l'insertion de code *C++* au sein d'un programme *Python*. *Pyrex* n'est pas un module, c'est un interpréteur *Python* modifié pour prendre en compte ces insertions *C++*. Il est essentiel de connaître la syntaxe du langage *C++*. *Pyrex* propose des temps d'exécution tout à fait acceptable par rapport à ceux du *C++* et il accroît fortement la vitesse d'écriture des programmes. Son installation n'est pas toujours évidente car il faut le relier au compilateur *C++ GCC* [65]. Une approche semblable est développée par l'extension `weave` [66] du module `scipy` [67]. *Pyrex* et *Weave* offre des performances bien supérieures à *Python* seul, proches de celle du langage *C*. *Weave* conviendra sans doute mieux à ceux qui ont déjà programmé en *C*.

58. http://www.pygame.org/
59. http://pyopengl.sourceforge.net/
60. http://sourceforge.net/projects/directpython/
61. http://python.net/crew/mhammond/win32/Downloads.html
62. comme http://wikipython.flibuste.net/CodeWindows#ExempleavecExcel
63. http://pybrary.net/pyPdf/
64. http://www.cosc.canterbury.ac.nz/greg.ewing/python/Pyrex/
65. http://gcc.gnu.org/, ce compilateur présent sur toute machine *Linux* est porté également sur *Microsoft Windows* (voir le site http://www.mingw.org/).
66. http://www.scipy.org/Weave
67. http://www.scipy.org/

L'accélération de l'exécution d'un programme *Python* suscite l'intérêt de nombreux informaticiens, c'est aussi l'objectif de l'outil *PyPy*[68] qui réécrit automatiquement certaines partie du code en langage *C* et l'intègre au programme existant.

## 2.7 Outils connexes

Ce paragraphe sort du cadre de ce livre. Néanmoins, on ne conçoit pas un programme d'envergure sans outil autre que *Python*. Lorsqu'on travaille à plusieurs, il faut pouvoir conserver les différentes versions d'un programme, être capable de fusionner des modifications effectuées par deux personnes, échanger de l'information. Ce sont ces tâches auxquelles les applications qui suivent tentent d'apporter des réponses.

### 2.7.1 Outils quotidiens

**SVN**[69] : c'est un logiciel de suivi de source, il permet de conserver différentes versions d'une arborescence de fichiers, de fusionner des modifications faites par plusieurs personnes. C'est très pratique pour travailler à plusieurs. Une interface graphique est également disponible : **TortoiseSVN**[70]. Pour travailler à plusieurs en réseau, il faudra également installer un serveur tel que **Apache**[71].

**MoinMoin**[72] : les wikis sont devenus des outils presque incontournables qui permettent de conserver tout type d'information. *Wikipedia* est le wiki le plus célèbre. Ces outils sont très utiles pour partager l'information au sein d'une entreprise, pour archiver les rapports de réunions par exemple. Ils intègrent souvent un moteur de recherche.

### 2.7.2 Outils occasionnels

**BeyondCompare**[73] : ce logiciel permet de comparer des fichiers textes entre eux pour voir les différences, il compare également les arborescences de fichiers. Il n'est pas gratuit. L'outil *SVN* permet déjà de voir les différences entre deux versions d'un même fichier mais celui-ci est plus complet en ce qui concernent les fonctionnalités de comparaison.

**epydoc**[74] : il peut être intéressant de construire une aide associée aux fonctions et aux classes d'un programme. C'est l'objectif de l'outil **epydoc** qui reprend le code pour produire des pages d'aide *HTML*. En collaboration avec *Graphviz*[75], l'aide est agrémentée de graphes d'héritage.

**doxygen**[76] : associé au module `doxypy`[77], cet outil offre une alternative plus riche que *epydoc*, il peut utiliser également *Graphviz*[78] pour produire des graphes.

**Sphinx**[79] : c'est l'outil qui génère l'aide associée au langage *Python* telle qu'elle apparaît à l'adresse http://docs.python.org/. Il crée des pages HTML et des documents au format PDF pour tout projet *Python*.

**HTML Help Workshop**[80] : cet outil permet de convertir une documentation au format HTML en un format CHM propre à *Microsoft Windows*. L'outil *doxygen* propose une option qui facilite cette conversion via ce logiciel.

68. http://codespeak.net/pypy/dist/pypy/doc/home.html
69. http://subversion.tigris.org/
70. http://tortoisesvn.tigris.org/
71. http://httpd.apache.org/
72. http://moinmo.in/
73. http://www.scootersoftware.com/
74. http://epydoc.sourceforge.net/
75. http://www.graphviz.org/
76. http://www.stack.nl/$\sim$dimitri/doxygen/
77. http://code.foosel.org/doxypy
78. http://www.graphviz.org/
79. http://sphinx.pocoo.org/
80. http://www.microsoft.com/downloads/ puis saisir *HTML Help Workshop* dans la barre de recherche

**InnoSetup**[81] : cette application permet de construire un installateur qui s'exécute automatiquement. Lorsque vous souhaitez transmettre votre travail sous *Python* à quelqu'un d'autre, celui-ci doit récupérer le langage, un éditeur, les fichiers *Python*, d'éventuelles données. L'installation permettra de regrouper l'ensemble de ces opérations sans que l'utilisateur final n'ait à se soucier de quoi que ce soit. Cette application ne fonctionne que sous *Microsoft Windows*.

### 2.7.3 Outils pour des besoins spécifiques

Le langage *Python* s'interface avec de nombreuses applications, les outils qui suivent peuvent aider des programmeurs confirmés à automatiser certaines tâches fastidieuses.

**BuildBot**[82] : cet utilitaire est écrit en *Python*, lorsqu'il est couplé avec *SVN*, il facilite la conception d'applications informatiques à plusieurs. A chaque mise à jour du code source, *BuildBot* recompile les sources, passe les tests de non-régression[83], avertit le ou les programmeurs concernés par des erreurs éventuellement introduites lors d'une mise à jour. Il permet de suivre en temps réel la stabilité d'une application. Le projet *PyBots*[84] regroupe plusieurs outils d'automatisation intéressants.

**MySQL**[85] : c'est un serveur de base de données. Les bases de données sont devenus incontournables, elles permettent de stocker et de consulter un grand nombre d'informations de façon beaucoup plus efficace que des fichiers quels qu'ils soient. Le langage *Python* permet facilement de dialoguer directement avec les logiciels de bases de données les plus connus dont *MySQL*. Le module `MySQLdb`[86] permet de s'y connecter depuis *Python*.

**SQLite**[87] : cette application traite également les bases de données mais sans le côté serveur. Les bases de données sont stockées sous forme de fichiers. Cet outil est intégré à la version 2.5 du langage *Python*.

**Remote Python Call (RPyC)**[88] : ce module facilite l'écriture de programmes *Python* qui doivent s'exécuter sur plusieurs machines avec tous les problèmes que cela comporte : synchronisation, partage d'informations, ...

**Mercurial**[89] : c'est un autre logiciel de suivi de source écrit en *Python*, son mode de fonctionnement - décentralisé - est différent de celui de *SVN*. Avec *Mercurial*, il n'y a plus une référence des fichiers sources d'un programme, *Mercurial* est capable d'assembler des versions d'où qu'elles viennent. L'avantage est de pouvoir travailler avec de nombreuses "branches".

### 2.7.4 Outils Internet

**django**[90] : cet outil facilite la conception de sites Internet qui ne sont pas réduits à de simples pages HTML mais incluent des parties écrites en *Python*. Il propose des exemples de pages, des accès à des bases de données. Il est souvent utilisé en combinaison avec l'outil suivant.

**CherryPy**[91] : il permet de réaliser des applications Internet. Il est suffisamment robuste pour être utilisé par de nombreux sites Internet. Il peut paraître fastidieux de s'y plonger mais ce genre d'outil offre un réel gain de temps lors de la conception de sites qui incluent de nombreux échanges avec les utilisateurs.

81. http://www.innosetup.com/
82. http://buildbot.net/trac
83. Les tests de non-régression ont pour objectif de limiter le nombre d'erreurs dans un programme. Pour une fonction retournant le résultat d'un calcul scientifique, un test de non-régression consiste à vérifier que ce calcul, pour des données bien précises, aboutit toujours au même résultat malgré les modifications apportées au programme. La détection d'une erreur débouche souvent sur un test de non-régression afin de s'assurer qu'elle ne se reproduira pas.
84. http://www.pybots.org/
85. http://www.mysql.fr/
86. http://sourceforge.net/projects/mysql-python
87. http://www.sqlite.org/
88. http://rpyc.wikispaces.com/
89. http://www.selenic.com/mercurial/wiki/
90. http://www.django-fr.org/
91. http://www.cherrypy.org/

**TurboGears**[92] : cet outil en regroupe plusieurs dont *CherryPy*, c'est un environement complet pour créer des sites Internet.

**Zope**[93] : c'est encore un outil pour concevoir des sites Internet, il est plus vieux et beaucoup plus riche. Il est également plus complexe à appréhender. Il est plutôt destiné au développement de sites conséquents.

92. http://turbogears.org/
93. http://www.zope.org/

# Chapitre 3

# Types et variables du langage *Python*

## 3.1 Variables

Il est impossible d'écrire un programme sans utiliser de variable. Ce terme désigne le fait d'attribuer un nom ou identificateur à des informations : en les nommant, on peut manipuler ces informations beaucoup plus facilement. L'autre avantage est de pouvoir écrire des programmes valables pour des valeurs qui varient : on peut changer la valeur des variables, le programme s'exécutera toujours de la même manière et fera les mêmes types de calculs quelque soient les valeurs manipulées. Les variables jouent un rôle semblable aux inconnues dans une équation mathématique.

L'ordinateur ne sait pas faire l'addition de plus de deux nombres mais cela suffit à calculer la somme de $n$ premiers nombres entiers. Pour cela, il est nécessaire de créer une variable intermédiaire qu'on appellera par exemple `somme` de manière à conserver le résultat des sommes intermédiaires.

```
somme = 0                   # initialisation : la somme est nulle
for i in range(1,n) :       # pour tous les indices de 1 à n
    somme = somme + i       # on ajoute le i ème élément à somme
```

**Définition 3.1 : variable**
Une variable est caractérisée par :

*un identificateur :* il peut contenir des lettres, des chiffres, des blancs soulignés mais il ne peut commencer par un chiffre. Minuscules et majuscules sont différenciées. Il est aussi unique.

*un type :* c'est une information sur le contenu de la variable qui indique à l'interpréteur *Python* la manière de manipuler cette information.

Comme le typage est dynamique en *Python*, le type n'est pas précisé explicitement, il est implicitement liée à l'information manipulée. Par exemple, en écrivant, `x = 3.4`, on ne précise pas le type de la variable `x` mais il est implicite car `x` reçoit une valeur réelle : `x` est de type réel ou `float` en *Python*. Pour leur première initialisation, une variable reçoit dans la plupart des cas une constante :

**Définition 3.2 : constante**
Les constantes sont le contraire des variables, ce sont toutes les valeurs numériques, chaînes de caractères, ..., tout ce qui n'est pas désigné par un nom. Les constantes possèdent un type mais pas d'identificateur.

Le langage *Python* possède un certain nombre de types de variables déjà définis ou types fondamentaux à partir desquels il sera possible de définir ses propres types (voir chapitre 5). Au contraire de langages tels que le *C*, il n'est pas besoin de déclarer une variable pour signifier qu'elle existe, il suffit de lui affecter une valeur. Le type de la variable sera défini par le type de la constante qui lui est affectée. Le type d'une variable peut changer, il correspond toujours au type de la dernière affectation.

```
x   = 3.5      # création d'une variable nombre réel appelée x initialisée à 3.5
               # 3.5 est un réel, la variable est de type "float"
sc = "chaîne"  # création d'une variable chaîne de caractères appelée str
               # initialisée à "chaîne", sc est de type "str"
```

**Remarque 3.3 : commentaires**
Pour tous les exemples qui suivront, le symbole # apparaîtra à maintes reprises. Il marque le début d'un commentaire que la fin de la ligne termine. Autrement dit, un commentaire est une information aidant à la compréhension du programme mais n'en faisant pas partie comme dans l'exemple qui suit.

```
x = 3          # affectation de la valeur entière 3 à la variable x
y = 3.0        # affectation de la valeur réelle 3.0 à la variable y
```

**Remarque 3.4 : instruction sur plusieurs lignes**
Le *Python* impose une instruction par ligne. Il n'est pas possible d'utiliser deux lignes pour écrire une affectation à moins de conclure chaque ligne qui n'est pas la dernière par le symbole \. L'exemple suivant est impossible.

```
x =
    5.5
```

Il devrait être rédigé comme suit :

```
x =  \
    5.5
```

Avec ce symbole, les longues instructions peuvent être écrites sur plusieurs lignes de manière plus lisibles, de sorte qu'elles apparaissent en entier à l'écran.

Les paragraphes suivant énumèrent les types incontournables en *Python*. Ils sont classés le plus souvent en deux catégories : types *immuables* ou *modifiables*. Néanmoins, tous les types du langage *Python* sont également des objets, c'est pourquoi on retrouve dans ce chapitre certaines formes d'écriture similaires à celles présentées plus tard dans le chapitre concernant les classes (chapitre 5).

## 3.2 Types immuables (ou *immutable*)

> **Définition 3.5 : type immuable (ou immutable)**
> Une variable de type immuable ne peut être modifiée. Une opération sur une variable de ce type entraîne nécessairement la création d'une autre variable du même type, même si cette dernière est temporaire.

Autrement dit, la simple instruction `x+ = 3` qui consiste à ajouter à la variable `x` la valeur `3` crée une seconde variable dont la valeur est celle de `x` augmentée de `3` puis à en recopier le contenu dans celui de la variable `x`. Les nombres sont des types immuables mais aussi les chaînes de caractères et les `tuple` qui sont des tableaux d'objets. Pour tous ces types, il n'est pas possible de modifier une variable de ce type, il faut en recréer une autre du même type qui intègrera la modification.

### 3.2.1 Type "rien"

*Python* propose un type `None` pour signifier qu'une variable ne contient rien. La variable est de type `None` et est égale à `None`.

```
s = None
print s    # affiche None
```

Certaines fonctions [1] utilisent cette convention lorsqu'il leur est impossible de retourner un résultat. Dans ce cas, il est possible de générer une erreur (ou exception, voir paragraphe 6), de retourner une valeur par

1. Les fonctions sont définies au paragraphe 4.4, plus simplement, ce sont des mini-programmes : elles permettent de découper un programme long en tâches plus petites.

défaut ou encore de retourner `None`. Il n'y a pas de choix meilleur que les autres, il suffit juste de préciser la convention choisie.

### 3.2.2 Nombres réels et entiers

Il existe deux types de nombres en *Python*, les nombres réels (`float`) et les nombres entiers (`int`). L'instruction `x = 3` crée une variable de type `int` initialisée à `3` tandis que `y = 3.5` crée une variable de type `float` initialisée à `3.5`. Le programme suivant permet de vérifier cela en affichant pour les variables `x` et `y`, leurs valeurs et leurs types respectifs grâce à la fonction `type`.

```
x = 3
y = 3.5
print "x =", x, type(x)
print "y =", y, type(y)
```

```
x = 3 <type 'int'>
y = 3.5 <type 'float'>
```

Voici la liste des opérateurs qui s'appliquent aux nombres réels et entiers. Les trois premiers résultats s'expliquent en utilisant la représentation en base deux. `8 << 1` s'écrit en base deux `100 << 1 = 1000`, ce qui vaut 16 en base décimale : les bits sont décalés vers la droite ce qui équivaut à multiplier par deux. De même, `7&2` s'écrit `1011&10 = 10`, qui vaut 2 en base décimale. Les opérateurs <<, >>, |, & sont des opérateurs bit à bit, ils se comprennent à partir de la représentation binaire des nombres entiers.

| **opérateur** | **signification** | **exemple** |
|---|---|---|
| << >> | décalage à gauche, à droite | `x = 8 << 1` (résultat = 16) |
| \| | opérateur logique *ou* bit à bit | `x = 8\|1` (résultat = 9) |
| & | opérateur logique *et* bit à bit | `x = 11&2` (résultat = 2) |
| + - | addition, soustraction | `x = y + z` |
| += -= | addition ou soustraction puis affectation | `x+ = 3` |
| * / | multiplication, division | `x = y * z` |
| // | division entière, le résultat est de type réel si l'un des nombres est réel | `x = y//3` |
| % | reste d'une division entière (modulo) | `x = y%3` |
| *= /= | multiplication ou division puis affectation | `x* = 3` |
| ** | puissance (entière ou non, racine carrée = ** 0.5) | `x = y * *3` |

Les fonctions `int` et `float` permettent de convertir un nombre quelconque ou une chaîne de caractères respectivement en un entier (arrondi) et en un nombre réel.

```
x = int (3.5)
y = float (3)
z = int ("3")
print "x:", type(x), "   y:", type(y), "   z:", type(z)
# affiche   x: <type 'int'>    y: <type 'float'>    z: <type 'int'>
```

**Remarque 3.6 : conversion en nombre entier**
Il peut arriver que la conversion en un nombre entier ne soit pas directe. Dans l'exemple qui suit, on cherche à convertir une chaîne de caractères (voir paragraphe 3.2.4) en entier mais cette chaîne représente un réel. Il faut d'abord la convertir en réel puis en entier, c'est à ce moment que l'arrondi sera effectué.

```
i = int ("3.5")          # provoque une erreur
i = int (float ("3.5"))  # fonctionne
```

Les opérateurs listés par le tableau ci-dessus ont des priorités différentes, triés par ordre croissant [2]. Toutefois, il est conseillé d'avoir recours aux parenthèses pour enlever les doutes : $3 * 2 * *4 = 3 * (2 * *4)$.

2. La page http://docs.python.org/reference/expressions.html#operator-summary est plus complète à ce sujet.

**Remarque 3.7 : division entière**
Il existe un cas pour lequel cet opérateur cache un sens implicite : lorsque la division opère sur deux nombres entiers ainsi que le montre l'exemple suivant.

```
x = 11
y = 2
z = x / y      # le résultat est 5 et non 5.5 car la division est entière
```

Pour obtenir le résultat juste incluant la partie décimale, il faut convertir les entiers en réels.

```
x = float (11)
y = float (2)
z = x / y      # le résultat est 5.5 car c'est une division entre deux réels
```

Pour éviter d'écrire le type `float`, on peut également écrire `11.0/2` de façon à spécifier explicitement que la valeur 11.0 est réelle et non entière. L'opérateur // permet d'effectuer une division entière lorsque les deux nombres à diviser sont réels, le résultat est un entier mais la variable est de type réel si l'un des nombres est de type réel.

### 3.2.3 Booléen

Les booléens sont le résultat d'opérations logiques et ont deux valeurs possibles : `True` ou `False`. Voici la liste des opérateurs qui s'appliquent aux booléens.

| opérateur | signification | exemple |
|---|---|---|
| `and` `or` | et, ou logique | `x` = `True or False` (résultat = True) |
| `not` | négation logique | `x` = `not x` |

```
x = 4 < 5
print x         # affiche True
print not x     # affiche False
```

Voici la liste des opérateurs de comparaisons qui retournent des booléens. Ceux-ci s'applique à tout type, aux entiers, réels, chaînes de caractères, t-uples... Une comparaison entre un entier et une chaîne de caractères est syntaxiquement correcte même si le résultat a peu d'intérêt.

| opérateur | signification | exemple |
|---|---|---|
| < > | inférieur, supérieur | `x` = 5 < 5 |
| <= >= | inférieur ou égal, supérieur ou égal | `x` = 5 <= 5 |
| == != | égal, différent | `x` = 5 == 5 |

A l'instar des nombres réels, il est préférable d'utiliser les parenthèses pour éviter les problèmes de priorités d'opérateurs dans des expressions comme : 3 < `x` `and` `x` < 7. Toutefois, pour cet exemple, *Python* accepte l'écriture résumée qui enchaîne des comparaisons : 3 < `x` `and` `x` < 7 est équivalent à 3 < `x` < 7. Il existe deux autres mots-clés qui retournent un résultat de type booléen :

| opérateur | signification |
|---|---|
| is | test d'identification |
| in | test d'appartenance |

Ces deux opérateurs seront utilisés ultérieurement, `in` avec les listes, les dictionnaires, les boucles (paragraphe 4.3.2), `is` lors de l'étude des listes (paragraphe 3.3.2) et des classes (chapitre 5). Bien souvent, les booléens sont utilisés de manière implicite lors de tests (paragraphe 4.2.1) ce qui n'empêche pas de les déclarer explicitement.

```
x = True
y = False
```

### 3.2.4 Chaîne de caractères

#### 3.2.4.1 Création d'une chaîne de caractères

**Définition 3.8 : chaîne de caractères**
Le terme "chaîne de caractères" ou *string* en anglais signifie une suite finie de caractères, autrement dit, du texte.

Ce texte est compris entre deux guillemets ou deux apostrophes, ces deux symboles sont interchangeables. L'exemple suivant montre comment créer une chaîne de caractères. Il ne faut pas confondre la partie entre guillemets ou apostrophes, qui est une constante, de la variable qui le contient.

```
t = "string = texte"
print type (t), t
t = 'string = texte, initialisation avec apostrophes'
print type (t), t

t = "morceau 1" \
    "morceau 2"    # second morceau ajouté au premier par l'ajout du symbole \,
                   # il ne doit rien y avoir après le symbole \,
                   # pas d'espace ni de commentaire
print t

t = """première ligne
seconde ligne"""   # chaîne de caractères qui s'étend sur deux lignes
print t
```

Le résultat de ce petit programme est le suivant :

```
<type 'str'> string = texte
<type 'str'> string = texte, initialisation avec apostrophes
morceau 1morceau 2
première ligne
seconde ligne
```

La troisième chaîne de caractères créée lors de ce programme s'étend sur deux lignes. Il est parfois plus commode d'écrire du texte sur deux lignes plutôt que de le laisser caché par les limites de fenêtres d'affichage. *Python* offre la possibilité de couper le texte en deux chaînes de caractères recollées à l'aide du symbole \ à condition que ce symbole soit le dernier de la ligne sur laquelle il apparaît. De même, lorsque le texte contient plusieurs lignes, il suffit de les encadrer entre deux symboles """ ou ''' pour que l'interpréteur *Python* considère l'ensemble comme une chaîne de caractères et non comme une série d'instructions.

Par défaut, le *Python* ne permet pas l'insertion de caractères tels que les accents dans les chaînes de caractères, le paragraphe 2.5.2 (page 21) explique comment résoudre ce problème. De même, pour insérer un guillemet dans une chaîne de caractères encadrée elle-même par des guillemets, il faut le faire précéder du symbole \. La séquence \" est appelée un extra-caractère (voir table 3.1).

**Remarque 3.9 : préfixe "r", chaîne de caractères**
Il peut être fastidieux d'avoir à doubler tous les symboles \ d'un nom de fichier. Il est plus simple dans ce cas de préfixer la chaîne de caractères par `r` de façon à éviter que l'utilisation du symbole \ ne désigne un caractère spécial. Les deux lignes suivantes sont équivalentes :

```
s = "C:\\Users\\Dupre\\exemple.txt"
s = r"C:\Users\Dupre\exemple.txt"
```

Sans la lettre "r", tous les \ doivent être doublés, dans le cas contraire, *Python* peut avoir des effets indésirables selon le caractère qui suit ce symbole.

| | |
|---|---|
| \" | guillemet |
| \' | apostrophe |
| \n | passage à la ligne |
| \\ | insertion du symbole \ |
| \% | pourcentage, ce symbole est aussi un caractère spécial |
| \t | tabulation |
| \r | retour à la ligne, peu usité, il a surtout son importance lorsqu'on passe d'un système *Windows* à *Linux* car *Windows* l'ajoute automatiquement à tous ses fichiers textes |

**Table 3.1 :** *Liste des extra-caractères les plus couramment utilisés à l'intérieur d'une chaîne de caractères (voir page http://www.pythonweb.org/projects/webmodules/doc/0.5.3/html_multipage/lib/node48.html ).*

### 3.2.4.2 Manipulation d'une chaîne de caractères

Une chaîne de caractères est semblable à un tableau et certains opérateurs qui s'appliquent aux tableaux s'appliquent également aux chaînes de caractères, celles-ci sont regroupées dans la table 3.2. La fonction `str` permet de convertir un nombre, un tableau, un objet (voir chapitre 5) en chaîne de caractères afin de pouvoir l'afficher. La fonction `len` retourne la longueur de la chaîne de caractères.

```
x = 5.567
s = str (x)
print type(s), s   # <type 'str'> 5.567
print len(s)       # affiche 5
```

| opérateur | signification | exemple |
|---|---|---|
| + | concaténation de chaînes de caractères | t = "abc" + "def" |
| * | répétition d'une chaîne de caractères | t = "abc" * 4 |
| [n] | obtention du n$^{ième}$ caractères, le premier caractère a pour indice 0 | t = "abc"<br>print t[0] # donne a |
| [i : j] | obtention des caractères compris entre les indices i et j − 1 inclus, le premier caractère a pour indice 0 | t = "abc"<br>print t[0 : 2] # donne ab |

**Table 3.2 :** *Opérations applicables aux chaînes de caractères.*

Il existe d'autres fonctions qui permettent de manipuler les chaînes de caractères.

**syntaxe 3.10 : fonctions et chaînes de caractères**

```
res = s.fonction (...)
```

Où `s` est une chaîne de caractères, `fonction` est le nom de l'opération que l'on veut appliquer à `s`, `res` est le résultat de cette manipulation.

La table 3.3 (page 35) présente une liste non exhaustive des fonctions disponibles dont un exemple d'utilisation suit. Cette syntaxe `variable.fonction(arguments)` est celle des classes [3].

3. voir chapitre 5

| | |
|---|---|
| count( sub[, st[, end]]) | Retourne le nombre d'occurrences de la chaîne de caractères sub, les paramètres par défaut st et end permettent de réduire la recherche entre les caractères d'indice st et end exclu. Par défaut, st est nul tandis que end correspond à la fin de la chaîne de caractères. |
| find( sub[, st[, end]]) | Recherche une chaîne de caractères sub, les paramètres par défaut st et end ont la même signification que ceux de la fonction count. Cette fonction retourne -1 si la recherche n'a pas abouti. |
| isalpha() | Retourne True si tous les caractères sont des lettres, False sinon. |
| isdigit() | Retourne True si tous les caractères sont des chiffres, False sinon. |
| replace( old, new[, co]) | Retourne une copie de la chaîne de caractères en remplaçant toutes les occurrences de la chaîne old par new. Si le paramètre optionnel co est renseigné, alors seules les co premières occurrences seront remplacées. |
| split( [sep[, msplit]]) | Découpe la chaîne de caractères en se servant de la chaîne sep comme délimiteur. Si le paramètre msplit est renseigné, au plus msplit coupures seront effectuées. |
| strip([s]) | Supprime les espaces au début et en fin de chaîne. Si le paramètre s est renseigné, la fonction strip supprime tous les caractères qui font partie de s au début et en fin de chaîne. |
| upper() | Remplace les minuscules par des majuscules. |
| lower() | Remplace les majuscules par des minuscules. |
| join(words) | Fait la somme d'un tableau de chaînes de caractères (une liste ou un T-uple), La chaîne de caractères sert de séparateur qui doit être ajouté entre chaque élément du tableau words. Un exemple de cette fonction figure en page 35. |

**Table 3.3 :** *Quelques fonctions s'appliquant aux chaînes de caractères, l'aide associée au langage Python fournira la liste complète (voir page http://docs.python.org/library/stdtypes.html#id4 ). Certains des paramètres sont encadrés par des crochets, ceci signifie qu'ils sont facultatifs.*

```
st = "langage python"
st = st.upper ()              # mise en lettres majuscules
i  = st.find ("PYTHON")       # on cherche "PYTHON" dans st
print i                       # affiche 8
print st.count ("PYTHON")     # affiche 1
print st.count ("PYTHON", 9)  # affiche 0
```

L'exemple suivant permet de retourner une chaîne de caractères contenant plusieurs éléments séparés par ";". La chaîne "un;deux;trois" doit devenir "trois;deux;un". On utilise pour cela les fonctionnalités split et join (voir table 3.3). L'exemple utilise également la fonctionnalité reverse des listes qui seront décrites plus loin dans ce chapitre. Il faut simplement retenir qu'une liste est un tableau. reverse retourne le tableau.

```
s    = "un;deux;trois"
mots = s.split (";")        # mots est égal à ['un', 'deux', 'trois']
mots.reverse ()             # retourne la liste, mots devient égal à
                            #                 ['trois', 'deux', 'un']
s2 = ";".join (mots)        # concaténation des éléments de mots séparés par ";"
print s2                    # affiche trois;deux;un
```

#### 3.2.4.3 Formatage d'une chaîne de caractères

*Python* offre une manière plus concise de former une chaîne de caractères à l'aide de plusieurs types d'informations en évitant la conversion explicite de ces informations (fonction str) et leur concaténation. Il est particulièrement intéressant pour les nombres réels qu'il est possible d'écrire en imposant un nombre

de décimales fixe. Le format est le suivant :

**syntaxe 3.11 : chaîne de caractères, formatage**

```
"....  %c1  ....  %c2 " % (v1,v2)
```

`c1` est un code choisi parmi ceux de la table 3.4 (page 37). Il indique le format dans lequel la variable `v1` devra être transcrite. Il en est de même pour le code `c2` associé à la variable `v2`. Les codes insérés dans la chaîne de caractères seront remplacés par les variables citées entre parenthèses après le symbole % suivant la fin de la chaîne de caractères. Il doit y avoir autant de codes que de variables, qui peuvent aussi être des constantes.

Voici concrètement l'utilisation de cette syntaxe :

```
x = 5.5
d = 7
s = "caractères"
res =   "un nombre réel %f et un entier %d, une chaîne de %s, \n" \
        "un réel d'abord converti en chaîne de caractères %s" % (x,d,s, str(x+4))
print res
res =   "un nombre réel " + str (x) + " et un entier " + str (d) + \
        ", une chaîne de " + s + \
        ",\n un réel d'abord converti en chaîne de caractères " + str(x+4)
print res
```

La seconde affectation de la variable `res` propose une solution équivalente à la première en utilisant l'opérateur de concaténation +. Les deux solutions sont équivalentes, tout dépend des préférences de celui qui écrit le programme.

```
un nombre réel 5.500000 et un entier 7, une chaîne de caractères,
un réel d'abord converti en chaîne de caractères 9.5
un nombre réel 5.5 et un entier 7, une chaîne de caractères,
un réel d'abord converti en chaîne de caractères 9.5
```

La première option permet néanmoins un formatage plus précis des nombres réels en imposant par exemple un nombre défini de décimal. Le format est le suivant :

**syntaxe 3.12 : chaîne de caractères, formatage des nombres**

```
"%n.df" % x
```

où `n` est le nombre de chiffres total et `d` est le nombre de décimales, `f` désigne un format réel indiqué par la présence du symbole %

```
x = 0.123456789
print x             # affiche 0.123456789
print "%1.2f" % x   # affiche 0.12
print "%06.2f" % x  # affiche 000.12
```

Il existe d'autres formats regroupés dans la table 3.4. L'aide reste encore le meilleur réflexe car le langage *Python* est susceptible d'évoluer et d'ajouter de nouveaux formats.

### 3.2.5 T-uple

**Définition 3.13 : T-uple**
Les T-uple sont un tableau d'objets qui peuvent être de tout type. Ils ne sont pas modifiables.

Un T-uple apparaît comme une liste d'objets comprise entre parenthèses et séparés par des virgules. Leur création reprend le même format :

| | |
|---|---|
| d | entier relatif |
| e | nombre réel au format exponentiel |
| f | nombre réel au format décimal |
| g | nombre réel, format décimal ou exponentiel si la puissance est trop grande ou trop petite |
| s | chaîne de caractères |

**Table 3.4 :** *Liste non exhaustive des codes utilisés pour formater des informations dans une chaîne de caractères (voir page http://docs.python.org/library/stdtypes.html#string-formatting-operations).*

```
x = (4,5)               # création d'un T-uple composé de 2 entiers
x = ("un",1,"deux",2)   # création d'un T-uple composé de 2 chaînes de caractères
                        # et de 2 entiers, l'ordre d'écriture est important
x = (3,)                # création d'un T-uple d'un élément, sans la virgule,
                        # le résultat est un entier
```

Ces objets sont des vecteurs d'objets. Il est possible d'effectuer les opérations regroupées dans la table 3.5. Etant donné que les chaînes de caractères sont également des tableaux, ces opérations reprennent en partie celles de la table 3.2, elles sont regroupées dans la table 3.5.

| | |
|---|---|
| `x in s` | vrai si `x` est un des éléments de `s` |
| `x not in s` | réciproque de la ligne précédente |
| `s + t` | concaténation de `s` et `t` |
| `s * n` | concatène `n` copies de `s` les unes à la suite des autres |
| `s[i]` | retourne une copie du i$^{ème}$ élément de `s` |
| `s[i : j]` | retourne un T-uple contenant une copie des éléments de `s` d'indices $i$ à $j$ exclu. |
| `s[i : j : k]` | retourne un T-uple contenant une copie des éléments de `s` dont les indices sont compris entre $i$ et $j$ exclu, ces indices sont espacés de $k$ : $i, i+k, i+2k, i+3k, ...$ |
| `len(s)` | nombre d'éléments de `s` |
| `min(s)` | plus petit élément de `s`, résultat difficile à prévoir lorsque les types des éléments sont différents |
| `max(s)` | plus grand élément de `s`, résultat difficile à prévoir lorsque les types des éléments sont différents |
| `sum(s)` | retourne la somme de tous les éléments |

**Table 3.5 :** *Opérations disponibles sur les T-uples, on suppose que* `s` *et* `t` *sont des T-uples,* `x` *est quant à lui quelconque.*

**Remarque 3.14 : T-uple, opérateur []**
Les T-uples ne sont pas modifiables, cela signifie qu'il est impossible de modifier un de leurs éléments. Par conséquent, la ligne d'affectation suivante n'est pas correcte :

```
a     = (4,5)
a [0] = 3      # déclenche une erreur d'exécution
```

Le message d'erreur suivant apparaît :

```
Traceback (most recent call last):
  File "<pyshell#78>", line 1, in -toplevel-
    a[0]=3
TypeError: object doesn't support item assignment
```

Pour changer cet élément, il est possible de s'y prendre de la manière suivante :

```
a = (4,5)
a = (3,) + a[1:2]  # crée un T-uple d'un élément concaténé
                   # avec la partie inchangée de a
```

### 3.2.6 Autres types

Il existe d'autres types comme le type `complex` permettant de représenter les nombres complexes. Ce type numérique suit les mêmes règles et fonctionne avec les mêmes opérateurs (excepté les opérateurs de comparaisons) que ceux présentés au paragraphe 3.2.2 et décrivant les nombres.

```
print complex (1,1)   # affiche (1+1j)
```

Le langage *Python* offre la possibilité de créer ses propres types immuables (voir le chapitre 5 grâce au mot-clé `__slots__`) mais ils seront définis à partir des types immuables présentés jusqu'ici.

## 3.3 Types modifiables (ou *mutable*)

Les types modifiables sont des conteneurs (ou containers en anglais) : ils contiennent d'autres objets, que ce soit des nombres, des chaînes de caractères ou des objets de type modifiable. Plutôt que d'avoir dix variables pour désigner dix objets, on en n'utilise qu'une seule qui désignera ces dix objets.

> **Définition 3.15 : type modifiable (ou mutable)**
> Une variable de type modifiable peut être modifiée, elle conserve le même type et le même identificateur. C'est uniquement son contenu qui évolue.

### 3.3.1 Liste

#### 3.3.1.1 Définition et fonctions

> **Définition 3.16 : liste**
> Les listes sont des collections d'objets qui peuvent être de tout type. Elles sont modifiables.

Une liste apparaît comme une succession d'objets compris entre crochets et séparés par des virgules. Leur création reprend le même format :

```
x = [4,5]               # création d'une liste composée de deux entiers
x = ["un",1,"deux",2]   # création d'une liste composée de
                        # deux chaînes de caractères
                        # et de deux entiers, l'ordre d'écriture est important
x = [3,]                # création d'une liste d'un élément, sans la virgule,
                        # le résultat reste une liste
x = [ ]                 # crée une liste vide
x = list ()             # crée une liste vide
```

Ces objets sont des listes chaînées d'autres objets de type quelconque (immuable ou modifiable). Il est possible d'effectuer les opérations regroupées dans la table 3.6. Ces opérations reprennent celle des T-uples (voir table 3.5) et incluent d'autres fonctionnalités puisque les listes sont modifiables (voir table 3.6). Il est donc possible d'insérer, de supprimer des éléments, de les trier. La syntaxe des opérations sur les listes est similaire à celle qui des opérations qui s'appliquent sur les chaînes de caractères, elles sont présentées par la table 3.7.

| | |
|---|---|
| `x in l` | vrai si `x` est un des éléments de `l` |
| `x not in l` | réciproque de la ligne précédente |
| `l + t` | concaténation de `l` et `t` |
| `l * n` | concatène `n` copies de `l` les unes à la suite des autres |
| `l[i]` | retourne l'élément i$^{\text{ème}}$ élément de `l`, à la différence des T-uples, l'instruction `l[i] = "a"` est valide, elle remplace l'élément `i` par "`a`". |
| `l[i : j]` | retourne une liste contenant les éléments de `l` d'indices $i$ à $j$ exclu. Il est possible de remplacer cette sous-liste par une autre en utilisant l'affectation `l[i : j] = l2` où `l2` est une autre liste (ou un T-uple) de dimension différente ou égale. |
| `l[i : j : k]` | retourne une liste contenant les éléments de `l` dont les indices sont compris entre $i$ et $j$ exclu, ces indices sont espacés de $k$ : $i, i+k, i+2k, i+3k, ...$ Ici encore, il est possible d'écrire l'affectation suivante : `l[i : j : k] = l2` mais `l2` doit être une liste (ou un T-uple) de même dimension que `l[i : j : k]`. |
| `len(l)` | nombre d'éléments de `l` |
| `min(l)` | plus petit élément de `l`, résultat difficile à prévoir lorsque les types des éléments sont différents |
| `max(l)` | plus grand élément de `l`, résultat difficile à prévoir lorsque les types des éléments sont différents |
| `sum(l)` | retourne la somme de tous les éléments |
| `del l[i : j]` | supprime les éléments d'indices entre `i` et `j` exclu. Cette instruction est équivalente à `l[i : j] = []`. |
| `list(x)` | convertit `x` en une liste quand cela est possible |

**Table 3.6 :** *Opérations disponibles sur les listes, identiques à celles des T-uples, on suppose que* `l` *et* `t` *sont des listes,* `i` *et* `j` *sont des entiers.* `x` *est quant à lui quelconque.*

#### 3.3.1.2 Exemples

L'exemple suivant montre une utilisation de la méthode `sort`.

```
x = [9,0,3,5,4,7,8]          # définition d'une liste
print x                      # affiche cette liste
x.sort ()                    # trie la liste par ordre croissant
print x                      # affiche la liste triée
```

Voici les deux listes affichées par cet exemple :

```
[9, 0, 3, 5, 4, 7, 8]
[0, 3, 4, 5, 7, 8, 9]
```

Pour classer les objets contenus par la liste mais selon un ordre différent, il faut définir une fonction qui détermine un classe entre deux éléments de la liste. C'est la fonction `compare` de l'exemple suivant.

```
def compare (x,y):           # crée une fonction
    if x > y : return -1     # qui retourne -1 si x<y,
    elif x == y : return 0   # 0 si x == y
    else: return 1           # 1 si x < y

x.sort (compare)             # trie la liste x à l'aide de la fonction compare
                             # cela revient à la trier par ordre décroissant
print x
```

Le programme affiche cette fois-ci la ligne :

```
[9, 8, 7, 5, 4, 3, 0]
```

| | |
|---|---|
| l.count(x) | Retourne le nombre d'occurrences de l'élément x. Cette notation suit la syntaxe des classes développée au chapitre 5. count est une méthode de la classe list. |
| l.index(x) | Retourne l'indice de la première occurrence de l'élément x dans la liste l. Si celle-ci n'existe, une exception est déclenchée (voir le paragraphe 6). Un exemple figure à la page 40. |
| l.append(x) | Ajoute l'élément x à la fin de la liste l. Si x est une liste, cette fonction ajoute la liste x en tant qu'élément, au final, la liste l ne contiendra qu'un élément de plus. |
| l.extend(k) | Ajoute tous les éléments de la liste k à la liste l. La liste l aura autant d'éléments supplémentaires qu'il y en a dans la liste k. |
| l.insert(i, x) | Insère l'élément x à la position i dans la liste l. |
| l.remove(x) | Supprime la première occurrence de l'élément x dans la liste l. S'il n'y a aucune occurrence de x, cette méthode déclenche une exception. |
| l.pop([i]) | Retourne l'élément l[i] et le supprime de la liste. Le paramètre i est facultatif, s'il n'est pas précisé, c'est le dernier élément dont la valeur est d'abord retournée puis il est supprimé de la liste. |
| l.reverse(x) | Retourne la liste, le premier et dernier élément échange leurs places, le second et l'avant dernier, et ainsi de suite. |
| l.sort([f, reverse]) | Cette fonction trie la liste par ordre croissant. Le paramètre f est facultatif, il permet de préciser la fonction de comparaison qui doit être utilisée lors du tri. Cette fonction prend comme paramètre deux éléments x et y de la liste et retourne les valeurs -1,0,1 selon que x < y, x == y ou x > y (voir paragraphe 4.4). Si reverse est True, alors le tri est décroissant. |

**Table 3.7 :** *Opérations permettant de modifier une liste on suppose que* l *est une liste,* x *est quant à lui quelconque.*

L'exemple suivant illustre un exemple dans lequel on essaye d'accéder à l'indice d'un élément qui n'existe pas dans la liste :

```
x = [9,0,3,5,0]
print x.index (1) # cherche la position de l'élément 1
```

Comme cet élément n'existe pas, il déclenche ce qu'on appelle une exception qui se traduit par l'affichage d'un message d'erreur. Le message indique le nom de l'exception générée (ValueError) ainsi qu'un message d'information permettant en règle générale de connaître l'événement qui en est la cause.

```
Traceback (most recent call last):
  File "c:/temp/temp", line 2, in -toplevel-
    print x.index(1)
ValueError: list.index(x): x not in list
```

Pour éviter cela, on choisit d'intercepter l'exception (voir paragraphe 6).

```
x = [9,0,3,5,0]
try:
    print x.index(1)
except ValueError:
    print "1 n'est pas présent dans la liste x"
else:
    print "trouvé"
```

Ce programme a pour résultat :

```
1 n'est pas présent dans la liste x
```

#### 3.3.1.3 Fonctions range, xrange

Les listes sont souvent utilisées dans des boucles ou notamment par l'intermédiaire de la fonction range. Cette fonction retourne une liste d'entiers.

> **syntaxe 3.17 :** range
>
> range (debut, fin [,marche])
>
> Retourne une liste incluant tous les entiers compris entre debut et fin exclu. Si le paramètre facultatif marche est renseigné, la liste contient tous les entiers n compris debut et fin exclu et tels que n − debut soit un multiple de marche.

Exemple :

```
print range(0,10,2)        # affiche [0, 2, 4, 6, 8]
```

La fonction xrange est d'un usage équivalent à range. Elle permet de parcourir une liste d'entiers sans la créer vraiment. Elle est plus rapide.

```
print xrange(0,10,2)       # affiche xrange(0,10,2)
```

Cette fonction est souvent utilisée lors de boucles (voir paragraphe 4.3.2). Pour parcourir tous les éléments d'un T-uple, d'une liste, d'un dictionnaire... Le programme suivant permet par exemple de calculer la somme de tous les entiers impairs compris entre 1 et 20 exclu.

```
s = 0
for n in range(1,20,2):    # ce programme est équivalent à
    s += n                 # s = sum (range(1,20,2))
```

Le programme suivant permet d'afficher tous les éléments d'une liste.

```
x = ["un", 1, "deux", 2, "trois", 3]
for n in range(0,len(x)):
    print "x [%d] = %s" % (n, x [n])

# le résultat est présenté à droite
```

```
x [0] = un
x [1] = 1
x [2] = deux
x [3] = 2
x [4] = trois
x [5] = 3
```

#### 3.3.1.4 Boucles et listes

Il est possible aussi de ne pas se servir des indices comme intermédiaires pour accéder aux éléments d'une liste quand il s'agit d'effectuer un même traitement pour tous les éléments de la liste x.

```
x = ["un", 1, "deux", 2]
for el in x :
    print "la liste inclut : ", el
```

L'instruction for el in x : se traduit littéralement par : *pour tous les éléments de la liste, faire...* Ce programme a pour résultat :

```
la liste inclut :  un
la liste inclut :  1
la liste inclut :  deux
la liste inclut :  2
```

Il existe également des notations abrégées lorsqu'on cherche à construire une liste à partir d'une autre. Le programme suivant construit la liste des entiers de 1 à 5 à partir du résultat retourné par la fonction range.

```
x = range(0,5)
y = list ()
for i in x : y.append (i+1)
print y                      # affiche [1,2,3,4,5]
```

Le langage *Python* offre la possibilité de résumer cette écriture en une seule ligne. Cette syntaxe sera reprise au paragraphe 4.3.2.2.

```
x = range(0,5)
y = [ i+1 for i in x]
print y                # affiche [1,2,3,4,5]
```

Cette définition de liste peut également inclure des tests ou des boucles imbriquées.

```
x = range(0,5)
y = [ i for i in x if i % 2 == 0]   # sélection les éléments pairs
print y                             # affiche [0,2,4]
z = [ i+j for i in x for j in x]    # construit tous les nombres i+j possibles
print z                             # affiche [0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 2,
                                    # 3, 4, 5, 6, 3, 4, 5, 6, 7, 4, 5, 6, 7, 8]
```

#### 3.3.1.5 Collage de séquences, fonction `zip`

On suppose qu'on dispose de $n$ séquences d'éléments (T-uple, liste, dictionnaire), toutes de longueur $l$. La fonction `zip` permet de construire une liste de T-uples qui est la juxtaposition de toutes ces séquences. Le $i^{\text{ième}}$ T-uple de la liste résultante contiendra tous les $i^{\text{ième}}$ éléments des séquences juxtaposées. Si les longueurs des séquences sont différentes, la liste résultante aura même taille que la plus courte des séquences.

```
a = (1,0,7,0,0,0)
b = [2,2,3,5,5,5]
c = [ "un", "deux", "trois", "quatre" ]
d = zip (a,b,c)
print d           # affiche [(1, 2, 'un'), (0, 2, 'deux'),
                  #       (7, 3, 'trois'), (0, 5, 'quatre')]
```

#### 3.3.1.6 Concaténation de chaîne de caractères

Il arrive fréquemment de constuire une chaîne de caractères petits bouts par petits bouts comme le montre le premier exemple ci-dessous. Cette construction peut s'avérer très lente lorsque le résultat est long. Dans ce cas, il est nettement plus rapide d'ajouter chaque morceau dans une liste puis de les concaténer en une seule fois grâce à la méthode `join`.

```
s = ""
while <condition> :
    s += ...
```

```
s = []
while <condition> :
    s.append ( ... )
s = "".join (s)
```

### 3.3.2 Copie

A l'inverse des objets de type immuable, une affectation ne signifie pas une copie. Afin d'éviter certaines opérations superflues et parfois coûteuses en temps de traitement, on doit distinguer la variable de son contenu. Une variable désigne une liste avec un mot, une affection permet de créer un second mot pour désigner la même liste. Il est alors équivalent de faire des opérations avec le premier mot ou le second comme le montre l'exemple suivant avec les listes `l` et `l2`.

```
l  = [4,5,6]
l2 = l
print l            # affiche [4,5,6]
print l2           # affiche [4,5,6]
l2 [1] = "modif"
print l            # affiche [4, 'modif', 6]
print l2           # affiche [4, 'modif', 6]
```

Dans cet exemple, il n'est pas utile de créer une seconde variable, dans le suivant, cela permet quelques raccourcis.

```
l      = [[0,1], [2,3]]
l1     = l [0]
l1 [0] = "modif" # ligne équivalente à : l [0][0] = "modif"
```

Par conséquent, lorsqu'on affecte une liste à une variable, celle-ci n'est pas recopiée, la liste reçoit seulement un nom de variable. L'affectation est en fait l'association d'un nom avec un objet (voir paragraphe 5.6). Pour copier une liste, il faut utiliser la fonction `copy` du module `copy`[4]. Ce point sera rappelé au paragraphe 5.6.3 (page 104).

```
import copy
l  = [4,5,6]
l2 = copy.copy(l)
print l            # affiche [4,5,6]
print l2           # affiche [4,5,6]
l2 [1] = "modif"
print l            # affiche [4,5,6]
print l2           # affiche [4, 'modif', 6]
```

L'opérateur == permet de savoir si deux listes sont égales même si l'une est une copie de l'autre. Le mot-clé `is` permet de vérifier si deux variables font référence à la même liste ou si l'une est une copie de l'autre comme le montre l'exemple suivant :

```
import copy
l  = [1,2,3]
l2 = copy.copy (l)
l3 = l

print l == l2  # affiche True
print l is l2  # affiche False
print l is l3  # affiche True
```

**Remarque 3.18 : fonction `copy` et `deepcopy`**
Le comportement de la fonction `copy` peut surprendre dans le cas où une liste contient d'autres listes. Pour être sûr que chaque élément d'une liste a été correctement recopiée, il faut utiliser la fonction `deepcopy`. La fonction est plus longue mais elle recopie toutes les listes que ce soit une liste incluse dans une liste elle-même incluse dans une autre liste elle-même incluse...

```
import copy
l  = [[1,2,3],[4,5,6]]
l2 = copy.copy (l)
l3 = copy.deepcopy (l)
l [0][0] = 1111
print l                # affiche [[1111, 2, 3], [4, 5, 6]]
print l2               # affiche [[1111, 2, 3], [4, 5, 6]]
print l3               # affiche [[1, 2, 3], [4, 5, 6]]
```

4. Le module `copy` est une extension interne. C'est une librairie de fonctions dont la fonction `copy`. Cette syntaxe sera vue au chapitre 7.

```
print l is l2          # affiche False
print l [0] is l2 [0]  # affiche True
print l [0] is l3 [0]  # affiche False
```

La fonction `deepcopy` est plus lente à exécuter car elle prend en compte les références récursives comme celles de l'exemple suivant où deux listes se contiennent l'une l'autre.

```
l     = [1,"a"]
ll    = [1,3]   # ll contient l
l [0] = ll      # l contient ll
print l         # affiche [[[...], 3], 'a']
print ll        # affiche [[[...], 'a'], 3]

import copy
z = copy.deepcopy (l)
print z         # affiche [[[...], 3], 'a']
```

### 3.3.3 Dictionnaire

Les dictionnaires sont des tableaux plus souples que les listes. Une liste référence les éléments en leur donnant une position : la liste associe à chaque élément une position entière comprise entre 0 et $n-1$ si $n$ est la longueur de la liste. Un dictionnaire permet d'associer à un élément autre chose qu'une position entière : ce peut être un entier, un réel, une chaîne de caractères, un T-uple contenant des objets immuables. D'une manière générale, un dictionnaire associe à une valeur ce qu'on appelle une clé de type immuable. Cette clé permettra de retrouver la valeur associée.

L'avantage principal des dictionnaires est la recherche optimisée des clés. Par exemple, on recense les noms des employés d'une entreprise dans une liste. On souhaite ensuite savoir si une personne ayant un nom précisé à l'avance appartient à cette liste. Il faudra alors parcourir la liste jusqu'à trouver ce nom ou parcourir toute la liste si jamais celui-ci ne s'y trouve pas [5]. Dans le cas d'un dictionnaire, cette recherche du nom sera beaucoup plus rapide à écrire et à exécuter.

#### 3.3.3.1 Définition et fonctions

> **Définition 3.19 : dictionnaire**
> Les dictionnaires sont des listes de couples. Chaque couple contient une clé et une valeur. Chaque valeur est indicée par sa clé. La valeur peut-être de tout type, la clé doit être de type immuable, ce ne peut donc être ni une liste, ni un dictionnaire. Chaque clé comme chaque valeur peut avoir un type différent des autres clés ou valeurs.

Un dictionnaire apparaît comme une succession de couples d'objets comprise entre accolades et séparés par des virgules. La clé et sa valeur sont séparées par le symbole :. Leur création reprend le même format :

```
x = { "cle1":"valeur1", "cle2":"valeur2" }
y = { }         # crée un dictionnaire vide
z = dict ()     # crée aussi un dictionnaire vide
```

Les indices ne sont plus entiers mais des chaînes de caractères pour cet exemple. Pour associer la valeur à la clé "cle1", il suffit d'écrire :

```
print x ["cle1"]
```

5. voir également le paragraphe 4.7.1, page 77

La plupart des fonctions disponibles pour les listes sont interdites pour les dictionnaires comme la concaténation ou l'opération de multiplication (∗). Il n'existe plus non plus d'indices entiers pour repérer les éléments, le seul repère est leur clé. La table 3.8 dresse la liste des opérations simples sur les dictionnaires tandis que la table 3.9 dresse la liste des méthodes plus complexes associées aux dictionnaires.

| | |
|---|---|
| `x in d` | vrai si `x` est une des clés de `d` |
| `x not in d` | réciproque de la ligne précédente |
| `d[i]` | retourne l'élément associé à la clé `i` |
| `len(d)` | nombre d'éléments de `d` |
| `min(d)` | plus petite clé |
| `max(d)` | plus grande clé |
| `del d[i]` | supprime l'élément associée à la clé `i` |
| `list(d)` | retourne une liste contenant toutes les clés du dictionnaire `d`. |
| `dict(x)` | convertit `x` en un dictionnaire si cela est possible, particulier, `d` est égal à `dict(d.items())` |

**Table 3.8 :** *Opérations disponibles sur les dictionnaires,* `d` *est un dictionnaire,* `x` *est quant à lui quelconque.*

| | |
|---|---|
| `d.copy()` | Retourne une copie de `d`. |
| `d.has_key(x)` | Retourne `True` si `x` est une clé de `d`. |
| `d.items()` | Retourne une liste contenant tous les couples (clé, valeur) inclus dans le dictionnaire. |
| `d.keys()` | Retourne une liste contenant toutes les clés du dictionnaire `d`. |
| `d.values()` | Retourne une liste contenant toutes les valeurs du dictionnaire `d`. |
| `d.iteritems()` | Retourne un itérateur sur les couples (clé, valeur). |
| `d.iterkeys()` | Retourne un itérateur sur les clés. |
| `d.itervalues()` | Retourne un itérateur sur les valeurs. |
| `d.get(k[,x])` | Retourne `d[k]`, si la clé `k` est manquante, alors la valeur est retourné à moins que le paramètre optionnel `x` soit renseigné, auquel cas, ce sera ce paramètre qui sera retourné. |
| `d.clear()` | Supprime tous les éléments du dictionnaire. |
| `d.update(d2)` | Pour chaque clé de `d1`, `d[k]` = `d1[k]`. |
| `d.setdefault(k[,x])` | Définit `d[k]` si la clé `k` existe, sinon, lui affecte `x` à `d[k]`. |
| `d.popitem()` | Retourne un élément et le supprime du dictionnaire. |

**Table 3.9 :** *Méthodes associées aux dictionnaires,* `d`, `d2` *sont des dictionnaires,* `x` *est quant à lui quelconque.*

Contrairement à une liste, un dictionnaire ne peut être trié car sa structure interne est optimisée pour effectuer des recherches rapides parmi les éléments. On peut aussi se demander quel est l'intérêt de la méthode `popitem` qui retourne un élément puis le supprime alors qu'il existe le mot-clé `del`. Cette méthode est simplement plus rapide car elle choisit à chaque fois l'élément le moins coûteux à supprimer, surtout lorsque le dictionnaire est volumineux.

Les itérateurs sont des objets qui permettent de parcourir rapidement un dictionnaire, ils seront décrits en détail au chapitre 5 sur les classes. Un exemple de leur utilisation est présenté dans le paragraphe suivant.

#### 3.3.3.2 Exemples

Il n'est pas possible de trier un dictionnaire. L'exemple suivant permet néanmoins d'afficher tous les éléments d'un dictionnaire selon un ordre croissant des clés. Ces exemples font appel aux paragraphes sur les boucles (voir chapitre 4).

```
x = { "un":1, "zéro":0, "deux":2, "trois":3, "quatre":4, "cinq":5, \
      "six":6, "sept":1, "huit":8, "neuf":9, "dix":10 }
```

```
key = d.keys ()
key.sort ()
for k in key:
    print k,d [k]
```

L'exemple suivant montre un exemple d'utilisation des itérateurs. Il s'agit de construire un dictionnaire inversé pour lequel les valeurs seront les clés et réciproquement.

```
d = { "un":1, "zero":0, "deux":2, "trois":3, "quatre":4, "cinq":5, \
       "six":6, "sept":1, "huit":8, "neuf":9, "dix":10 }

dinv = { }                      # création d'un dictionnaire vide
for key,value in d.items ()  :  # on parcourt les éléments du dictionnaire
                                # comme si c'était une liste de 2-uple
                                # (clé,valeur)
    dinv [value] = key          # on retourne le dictionnaire

print dinv                      # affiche {0: 'zero', 1: 'un', 2: 'deux',
                                # 3: 'trois', 4: 'quatre', 5: 'cinq', 6: 'six',
                                # 8: 'huit', 9: 'neuf', 10: 'dix'}
```

Pour être plus efficace, on peut remplacer la ligne `for key, value in d.items()` : par : `for key, value in d.iteritems()` :. De cette manière, aucune liste de couples (clé,valeur) n'est créée mais la fonction `iteritems` permet néanmoins de les parcourir. Le programme suivant montre la différence de ces deux fonctions bien qu'il soit équivalent d'utiliser l'une ou l'autre au sein d'une boucle.

```
d = { "un":1, "zero":0, "deux":2, "trois":3, "quatre":4, "cinq":5, \
       "six":6, "sept":1, "huit":8, "neuf":9, "dix":10 }
print d.items ()
print d.iteritems ()
```

Il a pour résultat :

```
[('trois', 3), ('sept', 1), ('neuf', 9), ('six', 6), ('zero', 0),
 ('un', 1), ('dix', 10), ('deux', 2), ('huit', 8), ('quatre', 4),
                                                    ('cinq', 5)]

<dictionary-itemiterator object at 0x0115DC40>
```

**Remarque 3.20 : modification d'un dictionnaire dans une boucle**
D'une manière générale, il faut éviter d'ajouter ou de supprimer un élément dans une liste ou un dictionnaire qu'on est en train de parcourir au sein d'une boucle `for` ou `while`. Cela peut marcher mais cela peut aussi aboutir à des résultats imprévisibles surtout avec l'utilisation d'itérateurs (fonction `iteritems`, `itervalues`, `iterkeys`). Il est préférable de terminer le parcours de la liste ou du dictionnaire puis de faire les modifications désirées une fois la boucle terminée. Dans le meilleur des cas, l'erreur suivante survient :

```
  File "essai.py", line 6, in <module>
    for k in d :
RuntimeError: dictionary changed size during iteration
```

#### 3.3.3.3 Copie

A l'instar des listes (voir paragraphe 3.3.2), les dictionnaires sont des objets et une affectation n'est pas équivalente à une copie comme le montre le programme suivant.

```
d  = {4:4,5:5,6:6}
d2 = d
print d            # affiche {4: 4, 5: 5, 6: 6}
```

```
print d2           # affiche {4: 4, 5: 5, 6: 6}
d2 [5] = "modif"
print d            # affiche {4: 4, 5: 'modif', 6: 6}
print d2           # affiche {4: 4, 5: 'modif', 6: 6}
```

Lorsqu'on affecte une liste à une variable, celle-ci n'est pas recopiée, la liste reçoit seulement un nom de variable. L'affectation est en fait l'association d'un nom avec un objet (voir paragraphe 5.6). Pour copier une liste, il faut utiliser la fonction `copy` du module `copy`.

```
d  = {4:4,5:5,6:6}
import copy
d2 = copy.copy(l)
print d            # affiche {4: 4, 5: 5, 6: 6}
print d2           # affiche {4: 4, 5: 5, 6: 6}
d2 [5] = "modif"
print d            # affiche {4: 4, 5: 5, 6: 6}
print d2           # affiche {4: 4, 5: 'modif', 6: 6}
```

Le mot-clé `is` a la même signification pour les dictionnaires que pour les listes, l'exemple du paragraphe 3.3.2 (page 42) est aussi valable pour les dictionnaires. Il en est de même pour la remarque concernant la fonction `deepcopy`. Cette fonction recopie les listes et les dictionnaires.

#### 3.3.3.4 Clés de type modifiable

Ce paragraphe concerne davantage des utilisateurs avertis qui souhaitent malgré tout utiliser des clés de type modifiable. Dans l'exemple qui suit, la clé d'un dictionnaire est également un dictionnaire et cela provoque une erreur. Il en serait de même si la variable `k` utilisée comme clé était une liste.

```
k = { 1:1}
d = { }
d [k] = 0
```

```
Traceback (most recent call last):
  File "cledict.py", line 3, in <module>
    d [k] = 0
TypeError: dict objects are unhashable
```

Cela ne veut pas dire qu'il faille renoncer à utiliser un dictionnaire ou une liste comme clé. La fonction `id` permet d'obtenir un entier qui identifie de manière unique un dictionnaire. Le code suivant est parfaitement correct.

```
k = { 1:1}
d = { }
d [id (k)] = 0
```

Toutefois, ce n'est pas parce que deux dictionnaires auront des contenus identiques que leurs identifiants retournés par la fonction `id` seront égaux. C'est ce qui explique l'erreur que provoque la dernière ligne du programme suivant.

```
k = {1:1}
d = { }
d [id (k)] = 0
b = k
print d [id(b)]  # affiche bien zéro
c = {1:1}
print d [id(c)]  # provoque une erreur car même si k et c ont des contenus égaux,
                 # ils sont distincts, la clé id(c) n'existe pas dans d
```

Il existe un cas où peut se passer de la fonction `id` mais il inclut la notion de classe définie au chapitre 5. L'exemple suivant utilise directement l'instance `k` comme clé. En affichant le dictionnaire `d`, on vérifie que la clé est liée au résultat de l'instruction `id(k)` même si ce n'est pas la clé.

```
class A : pass

k = A ()
d = { }
d [k] = 0
print d           # affiche {<__main__.A object at 0x0120DB90>: 0}
print id (k)      # affiche 18930576
print hex(id(k))  # affiche en hexadécimal          0x120db90
print d [id(k)]   # provoque une erreur
```

La fonction `hex` convertit un entier en notation hexadécimale. Les nombres affichés changent à chaque exécution. Pour conclure, ce dernier exemple montre comment se passer de la fonction `id` dans le cas d'une clé de type dictionnaire.

```
class A (dict):
    def __hash__(self):
        return id(self)

k = A ()
k ["t"]= 4
d = { }
d [k] = 0
print d         # affiche {{'t': 4}: 0}
```

## 3.4 Extensions

### 3.4.1 Mot-clé `print`, `repr`, conversion en chaîne de caractères

Le mot-clé `print` est déjà apparu dans les exemples présentés ci-dessus, il permet d'afficher une ou plusieurs variables préalablement définies, séparées par des virgules. Les paragraphes qui suivent donnent quelques exemples d'utilisation. La fonction `print` permet d'afficher n'importe quelle variable ou objet à l'écran, cet affichage suppose la conversion de cette variable ou objet en une chaîne de caractères. Deux fonctions permettent d'effectuer cette étape sans toutefois afficher le résultat à l'écran.

La fonction `str` (voir paragraphe 3.2.4.2) permet de convertir toute variable en chaîne de caractères. Il existe cependant une autre fonction `repr`, qui effectue cette conversion. Dans ce cas, le résultat peut être interprété par la fonction `eval` (voir paragraphe 3.4.2) qui se charge de la conversion inverse. Pour les types simples comme ceux présentés dans ce chapitre, ces deux fonctions retournent des résultats identiques. Pour l'exemple, `x` désigne n'importe quelle variable.

```
x == eval (repr(x)) # est toujours vrai (True)
x == eval (str (x)) # n'est pas toujours vrai
```

### 3.4.2 Informations fournies par *Python*

Bien que les fonctions ne soient définies que plus tard (paragraphe 4.4), il peut être intéressant de mentionner la fonction `dir` qui retourne la liste de toutes les variables créées et accessibles à cet instant du programme. L'exemple suivant :

```
x = 3
print dir ()
```

Retourne le résultat suivant :

```
['__builtins__', '__doc__', '__file__', '__name__', 'x']
```

Certaines variables - des chaînes des caractères - existent déjà avant même la première instruction. Elles contiennent différentes informations concernant l'environnement dans lequel est exécuté le programme *Python* :

| | |
|---|---|
| `__builtins__` | Ce module contient tous les éléments présents dès le début d'un programme *Python*, il contient entre autres les types présentés dans ce chapitre et des fonctions simples comme `range`. |
| `__doc__` | C'est une chaîne commentant le fichier, c'est une chaîne de caractères insérée aux premières lignes du fichiers et souvent entourée des symboles """ (voir chapitre 7). |
| `__file__` | Contient le nom du fichier où est écrit ce programme. |
| `__name__` | Contient le nom du module. |

La fonction `dir` est également pratique pour afficher toutes les fonctions d'un module. L'instruction `dir(sys)` affiche la liste des fonctions du module `sys` (voir chapitre 7).

**Remarque 3.21 : fonction `dir`**
La fonction `dir` appelée sans argument donne la liste des fonctions et variables définies à cet endroit du programme. Ce résultat peut varier selon qu'on se trouver dans une fonction, une méthode de classe ou à l'extérieur du programme. L'instruction `dir([])` donne la liste des méthodes qui s'appliquent à une liste.

De la même manière, la fonction `type` retourne une information concernant le type d'une variable.

```
x = 3
print x, type(x)     # affiche 3 <type 'int'>
x = 3.5
print x, type(x)     # affiche 3.5 <type 'float'>
```

Autre fonction intéressante, la fonction `eval` permet d'évaluer une chaîne de caractères. Le petit exemple suivant permet de tester toutes les opérations de calcul possibles entre deux entiers.

```
x  = 32
y  = 9
op = "+ - * / % // & | and or << >>".split ()
for o in op :
    s = str (x) + " " + o + "  " + str (y)
    print s, " = ", eval (s)
```

Ceci aboutit au résultat suivant :

```
32 +  9  =  41
32 -  9  =  23
32 *  9  =  288
32 /  9  =  3
32 %  9  =  5
32 //  9  =  3
32 &  9  =  0
32 |  9  =  41
32 and  9  =  9
32 or  9  =  32
32 <<  9  =  16384
32 >>  9  =  0
```

Le programme va créer une chaîne de caractères pour chacune des opérations et celle-ci sera évaluée grâce à la fonction `eval` comme si c'était une expression numérique. Il faut bien sûr que les variables que l'expression mentionne existent durant son évaluation.

### 3.4.3 Affectations multiples

Il est possible d'effectuer en *Python* plusieurs affectations simultanément.

```
x = 5       # affecte 5 à x
y = 6       # affecte 6 à y
x,y = 5,6   # affecte en une seule instruction 5 à x et 6 à y
```

Cette particularité reviendra lorsque les fonctions seront décrites puisqu'il est possible qu'une fonction retourne plusieurs résultats comme la fonction `divmod` illustrée par le programme suivant.

```
x,y = divmod (17,5)
print x,y                          # affiche 3 2
print "17 / 5 = 5 * ", x, " + ",y  # affiche 17 / 5 = 5 *  3  +  2
```

Le langage *Python* offre la possibilité d'effectuer plusieurs affectations sur la même ligne. Dans l'exemple qui suit, le couple $(5, 5)$ est affecté à la variable `point`, puis le couple `x`, `y` reçoit les deux valeurs du T-uple `point`.

```
x,y = point = 5,5
```

### 3.4.4 Ensemble

Le langage *Python* définit également ce qu'on appelle un ensemble. Il est définit par les classes `set` de type modifiable et la classe `frozenset` de type immuable. Ils n'acceptent que des types uniques et offre la plupart des opérations liées aux ensembles comme l'intersection, l'union. D'un usage moins fréquent, ils ne seront pas plus détaillés. La page http://docs.python.org/library/stdtypes.html#set décrit l'ensemble des fonctionnalités qui leur sont attachées.

```
print set ( (1,2,3) ) & set ( (2,3,5) )  # affiche set([2, 3])
```

Ce chapitre a présenté les différents types de variables définis par le langage *Python* pour manipuler des données ainsi que les opérations possibles avec ces types de données. Le chapitre suivant va présenter les tests, les boucles et les fonctions qui permettent de réaliser la plupart des programmes informatiques.

# Chapitre 4

# Syntaxe du langage *Python* (boucles, tests, fonctions)

---

Avec les variables, les boucles et les fonctions, on connaît suffisamment d'éléments pour écrire des programmes. Le plus difficile n'est pas forcément de les comprendre mais plutôt d'arriver à découper un algorithme complexe en utilisant ces briques élémentaires. C'est l'objectif des chapitres centrés autour des exercices. Toutefois, même si ce chapitre présente les composants élémentaires du langage, l'aisance qu'on peut acquérir en programmation vient à la fois de la connaissance du langage mais aussi de la connaissance d'algorithmes standards comme celui du tri ou d'une recherche dichotomique. C'est cette connaissance tout autant que la maîtrise d'un langage de programmation qui constitue l'expérience en programmation.

## 4.1 Les trois concepts des algorithmes

Les algorithmes sont composés d'instructions, ce sont des opérations élémentaires que le processeur exécute selon trois schémas :

| | |
|---|---|
| la séquence | enchaînement des instructions les unes à la suite des autres : passage d'une instruction à la suivante |
| le saut | passage d'une instruction à une autre qui n'est pas forcément la suivante (c'est une rupture de séquence) |
| le test | choix entre deux instructions |

Le saut n'apparaît plus de manière explicite dans les langages évolués car il est une source fréquente d'erreurs. Il intervient dorénavant de manière implicite au travers des boucles qui combinent un saut et un test comme le montre l'exemple suivant :

**Version avec boucles :**

```
initialisation de la variable moy à 0
faire pour i allant de 1 à N
      moy reçoit moy + ni
moy reçoit moy / N
```

**Version équivalente avec sauts :**

```
ligne 1 : initialisation de la variable moy à 0
ligne 2 : initialisation de la variable i à 1
ligne 3 : moy reçoit moy + ni
ligne 4 : i reçoit i + 1
ligne 5 : si i est inférieur ou égal à N alors aller à la ligne 3
ligne 6 : moy reçoit moy / N
```

Si tout programme peut se résumer à ces trois concepts, l'aisance que l'on peut acquérir dans tel ou tel langage dépend de la syntaxe qui les met en place. Cette syntaxe permet de transcrire des algorithmes rédigés en français comme celui de l'exemple ci-dessus pour aboutir au programme suivant, écrit en *Python* et interprétable par l'ordinateur via l'interpréteur associé au langage *Python*.

```
moy = 0
for i in range(1,N):
    moy += n [i]
moy /= N
```

Le premier élément de cette syntaxe est constitué de ses mots-clés (`for` et `in`) (voir également table 4.1) et des symboles (=, + =, / =, [, ], (, ), :) (voir table 4.2) La fonction `iskeyword` du module `keyword` [1] permet de savoir si un mot-clé donné fait partie du langage *Python*.

```
import keyword
print keyword.iskeyword("for")     # affiche True
print keyword.iskeyword("until")   # affiche False
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| and | as | assert | break | class | continue | def | del |
| elif | else | except | exec | finally | for | from | global |
| if | import | in | is | lambda | not | or | pass |
| print | raise | return | try | while | with | yield | |

**Table 4.1 :** *Mots-clés du langage Python.*

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + | − | * | ** | / | // | % | << | >> | & | \| | ˆ | \ |
| ∼ | < | > | <= | >= | == | ! = | <> | ( | ) | [ | ] | |
| { | } | ” | , | : | . | ′ | = | ; | + = | − = | * = | |
| / = | // = | % = | & = | \| = | ˆ= | >>= | <<= | ** = | ”””” | ′′′ | # | |

**Table 4.2 :** *Symbole du langage Python, certains ont plusieurs usages comme : qui est utilisé à chaque test ou boucle et qui permet aussi de déterminer une plage d'indices dans un tableau. L'opérateur <> est identique à ! = mais est déclaré obsolète (deprecated) : il est susceptible de disparaître dans une prochaine version de Python.*

**Remarque 4.1 : syntaxe et espaces**
Les espaces entre les mots-clés et les symboles ont peu d'importance, il peut n'y en avoir aucun comme dix. Les espaces servent à séparer un mot-clé, un nombre d'une variable.

1. Les modules seront décrits au chapitre 7. Dans ce cas-ci, un module est une librairie de fonctionnalités. La syntaxe `keyword.iskeyword` permet d'exécuter la fonctionnalité `iskeyword` du module `keyword`.

## 4.2 Tests

### 4.2.1 Définition et syntaxe

**Définition 4.2 : test**
Les tests permettent d'exécuter des instructions différentes selon la valeur d'une condition logique.

**syntaxe 4.3 : test**

```
if condition1 :
   instruction1
   instruction2
   ...
else :
   instruction3
   instruction4
   ...
```

La clause `else` est facultative. Lorsque la condition `condition1` est fausse et qu'il n'y a aucune instruction à exécuter dans ce cas, la clause `else` est inutile. La syntaxe du test devient :

```
if condition1 :
   instruction1
   instruction2
   ...
```

S'il est nécessaire d'enchaîner plusieurs tests d'affilée, il est possible de condenser l'écriture avec le mot-clé `elif` :

```
if condition1 :
   instruction1
   instruction2
   ...
elif condition2 :
   instruction3
   instruction4
   ...
elif condition3 :
   instruction5
   instruction6
   ...
else :
   instruction7
   instruction8
   ...
```

**Remarque 4.4 : indentation**
Le décalage des instructions par rapport aux lignes contenant les mots-clés `if`, `elif`, `else` est très important : il fait partie de la syntaxe du langage et s'appelle l'*indentation* (voir paragraphe 4.5). Celle-ci permet de grouper les instructions ensemble. Le programme suivant est syntaxiquement correct même si le résultat n'est pas celui désiré.

```
x = 1
if x > 0 :
   signe = 1
   print "le nombre est positif"
else :
   signe = -1
print "le nombre est négatif"  # ligne mal indentée
print "signe = ", signe
```

Une ligne est mal indentée (`print "le nombre est ngatif"`). Elle ne devrait être exécutée que si la condition `x > 0` n'est pas vérifiée. Le fait qu'elle soit alignée avec les premières instructions du programme fait que son exécution n'a plus rien à voir avec cette condition. La programme répond de manière erronée :

```
le nombre est positif
le nombre est négatif
signe = 1
```

Dans certains cas, l'interpréteur *Python* ne sait pas à quel bloc attacher une instruction, c'est le cas de l'exemple suivant, la même ligne a été décalée de deux espaces, ce qui est différent de la ligne qui précède et de la ligne qui suit.

```
x = 1
if x > 0 :
    signe = 1
    print "le nombre est positif"
else :
    signe = -1
  print "le nombre est négatif"     # ligne mal indentée
print "signe = ", signe
```

L'interpréteur retourne l'erreur suivante :

```
  File "test.py", line 7
    print "le nombre est négatif"
                                ^
IndentationError: unindent does not match any outer indentation level
```

### 4.2.2 Comparaisons possibles

Les comparaisons possibles entre deux entités sont avant tout numériques mais ces opérateurs peuvent être définis pour tout type (voir chapitre 5), notamment sur les chaînes de caractères pour lesquels les opérateurs de comparaison transcrivent l'ordre alphabétique.

| | |
|---|---|
| < , > | inférieur, supérieur |
| <= , >= | inférieur ou égal, supérieur ou égal |
| == , != | égal, différent |
| `is` , `not is` | `x is y` vérifie que `x` et `y` sont égaux, `not is`, différents, l'opérateur `is` est différent de l'opérateur ==, il est expliqué au paragraphe 3.3.2 (page 42). |
| `in`, `not in` | appartient, n'appartient pas |

### 4.2.3 Opérateurs logiques

Il existe trois opérateurs logiques qui combinent entre eux les conditions.

| | |
|---|---|
| `not` | négation |
| `and` | et logique |
| `or` | ou logique |

**Remarque 4.5 : priorité des opérations**
La priorité des opérations numériques est identique à celle rencontrée en mathématiques. L'opérateur puissance vient en premier, la multiplication/division ensuite puis l'addition/soustraction. Ces opérations sont prioritaires sur les opérateurs de comparaisons (>, <, ==, ...) qui sont eux-mêmes sur les opérateurs logiques `not`, `and`, `or`. Il est tout de même conseillé d'ajouter des parenthèses en cas de doute.

### 4.2.4 Ecriture condensée

Il existe deux écritures condensées de tests. La première consiste à écrire un test et l'unique instruction qui en dépend sur une seule ligne.

```
if condition :
    instruction1
else :
    instruction2
```

Ce code est équivalent à :

```
if condition : instruction1
else : instruction2
```

Le second cas d'écriture condensée concerne les comparaisons enchaînées. Le test `if 3 < x and x < 5 : instruction` peut être condensé par `if 3 < x < 5 : instruction`. Il est ainsi possible de juxtaposer autant de comparaisons que nécessaire : `if 3 < x < y < 5 : instruction`.

Le mot-clé `in` permet également de condenser certains tests lorsque la variable à tester est entière. `if x == 1 or x == 6 or x == 50 :` peut être résumé simplement par `if x in (1, 6, 50) :`.

### 4.2.5 Exemple

L'exemple suivant associe à la variable `signe` le signe de la variable `x`.

```
x = -5
if x < 0 :
   signe = -1
elif x == 0 :
   signe = 0
else :
   signe = 1
```

Son écriture condensée lorsqu'il n'y a qu'une instruction à exécuter :

```
x = -5
if x < 0 : signe = -1
elif x == 0 : signe = 0
else : signe = 1
```

Le programme suivant saisit une ligne au clavier et dit si c'est "oui" ou "non" qui a été saisi.

```
s = raw_input ("dites oui : ")    # voir remarque suivante
if s == "oui" or s [0:1] == "o" or s [0:1] == "0" or s == "1" :
    print "oui"
else :
    print "non"
```

**Remarque 4.6 : fonction `raw_input`**
La fonction `raw_input` invite l'utilisateur d'un programme à saisir une réponse lors de l'exécution du programme. Tant que la touche entrée n'a pas été pressée, l'exécution du programme ne peut continuer. Cette fonction est en réalité peu utilisée. Les interfaces graphiques sont faciles d'accès en *Python*, on préfère donc saisir une réponse via une fenêtre plutôt qu'en ligne de commande. L'exemple suivant montre comment remplacer cette fonction à l'aide d'une fenêtre graphique, le résultat apparaît dans la figure 4.1. On peut améliorer la fonction `question` en précisant une valeur par défaut par exemple, le chapitre 9 sera plus explicite à ce sujet.

```
import Tkinter
def question (legende) :
    reponse = [""]
    root = Tkinter.Tk ()
    root.title ("pseudo raw_input")
    Tkinter.Label (text = legende).pack (side = Tkinter.LEFT)
    s = Tkinter.Entry (text= "def", width=80)
    s.pack (side = Tkinter.LEFT)
    def rget () :
        reponse [0] = s.get ()
        root.destroy ()
    Tkinter.Button (text = "ok", command = rget).pack (side = Tkinter.LEFT)
    root.mainloop ()
    return reponse [0]

print "reponse ", question ("texte de la question")
```

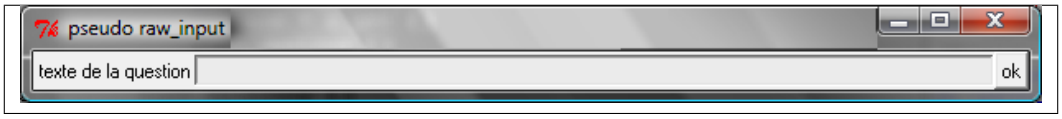



**Figure 4.1 :** *Fenêtre apparaissant lors de l'exécution du programme page 55 proposant une version graphique de la fonction* `raw_input`.

**Remarque 4.7 :** `if None, if 0`
L'écriture de certains tests peut encore être réduite lorsqu'on cherche à comparer une variable à 0, `None`, `False`, `True`.

```
if v != None :   # équivalent à     if v :
if v == None :   # équivalent à     if not v :
if v != 0 :      # équivalent à     if v :
if v == 0 :      # équivalent à     if not v :
if v == True :   # équivalent à     if v :
if v == False :  # équivalent à     if not v :
```

### 4.2.6 Passer, instruction `pass`

Dans certains cas, aucune instruction ne doit être exécutée même si un test est validé. En *Python*, le corps d'un test ne peut être vide, il faut utiliser l'instruction `pass`. Lorsque celle-ci est manquante, *Python* affiche un message d'erreur.

```
signe = 0
x = 0
if x < 0:
   signe = -1
elif x == 0:
   pass          # signe est déjà égal à 0
else :
   signe = 1
```

Dans ce cas précis, si l'instruction `pass` est oubliée, l'interpréteur *Python* génère l'erreur suivante :

```
File "nopass.py", line 6
    else :
    ^
IndentationError: expected an indented block
```

## 4.3 Boucles

**Définition 4.8 : boucle**
Les boucles permettent de répéter une séquence d'instructions tant qu'une certaine condition est vérifiée.

Le langage *Python* propose deux types de boucles. La boucle `while` suit scrupuleusement la définition 4.8. La boucle `for` est une boucle `while` déguisée (voir paragraphe 4.3.2), elle propose une écriture simplifiée pour répéter la même séquence d'instructions pour tous les éléments d'un même ensemble.

### 4.3.1 Boucle `while`

#### 4.3.1.1 Syntaxe et exemples

L'implémentation d'une boucle de type `while` suit le schéma d'écriture suivant :

**syntaxe 4.9 : boucle `while`**

```
while cond :
    instruction 1
    ...
    instruction n
```

Où `cond` est une condition qui détermine la poursuite de la répétition des instructions incluses dans la boucle. Tant que celle-ci est vraie, les instructions 1 à $n$ sont exécutées.

Tout comme les tests, la remarque 4.4 concernant l'indentation reste vraie. Le décalage des lignes d'un cran vers la droite par rapport à l'instruction `while` permet de les inclure dans la boucle comme le montre l'exemple suivant.

```
n = 0
while n < 3:
   print "à l'intérieur ", n
   n += 1
print "à l'extérieur ", n
```

Ce petit programme affiche à l'écran :

```
à l'intérieur  0
à l'intérieur  1
à l'intérieur  2
à l'extérieur  3
```

#### 4.3.1.2 Condition

Les conditions suivent la même syntaxe que celles définies lors des tests (voir paragraphes 4.2.2 et 4.2.3). A moins d'inclure l'instruction `break`[2] qui permet de sortir prématurément d'une boucle, la condition qui régit cette boucle doit nécessairement être modifiée à l'intérieur de celle-ci. Dans le cas contraire, on appelle une telle boucle une *boucle infinie* puisqu'il est impossible d'en sortir. L'exemple suivant contient une boucle infinie car le symbole = est manquant dans la dernière instruction. La variable `n` n'est jamais modifiée et la condition `n < 3` toujours vraie.

```
n = 0
while n < 3 :
```

2. L'instruction `break` est décrite au paragraphe 4.3.4.2, page 61.

```
   print n
   n + 1        # n n'est jamais modifié, l'instruction correcte serait n += 1,
```

### 4.3.2 Boucle `for`

#### 4.3.2.1 Syntaxe et exemples

L'implémentation d'une boucle de type `for` suit le schéma d'écriture suivant :

**syntaxe 4.10 : boucle `for`**

```
for x in set :
    instruction 1
    ...
    instruction n
```

Où `x` est un élément de l'ensemble `set`. Les instructions 1 à $n$ sont exécutées pour chaque élément `x` de l'ensemble `set`. Cet ensemble peut être une chaîne de caractères, un T-uple, une liste, un dictionnaire ou tout autre type incluant des itérateurs qui sont présentés au chapitre 5 concernant les classes (page 94).

Tout comme les tests, la remarque 4.4 concernant l'indentation reste vraie. L'exemple suivant affiche tous les éléments d'un T-uple à l'aide d'une boucle `for`.

```
t = (1,2,3,4)
for x in t:       # affiche les nombres 1,2,3,4
    print x       # chacun sur une ligne différente
```

Lors de l'affichage d'un dictionnaire, les éléments n'apparaissent pas triés ni dans l'ordre dans lequel ils y ont été insérés. L'exemple suivant montre comment afficher les clés d'un dictionnaire dans un ordre croissant.

```
d = { 1:2, 3:4, 5:6, 7:-1, 8:-2 }
print d               # affiche le dictionnaire {8: -2, 1: 2, 3: 4, 5: 6, 7: -1}
k = d.keys ()
print k               # affiche les clés [8, 1, 3, 5, 7]
k.sort ()
print k               # affiche les clés triées [1, 3, 5, 7, 8]
for x in k:           # affiche les éléments du dictionnaire
    print x,":",d [x]  # triés par clés croissantes
```

La boucle la plus répandue est celle qui parcourt des indices entiers compris entre 0 et $n-1$. On utilise pour cela la boucle `for` et la fonction `range` ou `xrange` comme dans l'exemple qui suit. La différence entre ces deux fonctions a déjà été présentée au paragraphe 3.3.1.3 (page 41).

```
sum = 0
N   = 10
for n in range(0,N):     # ou for n in xrange(0,N):  (plus rapide)
   sum += n              # additionne tous les entiers compris entre 0 et N-1
```

Ou encore pour une liste quelconque :

```
l   = [ 4, 5, 3, -6, 7, 9]
sum = 0
for n in range(0,len (l)):  # ou for n in xrange(0,len (l)) :  (plus rapide)
   sum += l [n]             # additionne tous les éléments de l
```

#### 4.3.2.2 Listes et boucle `for`

Le paragraphe 3.3.1.4 a montré comment le mot-clé `for` peut être utilisé pour simplifier la création d'une liste à partir d'une autre. La syntaxe suit le schéma suivant :

> **syntaxe 4.11 : boucle `for`**
>
> `[ expression for x in ensemble ]`
>
> Où `expression` est une expression numérique incluant ou non `x`, la variable de la boucle, `ensemble` est un ensemble d'éléments, T-uple, liste, dictionnaire.

Cette syntaxe permet de résumer en une ligne la création de la séquence `y` du programme suivant.

```
x = range(0,5)
y = list ()
for i in x :
   y.append (i+1)
print y             # affiche [1,2,3,4,5]
```

Les trois lignes du milieu sont maintenant résumées en une seule :

```
x = range(0,5)
y = [ i+1 for i in x]   # résume trois lignes du programme précédent
print y                 # affiche [1,2,3,4,5]
```

D'autres exemples de cette syntaxe réduite, ils impliquent des tests, des boucles imbriquées.

```
x = range(0,5)
y = [ i for i in x if i % 2 == 0]   # sélection des éléments pairs
print y                             # affiche [0,2,4]
z = [ i+j for i in x for j in x]    # construit tous les nombres i+j possibles
print z                             # affiche [0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 2, 3,
                                    # 4, 5, 6, 3, 4, 5, 6, 7, 4, 5, 6, 7, 8]
```

Cette écriture condensée est bien souvent plus lisible que la première. Tout dépend des préférences de celui qui programme.

#### 4.3.2.3 Itérateurs

Toute boucle `for` peut s'appliquer sur un objet muni d'un itérateur tels que les chaînes de caractères, T-uples, les listes, les dictionnaires.

```
d = ["un", "deux", "trois"]
for x in d:
   print x          # affichage de tous les éléments de d
```

Cette syntaxe réduite a déjà été introduite pour les listes et les dictionnaires au chapitre précédent. Il existe une version équivalent avec la boucle `while` utilisant de façon explicite les itérateurs. Il peut être utile de lire le chapitre suivant sur les classes et le chapitre 6 sur les exceptions avant de revenir sur la suite de cette section qui n'est de toutes façons pas essentielle. L'exemple précédent est convertible en une boucle `while` en faisant apparaître explicitement les itérateurs (voir paragraphe 5.4.2). Un itérateur est un objet qui permet de parcourir aisément un ensemble. La fonction `it = iter(e)` permet d'obtenir un itérateur `it` sur l'ensemble `e`. L'appel à l'instruction `it.next()` parcourt du premier élément jusqu'au dernier en retournant la valeur de chacun d'entre eux. Lorsqu'il n'existe plus d'élément, l'exception `StopIteration` est déclenchée (voir paragraphe 6). Il suffit de l'intercepter pour mettre fin au parcours.

```
d = ["un", "deux", "trois"]
it = iter (d)                    # obtient un itérateur sur d
```

```
while True:
    try: x = it.next ()           # obtient l'élément suivant, s'il n'existe pas
    except StopIteration: break   # déclenche une exception
    print x                       # affichage de tous les éléments de d
```

#### 4.3.2.4 Plusieurs variables de boucles

Jusqu'à présent, la boucle `for` n'a été utilisée qu'avec une seule variable de boucle, comme dans l'exemple suivant où on parcourt une liste de T-uple pour les afficher.

```
d = [ (1,0,0), (0,1,0), (0,0,1) ]
for v in d: print v
```

Lorsque les éléments d'un ensemble sont des T-uples, des listes ou des dictionnaires composés de taille fixe, il est possible d'utiliser une notation qui rappelle les affectations multiples (voir paragraphe 3.4.3). L'exemple précédent devient dans ce cas :

```
d = [ (1,0,0), (0,1,0), (0,0,1) ]
for x,y,z in d: print x,y,z
```

Cette écriture n'est valable que parce que chaque élément de la liste `d` est un T-uple composé de trois nombres. Lorsqu'un des éléments est de taille différente à celle des autres, comme dans l'exemple suivant, une erreur survient.

```
d = [ (1,0,0), (0,1,0,6), (0,0,1) ]  # un élément de taille quatre
for x,y,z in d: print x,y,z
```

Ce programme génère l'erreur suivante :

```
Traceback (most recent call last):
  File "c:\temp\delete.py", line 2, in -toplevel-
    for x,y,z in d: print x,y,z
ValueError: unpack tuple of wrong size
```

Cette syntaxe est très pratique associée à la fonction `zip` (voir paragraphe 3.3.1.5). Il est alors possible de parcourir plusieurs séquences (T-uple, liste, dictionnaire) simultanément.

```
a = range(0,5)
b = [x**2 for x in a]
for x,y in zip (a,b):
    print y, " est le carré de ", x
    # affichage à droite
```

```
0  est le carré de  0
1  est le carré de  1
4  est le carré de  2
9  est le carré de  3
16  est le carré de  4
```

### 4.3.3 Ecriture condensée

Comme pour les tests, lorsque les boucles ne contiennent qu'une seule instruction, il est possible de l'écrire sur la même ligne que celle de la déclaration de la boucle `for` ou `while`.

```
d = ["un", "deux", "trois"]
for x in d: print x          # une seule instruction
```

Il existe peu de cas où la boucle `while` s'écrit sur une ligne car elle inclut nécessairement une instruction permettant de modifier la condition d'arrêt.

```
d = ["un", "deux", "trois"]
i = 0
while d [i] != "trois" : i += 1
print "trois a pour position ", i
```

### 4.3.4 Pilotage d'une boucle

#### 4.3.4.1 Passer à l'itération suivante : `continue`

Pour certains éléments d'une boucle, lorsqu'il n'est pas nécessaire d'exécuter toutes les instructions, il est possible de passer directement à l'élément suivant ou l'itération suivante. Le programme suivant utilise la crible d'Eratosthène[3] pour dénicher tous les nombres premiers compris entre 1 et 99.

```
d = dict ()
for i in range(1,100):
    d [i] = True                # d [i] est vrai si i est un nombre premier
                                # au début, comme on ne sait pas, on suppose
                                # que tous les nombres sont premiers
for i in range(2,100):

   if not d [i]: continue       # si d [i] est faux,
                                # les multiples de i ont déjà été cochés
                                # et peut passer à l'entier suivant
   for j in range (2,100):
       if i*j < 100:
           d [i*j] = False      # d [i*j] est faux pour tous les multiples de i
                                # inférieurs à 100
print "liste des nombres premiers"
for i in d:
    if d [i]: print i
```

Ce programme est équivalent au suivant :

```
d = dict ()
for i in range(1,100): d [i] = True

for i in range(2,100):
   if d [i]:
       for j in range (2,100):
           if i*j < 100 :
               d [i*j] = False

print "liste des nombres premiers"
for i in d:
    if d [i]: print i
```

Le mot-clé `continue` évite de trop nombreuses indentations et rend les programmes plus lisibles.

#### 4.3.4.2 Sortie prématurée : `break`

Lors de l'écriture d'une boucle `while`, il n'est pas toujours adéquat de résumer en une seule condition toutes les raisons pour lesquelles il est nécessaire d'arrêter l'exécution de cette boucle. De même, pour une boucle `for`, il n'est pas toujours utile de visiter tous les éléments de l'ensemble à parcourir. C'est le cas par exemple lorsqu'on recherche un élément, une fois qu'il a été trouvé, il n'est pas nécessaire d'aller plus loin. L'instruction `break` permet de quitter l'exécution d'une boucle.

```
l = [6,7,5,4,3]
n = 0
c = 5
for x in l:
```

3. Le crible d'Eratosthène est un algorithme permettant de déterminer les nombres premiers. Pour un nombre premier $p$, il paraît plus simple de considérer tous les entiers de $p-1$ à 1 pour savoir si l'un d'eux divise $p$. C'est ce qu'on fait lorsqu'on doit vérifier le caractère premier d'un seul nombre. Pour plusieurs nombres à la fois, le crible d'Eratosthène est plus efficace : au lieu de s'intéresser aux diviseurs, on s'intéresse aux multiples d'un nombre. Pour un nombre $i$, on sait que $2i$, $3i$, ... ne sont pas premiers. On les raye de la liste. On continue avec $i+1$, $2(i+1)$, $3(i+1)$...

```
   if x == c: break   # l'élément a été trouvé, on sort de la boucle
   n += 1             # si l'élément a été trouvé, cette instruction
                      # n'est pas exécutée
print "l'élément ",c, " est en position ",
print n               # affiche l'élément 5 est en position 2
```

Si deux boucles sont imbriquées, l'instruction `break` ne sort que de la boucle dans laquelle elle est insérée. L'exemple suivant vérifie si un entier est la somme des carrés de deux entiers compris entre 1 et 20.

```
set = range (1,21)
n = 53
for x in set:
    for y in set:
        c = x*x + y*y
        if c == n: break
    if c == n: break   # cette seconde instruction break est nécessaire
                       # pour sortir de la seconde boucle
                       # lorsque la solution a été trouvée
if c == n:
    # le symbole \ permet de passer à la ligne sans changer d'instruction
    print n, " est la somme des carrés de deux entiers :", \
          x, "*", x, "+", y, "*", y, "=", n
else:
    print n, " n'est pas la somme des carrés de deux entiers"
```

Le programme affiche :

```
53  est la somme des carrés de deux entiers : 2 * 2 + 7 * 7 = 53
```

#### 4.3.4.3 Fin normale d'une boucle : `else`

Le mot-clé `else` existe aussi pour les boucles et s'utilise en association avec le mot-clé `break`. L'instruction `else` est placée à la fin d'une boucle, indentée au même niveau que `for` ou `while`. Les lignes qui suivent ne sont exécutées que si aucune instruction `break` n'a été exécutée dans le corps de la boucle. On reprend l'exemple du paragraphe précédent. On recherche cette fois-ci la valeur 1 qui ne se trouve pas dans la liste `L`. Les lignes suivant le test `if x == c` ne seront jamais exécutées au contraire de la dernière.

```
L = [6,7,5,4,3]
n = 0
c = 1
for x in L :
   if x == c :
       print "l'élément ", c, " est en position ", n
       break
   n += 1
else:
   print "aucun élément ", c, " trouvé"  # affiche aucun élément  1  trouvé
```

En fait, les lignes dépendant de la clause `else` seront exécutées dans tous les cas où l'exécution de la boucle n'est pas interrompue, par une instruction `break`, une instruction `return` (voir les fonctions, paragraphe 4.4), par la levée d'une exception (voir paragraphe 6).

#### 4.3.4.4 Suppression ou ajout d'éléments lors d'une boucle

En parcourant la liste en se servant des indices, il est possible de supprimer une partie de cette liste. Il faut néanmoins faire attention à ce que le code ne produise pas d'erreur comme c'est le cas pour le suivant. La boucle `for` parcourt la liste `range(0,len(li))` qui n'est pas modifiée en même temps que l'instruction `del li[i : i + 2]`.

```
li = range (0,10)
print li             # affiche [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for i in range (0, len (li)):
    if i == 5:
        del li [i:i+2]
    print li [i]     # affiche successivement 0, 1, 2, 3, 4, 7, 8, 9 et
                     # produit une erreur
print li
```

Le programme suivant marche parfaitement puisque cette fois-ci la boucle parcourt la liste `li`. En revanche, pour la suppression d'une partie de celle-ci, il est nécessaire de conserver en mémoire l'indice de l'élément visité. C'est le rôle de la variable `i`.

```
li = range (0,10)
print li             # affiche [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
i = 0
for t in li:
    if i == 5:
        del li [i:i+2]
    i = i+1
    print t          # affiche successivement 0, 1, 2, 3, 4, 5, 8, 9
print li             # affiche [0, 1, 2, 3, 4, 7, 8, 9]
```

Le langage *Python* offre la possibilité de supprimer des éléments d'une liste alors même qu'on est en train de la parcourir. Le programme qui suit ne marche pas puisque l'instruction `del i` ne supprime pas un élément de la liste mais l'identificateur `i` qui prendra une nouvelle valeur lors du passage suivant dans la boucle.

```
li = range (0,10)
print li        # affiche [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for i in li:
    if i == 5:
        del i
print li        # affiche [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

On pourrait construire des exemples similaires dans le cadre de l'ajout d'un élément à la liste. Il est en règle générale déconseillé de modifier une liste, un dictionnaire pendant qu'on le parcourt. Malgré tout, si cela s'avérait indispensable, il convient de faire plus attention dans ce genre de situations.

## 4.4 Fonctions

Les fonctions sont des petits programmes qui effectuent des tâches plus précises que le programme entier. On peut effectivement écrire un programme sans fonction mais ils sont en général illisibles. Utiliser des fonctions implique de découper un algorithme en tâches élémentaires. Le programme final est ainsi plus facile à comprendre. Un autre avantage est de pouvoir plus facilement isoler une erreur s'il s'en produit une : il suffit de tester une à une les fonctions pour déterminer laquelle retourne un mauvais résultat. L'avantage le plus important intervient lorsqu'on doit effectuer la même chose à deux endroits différentes d'un programme : une seule fonction suffit et elle sera appelée à ces deux endroits. Les programmes utilisant des fonctions sont incontestablement plus courts.

Enfin, pour les utilisateurs experts : en langage *Python*, les fonctions sont également des variables, elles ont un identificateur et une valeur qui est dans ce cas un morceau de code. Cette précision explique certaines syntaxes du chapitre 9 sur les interfaces graphiques ou celle introduite en fin de chapitre au paragraphe 4.7.10.

### 4.4.1 Définition, syntaxe

**Définition 4.12 : fonction**
Une fonction est une partie d'un programme - ou sous-programme - qui fonctionne indépendamment du reste du programme, qui reçoit une liste de paramètres et qui retourne un résultat. Le corps de la fonction désigne toute instruction du programme qui ne peut être exécutée que si la fonction est appelée.

Les fonctions sont couramment utilisées afin de ne pas recopier le code associé à une tâche couramment répétée. Elles rendent également le programme plus lisible dans la mesure où il est un assemblage de fonctions les plus courtes possibles. D'une manière générale, on évite les grandes fonctions pour préférer la scinder en plusieurs petites fonctions, chacune affectée à une tâche précise.

**syntaxe 4.13 : fonction, définition**

```
def fonction_nom (param_1, param_2, ..., param_n) :
    instruction_1
    instruction_2
    ...
    instruction_n
    return resultat_1, ..., resultat_n
```

Où `fonction_nom` est le nom de la fonction. Celui-ci commence par une lettre, peut inclure des chiffres mais ne doit pas contenir de lettres accentuées. `param_1` à `param_n` sont les noms des paramètres et `resultat_1` à `resultat_n` sont les résultats retournés par la fonction. S'il n'y en a aucun, l'instruction `return` est facultative ou peut être utilisée seule sans être suivie par une valeur ou une variable. Cette instruction peut apparaître plusieurs fois dans le code de la fonction mais une seule d'entre elles sera exécutée. A partir de ce moment, toute autre instruction de la fonction sera ignorée. Les instructions associées à une fonction doivent être indentées par rapport au mot-clé `def`.

Pour exécuter une fonction ainsi définie, il suffit de suivre la syntaxe suivante :

**syntaxe 4.14 : fonction, appel**

```
x_1, ..., x_n = fonction_nom (valeur_1, valeur_2, ..., valeur_n)}
```

Où `fonction_nom` est le nom de la fonction, `valeur_1` à `valeur_n` sont les noms des paramètres, `x_1` à `x_n` reçoivent les résultats retournés par la fonction. Cette affectation est facultative. On peut donc appeler la fonction comme ceci :

```
fonction_nom (valeur_1, valeur_2, ..., valeur_n)
```

Les résultats ne seront pas conservés.

Lorsqu'on commence à programmer, il arrive parfois qu'on confonde le rôle des mots-clés `print` et `return`. Dans ce cas, il faut se reporter à la remarque 11.3 page 229.

### 4.4.2 Exemple

Le programme suivant utilise deux fonctions. La première convertit des coordonnées cartésiennes en coordonnées polaires. Elle prend deux réels en paramètres et retourne deux autres réels. La seconde fonction affiche les résultats de la première pour tout couple de valeurs $(x, y)$. Elle ne retourne aucun résultat.

```
import math
def coordonnees_polaires (x,y):
    rho     = math.sqrt(x*x+y*y)   # calcul la racine carrée de x*x+y*y
    theta   = math.atan2 (y,x)     # calcule l'arc tangente de y/x en tenant
                                   # compte des signes de x et y
```

```
    return rho, theta

def affichage (x,y):
    r,t = coordonnees_polaires(x,y)
    print "cartésien (%f,%f) --> polaire (%f,%f degrés)" \
                % (x,y,r,math.degrees(t))

affichage (1,1)
affichage (0.5,1)
affichage (-0.5,1)
affichage (-0.5,-1)
affichage (0.5,-1)
```

Le programme affiche les lignes suivantes :

```
cartésien (1.000000,1.000000)   --> polaire (1.414214,  45.000000 degrés)
cartésien (0.500000,1.000000)   --> polaire (1.118034,  63.434949 degrés)
cartésien (-0.500000,1.000000)  --> polaire (1.118034, 116.565051 degrés)
cartésien (-0.500000,-1.000000) --> polaire (1.118034,-116.565051 degrés)
cartésien (0.500000,-1.000000)  --> polaire (1.118034, -63.434949 degrés)
```

### 4.4.3 Paramètres avec des valeurs par défaut

Lorsqu'une fonction est souvent appelée avec les mêmes valeurs pour ses paramètres, il est possible de spécifier pour ceux-ci une valeur par défaut.

**syntaxe 4.15 : fonction, valeur par défaut**

```
def fonction_nom (param_1, param_2 = valeur_2, ..., param_n = valeur_n):
    ...
```

Où `fonction_nom` est le nom de la fonction. `param_1` à `param_n` sont les noms des paramètres, `valeur_2` à `valeur_n` sont les valeurs par défaut des paramètres `param_2` à `param_n`. La seule contrainte lors de cette définition est que si une valeur par défaut est spécifiée pour un paramètre, alors tous ceux qui suivent devront eux aussi avoir leur valeur par défaut.

Exemple :

```
def commander_carte_orange (nom, prenom, paiement = "carte", nombre = 1, zone = 2):
    print "nom : ", nom
    print "prénom : ", prenom
    print "paiement : ", paiement
    print "nombre : ", nombre
    print "zone :", zone

commander_carte_orange ("Dupré", "Xavier", "chèque")
        # les autres paramètres nombre et zone auront pour valeur
        # leurs valeurs par défaut
```

Il est impossible qu'un paramètre sans valeur par défaut associée se situe après un paramètre dont une valeur par défaut est précisée. Le programme suivant ne pourra être exécuté.

```
def commander_carte_orange (nom, prenom, paiement = "carte", nombre = 1, zone):
    print "nom : ", nom
    # ...
```

Il déclenche l'erreur suivante :

```
  File "problem_zone.py", line 1
    def commander_carte_orange (nom, prenom, paiement = "carte", nombre = 1, zone):
SyntaxError: non-default argument follows default argument
```

**Remarque 4.16 : valeurs par défaut de type modifiable**
Les valeurs par défaut de type modifiable (liste, dictionnaire, classes) peuvent introduire des erreurs inattendues dans les programmes comme le montre l'exemple suivant :

```
def fonction (l = [0,0]) :
    l [0] += 1
    return l

print fonction ()         # affiche [1,0] : résultat attendu
print fonction ()         # affiche [2,0] : résultat surprenant
print fonction ( [0,0])   # affiche [1,0] : résultat attendu
```

L'explication provient du fait que la valeur par défaut est une liste qui n'est pas recréée à chaque appel : c'est la même liste à chaque fois que la fonction est appelée sans paramètre. Pour remédier à cela, il faudrait écrire :

```
import copy
def fonction (l = [0,0]) :
    l = copy.copy (l)
    l [0] += 1
    return l
```

L'exercice 13.4.7 (page 293) propose un exemple plus complet, voire retors.

### 4.4.4 Ordre des paramètres

Le paragraphe 4.4.1 a présenté la syntaxe d'appel a une fonction. Lors de l'appel, le nom des paramètres n'intervient plus, supposant que chaque paramètre reçoit pour valeur celle qui a la même position que lui lors de l'appel à la fonction. Il est toutefois possible de changer cet ordre en précisant quel paramètre doit recevoir quelle valeur.

> **syntaxe 4.17 : fonction, ordre des paramètres**
>
> `x_1, ..., x_n = fonction_nom (param_1 = valeur_1, ..., param_n = valeur_n)`
>
> Où `fonction_nom` est le nom de la fonction, `param_1` à `param_n` sont les noms des paramètres, `valeur_1` à `valeur_n` sont les valeurs que reçoivent ces paramètres. Avec cette syntaxe, l'ordre d'écriture n'importe pas. La valeur `valeur_i` sera toujours attribuée à `param_i`. Les variables `x_1` à `x_n` reçoivent les résultats retournés par la fonction. L'ordre des résultats ne peut pas être changé. S'il y a plusieurs résultats retournés, il est impossible de choisir lesquels conserver : soit tous, soit aucun.

Exemple :

```
def identite (nom, prenom):
    print "nom : ", nom, " prénom : ", prenom

identite("Xavier", "Dupré")                 # nom :  Xavier prénom :  Dupré
identite(prenom = "Xavier", nom = "Dupré")  # nom :  Dupré  prénom :  Xavier
```

Cette possibilité est intéressante surtout lorsqu'il y a de nombreux paramètres par défaut et que seule la valeur d'un des derniers paramètres doit être changée.

```
def commander_carte_orange (paiement = "carte", nombre = 1, zone = 2):
    print "paiement : ", paiement
    print "nombre : ", nombre
    print "zone :", zone

commander_carte_orange (zone = 5)  # seule la valeur par défaut
                                   # du paramètre zone sera changée
```

### 4.4.5 Surcharge de fonction

Contrairement à d'autres langages, *Python* n'autorise pas la surcharge de fonction. Autrement dit, il n'est pas possible que plusieurs fonctions portent le même nom même si chacune d'entre elles a un nombre différent de paramètres.

```
def fonction (a,b):
    return a + b

def fonction (a,b,c):
    return a + b + c

print fonction (5,6)
print fonction (5,6,7)
```

Le petit programme précédent est syntaxiquement correct mais son exécution génère une erreur parce que la seconde définition de la fonction `fonction` efface la première.

```
Traceback (most recent call last):
  File "cours4.py", line 7, in ?
    print fonction (5,6)
TypeError: fonction() takes exactly 3 arguments (2 given)
```

### 4.4.6 Commentaires

Le langage *Python* propose une fonction `help` qui retourne pour chaque fonction un commentaire ou mode d'emploi qui indique comment se servir de cette fonction. L'exemple suivant affiche le commentaire associé à la fonction `round`.

```
>>> help (round)
```

```
Help on built-in function round:

round(...)
    round(number[, ndigits]) -> floating point number

    Round a number to a given precision in decimal digits (default 0 digits).
    This always returns a floating point number.  Precision may be negative.
```

Lorsqu'on utilise cette fonction `help` sur la fonction `coordonnees_polaires` définie dans l'exemple du paragraphe précédent 4.4.2, le message affiche n'est pas des plus explicites.

```
>>> help (coordonnees_polaires)
```

```
Help on function coordonnees_polaires in module __main__:

coordonnees_polaires(x, y)
```

Pour changer ce message, il suffit d'ajouter en première ligne du code de la fonction une chaîne de caractères.

```
import math
def coordonnees_polaires (x,y):
    """convertit des coordonnées cartésiennes en coordonnées polaires
    (x,y) --> (pho,theta)"""
    rho     = math.sqrt(x*x+y*y)
    theta   = math.atan2 (y,x)
    return rho, theta
help (coordonnees_polaires)
```

Le programme affiche alors le message d'aide suivant nettement plus explicite :

```
Help on function coordonnees_polaires in module __main__:

coordonnees_polaires(x, y)
    convertit des coordonnées cartésiennes en coordonnées polaires
    (x,y) --> (pho,theta)
```

Il est conseillé d'écrire ce commentaire pour toute nouvelle fonction avant même que son corps ne soit écrit. L'expérience montre qu'on oublie souvent de l'écrire après.

### 4.4.7 Paramètres modifiables

Les paramètres de types immuables et modifiables se comportent de manières différentes à l'intérieur d'une fonction. Ces paramètres sont manipulés dans le corps de la fonction, voire modifiés parfois. Selon le type du paramètre, ces modifications ont des répercussions à l'extérieur de la fonction.

Les types immuables ne peuvent être modifiés et cela reste vrai. Lorsqu'une fonction accepte un paramètre de type immuable, elle ne reçoit qu'une copie de sa valeur. Elle peut donc modifier ce paramètre sans que la variable ou la valeur utilisée lors de l'appel de la fonction n'en soit affectée. On appelle ceci un passage de paramètre par valeur. A l'opposé, toute modification d'une variable d'un type modifiable à l'intérieur d'une fonction est répercutée à la variable qui a été utilisée lors de l'appel de cette fonction. On appelle ce second type de passage un passage par adresse.

L'exemple suivant utilise une fonction `somme_n_premier_terme` qui modifie ces deux paramètres. Le premier, `n`, est immuable, sa modification n'a aucune incidence sur la variable `nb`. En revanche, le premier élément du paramètre `liste` reçoit la valeur 0. Le premier élément de la liste `l` n'a plus la même valeur après l'appel de la fonction `somme_n_premier_terme` que celle qu'il avait avant.

```
def somme_n_premier_terme(n,liste):
    """calcul la somme des n premiers termes d'une liste"""
    somme = 0
    for i in liste:
        somme += i
        n -= 1              # modification de n (type immuable)
        if n <= 0: break
    liste[0] = 0            # modification de liste (type modifiable)
    return somme

l = [1,2,3,4]
nb = 3
print "avant la fonction ",nb,l   # affiche   avant la fonction  3 [1, 2, 3, 4]
s = somme_n_premier_terme (nb,l)
print "après la fonction ",nb,l   # affiche   après la fonction  3 [0, 2, 3, 4]
print "somme : ", s               # affiche   somme :  6
```

La liste `l` est modifiée à l'intérieur de la fonction `somme_n_premier_terme` comme l'affichage suivant le montre. En fait, à l'intérieur de la fonction, la liste `l` est désignée par l'identificateur `liste`, c'est la même liste. La variable `nb` est d'un type immuable. Sa valeur a été recopiée dans le paramètre `n` de la fonction `somme_n_premier_terme`. Toute modification de `n` à l'intérieur de cette fonction n'a aucune répercussion à l'extérieur de la fonction.

**Remarque 4.18 : passage par adresse**
Dans l'exemple précédent, il faut faire distinguer le fait que la liste passée en paramètre ne soit que modifiée et non changée. L'exemple suivant inclut une fonction qui affecte une nouvelle valeur au paramètre `liste` sans pour autant modifier la liste envoyée en paramètre.

```
def fonction (liste):
    liste = []
```

```
liste = [0,1,2]
print liste       # affiche [0,1,2]
fonction (liste)
print liste       # affiche [0,1,2]
```

Il faut considérer dans ce programme que la fonction `fonction` reçoit un paramètre appelé `liste` mais utilise tout de suite cet identificateur pour l'associer à un contenu différent. L'identificateur `liste` est en quelque sorte passé du statut de paramètre à celui de variable locale. La fonction associe une valeur à `liste` - ici, une liste vide - sans toucher à la valeur que cet identificateur désignait précédemment.

Le programme qui suit est différent du précédent mais produit les mêmes effets. Ceci s'explique par le fait que le mot-clé `del` ne supprime pas le contenu d'une variable mais seulement son identificateur. Le langage *Python* détecte ensuite qu'un objet n'est plus désigné par aucun identificateur pour le supprimer. Cette remarque est à rapprocher de celles du paragraphe 5.6.

```
def fonction (liste):
    del liste

liste = [0,1,2]
print liste       # affiche [0,1,2]
fonction (liste)
print liste       # affiche [0,1,2]
```

Le programme qui suit permet cette fois-ci de vider la liste `liste` passée en paramètre à la fonction `fonction`. La seule instruction de cette fonction modifie vraiment le contenu désigné par l'identificateur `liste` et cela se vérifie après l'exécution de cette fonction.

```
def fonction (liste):
    del liste[0:len(liste)]  # on peut aussi écrire : liste[:] = []

liste = [0,1,2]
print liste       # affiche [0,1,2]
fonction (liste)
print liste       # affiche []
```

### 4.4.8 Fonction récursive

> **Définition 4.19 : fonction récursive**
> Une fonction récursive est une fonction qui s'appelle elle-même.

La fonction récursive la plus fréquemment citée en exemple est la fonction factorielle. Celle-ci met en évidence les deux composantes d'une fonction récursive, la récursion proprement dite et la condition d'arrêt.

```
def factorielle(n):
    if n == 0 : return 1
    else : return n * factorielle(n-1)
```

La dernière ligne de la fonction `factorielle` est la récursion tandis que la précédente est la condition d'arrêt, sans laquelle la fonction ne cesserait de s'appeler, empêchant le programme de terminer son exécution. Si celle-ci est mal spécifiée ou absente, l'interpréteur *Python* affiche une suite ininterrompue de messages. *Python* n'autorise pas plus de 1000 appels récursifs : `factorielle(999)` provoque nécessairement une erreur d'exécution même si la condition d'arrêt est bien spécifiée.

```
Traceback (most recent call last):
  File "fact.py", line 5, in <module>
```

```
    factorielle(999)
  File "fact.py", line 3, in factorielle
    else : return n * factorielle(n-1)
  File "fact.py", line 3, in factorielle
    else : return n * factorielle(n-1)
  ...
```

La liste des messages d'erreurs est aussi longue qu'il y a eu d'appels à la fonction récursive. Dans ce cas, il faut transformer cette fonction en une fonction non récursive équivalente, ce qui est toujours possible.

```
def factorielle_non_recursive (n) :
    r = 1
    for i in range (2, n+1) :
        r *= i
    return r
```

### 4.4.9 Portée

#### 4.4.9.1 Portée des variables, des paramètres

Lorsqu'on définit une variable, elle n'est pas utilisable partout dans le programme. Par exemple, elle n'est pas utilisable avant d'avoir été déclarée au moyen d'une affectation. Le court programme suivant déclenche une erreur.

```
print x   # déclenche une erreur
```

```
Traceback (most recent call last):
  File "pas_declaree.py", line 1, in <module>
    print x
NameError: name 'x' is not defined
```

Il est également impossible d'utiliser une variable à l'extérieur d'une fonction où elle a été déclarée. Plusieurs fonctions peuvent ainsi utiliser le même nom de variable sans qu'à aucun moment, il n'y ait confusion. Le programme suivant déclenche une erreur identique à celle reproduite ci-dessus.

```
def portee_variable(x):
    var = x
    print var

portee_variable(3)
print var           # déclenche une erreur car var est déclarée dans
                    # la fonction portee_variable
```

> **Définition 4.20 : portée d'une variable**
> La portée d'une variable associée à un identificateur recouvre la portion du programme à l'intérieur de laquelle ce même identificateur la désigne. Ceci implique que, dans cette portion de code, aucune autre variable, aucune autre fonction, aucune autre classe, ne peut porter le même identificateur.

Une variable n'a donc d'existence que dans la fonction dans laquelle elle est déclarée. On appelle ce type de variable une variable locale. Par défaut, toute variable utilisée dans une fonction est une variable locale.

> **Définition 4.21 : variable locale**
> Une variable locale est une variable dont la portée est réduite à une fonction.

Par opposition aux variables locales, on définit les variables globales qui sont déclarées à l'extérieur de toute fonction.

**Définition 4.22 : variable globale**
Une variable globale est une variable dont la portée est l'ensemble du programme.

L'exemple suivant mélange variable locale et variable globale. L'identificateur `n` est utilisé à la fois pour désigner une variable globale égale à 1 et une variable locale égale à 1. A l'intérieur de la fonction, `n` désigne la variable locale égale à 2. A l'extérieur de la fonction, `n` désigne la variable globale égale à 1.

```
n = 1                   # déclaration d'une variable globale
def locale_globale():
    n = 2               # déclaration d'une variable locale
    print n             # affiche le contenu de la variable locale

print n                 # affiche 1
locale_globale()        # affiche 2
print n                 # affiche 1
```

Il est possible de faire référence aux variables globales dans une fonction par l'intermédiaire du mot-clé `global`. Celui-ci indique à la fonction que l'identificateur `n` n'est plus une variable locale mais désigne une variable globale déjà déclarée.

```
n = 1                   # déclaration d'une variable globale
def locale_globale():
    global n            # cette ligne indique que n désigne la variable globale
    n = 2               # change le contenu de la variable globale
    print n             # affiche le contenu de la variable globale

print n                 # affiche 1
locale_globale()        # affiche 2
print n                 # affiche 2
```

Cette possibilité est à éviter le plus possible car on peut considérer que `locale_globale` est en fait une fonction avec un paramètre caché. La fonction `locale_globale` n'est plus indépendante des autres fonctions puisqu'elle modifie une des données du programme.

#### 4.4.9.2 Portée des fonctions

Le langage *Python* considère les fonctions également comme des variables d'un type particulier. La portée des fonctions obéit aux mêmes règles que celles des variables. Une fonction ne peut être appelée que si elle a été définie avant son appel.

```
print type(factorielle)  # affiche <type 'function'>
```

Comme il est possible de déclarer des variables locales, il est également possible de définir des fonctions locales ou fonctions imbriquées. Une fonction locale n'est appelable qu'à l'intérieur de la fonction dans laquelle elle est définie. Dans l'exemple suivant, la fonction `affiche_pair` inclut une fonction locale qui n'est appelable que par cette fonction `affiche_pair`.

```
def affiche_pair():
    def fonction_locale(i):            # fonction locale ou imbriquée
        if i % 2 == 0 : return True
        else : return False
    for i in range(0,10):
        if fonction_locale(i):
            print i
```

```
affiche_pair()
fonction_locale(5)      # l'appel à cette fonction locale
                        # déclenche une erreur d'exécution
```

A l'intérieur d'une fonction locale, le mot-clé `global` désigne toujours les variables globales du programme et non les variables de la fonction dans laquelle cette sous-fonction est définie.

### 4.4.10 Nombre de paramètres variable

Il est possible de définir des fonctions qui prennent un nombre indéterminé de paramètres, lorsque celui-ci n'est pas connu à l'avance. Hormis les paramètres transmis selon le mode présenté dans les paragraphes précédents, des informations peuvent être ajoutées à cette liste lors de l'appel de la fonction, ces informations sont regroupées soit dans une liste de valeurs, soit dans une liste de couples (identificateur, valeur). La déclaration d'une telle fonction obéit à la syntaxe suivante :

**syntaxe 4.23 : fonction, nombre variable de paramètres**

```
def fonction (param_1, ..., param_n, *liste, **dictionnaire) :
```

Où `fonction` est un nom de fonction, `param_1` à `param_n` sont des paramètres de la fonction, `liste` est le nom de la liste qui doit recevoir la liste des valeurs seules envoyées à la fonction et qui suivent les paramètres (plus précisément, c'est un tuple), `dictionnaire` reçoit la liste des couples (identificateur, valeur).

L'appel à cette fonction suit quant à lui la syntaxe suivante :

**syntaxe 4.24 : fonction, nombre variable de paramètres (appel)**

```
fonction (valeur_1, ..., valeur_n, \
          liste_valeur_1, ..., liste_valeur_p, \
          nom_1 = v_1, ..., nom_q = v_q)
```

Où `fonction` est un nom de fonction, `valeur_1` à `valeur_n` sont les valeurs associées aux paramètres `param_1` à `param_n`, `liste_valeur_1` à `liste_valeur_p` formeront la liste `liste`, les couples `nom_1` : `v_1` à `nom_q` : `v_q` formeront le dictionnaire `dictionnaire`.

Exemple :

```
def fonction(p,*l,**d):
    print "p = ",p
    print "liste (tuple) l :", l
    print "dictionnaire d :", d

fonction (1,2,3,a=5,b=6) # 1 est associé au paramètre p
                         # 2 et 3 sont insérés dans la liste l
                         # a=5 et b=6 sont insérés dans le dictionnaire d
```

Ce programme affiche :

```
p =  1
liste l : (2, 3)
dictionnaire d : {'a': 5, 'b': 6}
```

A l'instar des paramètres par défaut, la seule contrainte de cette écriture est la nécessité de respecter l'ordre dans lequel les informations doivent apparaître. Lors de l'appel, les valeurs sans précision de nom de paramètre seront placées dans une liste (ici le tuple `l`). Les valeurs associées à un nom de paramètre seront placées dans un dictionnaire (ici `d`). Les valeurs par défaut sont obligatoirement placées après les paramètres non nommés explicitement.

Une fonction qui accepte des paramètres en nombre variable peut à son tour appeler une autre fonction acceptant des paramètres en nombre variable. Il faut pour cela se servir du symbole $*$ afin de transmettre à `fonction` les valeurs reçues par `fonction2`.

```
def fonction(p,*l,**d):
    print "p = ",p
    print "liste l :", l
    print "dictionnaire d :", d

def fonction2 (p, *l, **d) :
    l += (4,)              # on ajoute une valeur au tuple
    d ["c"] = 5            # on ajoute un couple (paramètre,valeur)
    fonction (p, *l, **d)  # ne pas oublier le symbole *

fonction2 (1,2,3,a=5,b=6)
```

Le programme affiche :

```
p =  1
liste l : (2, 3, 4)
dictionnaire d : {'a': 5, 'c': 5, 'b': 6}
```

### 4.4.11 Ecriture simplifiée pour des fonctions simples

Lorsque le code d'une fonction tient en une ligne et est le résultat d'une expression, il est possible de condenser son écriture à l'aide du mot-clé `lambda`.

**syntaxe 4.25 : fonction `lambda`**

```
nom_fonction = lambda param_1, ..., param_n : expression
```

`nom_fonction` est le nom de la fonction, `param_1` à `param_n` sont les paramètres de cette fonction (ils peuvent également recevoir des valeurs par défaut), `expression` est l'expression retournée par la fonction.

L'exemple suivant utilise cette écriture pour définir la fonction `min` retournant le plus petit entre deux nombres positifs.

```
min = lambda x,y : (abs (x+y) - abs (x-y))/2

print min (1,2)      # affiche 1
print min (5,4)      # affiche 4
```

Cette écriture correspond à l'écriture non condensée suivante :

```
def min(x,y):
    return (abs (x+y) - abs (x-y))/2

print min (1,2)      # affiche 1
print min (5,4)      # affiche 4
```

### 4.4.12 Fonctions générateur

Le mot-clé `yield` est un peu à part. Utilisé à l'intérieur d'une fonction, il permet d'interrompre le cours de son exécution à un endroit précis de sorte qu'au prochain appel de cette fonction, celle-ci reprendra le cours de son exécution exactement au même endroit avec des variables locales inchangées. Le mot-clé `return` ne doit pas être utilisé. Ces fonctions ou *générateurs* sont utilisées en couple avec le mot-clé `for` pour simuler un ensemble. L'exemple suivant implémente une fonction `fonction_yield` qui simule l'ensemble des entiers compris entre 0 et n exclu.

```
def fonction_yield(n):
    i = 0
    while i < n-1:
        print "yield 1" # affichage : pour voir ce que fait le programme
        yield i         # arrête la fonction qui reprendra
        i = i+1         # à la ligne suivante lors du prochain appel
    print "yield 2"     # affichage : pour voir ce que fait le programme
    yield i             # arrête la fonction qui ne reprendra pas
                        # lors du prochain appel car le code de la fonction
                        # prend fin ici

for a in fonction_yield(2):
    print a                 # affiche tous les éléments que retourne la
                            # fonction fonction_yield, elle simule la liste
                            # [0,1]
print "-----------------------------------------------"
for a in fonction_yield(3):
    print a                 # nouvel appel, l'exécution reprend
                            # au début de la fonction,
                            # affiche tous les éléments que retourne la
                            # fonction fonction_yield, elle simule la liste
                            # [0,1,2]
```

Le programme affiche tous les entiers compris entre 0 et 4 inclus ainsi que le texte "`yield 1`" ou "`yield 2`" selon l'instruction `yield` qui a retourné le résultat. Lorsque la fonction a finalement terminé son exécution, le prochain appel agit comme si c'était la première fois qu'on l'appelait.

```
yield 1
0
yield 2
1
-----------------------------------------------
yield 1
0
yield 1
1
yield 2
2
```

### 4.4.13 Identificateur appelable

La fonction `callable` retourne un booléen permettant de savoir si un identificateur est une fonction (voir chapitre 5), de savoir par conséquent si tel identificateur est appelable comme une fonction.

```
x = 5
def y () :
    return None
print callable (x)  # affiche False car x est une variable
print callable (y)  # affiche True car y est une fonction
```

### 4.4.14 Fonctions ajoutées lors de l'exécution du programme

#### 4.4.14.1 Fonction `eval`

Cette fonction a déjà été abordée lors des paragraphes 3.4.1 (paragraphe 3.4.2, page 49). Un exemple a déjà été présenté page 49. Elle évalue toute chaîne de caractères contenant une expression écrite avec la syntaxe du langage *Python*. Cette expression peut utiliser toute variable ou toute fonction accessible au moment où est appelée la fonction `eval`.

```
x = 3
y = 4
print eval ("x*x+y*y+2*x*y")  # affiche 49
print (x+y)**2                # affiche 49
```

Si l'expression envoyée à la fonction `eval` inclut une variable non définie, l'interpréteur *Python* génère une erreur comme le montre l'exemple suivant.

```
x = 3
y = 4
print eval ("x*x+y*y+2*x*y+z")
```

La variable `z` n'est pas définie et l'expression n'est pas évaluable.

```
Traceback (most recent call last):
  File "c:\temp\cours.py", line 3, in -toplevel-
    print eval ("x*x+y*y+2*x*y+z")
  File "<string>", line 0, in -toplevel-
NameError: name 'z' is not defined
```

L'erreur se produit dans une chaîne de caractères traduite en programme informatique, c'est pourquoi l'interpréteur ne peut pas situer l'erreur dans un fichier. L'erreur ne se produit dans aucun fichier, cette chaîne de caractères pourrait être définie dans un autre.

#### 4.4.14.2 Fonctions `compile`, `exec`

Plus complète que la fonction `eval`, la fonction `compile` permet d'ajouter une ou plusieurs fonctions au programme, celle-ci étant définie par une chaîne de caractères. Le code est d'abord compilé (fonction `compile`) puis incorporé au programme (fonction `exec`) comme le montre l'exemple suivant.

```
import math
str = """def coordonnees_polaires (x,y):
    rho     = math.sqrt(x*x+y*y)
    theta   = math.atan2 (y,x)
    return rho, theta"""       # fonction définie par une chaîne de caractères

obj = compile(str,"","exec")   # fonction compilée
exec obj                       # fonction incorporée au programme
print coordonnees_polaires(1,1)# affiche (1.4142135623730951, 0.78539816339744828)
```

La fonction `compile` prend en fait trois arguments. Le premier est la chaîne de caractères contenant le code à compiler. Le second paramètre ("" dans l'exemple) contient un nom de fichier dans lequel seront placées les erreurs de compilation. Le troisième paramètre est une chaîne de caractères à choisir parmi "exec" ou "eval". Selon ce choix, ce sera la fonction `exec` ou `eval` qui devra être utilisée pour agréger le résultat de la fonction `compile` au programme. L'exemple suivant donne un exemple d'utilisation de la fonction `compile` avec la fonction `eval`.

```
import math
str = """math.sqrt(x*x+y*y)"""  # expression définie par une chaîne de caractères

obj = compile(str,"","eval")    # expression compilée
x = 1
y = 2
print eval (obj)                # résultat de l'expression
```

## 4.5 Indentation

L'indentation est synonyme de décalage. Pour toute boucle, test, fonction, et plus tard, toute définition de classe, le fait d'indenter ou décaler les lignes permet de définir une dépendance d'un bloc de lignes par rapport à un autre. Les lignes indentées par rapport à une boucle `for` dépendent de celle-ci puisqu'elle seront exécutées à chaque passage dans la boucle. Les lignes indentées par rapport au mot-clé `def` sont considérées comme faisant partie du corps de la fonction. La remarque 4.4 page 53 précise l'erreur que l'interpréteur *Python* retourne en cas de mauvaise indentation.

Contrairement à d'autres langages comme le *C* ou *PERL*, *Python* n'utilise pas de délimiteurs pour regrouper les lignes. L'indentation, souvent présentée comme un moyen de rendre les programmes plus lisibles, est ici intégrée à la syntaxe du langage. Il n'y a pas non plus de délimiteurs entre deux instructions autre qu'un passage à la ligne. Le caractère \ placé à la fin d'une ligne permet de continuer l'écriture d'une instruction à la ligne suivante.

## 4.6 Fonctions usuelles

Certaines fonctions sont communes aux dictionnaires et aux listes, elles sont également définis pour de nombreux objets présents dans les extensions du langages. Quelque soit le contexte, le résultat attendu à la même signification. Les plus courantes sont présentées par la table 4.3.

La fonction `map` permet d'écrire des boucles de façon simplifiée. Elle est utile dans le cas où on souhaite appliquer la même fonction à tous les éléments d'un ensemble. Par exemple les deux dernières lignes du programme suivant sont équivalentes.

```
l = [0,3,4,4,5,6]
print [ est_pair (i) for i in l ]  # affiche [0, 1, 0, 0, 1, 0]
print map (est_pair, l)            # affiche [0, 1, 0, 0, 1, 0]
```

Elle peut aider à simplifier l'écriture lorsque plusieurs listes sont impliquées. Ici encore, les deux dernières lignes sont équivalentes.

```
def addition (x,y) : return x + y
l = [0,3,4,4,5,6]
m = [1,3,4,5,6,8]
print [ addition (l [i], m [i]) for i in range (0, len (l)) ]
print map (addition, l, m)   # affiche [1, 6, 8, 9, 11, 14]
```

Il est possible de substituer `None` à la fonction `f` pour obtenir l'équivalent de la fonction `zip`.

```
print map (None, l,m)  # affiche [(0, 1), (3, 3), (4, 4), (4, 5), (5, 6), (6, 8)]
print zip (l,m)        # affiche [(0, 1), (3, 3), (4, 4), (4, 5), (5, 6), (6, 8)]
```

| | |
|---|---|
| abs(x) | Retourne la valeur absolue de x. |
| callable(x) | Dit si la variable x peut être appelée. |
| chr(i) | Retourne le caractère associé au code numérique i. |
| cmp(x, y) | Compare x et y, retourne -1 si x < y, 0 en cas d'égalité, 1 sinon. |
| dir(x) | Retourne l'ensemble des méthodes associées à x qui peut être un objet, un module, un variable,... |
| help(x) | Retourne l'aide associée à x. |
| id(x) | Retourne un identifiant unique associé à l'objet x. Le mot-clé is est relié à cet identifiant. |
| isinstance( x, classe | Dit si l'objet x est de type classe, voir le chapitre 5 sur les classes. |
| issubclass( cl1, cl2) | Dit si la classe cl1 hérite de la classe cl2, voir le chapitre 5 sur les classes. |
| len(l) | Retourne la longueur de l. |
| map(f, l1, l2, ...) | Applique la fonction f sur les listes l1, l2... Voir l'exemple page 76. |
| max(l) | Retourne le plus grand élément de l. |
| min(l) | Retourne le plus petit élément de l. |
| ord(s) | Fonction réciproque de chr. |
| range(i, j[, k] | Construit la liste des entiers de i à j. Si k est précisé, va de k en k à partir de i. |
| reload(module) | Recharge un module (voir paragraphe 7). |
| repr(o) | Retourne une chaîne de caractères qui représente l'objet o. |
| round(x[, n]) | Arrondi x à n décimales près ou aucune si n n'est pas précisé. |
| str(o) | Retourne une chaîne de caractères qui représente l'objet o. |
| sum(l) | Retourne la somme de l'ensemble l. |
| type(o) | Retourne le type de la variable o. |
| xrange(i, j[, k] | Plus rapide que la fonction range mais utilisable que dans une boucle for. |
| zip(l1, l2, ...) | Construit une liste de tuples au lieu d'un tuple de listes. |

**Table 4.3 :** *Liste non exhaustive de fonctions définis par le langage Python sans qu'aucune extension ne soit nécessaire. Elles ne sont pas toujours applicables même si la syntaxe d'appel est correcte, elles produisent une erreur dans ce cas.*

## 4.7 Constructions classiques

Les paragraphes qui suivent décrivent des schémas qu'on retrouve dans les programmes dans de nombreuses situations. Ce sont des combinaisons simples d'une ou deux boucles, d'un test, d'une liste, d'un dictionnaire.

### 4.7.1 Recherche d'un élément

Rechercher un élément consiste à parcourir un ensemble jusqu'à ce qu'on le trouve : le résultat souhaité est sa position ou un résultat indiquant qu'il n'a pas été trouvé. Cette recherche est souvent insérée dans une fonction dont voici le schéma :

```
def recherche (li, c) :
    for i in range (0, len (li)) :
        if li [i] == c : return i
    return -1
li = [ 45, 32, 43, 56 ]
print recherche (li, 43)    # affiche 2
```

La méthode index permet de réduire ce programme.

```
print li.index (43)
```

Néanmoins, la fonction `recherche` peut être adaptée pour rechercher la première valeur trouvée parmi deux possibles, ce qu'il n'est pas possible de faire avec la méthode `index`.

```
def recherche (li, c,d) :
    for i in range (0, len (li)) :
        if li [i] in [c,d] : return i
    return -1
li = [ 45, 32, 43, 56 ]
print recherche (li, 43, 32)    # affiche 1
```

Ce cours exemple montre qu'il est utile de connaître quelques algorithmes simples. Ils ne sont pas forcément les plus efficaces mais ils aident à construire des programmes plus rapidement et à obtenir un résultat. S'il est positif alors il sera toujours temps de se pencher sur l'optimisation du programme pour qu'il soit plus efficace.

### 4.7.2 Maximum, minimum

La recherche d'un extremum comme le calcul d'une somme fait toujours intervenir une boucle. La différence est qu'on souhaite parfois obtenir la position du maximum.

```
li = [ 0, 434, 43, 6436, 5 ]
m  = li [0]             # initialisation
for l in li :           # boucle
    if m < l : m = l    # m est le maximum
```

Ce code est équivalent à l'instruction `max(li)`. Pour récupérer la position du maximum, il faut itérer sur les positions et non sur les éléments.

```
li = [ 0, 434, 43, 6436, 5 ]
m  = 0
for i in range (0, len (li)) :
    if li [m] < li [i] : m = i
```

Pour éviter la boucle, on peut utiliser l'astuce décrite au paragraphe 4.7.7 qui consiste à former une liste de couples avec la position initiale :

```
k = [ (li [i],i) for i in range (0, len (li)) ]
m = max (k) [1]
```

Il arrive fréquemment qu'on ne doive pas chercher le maximum d'une liste mais le maximum au sein d'un sous-ensemble de cette liste. Dans l'exemple qui suit, on cherche le tuple dont la première valeur est maximale et dont la seconde valeur est égale à 1.

```
li = [ (0,0), (434,0), (43,1), (6436,1), (5,0) ]
m  = -1 # -1 car le premier élément peut ne pas faire partie du sous-ensemble
for i in range (0, len (li)) :
    if li [i][1] == 0 and (m == -1 or li [m][0] < li [i][0]) : m = i
```

### 4.7.3 Recherche dichotomique

La recherche dichotomique est plus rapide qu'une recherche classique mais elle suppose que celle-ci s'effectue dans un ensemble trié. L'idée est de couper en deux l'intervalle de recherche à chaque itération. Comme l'ensemble est trié, en comparant l'élément cherché à l'élément central, on peut éliminer une partie de l'ensemble : la moitié inférieure ou supérieure.

```
def recherche_dichotomique (li, c) :
    a,b = 0, len (li)-1
    while a <= b :
```

```
        m = (a+b)/2
        if   c == li [m] : return m
        elif c <  li [m] : b = m-1   # partie supérieure éliminée
        else             : a = m+1   # partie inférieure éliminée
    return -1  # élément non trouvé

li = range (0,100,2)
print recherche_dichotomique (li, 48)  # affiche 24
print recherche_dichotomique (li, 49)  # affiche -1
```

### 4.7.4 Décomposition en matrice

Les quelques lignes qui suivent permettent de décomposer une chaîne de caractères en matrice. Chaque ligne et et chaque colonne sont séparées par des séparateurs différents. Ce procédé intervient souvent lorsqu'on récupère des informations depuis un fichier texte lui-même provenant d'un tableur.

```
s       = "case11;case12;case13|case21;case22;case23"
# décomposition en matrice
ligne   = s.split ("|")                     # lignes
mat     = [ l.split (";") for l in ligne ]  # colonnes
```

L'opération inverse :

```
ligne   = [ ";".join (l) for l in mat ]     # colonnes
s       = "|".join (ligne)                  # lignes
```

### 4.7.5 Somme

Le calcul d'une somme fait toujours intervenir une boucle car le langage *Python* ne peut faire des additions qu'avec deux nombres. Le schéma est toujours le même : initialisation et boucle.

```
li = [ 0, 434, 43, 6456 ]
s  = 0                      # initialisation
for l in li :               # boucle
    s += l                  # addition
```

Ce code est équivalent à l'instruction `sum(li)`. Dans ce cas où la somme intègre le résultat d'une fonction et non les éléments d'une liste, il faudrait écrire :

```
def fonction (x) : return x*x
s  = 0
for l in li : s += fonction (l)
```

Et ces deux lignes pourraient être résumées en une seule grâce à l'une de ces deux instructions.

```
s = sum ( [fonction (l) for l in li] )

s = sum ( map (fonction, li) )
```

L'avantage de la solution avec la boucle est qu'elle évite la création d'une liste intermédiaire, c'est un point à prendre en compte si la liste sur laquelle opère la somme est volumineuse. Il peut être intéressant de regarder l'exercice 13.1.9 page 265.

### 4.7.6 Tri

Le tri est une opération fréquente. On n'a pas toujours le temps de programmer le tri le plus efficace comme un tri *quicksort* et un tri plus simple suffit la plupart du temps. Le tri suivant consiste à recherche le plus petit élément puis à échanger sa place avec le premier élément du tableau du tableau. On recommence la même procédure à partir de la seconde position, puis la troisième et ainsi de suite jusqu'à la fin du tableau.

```
for i in range (0, len (li)) :
    # recherche du minimum entre i et len (li) exclu
    pos = i
    for j in range (i+1, len (li)) :
        if li [j] < li [pos] : pos = j
    # échange
    ech      = li [pos]
    li [pos] = li [i]
    li [i]   = ech
```

La méthode `sort` tri également une liste mais selon un algorithme plus efficace que celui-ci dont la logique est présentée par l'exercice 11.5.1, page 234. Ceci explique pourquoi on hésite toujours à programmer un tri quitte à avoir recours à une astuce telle que celle présentées au paragraphe suivant.

### 4.7.7 Tri et position initiale

Il arrive parfois qu'on souhaite trier un tableau puis appliquer la même transformation à un second tableau. Pour cela, il faut récupérer la position initiale des éléments dans le tableau trié. Une possibilité consiste à trier non pas le tableau mais une liste de couples (valeur, position) comme le montre l'exemple suivant :

```
tab = ["zéro", "un", "deux"]                        # tableau à trier
pos = [ (tab [i],i) for i in range (0, len (tab)) ] # tableau de couples
pos.sort ()                                         # tri
print pos               # affiche [('deux', 2), ('un', 1), ('zéro', 0)]
```

La liste [`p`[1] `for p in pos`] correspond à l'ensemble des positions initiales des éléments de la liste `tab`. Les deux lignes qui suivent permettent d'en déduire la position de chaque élément dans l'ensemble trié. `ordre`[`i`] est le rang de l'élément d'indice `i` avant le tri.

```
ordre = range (0, len (pos))
for i in range (0, len (pos)) : ordre [pos [i][1]] = i
```

### 4.7.8 Comptage

On souhaite ici compter le nombre d'occurrences de chaque élément d'un tableau. Par exemple, on pourrait connaître par ce moyen la popularité d'un mot dans un discours politique ou l'étendue du vocabulaire utilisé. L'exemple suivant compte les mots d'une liste de mots.

```
li = ["un", "deux", "un", "trois"]
d  = { }
for l in li :
    if l not in d : d [l] = 1
    else : d [l] += 1
print d   # affiche {'un': 2, 'trois': 1, 'deux': 1}
```

La structure la plus appropriée ici est un dictionnaire puisqu'on cherche à associer une valeur à un élément d'une liste qui peut être de tout type. Si la liste contient des éléments de type modifiable comme une liste, il faudrait convertir ceux-ci en un type immuable comme une chaîne de caractères. L'exemple suivant illustre ce cas en comptant les occurrences des lignes d'une matrice.

```
mat = [ [1,1,1], [2,2,2], [1,1,1]]
d  = { }
for l in mat :
    k = str (l)
    if k not in d : d [k] = 1
    else : d [k] += 1
print d   # affiche {'[1, 1, 1]': 2, '[2, 2, 2]': 1}
```

On peut également vouloir non pas compter le nombre d'occurrence mais mémoriser les positions des éléments tous identiques. On doit utiliser un dictionnaire de listes :

```
li = ["un", "deux", "un", "trois"]
d  = { }
for i in range (0, len (li)) :
    if li [i] not in d : d [li [i]] = [ i ]
    else : d [li [i]].append (i)
print d   # affiche {'un': [0, 2], 'trois': [3], 'deux': [1]}
```

### 4.7.9 Matrice en un vecteur

Dans un langage comme le *C++*, il arrive fréquemment qu'une matrice ne soit pas représentée par une liste de listes mais par une seule liste car cette représentation est plus efficace. Il faut donc convertir un indice en deux indices ligne et colonne. Il faut bien sûr que le nombre de colonnes sur chaque ligne soit constant. Le premier programme convertit une liste de listes en une seule liste.

```
mat = [[0,1,2],[3,4,5]]
lin = [ i * len (mat [i]) + j \
            for i in range (0, len (mat)) \
                for j in range (0, len (mat [i])) ]
```

Le programme suivant fait l'inverse. Il faut faire attention à la position des crochets et l'ordre des boucles `for`.

```
nc = len (mat [0])
mat = [ [ lin [i * nc + j] for j in range (0, len (mat [i])) ] \
            for i in range (0, len (mat)) ]
```

### 4.7.10 Fonctions et variables

Une fonction peut aussi recevoir en paramètre une autre fonction. L'exemple suivant inclut la fonction `calcul_n_valeur` qui prend comme paramètres `l` et `f`. Cette fonction calcule pour toutes les valeurs `x` de la liste `l` la valeur `f(x)`. `fonction_carre` ou `fonction_cube` sont passées en paramètres à la fonction `calcul_n_valeur` qui les exécute. Le paragraphe 5.10.1 page 118 présente le même programme mais rédigé avec des classes.

```
def fonction_carre(x) :  return x*x
def fonction_cube (x) :  return x*x*x

def calcul_n_valeur (l,f):
    res = [ f(i) for i in l ]
    return res

l = [0,1,2,3]
print l   # affiche [0, 1, 2, 3]

l1 = calcul_n_valeur (l, fonction_carre)
print l1  # affiche [0, 1, 4, 9]
```

```
l2 = calcul_n_valeur (l, fonction_cube)
print l2  # affiche [0, 1, 8, 27]
```

# Chapitre 5

# Classes

---

Imaginons qu'une banque détienne un fichier contenant des informations sur ses clients et qu'il soit impossible pour un client d'avoir accès directement à ces informations. Toutefois, il lui est en théorie possible de demander à son banquier quelles sont les informations le concernant détenues par sa banque. Il est en théorie également possible de les rectifier s'il estime qu'elles sont incorrectes [1] .

On peut comparer cette banque à un objet qui possède des informations et des moyens permettant de lire ces informations et de les modifier. Vu de l'extérieur, cette banque cache son fonctionnement interne et les informations dont elle dispose, mais propose des services à ses utilisateurs.

On peut considérer la banque comme un *objet* au sens informatique. Ce terme désigne une entité possédant des données et des méthodes permettant de les manipuler. Plus concrètement, une classe est un assemblage de variables appelées *attributs* et de fonctions appelées *méthodes*. L'ensemble des propriétés associées aux classes est regroupé sous la désignation de *programmation objet*.

## 5.1 Présentation des classes : méthodes et attributs

### 5.1.1 Définition, déclaration

**Définition 5.1 : classe**
Une classe est un ensemble incluant des variables ou *attributs* et des fonctions ou *méthodes*. Les attributs sont des variables accessibles depuis toute méthode de la classe où elles sont définies. En *Python*, les classes sont des types modifiables.

**syntaxe 5.2 : classe, définition**

```
class nom_classe :
    # corps de la classe
    # ...
```

Le corps d'une classe peut être vide, inclure des variables ou attributs, des fonctions ou méthodes. Il est en tout cas indenté de façon à indiquer à l'interpréteur *Python* les lignes qui forment le corps de la classe.

Les classes sont l'unique moyen en langage *Python* de définir de nouveaux types, propres à celui qui programme. Il n'existe pas de type "matrice" ou de type "graphe" en langage *Python* qui soit prédéfini [2]. Il est néanmoins possible de les définir au moyen des classes. Une matrice est par exemple un objet qui inclut les attributs suivant : le nombre de lignes, le nombre de colonnes, les coefficients de la matrice. Cette matrice inclut aussi des méthodes comme des opérations entre deux matrices telles que l'addition, la soustraction, la multiplication ou des opérations sur elle-même comme l'inversion, la transposition, la diagonalisation.

Cette liste n'est pas exhaustive, elle illustre ce que peut être une classe "matrice" - représentation informatique d'un objet "matrice" -, un type complexe incluant des informations de types variés (entier pour les dimensions, réels pour les coefficients), et des méthodes propres à cet objet, capables de manipuler ces

1. Tout fichier contenant des données à caractère personnel doit être recensé auprès de la Commission nationale de l'informatique et des libertés (CNIL). Cet organisme facilite l'accès à ces informations.

2. Les matrices sont définies dans une extension **externe** `numpy` (voir le chapitre 7).

informations.

Il est tout-à-fait possible de se passer des classes pour rédiger un programme informatique. Leur utilisation améliore néanmoins sa présentation et la compréhension qu'on peut en avoir. Bien souvent, ceux qui passent d'un langage uniquement fonctionnel à un langage objet ne font pas marche arrière. En résumé, les classes permettent de définir de nouveaux types.

**syntaxe 5.3 : classe, création d'une variable de type objet**

```
cl = nom_classe ()
```

La création d'une variable de type objet est identique à celle des types standards du langage *Python* : elle passe par une simple affectation. On appelle aussi `cl` une *instance* de la classe `nom_classe`.

Cette syntaxe est identique à la syntaxe d'appel d'une fonction. La création d'une instance peut également faire intervenir des paramètres (voir paragraphe 5.4). Le terme *instance* va de paire avec le terme *classe* :

**Définition 5.4 : instance**
Une instance d'une classe `C` désigne une variable de type `C`. Le terme instance ne s'applique qu'aux variables dont le type est une classe.

L'exemple suivant permet de définir une classe vide. Le mot-clé `pass` permet de préciser que le corps de la classe ne contient rien.

```
class classe_vide:
    pass
```

Il est tout de même possible de définir une instance de la classe `classe_vide` simplement par l'instruction suivante :

```
class classe_vide:
    pass
cl = classe_vide ()
```

**Remarque 5.5 : type d'une instance**
Dans l'exemple précédent, la variable `nb` n'est pas de type `exemple_classe` mais de type `instance` comme le montre la ligne suivante :

```
print type (cl)   # affiche <type 'instance'>
```

Pour savoir si une variable est une instance d'une classe donnée, il faut utiliser la fonction `isinstance` :

```
isinstance (cl,classe_vide)   # affiche True
```

### 5.1.2 Méthodes

**Définition 5.6 : méthode**
Les méthodes sont des fonctions qui sont associées de manière explicite à une classe. Elles ont comme particularité un accès privilégié aux données de la classe elle-même.

Ces données ou *attributs* sont définis plus loin. Les méthodes sont en fait des fonctions pour lesquelles la liste des paramètres contient obligatoirement un paramètre explicite qui est l'instance de la classe à

laquelle cette méthode est associée. Ce paramètre est le moyen d'accéder aux données de la classe.

**syntaxe 5.7 : classe, méthode, définition**

```
class nom_classe :
    def nom_methode (self, param_1, ..., param_n) :
        # corps de la méthode...
```

A part le premier paramètre qui doit de préférence s'appeler `self`, la syntaxe de définition d'une méthode ressemble en tout point à celle d'une fonction. Le corps de la méthode est indenté par rapport à la déclaration de la méthode, elle-même indentée par rapport à la déclaration de la classe.

L'appel à cette méthode obéit à la syntaxe qui suit :

**syntaxe 5.8 : classe, méthode, appel**

```
cl = nom_classe ()    # variable de type nom_classe
t  = cl.nom_methode (valeur_1, ..., valeur_n)
```

L'appel d'une méthode nécessite tout d'abord la création d'une variable. Une fois cette variable créée, il suffit d'ajouter le symbole "." pour exécuter la méthode. Le paramètre `self` est ici implicitement remplacé par `cl` lors de l'appel.

L'exemple suivant simule le tirage de nombres aléatoires à partir d'une suite définie par récurrence $u_{n+1} = (u_n * A) \ mod \ B$ où $A$ et $B$ sont des entiers très grands. Cette suite n'est pas aléatoire mais son comportement imite celui d'une suite aléatoire. Le terme $u_n$ est dans cet exemple contenu dans la variable globale `rnd`.

```
rnd = 42

class exemple_classe:
    def methode1(self,n):
        """simule la génération d'un nombre aléatoire
           compris entre 0 et n-1 inclus"""
        global rnd
        rnd = 397204094 * rnd % 2147483647
        return int (rnd % n)

nb  = exemple_classe ()
l   = [ nb.methode1(100) for i in range(0,10) ]
print l   # affiche [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]

nb2 = exemple_classe ()
l2  = [ nb2.methode1(100) for i in range(0,10) ]
print l2   # affiche [46, 42, 89, 66, 48, 12, 61, 84, 71, 41]
```

Deux instances `nb` et `nb2` de la classe `exemple_classe` sont créées, chacune d'elles est utilisée pour générer aléatoirement dix nombres entiers compris entre 0 et 99 inclus. Les deux listes sont différentes puisque l'instance `nb2` utilise la variable globale `rnd` précédemment modifiée par l'appel `nb.methode1(100)`.

**Remarque 5.9 : méthodes et fonctions**

Les méthodes sont des fonctions insérées à l'intérieur d'une classe. La syntaxe de la déclaration d'une méthode est identique à celle d'une fonction en tenant compte du premier paramètre qui doit impérativement être `self`. Les paramètres par défaut, l'ordre des paramètres, les nombres variables de paramètres présentés au paragraphe 4.4 sont des extensions tout autant applicables aux méthodes qu'aux fonctions.

### 5.1.3 Attributs

**Définition 5.10 : attribut**
Les attributs sont des variables qui sont associées de manière explicite à une classe. Les attributs de la classe se comportent comme des variables globales pour toutes les méthodes de cette classe.

Une classe permet en quelque sorte de regrouper ensemble des informations liées. Elles n'ont de sens qu'ensemble et les méthodes manipulent ces données liées. C'est le cas pour un segment qui est toujours défini par ces deux extrémités qui ne vont pas l'une sans l'autre.

**syntaxe 5.11 : classe, attribut, définition**

```
class nom_classe :
    def nom_methode (self, param_1, ..., param_n) :
        self.nom_attribut = valeur
```

Le paramètre `self` n'est pas un mot clé même si le premier paramètre est le plus souvent appelé `self`. Il désigne l'instance de la classe sur laquelle va s'appliquer la méthode. La déclaration d'une méthode inclut toujours un paramètre `self` de sorte que `self.nom_attribut` désigne un attribut de la classe. `nom_attribut` seul désignerait une variable locale sans aucun rapport avec un attribut portant le même nom. Les attributs peuvent être déclarés à l'intérieur de n'importe quelle méthode, voire à l'extérieur de la classe elle-même.

L'endroit où est déclaré un attribut a peu d'importance pourvu qu'il le soit avant sa première utilisation. Dans l'exemple qui suit, la méthode `methode1` utilise l'attribut `rnd` sans qu'il ait été créé.

```
class exemple_classe:
    def methode1(self,n):
        """simule la génération d'un nombre aléatoire
           compris entre 0 et n-1 inclus"""
        self.rnd = 397204094 * self.rnd % 2147483647
        return int (self.rnd % n)

nb = exemple_classe ()
l  = [ nb.methode1(100) for i in range(0,10) ]
print l
```

Cet exemple déclenche donc une erreur (ou exception) signifiant que l'attribut `rnd` n'a pas été créé.

```
Traceback (most recent call last):
  File "cours.py", line 8, in -toplevel-
    l = [ nb.methode1(100) for i in range(0,10) ]
  File "cours.py", line 4, in methode1
    self.rnd = 397204094 * self.rnd % 2147483647
AttributeError: exemple_classe instance has no attribute 'rnd'
```

Pour remédier à ce problème, il existe plusieurs endroits où il est possible de créer l'attribut `rnd`. Il est possible de créer l'attribut à l'intérieur de la méthode `methode1`. Mais le programme n'a plus le même sens puisqu'à chaque appel de la méthode `methode1`, l'attribut `rnd` reçoit la valeur 42. La liste de nombres aléatoires contient dix fois la même valeur.

```
class exemple_classe:
    def methode1(self,n):
        """simule la génération d'un nombre aléatoire
           compris entre 0 et n-1 inclus"""
        self.rnd = 42  # déclaration à l'intérieur de la méthode,
                       # doit être précédé du mot-clé self
        self.rnd = 397204094 * self.rnd % 2147483647
        return int (self.rnd % n)
```

```
nb = exemple_classe ()
l  = [ nb.methode1(100) for i in range(0,10) ]
print l  # affiche [19, 19, 19, 19, 19, 19, 19, 19, 19, 19]
```

Il est possible de créer l'attribut `rnd` à l'extérieur de la classe. Cette écriture devrait toutefois être évitée puisque la méthode `methode1` ne peut pas être appelée sans que l'attribut `rnd` ait été ajouté.

```
class exemple_classe:
    def methode1(self,n):
        """simule la génération d'un nombre aléatoire
           compris entre 0 et n-1 inclus"""
        self.rnd = 397204094 * self.rnd % 2147483647
        return int (self.rnd % n)

nb     = exemple_classe ()
nb.rnd = 42              # déclaration à l'extérieur de la classe,
                         # indispensable pour utiliser la méthode methode1
l = [ nb.methode1(100) for i in range(0,10) ]
print l  # affiche [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]
```

## 5.2 Constructeur

L'endroit le plus approprié pour déclarer un attribut est à l'intérieur d'une méthode appelée le *constructeur*. S'il est défini, il est implicitement exécuté lors de la création de chaque instance. Le constructeur d'une classe se présente comme une méthode et suit la même syntaxe à ceci près que son nom est imposé : `__init__`. Hormis le premier paramètre, invariablement `self`, il n'existe pas de contrainte concernant la liste des paramètres excepté que le constructeur ne doit pas retourner de résultat.

**syntaxe 5.12 : classe, constructeur**

```
class nom_classe :
    def __init__(self, param_1, ..., param_n):
        # code du constructeur
```

`nom_classe` est une classe, `__init__` est son constructeur, sa syntaxe est la même que celle d'une méthode sauf que le constructeur ne peut employer l'instruction `return`.

La modification des paramètres du constructeur implique également la modification de la syntaxe de création d'une instance de cette classe.

**syntaxe 5.13 : classe, instance**

```
x = nom_classe (valeur_1,...,valeur_n)
```

`nom_classe` est une classe, `valeur_1` à `valeur_n` sont les valeurs associées aux paramètres `param_1` à `param_n` du constructeur.

L'exemple suivant montre deux classes pour lesquelles un constructeur a été défini. La première n'ajoute aucun paramètre, la création d'une instance ne nécessite pas de paramètre supplémentaire. La seconde classe ajoute deux paramètres `a` et `b`. Lors de la création d'une instance de la classe `classe2`, il faut ajouter deux valeurs.

```
class classe1:
    def __init__(self):
        # pas de paramètre supplémentaire
        print "constructeur de la classe classe1"
        self.n = 1 # ajout de l'attribut n
```

```
x = classe1 ()     # affiche constructeur de la classe classe1
print x.n          # affiche 1

class classe2:
    def __init__(self,a,b):
        # deux paramètres supplémentaires
        print "constructeur de la classe classe2"
        self.n = (a+b)/2  # ajout de l'attribut n

x = classe2 (5,9)  # affiche constructeur de la classe classe2
print x.n          # affiche 7
```

**Remarque 5.14 : constructeur et fonction**
Le constructeur autorise autant de paramètres qu'on souhaite lors de la création d'une instance et celle-ci suit la même syntaxe qu'une fonction. La création d'une instance pourrait être considérée comme l'appel à une fonction à ceci près que le type du résultat est une instance de classe.

En utilisant un constructeur, l'exemple du paragraphe précédent simulant une suite de variable aléatoire permet d'obtenir une classe autonome qui ne fait pas appel à une variable globale ni à une déclaration d'attribut extérieur à la classe.

```
class exemple_classe:
    def __init__ (self) : # constructeur
        self.rnd = 42     # on crée l'attribut rnd, identique pour chaque instance
                          # --> les suites générées auront toutes le même début
    def methode1(self,n):
        self.rnd = 397204094 * self.rnd % 2147483647
        return int (self.rnd % n)

nb  = exemple_classe ()
l   = [ nb.methode1(100) for i in range(0,10) ]
print l   # affiche [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]

nb2 = exemple_classe ()
l2  = [ nb2.methode1(100) for i in range(0,10) ]
print l2   # affiche [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]
```

De la même manière qu'il existe un constructeur exécuté à chaque création d'instance, il existe un destructeur exécuté à chaque destruction d'instance. Il suffit pour cela de redéfinir la méthode `__del__`. A l'inverse d'autres langages comme le C++, cet opérateur est peu utilisé car le *Python* nettoie automatiquement les objets qui ne sont plus utilisés ou plus référencés par une variable. Un exemple est donné page 102.

## 5.3 Apport du langage *Python*

### 5.3.1 Liste des attributs

Chaque attribut d'une instance de classe est inséré dans un dictionnaire appelé `__dict__`, attribut implicitement présent dès la création d'une instance.

```
class exemple_classe:
    def __init__ (self) :
        self.rnd = 42
    def methode1(self,n):
        self.rnd = 397204094 * self.rnd % 2147483647
        return int (self.rnd % n)

nb = exemple_classe ()
print nb.__dict__        # affiche {'rnd': 42}
```

Ce dictionnaire offre aussi la possibilité de tester si un attribut existe ou non. Dans un des exemples du paragraphe précédent, l'attribut `rnd` était créé dans la méthode `methode1`, sa valeur était alors initialisée à chaque appel et la fonction retournait sans cesse la même valeur. En testant l'existence de l'attribut `rnd`, il est possible de le créer dans la méthode `methode1` au premier appel sans que les appels suivants ne réinitialisent sa valeur à 42.

```
class exemple_classe:
    def methode1(self,n):
        if "rnd" not in self.__dict__ :  # l'attribut existe-t-il ?
            self.rnd = 42                # création de l'attribut
            self.__dict__ ["rnd"] = 42   # autre écriture possible
        self.rnd = 397204094 * self.rnd % 2147483647
        return int (self.rnd % n)

nb = exemple_classe ()
l  = [ nb.methode1(100) for i in range(0,10) ]
print l  # affiche [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]
```

### 5.3.2 Attributs implicites

Certains attributs sont créés de manière implicite lors de la création d'une instance. Ils contiennent des informations sur l'instance.

| | |
|---|---|
| `__module__` | Contient le nom du module dans lequel est incluse la classe (voir chapitre 7). |
| `__class__` | Contient le nom de la classe de l'instance. Ce nom est précédé du nom du module suivi d'un point. |
| `__dict__` | Contient la liste des attributs de l'instance (voir paragraphe 5.3.1). |
| `__doc__` | Contient un commentaire associé à la classe (voir paragraphe 5.3.3). |

L'attribut `__class__` contient lui même d'autres d'attributs :

| | |
|---|---|
| `__doc__` | Contient un commentaire associé à la classe (voir paragraphe 5.3.3). |
| `__dict__` | Contient la liste des attributs statiques (définis hors d'une méthode) et des méthodes (voir paragraphe 5.5.2). |
| `__name__` | Contient le nom de l'instance. |
| `__bases__` | Contient les classes dont la classe de l'instance hérite (voir paragraphe 5.8). |

```
class classe_vide:
    pass
cl = classe_vide ()
print cl.__module__             # affiche __main__
print cl.__class__              # affiche __main__.classe_vide ()
print cl.__dict__               # affiche {}
print cl.__doc__                # affiche None  (voir paragraphe suivant)
print cl.__class__.__doc__      # affiche None
print cl.__class__.__dict__     # affiche {'__module__': '__main__',
                                #          '__doc__': None}
print cl.__class__.__name__     # affiche classe_vide
print cl.__class__.__bases__    # affiche ()
```

### 5.3.3 Commentaires, aide

Comme les fonctions et les méthodes, des commentaires peuvent être associés à une classe, ils sont affichés grâce à la fonction `help`. Cette dernière présente le commentaire associé à la classe, la liste des méthodes

ainsi que chacun des commentaires qui leur sont associés. Ce commentaire est affecté à l'attribut implicite `__doc__`. L'appel à la fonction `help` rassemble le commentaire de toutes les méthodes, le résultat suit le programme ci-dessous.

```
class exemple_classe:
    """simule une suite de nombres aléatoires"""
    def __init__ (self) :
        """constructeur : initialisation de la première valeur"""
        self.rnd = 42
    def methode1(self,n):
        """simule la génération d'un nombre aléatoire
        compris entre 0 et n-1 inclus"""
        self.rnd = 397204094 * self.rnd % 2147483647
        return int (self.rnd % n)
nb = exemple_classe ()
help (exemple_classe)     # appelle l'aide associée à la classe
```

```
class exemple_classe
 |  simule une suite de nombres aléatoires
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |      constructeur : initialisation de la première valeur
 |
 |  methode1(self, n)
 |      simule la génération d'un nombre aléatoire
 |      compris entre 0 et n-1 inclus
```

Pour obtenir seulement le commentaire associé à la classe, il suffit d'écrire l'une des trois lignes suivantes :

```
print exemple_classe.__doc__  # affiche simule une suite de nombres aléatoires
print nb.__doc__              # affiche simule une suite de nombres aléatoires
print nb.__class__.__doc__    # affiche simule une suite de nombres aléatoires
```

La fonction `help` permet d'accéder à l'aide associée à une fonction, une classe. Il existe des outils qui permettent de collecter tous ces commentaires pour construire une documentation au format *HTML* à l'aide d'outils comme *pydoc*[3], *epydoc*[4] ou *Doxygen*[5]. Ces outils sont souvent assez simples d'utilisation.

La fonction `dir` permet aussi d'obtenir des informations sur la classe. Cette fonction appliquée à la classe ou à une instance retourne l'ensemble de la liste des attributs et des méthodes. L'exemple suivant utilise la fonction `dir` avant et après l'appel de la méthode `meth`. Etant donné que cette méthode ajoute un attribut, la fonction `dir` retourne une liste plus longue après l'appel.

```
class essai_class:
    def meth(self):
        x      = 6
        self.y = 7

a = essai_class()
print dir (a)             # affiche ['__doc__', '__module__', 'meth']
a.meth ()
print dir (a)             # affiche ['__doc__', '__module__', 'meth', 'y']
print dir (essai_class)   # affiche ['__doc__', '__module__', 'meth']
```

La fonction `dir` appliquée à la classe elle-même retourne une liste qui inclut les méthodes et les attributs déjà déclarés. Elle n'inclut pas ceux qui sont déclarés dans une méthode jamais exécutée jusqu'à présent.

3. http://pydoc.org/
4. http://epydoc.sourceforge.net/
5. http://www.stack.nl/$\sim$dimitri/doxygen/ associé à *doxypy* (http://code.foosel.org/doxypy).

### 5.3.4 Classe incluse

Parfois, il arrive qu'une classe soit exclusivement utilisée en couple avec une autre, c'est par exemple le cas des itérateurs (voir paragraphe 5.4.2). Il est alors possible d'inclure dans la déclaration d'une classe celle d'une sous-classe.

L'exemple qui suit contient la classe `ensemble_element`. C'est un ensemble de points en trois dimensions (classe `element`) qui n'est utilisé que par cette classe. Déclarer la classe `element` à l'intérieur de la classe `ensemble_element` est un moyen de signifier ce lien.

```
class ensemble_element :

    class element :
        def __init__ (self) :
            self.x, self.y, self.z = 0,0,0

    def __init__ (self) :
        self.all = [ ensemble_element.element () for i in xrange (0,3) ]

    def barycentre (self) :
        b = ensemble_element.element ()
        for el in self.all :
            b.x += el.x
            b.y += el.y
            b.z += el.z
        b.x /= len (self.all)
        b.y /= len (self.all)
        b.z /= len (self.all)
        return b

f = ensemble_element ()
f.all [0].x, f.all [0].y, f.all [0].z = 4.5,1.5,1.5
b = f.barycentre ()
print b.x,b.y,b.z # affiche 1.5 0.5 0.5
```

Pour créer une instance de la classe `element`, il faut faire précéder son nom de la classe où elle est déclarée : `b = ensemble_element.element()` comme c'est le cas dans la méthode `barycentre` par exemple.

## 5.4 Opérateurs, itérateurs

Les opérateurs sont des symboles du langages comme $+$, $-$, $+=$, ... Au travers des opérateurs, il est possible de donner un sens à une syntaxe comme celle de l'exemple suivant :

```
class nouvelle_classe:
    pass
x = nouvelle_classe () + nouvelle_classe ()
```

L'addition n'est pas le seul symbole concerné, le langage *Python* permet de donner un sens à tous les opérateurs numériques et d'autres reliés à des fonctions du langage comme `len` ou `max`.

### 5.4.1 Opérateurs

Le programme suivant contient une classe définissant un nombre complexe. La méthode `ajoute` définit ce qu'est une addition entre nombres complexes.

```
class nombre_complexe:
    def __init__ (self, a = 0, b= 0) : self.a, self.b = a,b
    def get_module (self) :  return math.sqrt (self.a * self.a + self.b * self.b)
```

```
    def ajoute (self,c):
        return nombre_complexe (self.a + c.a, self.b + c.b)

c1 = nombre_complexe (0,1)
c2 = nombre_complexe (1,0)
c  = c1.ajoute (c2)         # c = c1 + c2
print c.a, c.b
```

Toutefois, on aimerait bien écrire simplement `c = c1 + c2` au lieu de `c = c1.ajoute(c2)` car cette syntaxe est plus facile à lire et surtout plus intuitive. Le langage *Python* offre cette possibilité. Il existe en effet des méthodes *clés* dont l'implémentation définit ce qui doit être fait dans le cas d'une addition, d'une comparaison, d'un affichage, ... A l'instar du constructeur, toutes ces méthodes clés, qu'on appelle des *opérateurs*, sont encadrées par deux blancs soulignés, leur déclaration suit invariablement le même schéma. Voici celui de l'opérateur `__add__` qui décrit ce qu'il faut faire pour une addition.

**syntaxe 5.15 : classe, opérateur `__add__`**

```
class nom_class :
    def __add__ (self, autre) :
        # corps de l'opérateur
        return ...   # nom_classe
```

`nom_classe` est une classe. L'opérateur `__add__` définit l'addition entre l'instance `self` et l'instance `autre` et retourne une instance de la classe `nom_classe`.

Le programme suivant reprend le précédent de manière à ce que l'addition de deux nombres complexes soit dorénavant une syntaxe correcte.

```
class nombre_complexe:
    def __init__ (self, a = 0, b= 0) : self.a, self.b = a,b
    def get_module (self) : return math.sqrt (self.a * self.a + self.b * self.b)

    def __add__(self, c):
        return nombre_complexe (self.a + c.a, self.b + c.b)

c1 = nombre_complexe (0,1)
c2 = nombre_complexe (1,0)
c  = c1 + c2          # cette expression est maintenant syntaxiquement correcte
c  = c1.__add__ (c2)  # même ligne que la précédente mais écrite explicitement
print c.a, c.b
```

L'avant dernière ligne appelant la méthode `__add__` transcrit de façon explicite ce que le langage *Python* fait lorsqu'il rencontre un opérateur + qui s'applique à des classes. Plus précisément, `c1` et `c2` pourraient être de classes différentes, l'expression serait encore valide du moment que la classe dont dépend `c1` a redéfini la méthode `__add__`. Chaque opérateur possède sa méthode-clé associée. L'opérateur + =, différent de + est associé à la méthode-clé `__iadd__`.

**syntaxe 5.16 : classe, opérateur `__iadd__`**

```
class nom_class :
    def __iadd__ (self, autre) :
        # corps de l'opérateur
        return self
```

`nom_classe` est une classe. L'opérateur `__iadd__` définit l'addition entre l'instance `self` et l'instance `autre`. L'instance `self` est modifiée pour recevoir le résultat. L'opérateur retourne invariablement l'instance modifiée `self`.

On étoffe la classe `nombre_complexe` à l'aide de l'opérateur `__iadd__`.

```
class nombre_complexe:
    def __init__ (self, a = 0, b= 0) : self.a, self.b = a,b
```

```
    def get_module (self) : return math.sqrt (self.a * self.a + self.b * self.b)
    def __add__(self, c)  : return nombre_complexe (self.a + c.a, self.b + c.b)

    def __iadd__(self, c) :
        self.a += c.a
        self.b += c.b
        return self

c1  = nombre_complexe (0,1)
c2  = nombre_complexe (1,0)
c1 += c2           # utilisation de l'opérateur +=
c1.__iadd__ (c2)   # c'est la transcription explicite de la ligne précédente
print c1.a, c1.b
```

Un autre opérateur souvent utilisé est `__str__` qui permet de redéfinir l'affichage d'un objet lors d'un appel à l'instruction `print`.

**syntaxe 5.17 : classe, opérateur `__str__`**

```
class nom_class :
    def __str__ (self) :
        # corps de l'opérateur
        return...
```

`nom_classe` est une classe. L'opérateur `__str__` construit une chaîne de caractères qu'il retourne comme résultat de façon à être affiché.

L'exemple suivant reprend la classe `nombre_complexe` pour que l'instruction `print` affiche un nombre complexe sous la forme $a + ib$.

```
class nombre_complexe:
    def __init__ (self, a = 0, b= 0) : self.a, self.b = a,b
    def __add__(self, c) : return nombre_complexe (self.a + c.a, self.b + c.b)

    def __str__ (self) :
        if   self.b == 0 : return "%f" % (self.a)
        elif self.b >  0 : return "%f + %f i" % (self.a, self.b)
        else             : return "%f - %f i" % (self.a, -self.b)

c1 = nombre_complexe (0,1)
c2 = nombre_complexe (1,0)
c3 = c1 + c2
print c3       # affiche 1.000000 + 1.000000 i
```

Il existe de nombreux opérateurs qu'il est possible de définir. La table 5.1 (page 121) présente les plus utilisés. Parmi ceux-là, on peut s'attarder sur les opérateurs `__getitem__` et `__setitem__`, ils redéfinissent l'opérateur [] permettant d'accéder à un élément d'une liste ou d'un dictionnaire. Le premier, `__getitem__` est utilisé lors d'un calcul, un affichage. Le second, `__setitem__`, est utilisé pour affecter une valeur.

L'exemple suivant définit un point de l'espace avec trois coordonnées. Il redéfinit ou *surcharge* les opérateurs `__getitem__` et `__setitem__` de manière à pouvoir accéder aux coordonnées de la classe `point_espace` qui définit un point dans l'espace. En règle générale, lorsque les indices ne sont pas corrects, ces deux opérateurs lèvent l'exception `IndexError` (voir le chapitre 6).

```
class point_espace:
    def __init__ (self, x,y,z): self._x, self._y, self._z = x,y,z

    def __getitem__(self,i):
        if i == 0 : return self._x
        if i == 1 : return self._y
        if i == 2 : return self._z
        # pour tous les autres cas --> erreur
```

```
        raise IndexError ("indice impossible, 0,1,2 autorisés")

    def __setitem__(self,i,x):
        if   i == 0 : self._x = x
        elif i == 1 : self._y = y
        elif i == 2 : self._z = z
        # pour tous les autres cas --> erreur
        raise IndexError ("indice impossible, 0,1,2 autorisés")

    def __str__(self):
        return "(%f,%f,%f)" % (self._x, self._y, self._z)

a = point_espace (1,-2,3)

print a                      # affiche (1.000000,-2.000000,3.000000)
a [1] = -3                   # (__setitem__) affecte -3 à a.y
print "abscisse : ", a [0]   # (__getitem__) affiche abscisse :  1
print "ordonnée : ", a [1]   # (__getitem__) affiche ordonnée :  -3
print "altitude : ", a [2]   # (__getitem__) affiche altitude :  3
```

Par le biais de l'exception `IndexError`, les expressions `a[i]` avec $\texttt{i} \neq 0, 1, 2$ sont impossibles et arrêtent le programme par un message comme celui qui suit obtenu après l'interprétation de `print a [4]` :

```
Traceback (most recent call last):
  File "point_espace.py", line 31, in ?
    print a [4]
  File "point_espace.py", line 13, in __getitem__
    raise IndexError, "indice impossible, 0,1,2 autorisés"
IndexError: indice impossible, 0,1,2 autorisés
```

### 5.4.2 Itérateurs

L'opérateur `__iter__` permet de définir ce qu'on appelle un itérateur. C'est un objet qui permet d'en explorer un autre, comme une liste ou un dictionnaire. Un itérateur est un objet qui désigne un élément d'un ensemble à parcourir et qui connaît l'élément suivant à visiter. Il doit pour cela contenir une référence à l'objet qu'il doit explorer et inclure une méthode `next` qui retourne l'élément suivant ou lève une exception si l'élément actuel est le dernier.

Par exemple, on cherche à explorer tous les éléments d'un objet de type `point_espace` défini au paragraphe précédent. Cette exploration doit s'effectuer au moyen d'une boucle `for`.

```
a = point_espace (1,-2,3)
for x in a:
    print x      # affiche successivement 1,-2,3
```

Cette boucle cache en fait l'utilisation d'un itérateur qui apparaît explicitement dans l'exemple suivant équivalent au précédent (voir paragraphe 4.3.2.3, page 59).

```
a = point_espace (1,-2,3)
it = iter (a)
while True:
    try : print it.next ()
    except StopIteration : break
```

Afin que cet extrait de programme fonctionne, il faut définir un itérateur pour la classe `point_espace`. Cet itérateur doit inclure la méthode `next`. La classe `point_espace` doit quant à elle définir l'opérateur `__iter__` pour retourner l'itérateur qui permettra de l'explorer.

```
class point_espace:
    def __init__ (self, x,y,z):
```

```
        self._x, self._y, self._z = x,y,z
    def __str__(self):
        return "(%f,%f,%f)" % (self._x, self._y, self._z)
    def __getitem__(self,i):
        if i == 0 : return self._x
        if i == 1 : return self._y
        if i == 2 : return self._z
        # pour tous les autres cas --> erreur
        raise IndexError ("indice impossible, 0,1,2 autorisés")

    class class_iter:
        """cette classe définit un itérateur pour point_espace"""
        def __init__ (self,ins):
            """initialisation, self._ins permet de savoir quelle
               instance de point_espace on explore,
               self._n mémorise l'indice de l'élément exploré"""
            self._n   = 0
            self._ins = ins

        def next (self):
            """retourne l'élément d'indice self._n et passe à l'élément suivant"""
            if self._n <= 2:
                v = self._ins [self._n]
                self._n += 1
                return v
            else :
                # si cet élément n'existe pas, lève une exception
                raise StopIteration

    def __iter__(self):
        """opérateur de la classe point_espace, retourne un itérateur
           permettant de l'explorer"""
        return point_espace.class_iter (self)

a = point_espace (1,-2,3)
for x in a:
    print x      # affiche successivement 1,-2,3
```

Cette syntaxe peut paraître fastidieuse mais elle montre de manière explicite le fonctionnement des itérateurs. Cette construction est plus proche de ce que d'autres langages objets proposent. *Python* offre néanmoins une syntaxe plus courte avec le mot-clé `yield` qui permet d'éviter la création de la classe `class_iter`. Le code de la méthode `__iter__` change mais les dernières lignes du programme précédent qui affichent successivement les éléments de `point_espace` sont toujours valides.

```
class point_espace:
    def __init__ (self, x,y,z):
        self._x, self._y, self._z = x,y,z
    def __str__(self):
        return "(%f,%f,%f)" % (self._x, self._y, self._z)
    def __getitem__(self,i):
        if i == 0 : return self._x
        if i == 1 : return self._y
        if i == 2 : return self._z
        # pour tous les autres cas --> erreur
        raise IndexError ("indice impossible, 0,1,2 autorisés")

    def __iter__(self):
        """itérateur avec yield"""
        _n = 0
        while _n <= 2 :
            yield self.__getitem__ (_n)
            _n += 1

a = point_espace (1,-2,3)
```

```
for x in a:
    print x      # affiche successivement 1,-2,3
```

## 5.5 Méthodes, attributs statiques et ajout de méthodes

### 5.5.1 Méthode statique

**Définition 5.18 : méthode statique**
Les méthodes statiques sont des méthodes qui peuvent être appelées même si aucune instance de la classe où elles sont définies n'a été créée.

L'exemple suivant définit une classe avec une seule méthode. Comme toutes les méthodes présentées jusqu'à présent, elle inclut le paramètre `self` qui correspond à l'instance pour laquelle elle est appelée.

```
class essai_class:
    def methode (self):
        print "méthode non statique"

x = essai_class ()
x.methode ()
```

Une méthode statique ne nécessite pas qu'une instance soit créée pour être appelée. C'est donc une méthode n'ayant pas besoin du paramètre `self`.

**syntaxe 5.19 : classe, méthode statique**

```
class nom_class :
    def nom_methode(params, ...) :
        # corps de la méthode
        ...
    nom_methode = staticmethod (nom_methode)
```

`nom_classe` est une classe, `nom_methode` est une méthode statique. Il faut pourtant ajouter la ligne suivante pour indiquer à la classe que cette méthode est bien statique à l'aide du mot-clé `staticmethod`.

Le programme précédent est modifié pour inclure une méthode statique. La méthode `methode` ne nécessite aucune création d'instance pour être appelée.

```
class essai_class:
    def methode ():
        print "méthode statique"
    methode = staticmethod(methode)

essai_class.methode ()
```

**Remarque 5.20 : déclaration d'une méthode statique**
Il est également possible de déclarer une fonction statique à l'extérieur d'une classe puis de l'ajouter en tant que méthode statique à cette classe. Le programme suivant déclare une fonction `methode` puis indique à la classe `essai_class` que la fonction est aussi une méthode statique de sa classe (avant-dernière ligne de l'exemple).

```
def methode ():
    print "méthode statique"

class essai_class:
```

```
    pass

essai_class.methode = staticmethod(methode)
essai_class.methode ()
```

Toutefois, il est conseillé de placer l'instruction qui contient `staticmethod` à l'intérieur de la classe. Elle n'y sera exécutée qu'une seule fois comme le montre l'exemple suivant :

```
class essai_class:
    print "création d'une instance de la classe essai_class"
    methode = staticmethod(methode)
cl = classe_vide () # affiche création d'une instance de la classe essai_class
ck = classe_vide () # n'affiche rien
```

Les méthodes statiques sont souvent employées pour créer des instances spécifiques d'une classe.

```
class Couleur :
    def __init__ (self, r, v, b) : self.r, self.v, self.b =  r, v, b
    def __str__ (self) : return str ( (self.r,self.v,self.b))
    def blanc () : return Couleur (255,255,255)
    def noir  () : return Couleur (0,0,0)
    blanc = staticmethod (blanc)
    noir  = staticmethod (noir)

c = Couleur.blanc ()
print c               # affiche (255, 255, 255)
c = Couleur.noir ()
print c               # affiche (0, 0, 0)
```

### 5.5.2 Attributs statiques

**Définition 5.21 : attribut statique**
Les attributs statiques sont des attributs qui peuvent être utilisés même si aucune instance de la classe où ils sont définis n'a été créée. Ces attributs sont partagés par toutes les instances.

**syntaxe 5.22 : classe, attribut statique**

```
class nom_class :
    attribut_statique = valeur
    def nom_methode (self,params, ...):
        nom_class.attribut_statique2 = valeur2
    def nom_methode_st (params, ...) :
        nom_class.attribut_statique3 = valeur3
    nom_methode_st = staticmethod (nom_methode_st)
```

`nom_classe` est une classe, `nom_methode` est une méthode non statique, `nom_methode_st` est une méthode statique. Les trois paramètres `attribut_statique`, `attribut_statique2`, `attribut_statique3` sont statiques, soit parce qu'ils sont déclarés en dehors d'une méthode, soit parce que leur déclaration fait intervenir le nom de la classe.

Pour le programme suivant, la méthode `meth` n'utilise pas `self.x` mais `essai_class.x`. L'attribut `x` est alors un attribut statique, partagé par toutes les instances. C'est pourquoi dans l'exemple qui suit l'instruction `z.meth()` affiche la valeur 6 puisque l'appel `y.meth()` a incrémenté la variable statique `x`.

```
class essai_class:
    x = 5
    def meth(self):
```

```
        print essai_class.x
        essai_class.x += 1

y = essai_class ()
z = essai_class ()
y.meth()    # affiche 5
z.meth()    # affiche 6
```

**Remarque 5.23 : ambiguïté**
Même si un attribut est statique, il peut être utilisé avec la syntaxe `self.attribut_statique` dans une méthode non statique à condition qu'un attribut non statique ne porte pas le même nom. Si tel est pourtant le cas, certaines confusions peuvent apparaître :

```
class exemple_classe:
    rnd = 42
    def incremente_rnd (self):
        self.rnd += 1
        return self.rnd

cl = exemple_classe()

print cl.__dict__                     # affiche {}
print cl.__class__.__dict__ ["rnd"]  # affiche 42
cl.incremente_rnd ()
print cl.__dict__                     # affiche {'rnd': 43}
print cl.__class__.__dict__ ["rnd"]  # affiche 42
```

Dans ce cas, ce sont en fait deux attributs qui sont créés. Le premier est un attribut statique créé avec la seconde ligne de l'exemple `rnd = 42`. Le second attribut n'est pas statique et apparaît dès la première exécution de l'instruction `self.rnd+ = 1` comme le montre son apparition dans l'attribut `__dict__` qui ne recense pas les attributs statiques.

### 5.5.3 Ajout de méthodes

Ce point décrit une fonctionnalité du langage *Python* rarement utilisée. Il offre la possibilité d'ajouter une méthode à une classe alors même que cette fonction est définie à l'extérieur de la déclaration de la classe. Cette fonction doit obligatoirement accepter un premier paramètre qui recevra l'instance de la classe. La syntaxe utilise le mot-clé `classmethod`.

**syntaxe 5.24 : classe, `classmethod`**

```
def nom_methode (cls) :
    # code de la fonction}

class nom_classe :
    # code de la classe
    nom_methode = classmethod (nom_methode)        # syntaxe 1

nom_classe.nom_methode = classmethod (nom_methode) # syntaxe 2
```

`nom_classe` est une classe, `nom_methode` est une méthode, `nom_methode` est une fonction qui est par la suite considérée comme une méthode de la classe `nom_methode` grâce à l'une ou l'autre des deux instructions incluant le mot-clé `classmethod`.

Dans l'exemple qui suit, cette syntaxe est utilisée pour inclure trois méthodes à la classe `essai_class` selon que la méthode est déclarée et affectée à cette classe à l'intérieur ou à l'extérieur du corps de `essai_class`.

```
def meth3 (cls): print "ok meth3", cls.x
def meth4 (cls): print "ok meth4", cls.x
```

```
class essai_classe:
    x = 5
    def meth(self): print "ok meth", self.x
    def meth2(cls): print "ok meth2", cls.x

    meth3 = classmethod (meth3)

x = essai_classe ()
x.meth ()                                # affiche ok meth 5
x.meth2 ()                               # affiche ok meth2 5
x.meth3 ()                               # affiche ok meth3 5

essai_classe.meth4 = classmethod (meth4)
x.meth4 ()                               # affiche ok meth4 5
```

### 5.5.4 Propriétés

Cette fonctionnalité est également peu utilisée, elle permet des raccourcis d'écriture. Les propriétés permettent de faire croire à l'utilisateur d'une instance de classe qu'il utilise une variable alors qu'il utilise en réalité une ou plusieurs méthodes. A chaque fois que le programmeur utilise ce faux attribut, il appelle une méthode qui calcule sa valeur. A chaque fois que le programmeur cherche à modifier la valeur de ce faux attribut, il appelle une autre méthode qui modifie l'instance.

**syntaxe 5.25 : classe, propriété**

```
class nom_classe :
    # code de la classe
    nom_propriete = property (fget, fset, fdel, doc)
```

Au sein de ces trois lignes, `nom_classe` est une classe, `nom_propriete` est le nom de la propriété, `fget` est la méthode qui doit retourner la valeur du pseudo-attribut `nom_propriete`, `fset` est la méthode qui doit modifier la valeur du pseudo-attribut `nom_propriete`, `fdel` est la méthode qui doit détruire le pseudo-attribut `nom_propriete`, `doc` est un commentaire qui apparaîtra lors de l'appel de la fonction `help(nom_class)` ou `help(nom_class.nom_propriete)`.

Pour illustrer l'utilisation des propriétés, on part d'une classe `nombre_complexe` qui ne contient que les parties réelle et imaginaire. Lorsqu'on cherche à obtenir le module [6], on fait appel à une méthode qui calcule ce module. Lorsqu'on cherche à modifier ce module, on fait appel à une autre méthode qui multiplie les parties réelle et imaginaire par un nombre réel positif de manière à ce que le nombre complexe ait le module demandé. On procède de même pour la propriété `arg`.

La propriété `conj` retourne quant à elle le conjugué du nombre complexe mais la réciproque n'est pas prévue. On ne peut affecter une valeur à `conj`.

```
import math

class nombre_complexe(object):           # voir remarque après l'exemple
    def __init__ (self, a = 0, b= 0):
        self.a = a
        self.b = b

    def __str__ (self) :
        if   self.b == 0 : return "%f" % (self.a)
        elif self.b >  0 : return "%f + %f i" % (self.a, self.b)
        else             : return "%f - %f i" % (self.a, -self.b)

    def get_module (self):
```

6. Le module désigne ici le module d'un nombre complexe qui est égal à sa norme. On le note $|a + ib| = \sqrt{a^2 + b^2}$.

```
        return math.sqrt (self.a * self.a + self.b * self.b)

    def set_module (self,m):
        r = self.get_module ()
        if r == 0:
            self.a = m
            self.b = 0
        else :
            d       = m / r
            self.a *= d
            self.b *= d

    def get_argument (self) :
        r = self.get_module ()
        if r == 0 : return 0
        else      : return math.atan2 (self.b / r, self.a / r)

    def set_argument (self,arg) :
        m       = self.get_module ()
        self.a  = m * math.cos (arg)
        self.b  = m * math.sin (arg)

    def get_conjugue (self):
        return nombre_complexe (self.a,-self.b)

    module = property (fget = get_module,   fset = set_module,   doc = "module")
    arg    = property (fget = get_argument, fset = set_argument, doc = "argument")
    conj   = property (fget = get_conjugue,                      doc = "conjugué")

c = nombre_complexe (0.5,math.sqrt (3)/2)
print "c = ",         c          # affiche c =  0.500000 + 0.866025 i
print "module = ",    c.module   # affiche module =  1.0
print "argument = ",  c.arg      # affiche argument =  1.0471975512

c           = nombre_complexe ()
c.module    = 1
c.arg       = math.pi * 2 / 3
print "c = ",         c          # affiche c =  -0.500000 + 0.866025 i
print "module = ",    c.module   # affiche module =  1.0
print "argument = ",  c.arg      # affiche argument =  2.09439510239
print "conjugué = ",  c.conj     # affiche conjugué =  -0.500000 - 0.866025 i
```

La propriété `conj` ne possède pas de fonction qui permet de la modifier. Par conséquent, l'instruction `c.conj = nombre_complexe(0,0)` produit l'erreur suivante :

```
Traceback (most recent call last):
  File "cour2.py", line 53, in ?
    c.conj = nombre_complexe (0,0)
AttributeError: can't set attribute
```

Etant donné qu'une propriété porte déjà le nom de `conj`, aucun attribut du même nom ne peut être ajouté à la classe `nombre_complexe`.

**Remarque 5.26 : propriété et héritage**
Afin que la propriété fonctionne correctement, il est nécessaire que la classe hérite de la classe `object` ou une de ses descendantes (voir également paragraphe 5.8).

## 5.6 Copie d'instances

### 5.6.1 Copie d'instance de classe simple

Aussi étrange que cela puisse paraître, le signe = ne permet pas de recopier une instance de classe. Il permet d'obtenir deux noms différents pour désigner le même objet. Dans l'exemple qui suit, la ligne `nb2 = nb` ne fait pas de copie de l'instance `nb`, elle permet d'obtenir un second nom `nb2` pour l'instance `nb`. Vu de l'extérieur, la ligne `nb2.rnd = 0` paraît modifier à la fois les objets `nb` et `nb2` puisque les lignes `print nb.rnd` et `print nb2.rnd` affichent la même chose. En réalité, `nb` et `nb2` désignent le même objet.

```
class exemple_classe:
    def __init__ (self) : self.rnd = 42
    def methode1(self,n):
        self.rnd = 397204094 * self.rnd % 2147483647
        return int (self.rnd % n)

nb  = exemple_classe ()
nb2 = nb
print nb.rnd        # affiche 42
print nb2.rnd       # affiche 42

nb2.rnd = 0

print nb2.rnd       # affiche 0, comme prévu
print nb.rnd        # affiche 0, si nb et nb2 étaient des objets différents,
                    # cette ligne devrait afficher 42
```

Pour créer une copie de l'instance `nb`, il faut le dire explicitement en utilisant la fonction `copy` du module `copy` (voir le chapitre 7).

**syntaxe 5.27 : fonction `copy`**

```
import copy
nom_copy = copy.copy (nom_instance)
```

`nom_instance` est une instance à copier, `nom_copy` est le nom désignant la copie.

L'exemple suivant applique cette copie sur la classe `exemple_classe` générant des nombres aléatoires.

```
class exemple_classe:
    def __init__ (self) : self.rnd = 42
    def methode1(self,n):
        self.rnd = 397204094 * self.rnd % 2147483647
        return int (self.rnd % n)

nb = exemple_classe ()

import copy           # pour utiliser le module copy
nb2 = copy.copy (nb)  # copie explicite

print nb.rnd    # affiche 42
print nb2.rnd   # affiche 42

nb2.rnd = 0

print nb2.rnd   # affiche 0
print nb.rnd    # affiche 42
```

**Remarque 5.28 : suppression de variables associées à la même instance**
Le symbole égalité ne fait donc pas de copie, ceci signifie qu'une même instance de classe peut porter plusieurs noms.

```
m  = [ 0, 1 ]
m2 = m
del m2   # supprime l'identificateur mais pas la liste
print m  # affiche [0, 1]
```

La suppression d'un objet n'est effective que s'il ne reste aucune variable le référençant. L'exemple suivant le montre.

```
class CreationDestruction (object) :

    def __init__ (self) :
        print "constructeur"

    def __new__ (self) :
        print "__new__"
        return object.__new__ (self)

    def __del__ (self) :
        print "__del__"

print "a"
m = CreationDestruction ()
print "b"
m2 = m
print "c"
del m
print "d"
del m2
print "e"
```

La sortie de ce programme est la suivante :

```
a
__new__
constructeur
b
c
d
__del__
e
```

Le destructeur est appelé autant de fois que le constructeur. Il est appelé lorsque plus aucun identificateur n'est relié à l'objet. Cette configuration survient lors de l'exemple précédent car le mot-clé `del` a détruit tous les identificateurs `m` et `m2` qui étaient reliés au même objet.

### 5.6.2 Copie d'instance de classes incluant d'autres classes

La fonction `copy` n'est pas suffisante lorsqu'une classe inclut des attributs qui sont eux-mêmes des classes incluant des attributs. Dans l'exemple qui suit, la classe `exemple_classe` inclut un attribut de type `classe_incluse` qui contient un attribut `attr`. Lors de la copie à l'aide de l'instruction `nb2 = copy.copy(nb)`, l'attribut `inclus` n'est pas copié, c'est l'instruction `nb2.inclus = nb.inclus` qui est exécutée. On se retrouve donc avec deux noms qui désignent encore le même objet.

```
class classe_incluse:
    def __init__ (self) : self.attr = 3

class exemple_classe:
    def __init__ (self) :
        self.inclus = classe_incluse ()
        self.rnd    = 42
```

```
nb = exemple_classe ()

import copy            # pour utiliser le module copy
nb2 = copy.copy (nb)   # copie explicite

print nb.inclus.attr   # affiche 3
print nb2.inclus.attr  # affiche 3

nb2.inclus.attr = 0

print nb.inclus.attr   # affiche 0 (on voudrait 3 ici)
print nb2.inclus.attr  # affiche 0
```

Pour effectivement copier les attributs dont le type est une classe, la première option - la plus simple - est de remplacer la fonction `copy` par la fonction `deepcopy`. Le comportement de cette fonction dans le cas des classes est le même que dans le cas des listes comme l'explique la remarque 3.18 page 43. La seconde solution, rarement utilisée, est d'utiliser l'opérateur `__copy__` et ainsi écrire le code associé à la copie des attributs de la classe.

**syntaxe 5.29 : classe, opérateur `__copy__`**

```
class nom_classe :
    def __copy__ () :
        copie = nom_classe (...)
        # ...
        return copie
```

`nom_classe` est le nom d'une classe. La méthode `__copy__` doit retourner une instance de la classe `nom_classe`, dans cet exemple, cette instance a pour nom `copie`.

L'exemple suivant montre un exemple d'implémentation de la classe `__copy__`. Cette méthode crée d'abord une autre instance `copie` de la classe `exemple_classe` puis initialise un par un ses membres. L'attribut `rnd` est recopié grâce à une affectation car c'est un nombre. L'attribut `inclus` est recopié grâce à la fonction `copy` du module `copy` car c'est une instance de classe. Après la copie, on vérifie bien que modifier l'attribut `inclus.attr` de l'instance `nb` ne modifie pas l'attribut `inclus.attr` de l'instance `nb2`.

```
import copy

class classe_incluse:
    def __init__ (self) : self.attr = 3

class exemple_classe:
    def __init__ (self) :
        self.inclus  = classe_incluse ()
        self.rnd     = 42
    def __copy__ (self):
        copie        = exemple_classe ()
        copie.rnd    = self.rnd
        copie.inclus = copy.copy (self.inclus)
        return copie

nb  = exemple_classe ()

nb2 = copy.copy (nb)   # copie explicite,
                       # utilise l'opérateur __copy__,
                       # cette ligne est équivalente à
                       # nb2 = nb.__copy__()

print nb.rnd           # affiche 42
print nb2.rnd          # affiche 42
print nb.inclus.attr   # affiche 3
print nb2.inclus.attr  # affiche 3
```

```
nb.inclus.attr = 0
nb.rnd         = 1

print nb.rnd           # affiche 1
print nb2.rnd          # affiche 42
print nb.inclus.attr   # affiche 0
print nb2.inclus.attr  # affiche 3 (c'est le résultat souhaité)
```

**Remarque 5.30 : affectation et copie**
On peut se demander pourquoi l'affectation n'est pas équivalente à une copie. Cela tient au fait que l'affectation en langage *Python* est sans cesse utilisée pour affecter le résultat d'une fonction à une variable. Lorsque ce résultat est de taille conséquente, une copie peut prendre du temps. Il est préférable que le résultat de la fonction reçoive le nom prévu pour le résultat.

```
def fonction_liste ():
    return range (4,7) # retourne la liste [4,5,6]
l = fonction_liste ()  # la liste [4,5,6] n'est pas recopiée,
                       # l'identificateur l lui est affecté
```

Lorsqu'une fonction retourne un résultat mais que celui-ci n'est pas attribué à un nom de variable. Le langage *Python* détecte automatiquement que ce résultat n'est plus lié à aucune variable. Il est détruit automatiquement.

```
def fonction_liste ():
    return range (4,7)
fonction_liste () # la liste [4,5,6] n'est pas recopiée,
                  # elle n'est pas non plus attribuée à une variable,
                  # elle est alors détruite automatiquement par le langage Python
```

### 5.6.3 Listes et dictionnaires

Les listes et les dictionnaires sont des types modifiables et aussi des classes. Par conséquent, l'affectation et la copie ont un comportement identique à celui des classes.

```
l  = [4,5,6]
l2 = l
print l       # affiche [4, 5, 6]
print l2      # affiche [4, 5, 6]
l2 [1] = 10
print l       # affiche [4, 10, 6]
print l2      # affiche [4, 10, 6]
```

Pour effectuer une copie, il faut écrire le code suivant :

```
l  = [4,5,6]
import copy
l2 = copy.copy (l)
print l       # affiche [4, 5, 6]
print l2      # affiche [4, 5, 6]
l2 [1] = 10
print l       # affiche [4, 5, 6]
print l2      # affiche [4, 10, 6]
```

La fonction `copy` ne suffit pourtant pas lorsque l'objet à copier est par exemple une liste incluant d'autres objets. Elle copiera la liste et ne fera pas de copie des objets eux-mêmes.

```
import copy
l  = [ [i] for i in range(0,3)]
```

```
ll = copy.copy (l)
print l, "  -  ", ll    # affiche [[0], [1], [2]]   -   [[0], [1], [2]]
ll [0][0] = 6
print l, "  -  ", ll    # affiche [[6], [1], [2]]   -   [[6], [1], [2]]
```

Il n'est pas possible de modifier la méthode `__copy__` d'un objet de type liste. Il existe néanmoins la fonction `deepcopy` qui permet de faire une copie à la fois de la liste et des objets qu'elle contient.

```
import copy
l  = [ [i] for i in range(0,3)]
ll = copy.deepcopy (l)
print l, "  -  ", ll    # affiche [[0], [1], [2]]   -   [[0], [1], [2]]
ll [0][0] = 6
print l, "  -  ", ll    # affiche [[0], [1], [2]]   -   [[6], [1], [2]]
```

### 5.6.4 `copy` et `deepcopy`

La fonction `copy` effectue une copie d'un objet, la fonction `deepcopy` effectue une copie d'un objet et de ceux qu'il contient. La fonction `copy` est associée à la méthode `__copy__` tandis que la fonction `deepcopy` est associée à la méthode `__deepcopy__`. Il est rare que l'une de ces deux méthodes doivent être redéfinies. L'intérêt de ce paragraphe est plus de montrer le mécanisme que cache la fonction `deepcopy` qui est la raison pour laquelle il existe deux fonctions de copie et non une seule.

**syntaxe 5.31 : fonction `deepcopy`**

```
import copy
memo = { }
nom_copy = copy.deepcopy (nom_instance [,memo])
```

`nom_instance` est une instance à copier, `nom_copy` est le nom désignant la copie. `memo` est un paramètre facultatif : s'il est envoyé à la fonction `deepcopy`, il contiendra alors la liste de toutes les copies d'objet effectuées lors de cet appel.

**syntaxe 5.32 : classe, opérateur `__deepcopy__`**

```
class nom_classe :
    def __deepcopy__ (self,memo) :
        copie = copy.copy (self)
        # ...
        return copie
```

`nom_classe` est le nom d'une classe. La méthode `__deepcopy__` doit retourner une instance de la classe `nom_classe`, dans cet exemple, cette instance a pour nom `copie`. Le paramètre `memo` permet de conserver la liste des copies effectuées à condition d'appeler `deepcopy` avec un dictionnaire en paramètre.

Le programme suivant reprend le second programme du paragraphe 5.6.2 et modifie la classe `classe_incluse` pour distinguer copie et copie profonde. Il peut être utile de lire le paragraphe 3.3.3.4 (page 47) pour comprendre pourquoi un dictionnaire utilisant comme clé une instance de classe est possible.

```
import copy

class classe_incluse:
    def __init__ (self) : self.attr = 3

class exemple_classe:
    def __init__ (self) :
        self.inclus = classe_incluse ()
```

```
        self.rnd    = 42
    def __copy__ (self):
        copie       = exemple_classe ()
        copie.rnd   = self.rnd
        return copie
    def __deepcopy__ (self,memo):
        if self in memo : return memo [self]
        copie        = copy.copy (self)
        memo [self]  = copie    # mémorise la copie de self qui est copie
        copie.inclus = copy.deepcopy (self.inclus,memo)
        return copie

nb = exemple_classe ()

nb2 = copy.deepcopy (nb)   # copie explicite à tous niveaux,
                           # utilise l'opérateur __copy__,
                           # cette ligne est équivalente à
                           # nb2 = nb.__deepcopy__()

print nb.rnd           # affiche 42
print nb2.rnd          # affiche 42
print nb.inclus.attr   # affiche 3
print nb2.inclus.attr  # affiche 3

nb.inclus.attr = 0
nb.rnd = 1

print nb.rnd           # affiche 1
print nb2.rnd          # affiche 42
print nb.inclus.attr   # affiche 0
print nb2.inclus.attr  # affiche 3  # résultat souhaité
```

On peut se demander quel est l'intérêt de la méthode `__deepcopy__` et surtout du paramètre `memo` modifié par la ligne `memo[self] = copie`. Ce détail est important lorsqu'un objet inclut un attribut égal à lui-même ou inclut un objet qui fait référence à l'objet de départ comme dans l'exemple qui suit.

```
import copy

class Objet1 :
    def __init__ (self, i) : self.i = i
    def __str__ (self) :
        return "o1 " + str (self.i) + " : " + str (self.o2.i)

class Objet2 :
    def __init__ (self, i, o) :
        self.i  = i
        self.o1 = o
        o.o2    = self
    def __str__ (self) :
        return "o2 " + str (self.i) + " : " + str (self.o1.i)

    def __deepcopy__ (self,memo) : return Objet2 (self.i, self.o1)

o1 = Objet1 (1)
o2 = Objet2 (2, o1)
print o1  # affiche o1 1 : 2
print o2  # affiche o2 2 : 1

o3   = copy.deepcopy (o2)
o3.i = 4
print o1  # affiche o1 1 : 4    --> on voudrait 2
print o2  # affiche o2 2 : 1
print o3  # affiche o2 4 : 1
```

On modifie le programme comme suit pour obtenir une recopie d'instances de classes qui pointent les unes sur vers les autres. Le paramètre `memo` sert à savoir si la copie de l'objet a déjà été effectuée ou non. Si non, on fait une copie, si oui, on retourne la copie précédemment effectuée et conservée dans `memo`.

```
import copy

class Objet1 :
    def __init__ (self, i) : self.i = i
    def __str__ (self) :
        return "o1 " + str (self.i) + " : " + str (self.o2.i)
    def __deepcopy__ (self,memo={}) :
        if self in memo : return memo [self]
        r           = Objet1 (self.i)
        memo [self] = r
        r.o2        = copy.deepcopy (self.o2, memo)
        return r

class Objet2 :
    def __init__ (self, i, o) :
        self.i  = i
        self.o1 = o
        o.o2    = self
    def __str__ (self) :
        return "o2 " + str (self.i) + " : " + str (self.o1.i)

    def __deepcopy__ (self,memo = {}) :
        if self in memo : return memo [self]
        r           = Objet2 (self.i, self.o1)
        memo [self] = r
        r.o1        = copy.deepcopy (self.o1, memo)
        return r

o1 = Objet1 (1)
o2 = Objet2 (2, o1)

print o1  # affiche o1 1 : 2
print o2  # affiche o2 2 : 1

o3   = copy.deepcopy (o2)
o3.i = 4
print o1  # affiche o1 1 : 2    --> on a 2 cette fois-ci
print o2  # affiche o2 2 : 1
print o3  # affiche o2 4 : 1
```

Ces problématiques se rencontrent souvent lorsqu'on aborde le problème de la sérialisation d'un objet qui consiste à enregistrer tout objet dans un fichier, même si cet objet inclut des références à des objets qui font référence à lui-même. C'est ce qu'on appelle des références circulaires. L'enregistrement d'un tel objet avec des références circulaires et sa relecture depuis un fichier se résolvent avec les mêmes artifices que ceux proposés ici pour la copie. L'utilisation des opérateurs `__copy__` et `__deepcopy__` est peu fréquente. Les fonctions `copy` et `deepcopy` du module `copy` suffisent dans la plupart des cas.

## 5.7 Attributs figés

Il arrive parfois qu'une classe contienne peu d'informations et soit utilisée pour créer un très grand nombre d'instances. Les paragraphes précédents ont montré que l'utilisation des attributs était assez souple. Il est toujours possible d'ajouter un attribut à n'importe quelle instance. En contrepartie, chaque instance conserve en mémoire un dictionnaire `__dict__` qui recense tous les attributs qui lui sont associés. Pour une classe susceptible d'être fréquemment instanciée comme un point dans l'espace (voir paragraphe 5.5.4), chaque instance n'a pas besoin d'avoir une liste variable d'attributs. Le langage *Python* offre la possibilité

de figer cette liste.

**syntaxe 5.33 : classe, __slots__**

```
class nom_classe (object) :
    __slots__ = "attribut_1", ..., "attribut_n"
```

nom_classe est le nom de la classe, elle doit hériter de object ou d'une classe qui en hérite elle-même (voir paragraphe 5.8). Il faut ensuite ajouter au début du corps de la classe la ligne __slots__ = "attribut_1",...,"attribut_n" où attribut_1 à attribut_n sont les noms des attributs de la classe. Aucun autre ne sera accepté.

L'exemple suivant utilise cette syntaxe pour définir un point avec seulement trois attributs _x, _y, _z.

```
class point_espace(object):
    __slots__ = "_x", "_y", "_z"

    def __init__ (self, x,y,z): self._x, self._y, self._z = x,y,z
    def __str__(self): return "(%f,%f,%f)" % (self._x, self._y, self._z)

a = point_espace (1,-2,3)
print a
```

Etant donné que la liste des attributs est figée, l'instruction a.j = 6 qui ajoute un attribut j à l'instance a déclenche une exception (voir paragraphe 6). La même erreur se déclenche si on cherche à ajouter cet attribut depuis une méthode (self.j = 6).

```
Traceback (most recent call last):
  File "cours4.py", line 15, in ?
    a.j = 6
AttributeError: 'point_espace' object has no attribute 'j'
```

L'attribut __dict__ n'existe pas non plus, par conséquent, l'expression a.__dict__ génère la même exception. La présence de l'instruction __slots__ = ... n'a aucun incidence sur les attributs statiques.

## 5.8 Héritage

L'héritage est un des grands avantages de la programmation objet. Il permet de créer une classe à partir d'une autre en ajoutant des attributs, en modifiant ou en ajoutant des méthodes. En quelque sorte, on peut modifier des méthodes d'une classe tout en conservant la possibilité d'utiliser les anciennes versions.

### 5.8.1 Exemple autour de pièces de monnaie

On désire réaliser une expérience à l'aide d'une pièce de monnaie. On effectue cent tirages successifs et on compte le nombre de fois où la face pile tombe. Le programme suivant implémente cette expérience sans utiliser la programmation objet.

```
import random  # extension interne incluant des fonctions
               # simulant des nombres aléatoires,
               # random.randint (a,b) --> retourne un nombre entier entre a et b
               # cette ligne doit être ajoutée à tous les exemples suivant
               # même si elle n'y figure plus

def cent_tirages () :
    s = 0
    for i in range (0,100) : s += random.randint (0,1)
    return s
```

```
print cent_tirages ()
```

On désire maintenant réaliser cette même expérience pour une pièce truquée pour laquelle la face pile sort avec une probabilité de $0,7$. Une solution consiste à réécrire la fonction `cent_tirages` pour la pièce truquée.

```
def cent_tirages () :
    s = 0
    for i in range (0,100) :
        t = random.randint (0,10)
        if t >= 3 : s += 1
    return s

print cent_tirages ()
```

Toutefois cette solution n'est pas satisfaisante car il faudrait réécrire cette fonction pour chaque pièce différente pour laquelle on voudrait réaliser cette expérience. Une autre solution consiste donc à passer en paramètre de la fonction `cent_tirages` une fonction qui reproduit le comportement d'une pièce, qu'elle soit normale ou truquée.

```
def piece_normale () :
    return random.randint (0,1)

def piece_truquee () :
    t = random.randint (0,10)
    if t >= 3 : return 1
    else : return 0

def cent_tirages (piece) :
    s = 0
    for i in range (0,100) : s += piece ()
    return s

print cent_tirages (piece_normale)
print cent_tirages (piece_truquee)
```

Mais cette solution possède toujours un inconvénient car les fonctions associées à chaque pièce n'acceptent aucun paramètre. Il n'est pas possible de définir une pièce qui est normale si la face *pile* vient de sortir et qui devient truquée si la face *face* vient de sortir [7]. On choisit alors de représenter une pièce normale par une classe.

```
class piece_normale :
    def tirage (self) :
        return random.randint (0,1)

    def cent_tirages (self) :
        s = 0
        for i in range (0,100) : s += self.tirage ()
        return s

p = piece_normale ()
print p.cent_tirages ()
```

7. Il faudrait pour cela créer une variable globale.

```
avant = 0
def piece_tres_truquee() :
    global avant
    if avant == 0 : avant = piece_truquee()
    else : avant = piece_normale()
    return avant
```

C'est comme si la fonction `piece_tres_truquee` avait un paramètre caché. Cette solution n'est pas conseillée car c'est le genre de détail qu'on oublie par la suite.

On peut aisément recopier et adapter ce code pour la pièce truquée.

```
class piece_normale :
    def tirage (self) :
        return random.randint (0,1)

    def cent_tirages (self) :
        s = 0
        for i in range (0,100) : s += self.tirage ()
        return s

class piece_truquee :
    def tirage (self) :
        t = random.randint (0,10)
        if t >= 3 : return 1
        else : return 0

    def cent_tirages (self) :
        s = 0
        for i in range (0,100) : s += self.tirage ()
        return s

p  = piece_normale ()
print p.cent_tirages ()
p2 = piece_truquee ()
print p2.cent_tirages ()
```

Toutefois, pour les deux classes `piece_normale` et `piece_truquee`, la méthode `cent_tirage` est exactement la même. Il serait préférable de ne pas répéter ce code puisque si nous devions modifier la première - un nombre de tirages différent par exemple -, il faudrait également modifier la seconde. La solution passe par l'héritage. On va définir la classe `piece_truquee` à partir de la classe `piece_normale` en remplaçant seulement la méthode `tirage` puisqu'elle est la seule à changer.

On indique à la classe `piece_truquee` qu'elle hérite - ou dérive - de la classe `piece_normale` en mettant `piece_normale` entre parenthèses sur la ligne de la déclaration de la classe `piece_truquee`. Comme la méthode `cent_tirages` ne change pas, elle n'a pas besoin d'apparaître dans la définition de la nouvelle classe même si cette méthode est aussi applicable à une instance de la classe `piece_truquee`.

```
class piece_normale :
    def tirage (self) :
        return random.randint (0,1)

    def cent_tirages (self) :
        s = 0
        for i in range (0,100) : s += self.tirage ()
        return s

class piece_truquee (piece_normale) :
    def tirage (self) :
        t = random.randint (0,10)
        if t >= 3 : return 1
        else : return 0

p  = piece_normale ()
print p.cent_tirages ()
p2 = piece_truquee ()
print p2.cent_tirages ()
```

Enfin, on peut définir une pièce très truquée qui devient truquée si *face* vient de sortir et qui redevient normale si *pile* vient de sortir. Cette pièce très truquée sera implémentée par la classe `piece_tres_truquee`. Elle doit contenir un attribut `avant` qui conserve la valeur du précédent tirage. Elle doit redéfinir la méthode `tirage` pour être une pièce normale ou truquée selon la valeur de l'attribut `avant`. Pour éviter de réécrire

des méthodes déjà écrites, la méthode `tirage` de la classe `piece_tres_truquee` doit appeler la méthode `tirage` de la classe `piece_truquee` ou celle de la classe `piece_normale` selon la valeur de l'attribut `avant`.

```
class piece_normale :
    def tirage (self) :
        return random.randint (0,1)

    def cent_tirages (self) :
        s = 0
        for i in range (0,100) : s += self.tirage ()
        return s

class piece_truquee (piece_normale) :
    def tirage (self) :
        t = random.randint (0,10)
        if t >= 3 : return 1
        else : return 0

class piece_tres_truquee (piece_truquee) :
    def __init__(self) :
        # création de l'attribut avant
        self.avant = 0

    def tirage (self) :
        if self.avant == 0 :
            # appel de la méthode tirage de la classe piece_truquee
            self.avant = piece_truquee.tirage (self)
        else :
            # appel de la méthode tirage de la classe piece_normale
            self.avant = piece_normale.tirage (self)
        return self.avant

p = piece_normale ()
print "normale ", p.cent_tirages ()
p2 = piece_truquee ()
print "truquee ", p2.cent_tirages ()
p3 = piece_tres_truquee ()
print "tres truquee ", p3.cent_tirages ()
```

L'héritage propose donc une manière élégante d'organiser un programme. Il rend possible la modification des classes d'un programme sans pour autant les altérer.

> **Définition 5.34 : héritage**
> On dit qu'une classe $B$ hérite d'une autre classe $A$ si la déclaration de $B$ inclut les attributs et les méthodes de la classe $A$.

La surcharge est un autre concept qui va de pair avec l'héritage. Elle consiste à redéfinir des méthodes déjà définies chez l'ancêtre. Cela permet de modifier le comportement de méthodes bien que celles-ci soient utilisées par d'autres méthodes dont la définition reste inchangée.

> **Définition 5.35 : surcharge**
> Lorsqu'une classe $B$ hérite de la classe $A$ et redéfinit une méthode de la classe $A$ portant le même nom, on dit qu'elle surcharge cette méthode. S'il n'est pas explicitement précisé qu'on fait appel à une méthode d'une classe donnée, c'est toujours la méthode surchargée qui est exécutée.

### 5.8.2 Syntaxe

**syntaxe 5.36 : classe, héritage**

```
class nom_classe (nom_ancetre) :
    # corps de la classe
    # ...
```

`nom_classe` désigne le nom d'une classe qui hérite ou dérive d'une autre classe `nom_ancetre`. Celle-ci `nom_ancetre` doit être une classe déjà définie.

L'utilisation de la fonction `help` permet de connaître tous les ancêtres d'une classe. On applique cette fonction à la classe `piece_tres_truquee` définie au paragraphe précédent.

```
help (piece_tres_truquee)
```

On obtient le résultat suivant :

```
Help on class piece_tres_truquee in module __main__:

class piece_tres_truquee(piece_truquee)
 |  Method resolution order:
 |      piece_tres_truquee
 |      piece_truquee
 |      piece_normale
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |
 |  tirage(self)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from piece_normale:
 |
 |  cent_tirages(self)
```

**Remarque 5.37 : Method resolution order**

La rubrique `Method resolution order` indique la liste des héritages successifs qui ont mené à la classe `piece_tres_truquee`. Cette rubrique indique aussi que, lorsqu'on appelle une méthode de la classe `piece_tres_truquee` si elle n'est pas redéfinie dans cette classe, le langage *Python* la cherchera chez l'ancêtre direct, ici, la classe `piece_truquee`. Si elle ne s'y trouve toujours pas, *Python* ira la chercher aux niveaux précédents jusqu'à ce qu'il la trouve.

L'attribut `__bases__` d'une classe (voir paragraphe 5.3.2) contient le (ou les ancêtres, voir paragraphe 5.8.4). Il suffit d'interroger cet attribut pour savoir si une classe hérite d'une autre comme le montre l'exemple suivant.

```
for l in piece_tres_truquee.__bases__ :
    print l   # affiche __main__.piece_truquee
print piece_normale in piece_tres_truquee.__bases__  # affiche False
print piece_truquee in piece_tres_truquee.__bases__  # affiche True
```

La fonction `issubclass` permet d'obtenir un résultat équivalent. `issubclass(A,B)` indique si la classe `A` hérite directement ou indirectement de la classe `B`. Le paragraphe 5.8.5 revient sur cette fonction.

```
print issubclass (piece_tres_truquee, piece_normale)  # affiche True
print issubclass (piece_truquee, piece_normale)       # affiche True
```

Dans les exemples précédents, `piece_normale` ne dérive d'aucune autre classe. Toutefois, le langage *Python* propose une classe d'objets dont héritent toutes les autres classes définies par le langage : c'est la classe

`object`. Les paragraphes précédents ont montré qu'elle offrait certains avantages (voir paragraphe 5.5.4 sur les propriétés ou encore paragraphe 5.7 sur les attributs non liés).

Le paragraphe précédent a montré qu'il était parfois utile d'appeler dans une méthode une autre méthode appartenant explicitement à l'ancêtre direct de cette classe ou à un de ses ancêtres. La syntaxe est la suivante.

**syntaxe 5.38 : classe, appel d'une méthode de l'ancêtre**

```
class nom_classe (nom_ancetre) :
    def nom_autre_methode (self, ...) :
        # ...
    def nom_methode (self, ...) :
        nom_ancetre.nom_methode (self, ...)
            # appel de la méthode définie chez l'ancêtre
        nom_ancetre.nom_autre_methode (self, ...)
            # appel d'une autre méthode définie chez l'ancêtre
        self.nom_autre_methode (...)
            # appel d'une méthode surchargée
```

`nom_classe` désigne le nom d'une classe, `nom_ancetre` est le nom de la classe dont `nom_classe` hérite ou dérive. `nom_methode` est une méthode surchargée qui appelle la méthode portant le même nom mais définie dans la classe `nom_ancetre` ou un de ses ancêtres. `nom_autre_methode` est une autre méthode. La méthode `nom_methode` de la classe `nom_classe` peut faire explicitement appel à une méthode définie chez l'ancêtre `nom_ancetre` même si elle est également surchargée ou faire appel à la méthode surchargée.

Ces appels sont très fréquents en ce qui concerne les constructeurs qui appellent le constructeur de l'ancêtre. Il est même conseillé de le faire à chaque fois.

```
class A :
    def __init__ (self) :
        self.x = 0
class B (A) :
    def __init__ (self) :
        A.__init__ (self)
        self.y = 0
```

**Remarque 5.39 : surcharge d'attributs**

Contrairement aux méthodes, la surcharge d'attributs n'est pas possible. Si un ancêtre possède un attribut d'identificateur `a`, les classes dérivées le possèdent aussi et ne peuvent en déclarer un autre du même nom. Cela tient au fait que quelque soit la méthode utilisée, celle de l'ancêtre ou celle d'une classe dérivée, c'est le même dictionnaire d'attributs `__dict__` qui est utilisé. En revanche, si la classe ancêtre déclare un attribut dans son constructeur, il ne faut pas oublier de l'appeler dans le constructeur de la classe fille afin que cette attribut existe pour la classe fille.

```
class ancetre :
    def __init__(self) :
        self.a = 5
    def __str__ (self) :
        return "a = " + str (self.a)

class fille (ancetre) :
    def __init__(self) :
        ancetre.__init__(self)     # cette ligne est importante
                                   # car sans elle, l'attribut a n'existe pas
        self.a += 1
    def __str__ (self) :
        s = "a = " + str (self.a)
        return s
```

```
x = ancetre ()
print x         # affiche a = 5
y = fille ()
print y         # affiche a = 6
```

### 5.8.3 Sens de l'héritage

Il n'est pas toujours évident de concevoir le sens d'un héritage. En mathématique, le carré est un rectangle dont les côtés sont égaux. A priori, une classe `carre` doit dériver d'une classe `rectangle`.

```
class rectangle :
    def __init__(self,a,b) :
        self.a,self.b = a,b
    def __str__ (self) :
        return "rectangle " + str (self.a) +  " x " +  str (self.b)

class carre (rectangle) :
    def __init__( self, a) :
        rectangle.__init__ (self, a,a)

r = rectangle (3,4)
print r  # affiche rectangle 3 x 4
c = carre (5)
print c  # affiche rectangle 5 x 5
```

Toutefois, on peut aussi considérer que la classe `carre` contient une information redondante puisqu'elle possède deux attributs qui seront toujours égaux. On peut se demander s'il n'est pas préférable que la classe `rectangle` hérite de la classe `carre`.

```
class carre :
    def __init__( self, a) :
        self.a = a
    def __str__ (self) :
        return "carre " + str (self.a)

class rectangle (carre):
    def __init__(self,a,b) :
        carre.__init__(self, a)
        self.b = b
    def __str__ (self) :
        return "rectangle " + str (self.a) +  " x " +  str (self.b)

r = rectangle (3,4)
print r  # affiche rectangle 3 x 4
c = carre (5)
print c  # affiche carre 5
```

Cette seconde version minimise l'information à mémoriser puisque la classe `carre` ne contient qu'un seul attribut et non deux comme dans l'exemple précédent. Néanmoins, il a fallu surcharger l'opérateur `__str__` afin d'afficher la nouvelle information.

Il n'y a pas de meilleur choix parmi ces deux solutions proposées. La première solution va dans le sens des propriétés croissantes, les méthodes implémentées pour les classes de bases restent vraies pour les suivantes. La seconde solution va dans le sens des attributs croissants, des méthodes implémentées pour les classes de bases doivent souvent être adaptées pour les héritiers. En contre partie, il n'y a pas d'information redondante.

Ce problème d'héritage ne se pose pas à chaque fois. Dans l'exemple du paragraphe 5.8.1 autour des pièces truquées, il n'y a pas d'ambiguïté sur le sens de l'héritage. Le programme perdrait en clarté si une pièce normale devait hériter d'une pièce truquée. Cette ambiguïté n'existe pas car les propriétés communes à

toutes les classes héritant de la classe `piece_normale` sont connues, elles sont imposées par le problème à résoudre.

Dans le cas des classes `carre` et `rectangle`, il n'est pas possible de déterminer la meilleure solution tant que leur usage ultérieur n'est pas connu. Ce problème revient également lorsqu'on définit des matrices et des vecteurs. Un vecteur est une matrice d'une seul colonne, il ne possède qu'une seule dimension au lieu de deux pour une matrice. L'exercice 13.1.10 page 267 revient sur ce dilemme.

### 5.8.4 Héritage multiple

Jusqu'à présent, tous les exemples d'héritages entre classes n'ont fait intervenir que deux classes, la classe ancêtre dont hérite la classe descendante. L'héritage multiple part du principe qu'il peut y avoir plusieurs ancêtres pour une même classe. La classe descendante hérite dans ce cas de tous les attributs et méthodes de tous ses ancêtres.

Dans l'exemple qui suit, la classe `C` hérite des classes `A` et `B`. Elle hérite donc des méthodes de `carre` et `cube`. Chacune des classes `A` et `B` contient un constructeur qui initialise l'attribut `a`. Le constructeur de la classe `C` appelle le constructeur de la classe `A` pour initialiser cet attribut.

```
class A :
    def __init__ (self) : self.a = 5
    def carre (self) : return self.a ** 2

class B :
    def __init__ (self) : self.a = 6
    def cube (self) : return self.a ** 3

class C (A,B) :
    def __init__ (self):
        A.__init__ (self)

x = C ()
print x.carre ()    # affiche 25
print x.cube ()     # affiche 125
```

Mais ces héritages multiples peuvent parfois apporter quelques ambiguïtés comme le cas où au moins deux ancêtres possèdent une méthode du même nom. Dans l'exemple qui suit, la classe `C` hérite toujours des classes `A` et `B`. Ces deux classes possèdent une méthode `calcul`. La classe `C`, qui hérite des deux, possède aussi une méthode `calcul` qui, par défaut, sera celle de la classe `A`.

```
class A :
    def __init__ (self) : self.a = 5
    def calcul (self) : return self.a ** 2

class B :
    def __init__ (self) : self.a = 6
    def calcul (self) : return self.a ** 3

class C (A,B) :
    def __init__ (self):
        A.__init__ (self)

x = C ()
print x.calcul ()  # affiche 25
```

Cette information est disponible via la fonction `help` appliquée à la classe `C`. C'est dans ce genre de situations que l'information apportée par la section `Method resolution order` est importante (voir remarque 5.37 page 112).

```
class C(A, B)
 |  Method resolution order:
```

```
 |      C
 |      A
 |      B
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |
 |  calcul(self)
```

Pour préciser que la méthode `calcul` de la classe `C` doit appeler la méthode `calcul` de la classe `B` et non `A`, il faut l'écrire explicitement en surchargeant cette méthode.

```
class A :
    def __init__ (self) : self.a = 5
    def calcul (self) : return self.a ** 2

class B :
    def __init__ (self) : self.a = 6
    def calcul (self) : return self.a ** 3

class C (A,B) :
    def __init__ (self):
        A.__init__ (self)

    def calcul (self) :
        return B.calcul (self)

x = C ()
print x.calcul ()  # affiche 125
```

**Remarque 5.40 : héritage multiple et constructeur**
L'exemple précédent est un cas particulier où il n'est pas utile d'appeler les constructeurs des deux classes dont la classe `C` hérite mais c'est un cas particulier. Le constructeur de la classe `C` devrait être ainsi :

```
class C (A,B) :
    def __init__ (self):
        A.__init__ (self)
        B.__init__ (self)
```

### 5.8.5 Fonctions `issubclass` et `isinstance`

La fonction `issubclass` permet de savoir si une classe hérite d'une autre.

> **syntaxe 5.41 : fonction `isubclass`**
>
> ```
> issubclass (B,A)
> ```
>
> Le résultat de cette fonction est vrai si la classe `B` hérite de la classe `A`, le résultat est faux dans tous les autres cas. La fonction prend comme argument des classes et non des instances de classes.

L'exemple qui suit utilise cette fonction dont le résultat est vrai même pour des classes qui n'héritent pas directement l'une de l'autre.

```
class A (object) : pass
class B (A)      : pass
class C (B)      : pass

print issubclass (A, B)     # affiche False
print issubclass (B, A)     # affiche True
```

```
print issubclass (A, C)     # affiche False
print issubclass (C, A)     # affiche True
```

Lorsqu'on souhaite appliquer la fonction à une instance de classe, il faut faire appel à l'attribut `__class__`. En reprenant les classes définies par l'exemple précédant cela donne :

```
a = A ()
b = B ()
print issubclass (a.__class__, B)     # affiche False
print issubclass (b.__class__, A)     # affiche True
print issubclass (a.__class__, A)     # affiche True
```

La fonction `isinstance` permet de savoir si une instance de classe est d'une type donné. Elle est équivalente à la fonction `issubclass` à ceci près qu'elle prend comme argument une instance et une classe. L'exemple précédent devient avec la fonction `isinstance` :

```
a = A ()
b = B ()
print isinstance (a, B)     # affiche False
print isinstance (b, A)     # affiche True
print isinstance (a, A)     # affiche True
```

L'utilisation des fonctions `issubclass` et `isinstance` n'est pas très fréquente mais elle permet par exemple d'écrire une fonction qui peut prendre en entrée des types variables.

```
def fonction_somme_list (ens) :
    r = "list "
    for e in ens : r += e
    return r

def fonction_somme_dict (ens) :
    r = "dict "
    for k,v in ens.items () : r += v
    return r

def fonction_somme (ens) :
    if   isinstance (ens, dict) : return fonction_somme_dict (ens)
    elif isinstance (ens, list) : return fonction_somme_list (ens)
    else                        : return "erreur"

li = ["un", "deux", "trois"]
di = {1:"un", 2:"deux", 3:"trois"}
tu = ("un", "deux", "trois")
print fonction_somme (li)  # affiche list undeuxtrois
print fonction_somme (di)  # affiche dict undeuxtrois
print fonction_somme (tu)  # affiche erreur
```

L'avantage est d'avoir une seule fonction capable de s'adapter à différents type de variables, y compris des types créés par un programmeur en utilisant les classes.

## 5.9 Compilation de classes

La compilation de classe fonctionne de manière similaire à celle de la compilation de fonctions (voir paragraphe 4.4.14.2, page 75). Il s'agit de définir une classe sous forme de chaîne de caractères puis d'appeler la fonction `compile` pour ajouter cette classe au programme et s'en servir. L'exemple suivant reprend deux classes décrites au paragraphe 5.8.3. Elles sont incluses dans une chaîne de caractères, compilées puis incluses au programme (fonction `exec`).

```
s = """class carre :
    def __init__( self, a) : self.a = a
    def __str__ (self)     : return "carre " + str (self.a)

class rectangle (carre):
    def __init__(self,a,b) :
        carre.__init__(self, a)
        self.b = b
    def __str__ (self) :
        return "rectangle " + str (self.a) +  " x " +  str (self.b)"""

obj = compile(s,"","exec")       # code à compiler
exec (obj)                       # classes incorporées au programme

r = rectangle (3,4)
print r  # affiche rectangle 3 x 4
c = carre (5)
print c  # affiche carre 5
```

Comme toute fonction, la fonction `compile` génère une exception lorsque la chaîne de caractères contient une erreur. Le programme qui suit essaye de compiler une chaîne de caractères confondant `self` et `seilf`.

```
# -*- coding: cp1252 -*-
"""erreur de compilation incluses dans le code inséré dans la
chaîne de caractère s"""
s = """class carre :
    def __init__( self, a) :
        seilf.a = a   # erreur de compilation
    def __str__ (self) :
        return "carre " + str (self.a) """

obj = compile(s,"variable s","exec")   # code à compiler
exec (obj)                             # classes incorporées au programme

c = carre (5)
print c  # affiche carre 5
```

L'exécution de ce programme affiche le message suivant :

```
Traceback (most recent call last):
  File "C:\temp\cours.py", line 14, in -toplevel-
    c = carre (5)
  File "variable s", line 3, in __init__
NameError: global name 'seilf' is not defined
```

Le message d'erreur est le même que pour un programme ne faisant pas appel à la fonction `compile` à ceci près que le fichier où a lieu l'erreur est `variable s` qui est le second paramètre envoyé à la fonction `compile`. Le numéro de ligne fait référence à la troisième ligne de la chaîne de caractères `s` et non à la troisième ligne du programme.

## 5.10 Constructions classiques

### 5.10.1 Héritage

Le premier exemple est classique puisqu'il reprend le programme du paragraphe 4.7.10 de la page 81 pour le réécrire avec des classes et éviter de passer des fonctions comme paramètre d'une autre fonction. La première classe définit le module des suivantes. La méthode `calcul` n'accepte qu'un seul paramètre `x` mais pourrait également prendre en compte des constantes si celles-ci sont renseignées via le constructeur de la classe. C'est l'avantage de cette solution déjà illustrée par les pièces normales et truquées.

```
class Fonction :
    def calcul (self, x) : pass
    def calcul_n_valeur (self, l) :
        res = [ self.calcul (i) for i in l ]
        return res

class Carre (Fonction) :
    def calcul (self, x) : return x*x

class Cube (Fonction) :
    def calcul (self, x) : return x*x*x

l = [0,1,2,3]
print l   # affiche [0, 1, 2, 3]

l1 = Carre ().calcul_n_valeur (l)  # l1 vaut [0, 1, 4, 9]
l2 = Cube () .calcul_n_valeur (l)  # l2 vaut [0, 1, 8, 27]
```

La version suivante mélange héritage et méthodes envoyées comme paramètre à une fonction. Il est préférable d'éviter cette construction même si elle est très utilisée par les interfaces graphiques. Elle n'est pas toujours transposable dans tous les langages de programmation tandis que le programme précédent aura un équivalent dans tous les langages objet.

```
class Fonction :
    def calcul (self, x) : pass
class Carre (Fonction) :
    def calcul (self, x) : return x*x
class Cube (Fonction) :
    def calcul (self, x) : return x*x*x

def calcul_n_valeur (l,f):
    res = [ f(i) for i in l ]
    return res

l = [0,1,2,3]
l1 = calcul_n_valeur (l, Carre ().calcul) # l1 vaut [0, 1, 4, 9]
l2 = calcul_n_valeur (l, Cube ().calcul)  # l2 vaut [0, 1, 8, 27]
```

### 5.10.2 Deux lignées d'héritages

C'est une configuration qui arrive fréquemment lorsqu'on a d'un côté différentes structures pour les mêmes données et de l'autre différents algorithmes pour le même objectif. Par exemple, une matrice peut être définie comme une liste de listes ou un dictionnaire de 2-uples [8]. La multiplication de deux matrices peut être une multiplication classique ou celle de *Strassen* [9].

La question est comment modéliser ces deux structures et ces deux multiplications sachant que les quatre paires *structure - algorithme* doivent fonctionner. On pourrait simplement créer deux classes faisant référence aux deux structures différentes et à l'intérieur de chacune d'entre elles, avoir deux méthodes de multiplication. Néanmoins, si une nouvelle structure ou un nouvel algorithme apparaît, la mise à jour peut être fastidieuse.

Il est conseillé dans ce cas d'avoir quatre classes et de définir une interface d'échanges communes. L'algorithme de multiplication ne doit pas savoir quelle structure il manipule : il doit y accéder par des méthodes. De cette manière, c'est la classe qui indique l'algorithme choisi et non une méthode. Ajouter un troisième algorithme ou une troisième structure revient à ajouter une classe : l'interface d'échange ne change pas. Le programme pourrait suivre le schéma qui suit.

8. voir exercice 11.2.1, page 217

9. L'algorithme de multiplication de deux matrices de *Strassen* est plus rapide qu'une multiplication classique pour de grandes matrices. Son coût est en $O(n^{\frac{\ln 7}{\ln 2}}) \sim O(n^{2,807})$ au lieu de $O(n^3)$.

```
class Matrice :
    def __init__ (self,lin,col,coef):
        self.lin, self.col = lin, col

    # interface d'échange
    def get_lin () : return self.lin
    def get_col () : return self.col
    def __getitem__(self,i,j): pass
    def __setitem__(self,i,j,v): pass
    def get_submat(self, i1,j1,i2,j2): pass
    def set_submat(self, i1,j1,mat): pass
    # fin de l'interface d'échange

    def trace (self) :
        t = 0
        for i in xrange (0, self.lin):
            t += self (i,i)
        return t

class MatriceList (Matrice) :
    def __init__ (self,lin,col,coef):
        Matrice.__init__ (self, \
                      lin, col, coef)
        #...

    def __getitem__ (self, i,j) : #...
    def __setitem__ (self, i,j, v) : #...
    def get_submat(self, i1,j1,i2,j2): #...
    def set_submat(self, i1,j1,mat): #...

class MatriceDict (Matrice) :
    def __init__ (self,lin,col,coef):
        Matrice.__init__ (self, \
                       lin, col, coef)
        #...

    def __getitem__ (self, i,j) : #...
    def __setitem__ (self, i,j, v) : #...
    def get_submat(self, i1,j1,i2,j2): #...
    def set_submat(self, i1,j1,mat): #...
```

```
class Produit :
    def calcul (self, mat1, mat2):
        pass

class ProduitClassique (Produit) :
    def calcul (self, mat1, mat2):
        #...
        return

class ProduitStrassen (Produit) :
    def calcul (self, mat1,mat2):
        #...
        return
```

Cette construction autorise même la multiplication de matrices de structures différentes. Très répandue, cette architecture est souvent plus coûteuse au moment de la conception car il faut bien penser l'interface d'échange mais elle l'est beaucoup moins par la suite. Il existe d'autres assemblages de classes assez fréquents, regroupés sous le terme de *Design Patterns*. Pour peu que ceux-ci soient connus de celui qui conçoit un programme, sa relecture et sa compréhension en sont facilitées si les commentaires font mention du *pattern* utilisé.

| | |
|---|---|
| `__cmp__(self,x)` | Retourne un entier égale à -1, 0, 1, chacune de ces valeurs étant associés respectivement à : `self` < `x`, `self` == `x`, `self` >= `x`. Cet opérateur est appelé par la fonction `cmp`. |
| `__str__(self)` | Convertit un objet en une chaîne de caractère qui sera affichée par la fonction `print` ou obtenu avec la fonction `str`. |
| `__contains__(self,x)` | Retourne `True` ou `False` selon que `x` appartient à `self`. Le mot-clé `in` renvoie à cet opérateur. En d'autres termes, `if x in obj` : appelle `obj.__contains__(x)`. |
| `__len__(self)` | Retourne le nombre d'élément de `self`. Cet opérateur est appelé par la fonction `len`. |
| `__abs__(self)` | Cet opérateur est appelé par la fonction `abs`. |
| `__getitem__(self,i)` | Cet opérateur est appelé lorsqu'on cherche à accéder à un élément de l'objet `self` d'indice `i` comme si c'était une liste. Si l'indice `i` est incorrect, l'exception `IndexError` doit être levée. |
| `__setitem__(self,i,v)` | Cet opérateur est appelé lorsqu'on cherche à affecter une valeur `v` à un élément de l'objet `self` d'indice `i` comme si c'était une liste ou un dictionnaire. Si l'indice `i` est incorrect, l'exception `IndexError` doit être levée. |
| `__delitem__(self,i)` | Cet opérateur est appelé lorsqu'on cherche à supprimer l'élément de l'objet `self` d'indice `i` comme si c'était une liste ou un dictionnaire. Si l'indice `i` est incorrect, l'exception `IndexError` doit être levée. |
| `__int__(self)`<br>`__float__(self)`<br>`__complex__(self)` | Ces opérateurs implémente la conversion de l'instance `self` en entier, réel ou complexe. |
| `__add__(self,x)`<br>`__div__(self,x)`<br>`__mul__(self,x)`<br>`__sub__(self,x)`<br>`__pow__(self,x)`<br>`__lshift__(self,x)`<br>`__rshift__(self,x)` | Opérateurs appelés pour les opérations +, /, ∗, −, ∗∗, <<, >> |
| `__iadd__(self,x)`<br>`__idiv__(self,x)`<br>`__imul__(self,x)`<br>`__isub__(self,x)`<br>`__ipow__(self,x)`<br>`__ilshift__(self,x)`<br>`__irshift__(self,x)` | Opérateurs appelés pour les opérations + =, / =, ∗ =, − =, ∗∗ =, <<=, >>= |

**Table 5.1 :** *Opérateurs ou fonctions spéciales les plus utilisées. La page http://docs.python.org/reference/datamodel.html#special-method-names (voir également http://docs.python.org/library/operator.html#mapping-operators-to-functions) recense toutes les fonctions spéciales qu'il est possible de redéfinir pour une classe.*

# Chapitre 6

# Exceptions

---

Le petit programme suivant déclenche une erreur parce qu'il effectue une division par zéro.

```
x = 0
y = 1.0 / x
```

Il déclenche une erreur ou ce qu'on appelle une *exception*.

```
Traceback (most recent call last):
  File "cours.py", line 2, in ?
    y = 1.0 / x
ZeroDivisionError: float division
```

Le mécanisme des exceptions permet au programme de "rattraper" les erreurs, de détecter qu'une erreur s'est produite et d'agir en conséquence afin que le programme ne s'arrête pas.

## 6.1 Principe des exceptions

### 6.1.1 Attraper toutes les erreurs

> **Définition 6.1 : exception**
> Une exception est un objet qui indique que le programme ne peut continuer son exécution.

On décide par exemple qu'on veut rattraper toutes les erreurs du programme et afficher un message d'erreur. Le programme suivant appelle la fonction `inverse` qui retourne l'inverse d'un nombre.

```
def inverse (x):
    y = 1.0 / x
    return y

a = inverse (2)
print a
b = inverse (0)
print b
```

Lorsque le paramètre `x = 0`, le programme effectue une division par zéro et déclenche une erreur qui paraît différente de la première d'après la longueur du message. C'est pourtant la même erreur : cette liste correspond en fait à ce qu'on appelle la pile d'appels [1]. Si l'erreur se produit dans une fonction elle-même appelée par une autre... la pile d'appel permet d'obtenir la liste de toutes les fonctions pour remonter jusqu'à celle où l'erreur s'est produite.

```
Traceback (most recent call last):
  File "cours.py", line 8, in ?
    b = inverse (0)
```

1. *call stack* en anglais

```
  File "cours.py", line 3, in inverse
    y = 1.0 / x
ZeroDivisionError: float division
```

Afin de rattraper l'erreur, on insère le code susceptible de produire une erreur entre les mots clés `try` et `except`.

```
def inverse (x):
    y = 1.0 / x
    return y

try :
    a = inverse (2)
    print a
    b = inverse (0)  # déclenche une exception
    print b
except :
    print "le programme a déclenché une erreur"
```

Le programme essaye d'exécuter les quatre instructions incluses entre les instructions `try` et `except`. Si une erreur se produit, le programme exécute alors les lignes qui suivent l'instruction `except`. L'erreur se produit en fait à l'intérieur de la fonction `inverse` mais celle-ci est appelée à l'intérieur d'un code "protégé" contre les erreurs. Le programme précédent affiche les deux lignes suivantes.

```
0.5
le programme a déclenché une erreur
```

Il est aussi possible d'ajouter une clause `else` qui sert de préfixe à une liste d'instructions qui ne sera exécutée que si aucune exception n'est déclenchée.

```
def inverse (x):
    y = 1.0 / x
    return y

try :
    print inverse (2)  # pas d'erreur
    print inverse (1)  # pas d'erreur non plus
except :
    print "le programme a déclenché une erreur"
else :
    print "tout s'est bien passé"
```

Ce dernier programme ne déclenche aucune exception et affiche les lignes suivantes.

```
0.5
1.0
tout s'est bien passé
```

Pour résumer, la syntaxe suivante permet d'attraper toutes les erreurs qui se produisent pendant l'exécution d'une partie du programme. Cette syntaxe permet en quelque sorte de protéger cette partie du programme

contre les erreurs.

**syntaxe 6.2 : exception**

```
try :
    # ... instructions à protéger
except :
    # ... que faire en cas d'erreur
else :
    # ... que faire lorsque aucune erreur n'est apparue
```

Toute erreur déclenchée alors que le programme exécute les instructions qui suivent le mot-clé `try` déclenche immédiatement l'exécution des lignes qui suivent le mot-clé `except`. Dans le cas contraire, le programme se poursuit avec l'exécution des lignes qui suivent le mot-clé `else`. Cette dernière partie est facultative, la clause `else` peut ou non être présente. Dans tous les cas, l'exécution du programme ne s'arrête pas.

**Remarque 6.3 : plusieurs erreurs**

Lorsqu'une section de code est protégée contre les exceptions, si elle contient plusieurs erreurs, son exécution s'arrête à la première des erreurs découvertes. Par exemple, dès la première erreur qui correspond au calcul d'une puissance non entière d'un nombre négatif, l'exécution du programme suivant est dirigée vers l'instruction qui suit le mot-clé `except`.

```
def inverse (x):
    y = 1.0 / x
    return y

try :
    print (-2.1) ** 3.1  # première erreur
    print inverse (2)
    print inverse (0)    # seconde erreur
except :
    print "le programme a déclenché une erreur"
```

### 6.1.2 Obtenir le type d'erreur, attraper un type d'exception

Parfois, plusieurs types d'erreurs peuvent être déclenchés à l'intérieur d'une portion de code protégée. Pour avoir une information sur ce type, il est possible de récupérer une variable de type `Exception`.

```
def inverse (x):
    y = 1.0 / x
    return y

try :
    print inverse (2)
    print inverse (0)
except Exception, exc:
    print "exception de type ", exc.__class__
         # affiche exception de type  exceptions.ZeroDivisionError
    print "message ", exc
         # affiche le message associé à l'exception
```

Le programme précédent récupère une exception sous la forme d'une variable appelée `exc`. Cette variable est en fait une instance d'une classe d'erreur, `exc.__class__` correspond au nom de cette classe. A l'aide de la fonction `isinstance`, il est possible d'exécuter des traitements différents selon le type d'erreur.

```
def inverse (x):
    y = 1.0 / x
    return y
```

```
try :
    print (-2.1) ** 3.1  # première erreur
    print inverse (2)
    print inverse (0)    # seconde erreur
except Exception, exc:
    if isinstance (exc, ZeroDivisionError) :
        print "division par zéro"
    else :
        print "erreur insoupçonnée : ", exc.__class__
        print "message ", exc
```

L'exemple précédent affiche le message qui suit parce que la première erreur intervient lors du calcul de $(-2.1) * *3.1$.

```
erreur insoupçonnée :  exceptions.ValueError
```

Une autre syntaxe plus simple permet d'attraper un type d'exception donné en accolant au mot-clé `except` le type de l'exception qu'on désire attraper. L'exemple précédent est équivalent au suivant mais syntaxiquement différent.

```
def inverse (x):
    y = 1.0 / x
    return y

try :
    print (-2.1) ** 3.1
    print inverse (2)
    print inverse (0)
except ZeroDivisionError:
    print "division par zéro"
except Exception, exc:
    print "erreur insoupçonnée : ", exc.__class__
    print "message ", exc
```

Cette syntaxe obéit au schéma qui suit.

**syntaxe 6.4 : exception d'un type donné**

```
try :
    # ... instructions à protéger
except type_exception_1 :
    # ... que faire en cas d'erreur de type type_exception_1
except type_exception_i :
    # ... que faire en cas d'erreur de type type_exception_i
except type_exception_n :
    # ... que faire en cas d'erreur de type type_exception_n
except :
    # ... que faire en cas d'erreur d'un type différent de tous
    #     les précédents types
else :
    # ... que faire lorsque une erreur aucune erreur n'est apparue
```

Toute erreur déclenchée alors que le programme exécute les instructions qui suivent le mot-clé `try` déclenche immédiatement l'exécution des lignes qui suivent un mot-clé `except`. Le programme compare le type d'exception aux types `type_exception_1` à `type_exception_n`. S'il existe une correspondance alors ce sont les instructions de la clause `except` associée qui seront exécutées et uniquement ces instructions. La dernière clause `except` est facultative, elle est utile lorsque aucun type de ceux prévus ne correspond à l'exception générée. La clause `else` est aussi facultative. Si la dernière clause `except` n'est pas spécifiée et que l'exception déclenchée ne correspond à aucune de celle listée plus haut, le programme s'arrête sur cette erreur à moins que celle-ci ne soit attrapée plus tard.

Le langage *Python* propose une liste d'exceptions standards (voir table 6.1, page 126). Lorsqu'une erreur ne correspond pas à l'une de ces exceptions, il est possible de créer une exception propre à un certain type d'erreur (voir paragraphe 6.2). Lorsqu'une fonction ou une méthode déclenche une exception non standard, généralement, le commentaire qui lui est associée l'indique.

| | |
|---|---|
| `Exception` | Elle n'est en principe jamais explicitement générée par les fonctions *Python* mais elle permet d'attraper toutes les exceptions car toutes dérivent de la classe `Exception`. |
| `AttributeError` | Une référence à un attribut inexistant ou une affectation a échoué. |
| `ArithmeticError` | Une opération arithmétique a échoué. |
| `FloatingPointError` | Une opération avec des nombres réels a échoué. |
| `IOError` | Une opération concernant les entrées/sorties (Input/Output) a échoué. Cette erreur survient par exemple lorsqu'on cherche à lire un fichier qui n'existe pas. |
| `ImportError` | Cette erreur survient lorsqu'on cherche à importer un module qui n'existe pas. |
| `IndentationError` | L'interpréteur ne peut interpréter une partie du programme à cause d'un problème d'indentation. Il n'est pas possible d'exécuter un programme mal indenté mais cette erreur peut se produire lors de l'utilisation de la fonction `compile` (voir le paragraphe 4.4.14.2, page 75). |
| `IndexError` | On utilise un index erroné pour accéder à un élément d'une liste, d'un dictionnaire ou de tout autre tableau. |
| `KeyError` | Une clé est utilisée pour accéder à un élément d'un dictionnaire dont elle ne fait pas partie. |
| `NameError` | On utilise une variable, une fonction, une classe qui n'existe pas. |
| `OverflowError` | Un calcul sur des entiers ou des réels dépasse les capacités de codage des entiers ou des réels. |
| `StopIteration` | Cette exception est utilisée pour signifier qu'un itérateur atteint la fin d'un ensemble, un tableau, un dictionnaire. |
| `TypeError` | Erreur de type, une fonction est appliquée sur un objet qu'elle n'est pas censée manipuler. |
| `UnicodeError` | Il faut parfois convertir des chaînes de caractères de type `unicode` en chaînes de caractères de type `str` et réciproquement. Cette exception intervient lorsque la conversion n'est pas possible (voir également le paragraphe 8.8 page 173). |
| `ValueError` | Cette exception survient lorsqu'une valeur est inappropriée pour une certaine opération, par exemple, l'obtention du logarithme d'un nombre négatif. |
| `ZeroDivisionError` | Cette exception survient pour une division par zéro. |

**Table 6.1 :** *Exceptions standard les plus couramment utilisées. Ces exceptions sont définies par le langage Python. Il n'est pas nécessaire d'inclure un module. La page http://docs.python.org/library/exceptions.html recense toutes les exceptions prédéfinies.*

### 6.1.3 Lancer une exception

Lorsqu'une fonction détecte une erreur, il lui est possible de déclencher une exception par l'intermédiaire du mot-clé `raise`. La fonction `inverse` compare `x` à `0` et déclenche l'exception `ValueError` si `x` est nul. Cette exception est attrapée plus bas.

```
def inverse (x):
    if x == 0 :
        raise ValueError
    y = 1.0 / x
    return y

try :
    print inverse (0)  # erreur
```

```
except ValueError:
    print "erreur"
```

Il est parfois utile d'associer un message à une exception afin que l'utilisateur ne soit pas perdu. Le programme qui suit est identique au précédent à ceci près qu'il associe à l'exception `ValueError` qui précise l'erreur et mentionne la fonction où elle s'est produite. Le message est ensuite intercepté plus bas.

```
def inverse (x):
    if x == 0 :
        raise ValueError ("valeur nulle interdite, fonction inverse")
    y = 1.0 / x
    return y

try :
    print inverse (0)  # erreur
except ValueError, exc:
    print "erreur, message : ", exc
```

Le déclenchement d'une exception suit la syntaxe suivante.

**syntaxe 6.5 : exception,** `raise`

`raise exception_type (message) :`

Cette instruction lance l'exception `exception_type` associée au message `message`. Le message est facultatif, lorsqu'il n'y en a pas, la syntaxe se résume à `raise exception_type`.

Et pour attraper cette exception et le message qui lui est associé, il faut utiliser la syntaxe 6.4 décrite au paragraphe précédent.

### 6.1.4 Héritage et exception

L'instruction `help(ZeroDivisionError)` retourne l'aide associée à l'exception `ZeroDivisionError`. Celle-ci indique que l'exception `ZeroDivisionError` est en fait un cas particulier de l'exception `ArithmeticError`, elle-même un cas particulier de `StandardError`.

```
class ZeroDivisionError(ArithmeticError)
 |  Second argument to a division or modulo operation was zero.
 |
 |  Method resolution order:
 |      ZeroDivisionError
 |      ArithmeticError
 |      StandardError
 |      Exception
```

Toutes les exceptions sont des cas particuliers de l'exception de type `Exception`. C'est pourquoi l'instruction `except Exception, e :` attrape toutes les exceptions. L'instruction `except ArithmeticError :` attrape toutes les erreurs de type `ArithmeticError`, ce qui inclut les erreurs de type `ZeroDivisionError`. Autrement dit, toute exception de type `ZeroDivisionError` est attrapée par les instructions suivantes :

- `except ZeroDivisionError :`
- `except ArithmeticError :`
- `except StandardError :`
- `except Exception :`

Plus précisément, chaque exception est une classe qui dérive directement ou indirectement de la classe `Exception`. L'instruction `except ArithmeticError :` par exemple attrape toutes les exceptions de type `ArithmeticError` et toutes celles qui en dérivent comme la classe `ZeroDivisionError` (voir également le paragraphe 6.2).

### 6.1.5 Instructions `try`, `except` imbriquées

Comme pour les boucles, il est possible d'imbriquer les portions protégées de code les unes dans les autres. Dans l'exemple qui suit, la première erreur est l'appel à une fonction non définie, ce qui déclenche l'exception `NameError`.

```
def inverse (x):
    y = 1.0 / x
    return y

try :
    try :
        print inverses (0)  # fonction inexistante --> exception NameError
        print inverse (0)   # division par zéro --> ZeroDivisionError
    except NameError:
        print "appel à une fonction non définie"
except ZeroDivisionError, exc:
    print "erreur ", exc
```

En revanche, dans le second exemple, les deux lignes `print inverse(0)` et `print inverses(0)` ont été permutées. La première exception déclenchée est la division par zéro. La première clause `except` n'interceptera pas cette erreur puisqu'elle n'est pas du type recherché.

```
def inverse (x):
    y = 1.0 / x
    return y

try :
    try :
        print inverse (0)   # division par zéro --> ZeroDivisionError
        print inverses (0)  # fonction inexistante --> exception NameError
    except NameError:
        print "appel à une fonction non définie"
except ZeroDivisionError, exc:
    print "erreur ", exc
```

Une autre imbrication possible est l'appel à une fonction qui inclut déjà une partie de code protégée. L'exemple suivant appelle la fonction `inverse` qui intercepte les exceptions de type `ZeroDivisionError` pour retourner une grande valeur lorsque `x = 0`. La seconde exception générée survient lors de l'appel à la fonction `inverses` qui déclenche l'exception `NameError`, elle aussi interceptée.

```
def inverse (x):
    try :
        y = 1.0 / x
    except ZeroDivisionError, exc:
        print "erreur ", exc
        if x > 0 : return 1000000000
        else : return -1000000000
    return y

try :
    print inverse (0)   # division par zéro --> ZeroDivisionError
    print inverses (0)  # fonction inexistante --> exception NameError
except NameError:
    print "appel à une fonction non définie"
```

## 6.2 Définir ses propres exceptions

### 6.2.1 Dériver une classe d'exception

Pour définir sa propre exception, il faut créer une classe qui dérive d'une classe d'exception existante (voir paragraphe 6.1.4), par exemple, la classe `Exception`. L'exemple suivant crée une exception `AucunChiffre` qui est lancée par la fonction `conversion` lorsque la chaîne de caractères qu'elle doit convertir ne contient pas que des chiffres.

```
class AucunChiffre (Exception) :
    """chaîne de caractères contenant aussi autre chose que des chiffres"""

def conversion (s) :
    """conversion d'une chaîne de caractères en entier"""
    if not s.isdigit () :
        raise AucunChiffre (s)
    return int (s)

try :
    s = "123a"
    print s, " = ", conversion (s)
except AucunChiffre, exc :
    # on affiche ici le commentaire associé à la classe d'exception
    # et le message associé
    print AucunChiffre.__doc__, " : ", exc
```

**Remarque 6.6 : exception et opérateur `__str__`**
En redéfinissant l'opérateur `__str__` d'une exception, il est possible d'afficher des messages plus explicites avec la seule instruction `print`.

```
class AucunChiffre (Exception) :
    """chaîne de caractères contenant aussi autre chose que des chiffres"""
    def __str__ (self) :
        return self.__doc__ + " " + Exception.__str__ (self)
```

### 6.2.2 Personnalisation d'une classe d'exception

Il est parfois utile qu'une exception contienne davantage d'informations qu'un simple message. L'exemple suivant reprend l'exemple du paragraphe précédent. L'exception `AucunChiffre` inclut cette fois-ci un paramètre supplémentaire contenant le nom de la fonction où l'erreur a été déclenchée.

La classe `AucunChiffre` possède dorénavant un constructeur qui doit recevoir deux paramètres : une valeur et un nom de fonction. L'exception est levée à l'aide de l'instruction `raise AucunChiffre (s,"conversion")` qui regroupe dans un T-uple les paramètres à envoyer à l'exception.

L'opérateur `__str__` a été modifié de façon à ajouter ces deux informations dans le message associé à l'exception. Ainsi, l'instruction `print exc` présente à l'avant dernière ligne de cet exemple affiche un message plus complet.

```
class AucunChiffre (Exception) :
    """chaîne de caractères contenant aussi autre chose que des chiffres"""
    def __init__(self, s, f = "") :
        Exception.__init__(self, s)
        self.s = s
        self.f = f
    def __str__(self) :
        return """exception AucunChiffre, depuis la fonction """ + self.f + \
                  " avec le paramètre " + self.s

def conversion (s) :
```

```
    """conversion d'une chaîne de caractères en entier"""
    if not s.isdigit () :
        raise AucunChiffre (s, "conversion")
    return int (s)

try :
    s = "123a"
    i = conversion (s)
    print s, " = ", i
except AucunChiffre, exc :
    print exc
    print "fonction : ", exc.f
```

Etant donné que le programme déclenche une exception dans la section de code protégée, les deux derniers affichages sont les seuls exécutés correctement. Ils produisent les deux lignes qui suivent.

```
exception AucunChiffre, depuis la fonction conversion avec le paramêtre 123a
fonction :  conversion
```

## 6.3 Exemples d'utilisation des exceptions

### 6.3.1 Les itérateurs

Les itérateurs sont des outils qui permettent de parcourir des objets qui sont des ensembles, comme une liste, un dictionnaire. Ils fonctionnent toujours de la même manière. Dans l'exemple qui suit (déjà présenté au paragraphe 5.4.2, page 94), on définit une classe contenant trois coordonnées, ainsi qu'un itérateur permettant de parcourir ces trois coordonnées. Arrivée à la troisième itération, l'exception `StopIteration` est déclenchée. Cette exception indique à une boucle `for` de s'arrêter. Une boucle `while` fait apparaître explicitement le rôle de l'exception.

```
class point_espace:

    def __init__ (self, x,y,z): self._x, self._y, self._z = x,y,z
    def __str__(self): return "(%f,%f,%f)" % (self._x, self._y, self._z)
    def __getitem__ (self, n) :
        if   n == 0 : return self._x
        elif n == 1 : return self._y
        elif n == 2 : return self._z
        else : raise IndexError

    class class_iter:
        def __init__ (self,ins):
            self._n   = 0
            self._ins = ins
        def next (self):
            if self._n <= 2:
                v = self._ins [self._n]
                self._n += 1
                return v
            else :
                raise StopIteration

    def __iter__(self):
        return point_espace.class_iter (self)

a = point_espace (1,-2,3)
for x in a:
    print x      # affiche successivement 1,-2,3

print "----- autre solution ------"'
```

```
a = point_espace (1,-2,3)
it = iter (a)
while True:
    try :
        print it.next ()  # affiche successivement 1,-2,3
    except StopIteration :
        break
```

Cet exemple montre seulement que les exceptions n'interviennent pas seulement lors d'erreurs mais font parfois partie intégrante d'un algorithme.

### 6.3.2 Exception ou valeur aberrante

Sans exception, une solution pour indiquer un cas de mauvaise utilisation d'une fonction est de retourner une valeur aberrante. Retourner -1 pour une fonction dont le résultat est nécessairement positif est une valeur aberrante. Cette convention permet de signifier à celui qui appelle la fonction que son appel n'a pu être traité correctement. Dans l'exemple qui suit, la fonction `racine_carree` retourne un couple de résultats, `True` ou `False` pour savoir si le calcul est possible, suivi du résultat qui n'a un sens que si `True` est retournée en première valeur.

```
def racine_carree(x) :
    if x < 0 : return False, 0
    else : return True, x ** 0.5
print racine_carree (-1)  # (False, 0)
print racine_carree (1)   # (True, 1.0)
```

Plutôt que de compliquer le programme avec deux résultats ou une valeur aberrante, on préfère souvent déclencher une exception, ici, `ValueError`. La plupart du temps, cette exception n'est pas déclenchée. Il est donc superflu de retourner un couple plutôt qu'une seule valeur.

```
def racine_carree(x) :
    if x < 0 : raise ValueError ("valeur négative")
    return x ** 0.5
print racine_carree (-1)  # déclenche une exception
print racine_carree (1)
```

### 6.3.3 Le piège des exceptions

Ce paragraphe évoque certains problèmes lorsqu'une exception est levée. L'exemple utilise les fichiers décrits au chapitre 8. Lorsqu'une exception est levée à l'intérieur d'une fonction, l'exécution de celle-ci s'interrompt. Si l'exception est attrapée, le programme continue sans problème ; les objets momentanément créés seront détruits par le garbage collector. Il faut pourtant faire attention dans le cas par exemple où l'exception est levée alors qu'un fichier est ouvert : il ne sera pas fermé.

```
# -*- coding: latin-1 -*-
def ajoute_resultat_division (nom, x, y) :
    """ajoute le résultat de la division x/y au fichier nom"""
    f = open (nom, "a")
    f.write (str (x) + "/" + str (y) + "= ")                                    5
    f.write ( str ((float (x)/y)) + "\n" )
    f.close ()

for i in range (0, 5) :
    try :                                                                       10
        print i-1,i-2
        ajoute_resultat_division ("essai.txt", i-1,i-2)
    except Exception, e : print "erreur avec i = ", i, "\nerreur ", e
```

Les écritures dans le fichier se font en mode ajout ("`a`"), le fichier "`essai.txt`" contiendra tout ce qui aura été écrit.

```
-1 -2
0 -1
1 0
erreur avec i =  2
erreur  float division
2 1
3 2
```

*côté affichage*

```
-1/-2= 0.5
0/-1= 0.0
2/1= 2.0
3/2= 1.5
1/0=
```

*côté fichier*

Il n'y pas de surprise du côté des affichages produit par l'instruction `print`. La différence vient de la dernière ligne du fichier qui intervient de façon surprenante en dernière position. Il faut fermer le fichier avant de sortir de la fonction quoiqu'il arrive. L'ouverture d'un fichier en mode ajout est assez fréquente lorsqu'on souhaite stocker des informations tout au long de l'exécution d'un processus.

# Chapitre 7

# Modules

Il est souvent préférable de répartir le code d'un grand programme sur plusieurs fichiers. Parmi tous ces fichiers, un seul est considéré comme fichier principal, il contient son point d'entrée, les premières instructions exécutées. Les autres fichiers sont considérés comme des modules, en quelque sorte, des annexes qui contiennent tout ce dont le fichier principal a besoin.

## 7.1 Modules et fichiers

### 7.1.1 Exemple

Cet exemple montre comment répartir un programme sur deux fichiers. Le premier est appelé *module* car il n'inclut pas le point d'entrée du programme.

> **Définition 7.1 : point d'entrée du programme**
> Le point d'entrée d'un programme est la première instruction exécutée par l'ordinateur lors de l'exécution de ce programme.

Cet exemple de module contient une fonction, une classe et une variable. Ces trois éléments peuvent être utilisés par n'importe quel fichier qui importe ce module. Le nom d'un module correspond au nom du fichier sans son extension.

**Fichier :** `module_exemple.py`

```
# -*- coding: cp1252 -*-
"""exemple de module, aide associée"""

exemple_variable = 3

def exemple_fonction () :
    """exemple de fonction"""
    return 0

class exemple_classe :
    """exemple de classe"""
    def __str__ (self) :
        return "exemple_classe"
```

**Fichier :** `exemple.py`

```
import module_exemple

c = module_exemple.exemple_classe ()
print c
print module_exemple.exemple_fonction()
help (module_exemple)
```

Pour importer un module, il suffit d'insérer l'instruction `import nom_module` avant d'utiliser une des choses qu'il définit. Ces importations sont souvent regroupées au début du programme, elles sont de cette façon mises en évidence même s'il est possible de les faire n'importe où. L'exemple qui suit importe le module défini ci-dessus. Les modules commencent le plus souvent par une chaîne de caractères comme dans l'exemple précédent, celle-ci contient l'aide associée à ce module. Elle apparaît avec l'instruction `help(module_exemple)`.

Rien ne différencie les deux fichiers `module_exemple.py` et `exemple.py` excepté le fait que le second utilise des éléments définis par le premier. Dans un programme composé de plusieurs fichiers, un seul contient le point d'entrée et tous les autres sont des modules.

La syntaxe d'appel d'un élément d'un module est identique à celle d'une classe. On peut considérer un module comme une classe avec ses méthodes et ses attributs à la seule différence qu'il ne peut y avoir qu'une seule instance d'un même module. La répétition de l'instruction `import module_exemple` n'aura aucun effet : un module n'est importé que lors de la première instruction `import nom_module` rencontré lors de l'exécution du programme.

**Remarque 7.2 : fichier** *.pyc

L'utilisation d'un module qu'on a soi-même conçu provoque l'apparition d'un fichier d'extension. `pyc`. Il correspond à la traduction du module en *bytecode* plus rapidement exploitable par l'interpréteur *Python*. Ce fichier est généré à chaque modification du module. Lorsqu'un module est importé, *Python* vérifie la date des deux fichiers d'extension `py` et `pyc`. Si le premier est plus récent, le second est recréé. Cela permet un gain de temps lorsqu'il y a un grand nombre de modules. Il faut faire attention lorsque le fichier source d'extension `py` est supprimé, il est alors encore possible de l'importer tant que sa version d'extension `pyc` est présente.

**Remarque 7.3 : recharger un module**

Le module `module_exemple` contient une variable `exemple_variable` peut être modifiée au cours de l'exécution du programme. Il est possible de revenir à sa valeur initiale en forçant *Python* à recharger le module grâce à la fonction `reload`.

```
import module_exemple
module_exemple.exemple_variable = 10
reload (module_exemple)
print module_exemple.exemple_variable   # affiche 3
```

### 7.1.2 Autres syntaxes

Il existe trois syntaxes différentes pour importer un module. La première est décrite au paragraphe précédent. Il en existe une autre qui permet d'affecter à un module un identificateur différent du nom du fichier dans lequel il est décrit. En ajoutant l'instruction `as` suivi d'un autre nom `alias`, le module sera désigné par la suite par l'identificateur `alias` comme le montre l'exemple suivant.

**Fichier :** `exemple2.py`

```
import module_exemple as alias

c = alias.exemple_classe ()
print c
print alias.exemple_fonction ()
help (alias)
```

Une autre syntaxe permet de se passer d'identificateur pour un module en utilisant le mot-clé `from`. En utilisant la syntaxe `from module import *`, tous les identificateurs (fonctions, classes, variables) sont directement accessibles sans les faire précéder d'un identificateur de module ainsi que le montre l'exemple suivant.

**Fichier :** `exemple3.py`

```
from module_exemple import *

c = exemple_classe ()
print c
print exemple_fonction ()
```

La partie `import *` permet d'importer toutes les classes, attributs ou fonctions d'un module mais il est possible d'écrire `from module_exemple import exemple_class` pour n'importer que cette classe. La fonction `exemple_fonction` ne sera pas accessible. Toutefois, cette syntaxe est déconseillée. Elle réduit la longueur de certaines fonctions ou classes puisqu'il n'est plus besoin de faire apparaître le module d'où elle proviennent

et cela ne permet plus de distinguer une classe ou une fonction définie dans ce fichier de celles définies dans un module importé.

Il existe une dernière syntaxe d'importation d'un module qui est utile quand on ne sait pas encore au moment d'écriture du programme le nom du module à importer. Celui-ci sera précisé à l'aide d'une chaîne de caractères au moyen de la fonction `__import__`.

**Fichier :** `exemple4.py`

```
alias = __import__ ("module_exemple")

c = alias.exemple_classe ()
print c
print alias.exemple_fonction ()
help (alias)
```

### 7.1.3 Nom d'un module

Le nom d'un module est défini par le nom du fichier sous lequel il est enregistré. Dans l'exemple du paragraphe précédent, le module avait pour nom de fichier `module_exemple.py`, le nom de ce module est donc `module_exemple`.

Néanmoins, ce module peut également être exécuté comme un programme normal. Si tel est le cas, son nom devient `__main__`. C'est pourquoi, les quelques lignes qui suivent apparaissent souvent. Elles ne sont exécutées que si ce fichier a pour nom `__main__`. Un seul fichier peut porter ce nom : celui qui contient le point d'entrée.

```
if __name__ == "__main__" :
    print "ce fichier est le programme principal"
```

Cette astuce est régulièrement utilisée pour tester les fonctions et classes définies dans un module. Etant donnée que cette partie n'est exécutée que si ce fichier est le programme principal, ajouter du code après le test `if __name__ == "__main__"` : n'a aucune incidence sur tout programme incluant ce fichier comme module.

### 7.1.4 Emplacement d'un module

Lorsque le module est placé dans le même répertoire que le programme qui l'utilise, l'instruction `import nom_module_san` suffit. Cette instruction suffit également si ce module est placé dans le répertoire `site – packages` présent dans le répertoire d'installation de *Python*. Si ce n'est pas le cas, il faut préciser à l'interpréteur *Python* où il doit chercher ce module :

```
import sys
sys.path.append (sys.path [0] + "/../common")
################ sys.path [0] = répertoire de ce programme
import nom_module
```

La variable `sys.path` contient les répertoires où *Python* va chercher les modules. Le premier d'entre eux est le répertoire du programme. Il suffit d'ajouter à cette liste le répertoire désiré, ici, un répertoire appelé `common` situé au même niveau que le répertoire du programme. A ce sujet, il est conseillé d'utiliser le plus souvent possible des chemins relatifs et non absolus [1]. De cette façon, on peut recopier le programme et ses modules à un autre endroit du disque dur sans altérer leur fonctionnement.

1. Depuis un répertoire courant, les chemins relatifs permettent de faire référence à d'autres répertoires sans avoir à prendre en compte leur emplacement sur le disque dur contrairement aux chemins absolus comme C:/Python25/python.exe.

### 7.1.5 Ajouter un module en cours d'exécution

De la même façon que *Python* est capable d'inclure de nouvelles portions de code en cours d'exécution [2], il est également capable d'inclure en cours d'exécution des modules dont on ne connaît pas le nom au début de l'exécution. Cela s'effectue grâce à la fonction `__import__` déjà présentée ci-dessus. Néanmoins, cette fonction ne peut pas importer un module si celui-ci est désigné par un nom de fichier incluant son répertoire. Il faut d'abord déterminer le répertoire où est le module grâce à la fonction `split` du module `os.path`. Le programme suivant illustre cette possibilité en proposant une fonction qui importe un module connaissant le nom du fichier qui le contient.

```
# -*- coding: cp1252 -*-
def import_fichier (module) :
    import os.path
    import sys
    if os.path.exists (module) :           # on teste l'existence du fichier
        folder,name = os.path.split (module)  # on obtient le répertoire du module
        if folder not in sys.path :
            sys.path.append (folder)       # on ajoute le répertoire dans la liste
                                           # des répertoires autorisés
        name = name.replace (".py", "")    # on enlève l'extension
        module = __import__ (name)         # on importe le module
        return module
    else :
        # si le fichier n'existe pas --> on lève une exception
        raise ImportError ("impossible d'importer le module " + module)

# on importe un module
mod = import_fichier (r"D:\Dupre\informatique\programme\corde.py")
# on affiche l'aide associée
help (mod)
```

### 7.1.6 Liste des modules importés

Le dictionnaire `modules` du module `sys` contient l'ensemble des modules importés. Le programme suivant affiche cette liste.

```
import sys
for m in sys.modules :
    print m, " " * (14 - len(str(m))), sys.modules [m]
```

```
os              <module 'os' from 'c:\python25\lib\os.pyc'>
os.path         <module 'ntpath' from 'c:\python25\lib\ntpath.pyc'>
re              <module 're' from 'c:\python25\lib\re.pyc'>
site            <module 'site' from 'c:\python25\lib\site.pyc'>
sys             <module 'sys' (built-in)>
types           <module 'types' from 'c:\python25\lib\types.pyc'>
...
```

Lorsque le programme stipule l'import d'un module, *Python* vérifie s'il n'est pas déjà présent dans cette liste. Dans le cas contraire, il l'importe. Chaque module n'est importé qu'une seule fois. La première instruction `import module_exemple` rencontrée introduit une nouvelle entrée dans le dictionnaire `sys.modules` :

```
module_exemple  <module 'module_exemple' from 'D:\python_cours\module_exemple.py'>
```

Le dictionnaire `sys.modules` peut être utilisé pour vérifier la présence d'un module ou lui assigner un autre identificateur. Un module est un objet qui n'autorise qu'une seule instance.

2. grâce à la fonction `exec`

```
if "module_exemple" in sys.modules :
    m = sys.modules ["module_exemple"]
    m.exemple_fonction ()
```

### 7.1.7 Attributs communs à tout module

Une fois importés, tous les modules possèdent cinq attributs qui contiennent des informations comme leur nom, le chemin du fichier correspondant, l'aide associée.

| | |
|---|---|
| `__all__` | Contient toutes les variables, fonctions, classes du module. |
| `__builtins__` | Ce dictionnaire contient toutes les fonctions et classes inhérentes au langage *Python* utilisées par le module. |
| `__doc__` | Contient l'aide associée au module. |
| `__file__` | Contient le nom du fichier qui définit le module. |
| `__name__` | Cette variable contient a priori le nom du module sauf si le module est le point d'entrée du programme auquel cas cette variable contient "`__main__`". |

Ces attributs sont accessibles si le nom du module est utilisé comme préfixe. Sans préfixe, ce sont ceux du module lui-même.

```
import os
print os.__name__, os.__doc__
if __name__ == "__main__" : print "ce fichier est le point d'entrée"
else : print "ce fichier est importé"
```

### 7.1.8 Arborescence de modules, paquetage

Lorsque le nombre de modules devient conséquent, il est parfois souhaitable de répartir tous ces fichiers dans plusieurs répertoires. Il faudrait alors inclure tous ces répertoires dans la liste `sys.path` du module `sys` ce qui paraît fastidieux. *Python* propose la définition de paquetage, ce dernier englobe tous les fichiers *Python* d'un répertoire à condition que celui contienne un fichier `__init__.py` qui peut être vide. La figure 7.1 présente une telle organisation et l'exemple suivant explicite comment importer chacun de ces fichiers sans avoir à modifier les chemins d'importation.

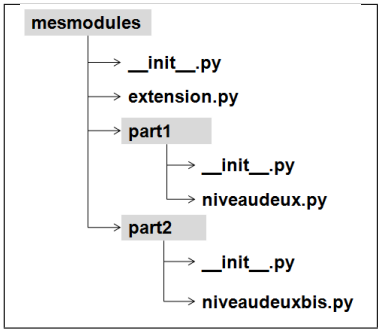



**Figure 7.1 :** *Arborescence de modules, un paquetage est défini par un ensemble de fichiers Python et d'un fichier* `__init__.py`. *Les répertoires sont grisées tandis que les fichiers apparaissent avec leur extension.*

```
import mesmodules.extension
import mesmodules.part1.niveaudeux
import mesmodules.part2.niveaudeuxbis
```

Lors de la première instruction `import mesmodules.extension`, le langage *Python* ne s'intéresse pas qu'au seul fichier `extension.py`, il exécute également le contenu du fichier `__init__.py`. C'est ici qu'il faut insérer des instructions à exécuter avant l'import de n'importe quel module du paquetage.

## 7.2 Modules internes

*Python* dispose de nombreux modules préinstallés [3]. Cette liste est trop longue pour figurer dans ce document, elle est aussi susceptible de s'allonger au fur et à mesure du développement du langage *Python*. La table 7.2 (page 138) regroupe les modules les plus utilisés.

| | |
|---|---|
| `calendar` | Gérer les calendriers, les dates (voir chapitre 8). |
| `cgi` | Utilisé dans les scripts CGI (programmation Internet) |
| `cmath` | Fonctions mathématiques complexes. |
| `codecs` | Jeux de caractères (voir paragraphe 8.8) |
| `copy` | Copies d'instances de classes. |
| `csv` | Gestion des fichiers au format CSV (utilisés par *Microsoft Excel*). |
| `datetime` | Calculs sur les dates et heures (voir chapitre 8). |
| `gc` | Gestion du garbage collector. |
| `getopt` | Lire les options des paramètres passés en arguments d'un programme *Python*. |
| `glob` | Chercher des fichiers (voir chapitre 8). |
| `hashlib` | Fonctions de cryptage. |
| `htmllib` | Lire le format HTML. |
| `math` | Fonctions mathématiques standard telles que `cos`, `sin`, `exp`, `log`... |
| `os` | Fonctions systèmes dont certaines fonctions permettant de gérer les fichiers (voir chapitre 8). |
| `os.path` | Manipulations de noms de fichiers (voir chapitre 8). |
| `pickle` | Sérialisation d'objets, la sérialisation consiste à convertir des données structurées de façon complexe en une structure linéaire facilement enregistrable dans un fichier (voir chapitre 8). |
| `profile` | Etudier le temps passé dans les fonctions d'un programme. |
| `random` | Génération de nombres aléatoires. |
| `re` | Expressions régulières (voir paragraphe 8.6). |
| `shutil` | Copie de fichiers (voir chapitre 8). |
| `sqlite3` | Accès aux fonctionnalités du gestionnaire de base de données SQLite3. |
| `string` | Manipulations des chaînes de caractères. |
| `sys` | Fonctions systèmes, fonctions liées au langage *Python* (voir chapitre 8). |
| `threading` | Utilisation de threads (voir chapitre 10). |
| `time` | Accès à l'heure, l'heure système, l'heure d'une fichier. |
| `Tkinter` | Interface graphique (voir chapitre 9). |
| `unittest` | Tests unitaires (ou comment améliorer la fiabilité d'un programme). |
| `urllib` | Pour lire le contenu de page HTML sans utiliser un navigateur. |
| `xml.dom` | Lecture du format XML. |
| `xml.sax` | Lecture du format XML. |
| `zipfile` | Lecture de fichiers ZIP (voir chapitre 8). |

**Table 7.1 :** *Liste de modules souvent utilisés. La liste exhaustive est disponible à l'adresse http://docs.python.org/modindex.html. Les moteurs de recherche sur Internet retournent des résultats assez pertinents sur des requêtes du type python + le nom du module. Les résultats sont principalement en langue anglaise.*

Certains de ces modules sont présentés dans les chapitres qui suivent. Le programme suivant par exemple utilise les modules `random` et `math` pour estimer le nombre `pi`. Pour cela, on tire aléatoirement deux nombres $(x, y)$ dans l'intervalle $[0, 1]$, si $\sqrt{x^2 + y^2} \leqslant 1$, on compte 1 sinon 0. Au final, $\hat{\pi} = \mathbb{E}\left(\mathbb{1}_{\left\{\sqrt{x^2+y^2}\leqslant 1\right\}}\right)$.

```
# -*- coding: cp1252 -*-
```

3. La page http://docs.python.org/modindex.html recense tous les modules disponibles avec *Python*.

```
import random
import math

somme = 0
nb    = 1000000
for i in range (0,nb) :
    x = random.random ()  # nombre aléatoire entre [0,1]
    y = random.random ()
    r = math.sqrt (x*x + y*y)  # racine carrée
    if r <= 1 : somme += 1

print "estimation ", 4 * float (somme) / nb
print "PI = ", math.pi
```

Le programme suivant calcule l'intégrale de Monte Carlo de la fonction $f(x) = \sqrt{x}$ qui consiste à tirer des nombres aléatoires dans l'intervalle `a,b` puis à faire la moyenne des $\sqrt{x}$ obtenu.

```
import random  # import du module random : simulation du hasard
import math    # import du module math : fonctions mathématiques

def integrale_monte_carlo (a,b,f,n) :
    somme = 0.0
    for i in range (0,n) :
        x = random.random () * (b-a) + a
        y = f(x)
        somme += f(x)
    return somme / n

def racine (x) : return math.sqrt (x)

print integrale (0,1,racine,100000)
```

Le programme suivant utilise le module `urllib` pour télécharger le contenu d'une page et l'afficher.

```
def get_page_html (url):
    import urllib
    d = urllib.urlopen(url)
    res = d.readlines()
    d.close ()
    return res

url = "http://www.lemonde.fr"
print get_page_html (url)
```

## 7.3 Modules externes

Les modules externes ne sont pas fournis avec *Python*, ils nécessitent une installation supplémentaire. Il serait impossible de couvrir tous les thèmes abordés par ces extensions. La simplicité d'utilisation du langage *Python* et son interfaçage facile avec le langage *C* contribue à sa popularité. Il permet de relier entre eux des projets conçus dans des environnements différents, dans des langages différents. Depuis les versions 2.3, 2.4 du langage *Python*, la plupart des modules externes sont faciles à installer, faciles à utiliser d'après les exemples que fournissent de plus en plus les sites Internet qui les hébergent. De plus, il s'écoulent peu de temps entre la mise à disposition d'une nouvelle version du langage *Python* et la mise à jour du module pour cette version. Le paragraphe 2.6 (page 22) donne une liste de modules utiles du point de vue d'un ingénieur généraliste.

Néanmoins, de nombreux modules ont été conçus pour un besoin spécifique et ne sont plus maintenus. Cela convient lors de l'écriture d'un programme qui remplit un besoin ponctuel. Pour une application

plus ambitieuse, il est nécessaire de vérifier quelques informations comme la date de création, celle de la dernière mise à jour, la présence d'une documentation, une prévision pour la sortie de la future version, si c'est une personne lambda qui l'a conçu ou si c'est une organisation comme celle qui fournit le module `pygoogledesktop`[4]. C'est le cas de tous les modules présentés au pragraphe 2.6.

Concernant leur installation, certains modules externes comme `SciPy` peuvent être installés à l'aide d'un fichier exécutable sous *Windows*. Il suffit d'exécuter ce fichier pour pouvoir se servir du module par la suite dans un programme *Python*. Ce mode d'installation est disponible pour la plupart des modules de taille conséquente.

D'autres modules apparaissent compressés dans un fichier. Une fois décompressés, ils incluent un fichier `setup.py`. Le langage *Python* fournit une procédure d'installation standard : il suffit d'écrire quelques lignes dans une fenêtre de commande ouverte dans le répertoire où a été décompressé le fichier contenant le module à installer.

```
c:\python25\python setup.py install
```

Sous *Linux* et *Mac OS X*, cette ligne devient :

```
python setup.py install
```

## 7.4 *Python* et les autres langages

Il est impossible d'obtenir avec le même langage à la fois une grande vitesse de développement et une grande vitesse d'exécution. La façon dont est gérée la mémoire est un élément important qui explique l'appartenance d'un langage à l'une ou l'autre des deux catégories. *Python* inclut une fonctionnalité qu'on appelle le *garbage collector*. A chaque objet créé correspond un emplacement mémoire. Celui-ci peut être créé et détruit de façon explicite ou être totalement géré par le langage, ce que fait *Python*. Il est inutile de se soucier d'un objet dont on ne sert plus, il sera détruit automatiquement et la place mémoire qu'il utilisait sera de nouveau exploitable. Cette aisance de programmation suppose que le langage sache déterminer avec exactitude quand une variable n'est plus utilisée. Ceci signifie que *Python* mémorise des informations supplémentaires sur tous les objets créés en cours d'exécution pour assurer leur référencement. *Python* est à tout moment en mesure de connaître l'ensemble des noms de variables utilisées pour désigner le même objet. Le *garbage collector* alourdit un programme de façon cachée, il le ralentit tout en facilitant la conception des programmes.

C'est pour accélérer l'exécution que le langage *Python* est parfois associé à d'autres langages. Le programme final écrit en *Python* utilise des fonctionnalités haut niveau codées dans un autre langage plus rapide.

### 7.4.1 Langage *Java*

Il est possible d'utiliser des classes *Java* en *Python*. Cette possibilité ne sera pas plus détaillée ici car le langage *Java* est également un langage interprété même s'il est nettement plus rapide. Pour cela, il faut utiliser une version de l'interpréteur *Python* codée lui-même en *Java* ce qui est le cas de la version *Java* de *Python* : *Jython*[5].

### 7.4.2 Langage *C*

Il existe deux façons de construire un programme mêlant *Python* et langage C. La première méthode est la réalisation de modules *Python* écrits en C ou C++. A moins de devoir construire un module le plus efficace possible, il est peu conseillé de le faire directement car cela implique de gérer soi-même le référencement

4. http://code.google.com/p/pythongoogledesktop/
5. http://www.jython.org/Project/

des variables créées en C ou C++ et exportées vers *Python*[6]. Il est préférable d'utiliser des outils comme la librairie *SWIG*[7], ou encore la librairie *Boost Python*[8] qui simplifie le travail d'intégration. C'est cette dernière option qui est détaillée dans ce livre.

L'utilisation d'un autre langage est en effet indispensable lorsque l'exigence de rapidité est trop grande. Les modules scientifiques tels que `scipy` ou `numpy` ne répondent pas à tous les besoins. Il faut parfois développer soi-même des outils numériques en *C++*[9] même si la partie interface est assurée par *Python*.

L'autre façon de mélanger les deux langages est l'inclusion de petites parties *C* dans un programme *Python*. Ce point a déjà été évoqué au paragraphe 2.6.7 (page 25). Cela permet une optimisation essentiellement locale et numérique mais pas l'utilisation sous *Python* d'un projet existant programmé en *C++* ou *C*.

D'une manière générale, il est préférable de scinder nettement les parties d'un même programme qui s'occupent de calculs numériques de celles qui prennent en charge l'interface graphique. Construire un module écrit en *C++* pour gérer les calculs numériques est une façon drastique et efficace d'opérer cette scission.

## 7.5 Ecrire un module en langage *C++*

Avant de décrire la librairie *Boost Python* qui facilite l'écriture d'extension en langage *C++*, il peut être utile de voir un exemple d'un module écrit dans ce langage. L'inconvénient majeur est le comptage des références : le garbage collector du langage *Python*détruit automatique toute variable qui n'est plus utilisée. Cette information est gérée automatiquement par le langage *Python*mais doit l'être manuellement au sein des extensions écrites en langage *C++*.

Un préalable est cependant nécessaire pour les utilisateurs du système d'exploitation *Windows*. Il faut que le compilateur *MinGW*[10] soit installé. C'est l'adaptation du compilateur *C++ gcc* qui est présent sur tout système d'exploitation *Linux*.

L'exemple suivant :

```
#include <Python.h>
#include <stdio.h>

/**
Une fois importe, le module definit deux fonctions :
    - exemple qui prend une chaine en argument et retourne 0
    - exemple2 qui leve une exception creee pour l'occasion
*/



static PyObject* ExempleErreur;

static PyObject* exemple (PyObject* self, PyObject* args)
{
  const char* chaine;

  if (!PyArg_ParseTuple (args, "s", &chaine))
    return NULL;

  return Py_BuildValue("i", 2);
}

static PyObject* exemple2(PyObject* self, PyObject* args)
```

6. Il faut tenir compte dans le module en *C* des opérations nécessaires au garbage collector.
7. http://www.swig.org/
8. http://www.boost.org/doc/libs/1_36_0/libs/python/doc/index.html
9. en finance par exemple
10. http://www.mingw.org/

```
{
  //const char* chaine ;

  PyErr_SetString (ExempleErreur, "Exemple de levée d'erreur") ;
  return NULL;
}

////////////////////////////////////////////////
////////////// export des fonctions ////////////////
////////////////////////////////////////////////

const char * module_name = "sample_module" ;
char buffer [100] ;

static PyMethodDef fonctions [] = {
  {"exemple",   exemple,    METH_VARARGS, "Un commentaire"},
  {"exemple2",  exemple2,   METH_VARARGS, "Une methode levant une exception"},
  {NULL, NULL, 0, NULL}
} ;

PyMODINIT_FUNC initsample_module(void)
{
  PyObject* m ;
  m = Py_InitModule (module_name, fonctions) ;


  sprintf (buffer, "%s.Exception", module_name) ;
  ExempleErreur = PyErr_NewException(buffer, NULL, NULL) ;
  Py_INCREF (ExempleErreur) ;
  PyModule_AddObject (m, "Exception", ExempleErreur) ;
}
```

Le module suivant évite l'installation manuelle d'un module écrit en *C++*. Il détecte que le module a été modifié et qu'il doit être recompilé. Toutefois, il ne gère qu'un seul fichier.

```
# -*- coding: latin-1 -*-
"""
import d'un module C (un fichier), inclut la recompilation si le module est trop vieux
"""

import os
import sys
import re

def _find_compiled_file (path, name) :
    """
    """
    exp = re.compile (name + "[.]o")
    file, rep = [], []
    for r, d, f in os.walk (path) :
        for a in f :
            if exp.search (a) :
                return r,a
    return None

def import_c_module (name, mingw = r"c:\MinGW\bin", cpp = True) :
    """
    import d'un module C
    @param      name        nom du module C (nom du fichier complet avec extension)
    @param      mingw       emplacement du compilateur mingw
    @param      cpp         extension cpp ou c ?
    @return                 objet module

    exemple :
```

```
@code
module_name = import_c_module (module_name)
#endcode
"""
if os.path.splitext (name) [1] == "" :
    if cpp :
        name += ".cpp"
        ext   = "cpp"
    else :
        name += ".c"
        ext   = "c"
else :
    ext = os.path.splitext (name) [1] [1:]

mfile   = name
mod     = os.path.splitext (mfile) [0]

for path in ["."] + sys.path :
    whole = os.path.join (path, name)
    if os.path.exists (whole) :
        break
else :
    raise ImportError ("unable to find file %s in any import path" % name)

if sys.platform == "win32" : fsec = mod + ".pyd"
else : fsec = mod + ".so"
comp = os.path.join (path, fsec)
if not os.path.exists (comp) :
    cond = True
else :
    r,f     = _find_compiled_file (path, mfile)
    if f != None :
        wholeo  = os.path.join (r,f)
        datec   = os.path.getmtime (whole)
        date    = os.path.getmtime (wholeo)
        cond    = datec > date
    else :
        cond = True

if cond :
    mfile = mod

    file =  ("""
            # -*- coding: latin-1 -*-
            from distutils.core import setup
            from distutils.core import Extension

            setup(name          = '%s',
                  version       = '0.1',
                  ext_modules   = [Extension('%s', ['%s.%s']), ],
                  url           = '',
                  author        = '',
                  author_email  = '...',
                  )
            """ % (mfile,mfile,mfile,ext)).replace ("                ", "")
    wr   = os.path.join (path, "setup.py")
    f = open (wr, "w")
    f.write (file)
    f.close ()

    env = os.getenv ("PATH")
    if mingw not in env :
        os.putenv ("PATH", env + ";" + mingw)

    cwd = os.getcwd ()
```

```
        os.chdir (path)

        py   = sys.executable.replace ("pythonw.exe", "python.exe")
        if sys.platform == "win32" :    cmd  = "%s setup.py build_ext --inplace -c mingw32" % (py)
        else :                          cmd  = "%s setup.py build_ext --inplace" % (py)
        child_stdin, stdout, child_stderr = os.popen3 (cmd)
        res = stdout.read ()
        err = child_stderr.read ()
        stdout.close ()

        os.chdir (cwd)

        warning = 'cc1plus: warning: command line option '\
                  '"-Wstrict-prototypes" is valid for C/ObjC but not for C++'
        if len (err) > 0 and (warning not in err or len (err) > len (warning) + 10) :
            message  = "\nOUTPUT:\n" + res + "\n\n"
            message += "ERR:\n" + err
            raise Exception ("unable to compile %s\n%s" % (name, message))

        mod = __import__ (mfile)
        return mod

    else :
        mfile   = mod
        mod     = __import__ (mfile)
        return mod


if __name__ == "__main__" :
    sample_module = import_c_module ("sample_module")
    print sample_module

    print sample_module.exemple("e")
    print sample_module.exemple2()
```

## 7.6 *Boost Python*

La librairie *Boost*[11] est une librairie écrite en C++ qui a l'avantage d'être portable : elle est utilisable sur les systèmes d'exploitation les plus courants. Régulièrement mise à jour, elle propose une extension de la librairie *Standard Template Library (STL)* incluant notamment les expressions régulières, les graphes, les threads, ... *Boost Python* est une partie de cette librairie qui se présente également sous forme de *template*[12].

Il est préférable d'avoir programmé en *C++* et en *Python* pour pouvoir mettre en œuvre les exemples proposés ci-dessous. La prise en main nécessite un investissement, une certaine habitude des longs messages d'erreurs issus des *template* mais une fois que cela est fait, la librairie accélère notablement l'écriture d'un module. Il n'est plus utile de se soucier de l'interfaçage avec *Python* et notamment du référencement déréférencement des objets pour le *garbage collector*[13].

**Remarque 7.4 : utilisation d'un logiciel de suivi de source**
Le nombre de fichiers est assez important, il est conseillé de conserver l'historique des modifications grâce

11. http://www.boost.org/

12. Les *template* ou *patron de classe* sont un concept de métaprogrammation. Lorsque deux classes sont presque identiques à l'exception d'un type, le langage *C++* autorise la création d'un patron unique pour ces deux classes. Le *C++* est à typage statique : il faut écrire deux classes pour créer un tableau d'entiers et un tableau de réels même si le code est le même. Les *template* permettent de factoriser le code de ces deux classes.

13. Le langage *Python* détruit automatique les objets qui ne sont plus utilisés. Cette fonctionnalité s'appuie sur un référencement de tous les objets créés. Alors qu'il est implicite en *Python*, il est explicite dans un module *Python* écrit en langage *C* à moins d'utiliser une librairie comme *Boost Python* prend en charge une partie de cette tâche (voir la page http://docs.python.org/c-api/intro.html#objects-types-and-reference-counts).

à un logiciel de suivi de source tels que *TortoiseSVN*[14]. Cela évite de perdre parfois une demi-journée de travail.

### 7.6.1 Exemple

Vous pourrez trouver à l'adresse suivante [15] l'ensemble des sources nécessaires à la réalisation d'un module avec *Boost Python*[16]. Ce projet est configuré pour *Microsoft Visual Studio C++* et définit un premier module appelé *PythonSample* qui contient quelques fonctions et une classe. Le résultat est une DLL (Dynamic Link Library) qui sera appelée par le langage *Python*. Cet exemple est conçu pour *Windows*.

Un point important est l'utilisation de la librairie `c : \python25\python25.lib` qui explique qu'un module compilé pour une version donnée de *Python* ne puisse être utilisé avec une autre version : un module compilé avec la version 2.4 ne fonctionnera pas avec la version 2.5.

L'exemple proposé contient également différentes fonctions de conversion des structures *Python* vers des containers de la *STL* ou *Standard Template Library*. Il suffit d'ouvrir le fichier `pythonsample.sln` avec *Microsoft Visual C++* pour avoir accès au code source, c'est ce que montre la figure 7.2. Le fichier `exesample.sln` définit un programme exécutable permettant de tester le module sans *Python* (voir figure 7.3). Les paragraphes qui suivent présentent quelques éléments de syntaxe et se concluent par un exercice qui consiste à ajouter une fonction au module.

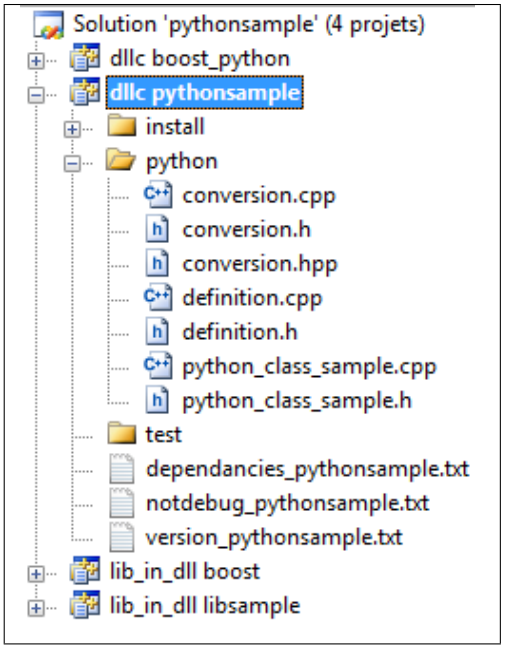



**Figure 7.2 :** *Copie d'écran du projet ouvert avec Microsoft Visual C++. Les deux projets* `dllc boost_python` *et* `lib_in_dll boost` *contiennent les sources Boost Python. Le projet* `lib_in_dll libsample` *contient le code d'une librairie qui est écrite sans tenir compte du fait que celle-ci sera utilisée sous forme de module Python. Le dernier projet* `dllc pythonsample` *ajoute la couche qui fera le pont entre la librairie* `libsample` *et le langage Python. Les fichiers* `conversion.*` *contiennent des fonctions de conversion des types Python vers des types C++ ou plus précisément de la STL (Standard Template Library). Le fichier* `definition.cpp` *contient le point d'entrée du module : la macro* `BOOST_PYTHON_MODULE` *qui déclare à Python les fonctions et classes du module.*

### 7.6.2 Les grandes lignes

Un module construit avec *Boost Python* se compose de trois éléments, une inamovible DLL `boost_python.dll`, un fichier < `module` > `.py` portant le nom du module, une DLL < `module` > `.dll`. Les deux premiers fichiers ne changent pas, le dernier contient les classes et fonctions proposées par le module. L'organisation proposée dans l'exemple du paragraphe 7.6.1 suit le schéma de la figure 7.3. L'utilisation d'un programme exécutable n'est pas indispensable. Toutefois, cela permet de tester le module sans *Python* et ainsi de faciliter sa conception.

14. http://tortoisesvn.tigris.org/
15. http://www.xavierdupre.fr/
16. L'adresse suivante fournit une aide plus complète que ce document : http://www.boost.org/libs/python/doc/tutorial/doc/html/index.html.

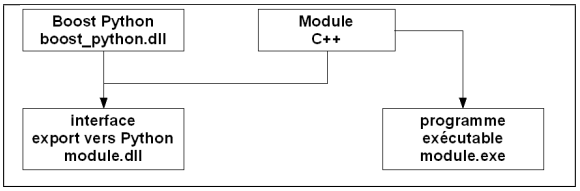



**Figure 7.3 :** *Schéma général de l'exemple introduit au paragraphe 7.6.1. Le code de la librairie est utilisé selon deux façons, à l'intérieur d'un module Python ce qui est l'objectif recherché et à l'intérieur d'un programme exécutable utilisé pour tester cette librairie sans passer par Python.*

Une fois le module prêt, il est utilisable dans tout programme *Python*. Il suffit de l'importer. Lors de cette étape, une fonction va être exécutée pour indiquer au langage *Python* l'ensemble des classes et des fonctions présentes dans le module.

#### 7.6.2.1 Initialiser le module

L'initialisation d'un module correspond à la déclaration au langage *Python* de l'ensemble des classes et des fonctions à exporter. Cette partie peut contenir des tâches qui doivent être faites la première fois que *Python* importe le module (initialisation de variables globales par exemple, chargement de données, ...). L'objectif de cette initialisation est de construire pour chaque fonction ou classe ou méthode un objet *Python* correspondant.

```
BOOST_PYTHON_MODULE(PythonSample)
{
  // étapes d'initialisation
} ;
```

#### 7.6.2.2 Ajout d'une fonction

La fonction `boost :: python :: def` permet de déclarer une fonction. On peut y préciser les arguments, des valeurs par défaut, un message d'aide retourné par l'instruction *Python* `help`.

```
int fonction (int r, const char * s, double x = 4) ;
BOOST_PYTHON_MODULE(PythonSample)
{
  boost::python::def ("fonction", fonction,
                      (boost::python::arg ("r"),
                       boost::python::arg ("s"),
                       boost::python::arg ("x") = 4),
                       "aide associée à la fonction") ;
} ;
```

Les arguments de `boost :: python :: def` sont les suivants :

1. Le premier argument est le nom de la fonction exportée en *Python*. Ce nom peut tout à fait être différent de celui de la fonction *C++*.
2. Le second argument est la fonction elle-même.
3. Le troisième argument est la liste des arguments suivis ou non d'une valeur par défaut.
4. Le dernier argument est l'aide associée à la fonction.

Côté *Python*, cela donne :

```
import PythonSample
print PythonSample.fonction (4, "second")
print PythonSample.fonction (4, "second", 5)
```

La surcharge de fonction est possible avec *Boost Python* mais elle ne sera pas détaillée ici. Le langage *Python* n'autorise pas la surcharge, la rendre possible revient à créer une fonction qui appellera telle ou telle autre fonction selon le type des paramètres qu'elle aura reçus.

#### 7.6.2.3 Ajout d'une classe

La déclaration d'une classe suit le même schéma que pour une fonction mais avec le template `boost :: python :: class_`. Lorsque le constructeur n'a pas d'argument, la déclaration suit le schéma suivant :

```
boost::python::class_<PythonClassSample> obj (
          "ClassSample",
          "help on PythonClassSample") ) ;
```

Le template `class_` est instancié sur la classe à exporter, ici `PythonClassSample`. On crée un objet `obj` de type `boost :: python :: class_ < PythonClassSample >`. Son constructeur prend comme arguments :

1. Le nom de la classe en *Python*.
2. L'aide qui lui est associée. Ce second paramètre est facultatif mais il est conseillé de le renseigner.

Au cas où la classe aurait un constructeur avec des paramètres, il faudrait ajouter le code suivant :

```
boost::python::class_<PythonClassSample> obj (
        "ClassSample",
        "help on PythonClassSample",
        boost::python::init<int,const char*> (                    // ajout
             (boost::python::arg ("a"),                           // ajout
              boost::python::arg ("s") = "default value for s"),  // ajout
              "help on PythonClassSample constructor" )           // ajout
        ) ;
```

C'est-à-dire un autre template `boost :: python :: init < ... >` dont les arguments sont les types des paramètres du constructeur, c'est-à-dire entier et chaîne de caractères dans l'exemple précédent. La suite est identique à la déclaration d'une fonction. Il est également possible de définir un héritage de la manière qui suit :

```
boost::python::class_<PythonClassSample,
                      boost::python::bases<ClassBase> > obj (
                           "ClassSample",
                           "help on PythonClassSample") ;
```

Le template `class_` peut recevoir un autre argument qui est `boost :: python :: bases < ClassBase >` avec `ClassBase` désignant la classe mère de la classe déclarée. Cette dernière doit avoir été déclarée au préalable avec cette même syntaxe `boost :: python :: class_`.

```
boost::python::class_<PythonClassSample,
           boost::python::bases<ClassBase> > obj (
         // ...
```

**Remarque 7.5 : template *C++***
Lorsqu'un template est un argument d'un autre template, il faut insérer un espace entre deux symboles < ou > consécutifs pour éviter la confusion avec les opérateurs << ou >> qui existent en *C++*.

#### 7.6.2.4 Ajout d'une méthode

La déclaration d'une classe passe par la déclaration d'une instance du template `class_`. Dans les exemples utilisés, cette instance porte le nom `obj`. C'est cette instance qui va permettre de déclarer une méthode exactement de la même manière qu'une fonction :

```
obj.def (   "Random",
            &PythonClassSample::Random,
            (boost::python::arg ("pos")),
            "help on the method") ;
```

Cette même méthode `def` permet également de surcharge le constructeur au cas où la classe pourrait être initialisée de différentes manières. Cela donne :

```
boost::python::class_<MyVector> obj ("Vector", "help on vector",
        boost::python::init<int> (
        (PY_ARG ("n")),
        "crée un vecteur de dimension n")) ;
// ajout d'un constructeur sans paramètre
obj.def (boost::python::init<MyVector> ()) ;
```

La classe *Python* `Vector` peut être créée avec un ou aucun paramètre :

```
v  = PythonSample.Vector ()
v2 = PythonSample.Vector (10)
```

#### 7.6.2.5 Ajout d'un attribut

La déclaration d'un attribut n'est pas très différente avec la méthode `def_readwrite` :

```
obj.def_readwrite ("a", &PythonClassSample::_a, "retourne un accès à a") ;
```

Le premier paramètre sera son nom sous *Python*, le second le paramètre lui-même, le troisième l'aide associée. La méthode `def_readwrite` permet d'exporter un attribut de sorte qu'il soit lisible et modifiable.

### 7.6.3 Exemple concret : ajout d'une fonction

L'objectif est d'ajouter une fonction au module. La première étape consiste à décompresser le fichier téléchargé [17], à compiler l'ensemble des fichiers sous *Microsoft Visual C++* en version *release* et *debug*. L'exécution du programme `PythonSample/test/test.py` doit s'effectuer sans problème. Ajouter une fonction revient ensuite à suivre les étapes suivantes :

1. Déclarer la fonction dans le fichier `libSample/mafonction.h` qu'on crée.
2. Implémenter la fonction dans le fichier `libSample/mafonction.cpp` qu'on crée également.

La partie purement *C++* est terminée [18], il faut maintenant définir la transition *C++*-*Python* grâce aux étapes suivantes :

1. On déclare une fonction dans un fichier `PythonSample/mafonction.h` qu'on crée. Elle peut par exemple porter le même nom que la précédente mais préfixé par `python_`.
2. On peut passer à l'implémentation de la fonction *Python* dans un fichier `PythonSample/mafonction.cpp` qu'on crée aussi. Le code est sensiblement le même pour chaque fonction ajoutée : il faut convertir les paramètres depuis des structures *Python* en type *C++*, appeler la fonction *C++* puis effectuer la conversion inverse. On pourra s'aider des fonctions contenues dans le fichier `PythonSample/python/conversion.h`.
3. Il suffit de déclarer la fonction au sein de la macro `BOOST_PYTHON_MODULE` dans le fichier `PythonSample/python/definitio`

Il ne reste plus qu'à tester la fonction nouvellement incluse depuis le programme `PythonSample/test/test.py`. Le contenu de chaque fichier modifié lors de ces étapes est présenté par la figure 7.4 page 152.

### 7.6.4 Garbage collector et pointeur

Le langage *Python* gère lui-même la destruction des objets au travers d'un *garbage collector*. Cela signifie qu'un objet créé en *Python* provenant d'un module défini dans module en *C++* sera lui aussi détruit par

17. depuis l'adresse http://www.xavierdupre.fr/
18. C'est facultatif mais il est également conseillé d'écrire d'écrire un code pour tester cette fonction dans le fichier `exeSample/source/main.cpp`. Cela réduit le risque d'erreurs d'exécution dans cette partie.

*Python*. A partir du moment où le destructeur d'un objet libère les pointeurs qu'il a alloué, la mémoire ne sera pas corrompue. Ce schéma ne permet pas de prévoir quand l'objet sera détruit.

Pour parer aux problèmes éventuels qui pourraient survenir notamment lorsqu'un objet contient des pointeurs sur des données détenues par un autre objet, il est possible de spécifier à *Boost Python* qu'un résultat est une référence : son identificateur *Python* peut être détruit mais pas l'objet qu'il désigne. C'est l'objet du template `boost :: python :: return_internal_reference <> ()`. Il est utilisé pour spécifier qu'une fonction retourne un résultat qui ne doit pas être détruit.

Par exemple, on utilise cette fonctionnalité pour définir un opérateur d'affectation comme l'opérateur + =. La fonction suivante a la même signature *C++* que l'opérateur + =.

```
class PythonClassSample
{
    ...
    PythonClassSample & __iadd__ (const PythonClassSample &a) ;
    ...
} ;
```

Son résultat est une référence sur un objet existant. Si aucune mention spécifique n'est précisée lors de l'export de la méthode (avec `def`), lors d'un appel à l'opérateur + =, *Python* va considérer que le résultat est un objet différent alors qu'il s'agit de deux identificateurs *Python* faisant référence au même objet *C++*. C'est pour cela qu'on ajoute l'argument suivant à la méthode `def` :

```
    x.def ("__iadd__",
           &PythonClassSample::__iadd__,
           boost::python::return_internal_reference<>(),
           "addition") ;
```

De cette façon, il y aura bien deux identificateurs *Python* faisant référence au même objet *C++*. C'est aussi cette syntaxe qui permet de déclarer une fonction retournant un pointeur sur un objet *C++*.

### 7.6.5 Utilisation, installation

Le module terminé se compose de trois fichiers :

1. la DLL compilée avec le code C++ du module,
2. un fichier portant le même nom que la DLL mais d'extension .py : `PythonSample.py`,
3. un fichier `__init__.py`.

Voici ce fichier pour l'exemple présenté au paragraphe 7.6.1. La fonction importante de ce fichier est `load_dynamic` qui charge la DLL en mémoire au cas où elle n'aurait jamais été chargée.

```
import sys
if "PythonSample" not in sys.modules :
    PythonSample = imp.load_dynamic ('PythonSample', PythonSample.dll)
    sys.modules ["PythonSample"]  = PythonSample
```

### 7.6.6 Détails sur l'exemple du paragraphe 7.6.1

Certains fichiers exposés ici ont été simplifiés par rapport à l'exemple téléchargeable. Celui-ci a été configuré de manière à produire deux DLL, une autre pour la version *release* `PythonSample.dll` et une pour la version *debug* `PythonSampled.dll`. Ceci permet d'utiliser l'une ou l'autre des versions dans le même programme. La version *debug* facilite la correction des erreurs qui pourraient survenir lors de l'utilisation du module sous *Python*(voir le paragraphe 7.6.7). Il suffit alors d'écrire :

```
import PythonSampled as PythonSample
# au lieu de import PythonSample
```

De cette façon, passer d'une DLL à l'autre est simple et permet de débugger, tâche qu'introduit le paragraphe suivant. Si le programme *Python* s'étale sur plusieurs fichiers, il est plus pratique d'ajouter au début de chacun fichier qui utilise le module *C++* :

```
import sys
if "PythonSampled" in sys.modules : PythonSample = sys.modules ["PythonSampled"]
else : import PythonSample
```

### 7.6.7 Debuggage

Il ne serait pas pensable de développer des outils sans pouvoir utiliser un débuggeur. Dans l'exemple présenté au paragraphe 7.6.1, le fichier `testd.py` n'utilise pas le module compilé dans sa version *release* mais dans sa version ***debug***. L'exécution est plus lente mais permet de faire appel au débuggeur pour corriger une éventuelle erreur d'exécution.

Une manière de débugger consiste à momentanément arrêter l'exécution du programme *Python* (à l'aide de la fonction `raw_input` par exemple), le temps d'attacher le débuggeur de *Microsoft Visual C++* au processus dont le nom est *python.exe* ou *pythonw.exe*. Il ne reste plus qu'à placer des pointeurs d'arrêt puis à continuer l'exécution du programme *Python* jusqu'à ce que ces pointeurs d'arrêt soient atteints.

Une seconde méthode consiste à volontairement insérer dans le code *C++* du module l'instruction `__debugbreak`(). Lorsque le module est exécuté via un programme *Python* et qu'il arrive sur cette ligne, il vous demande l'autorisation de continuer l'exécution via un débuggeur. Il faudra penser à enlever la ligne une fois le module corrigé.

Il reste toutefois une dernière façon plus classique de débugger qui est l'ajout dans le code d'instructions écrivant des messages dans un fichier texte. On écrit des *logs* ou des *traces*. De cette façon, lorsque le programme provoque une erreur, on connaît la séquence de messages générés par le programme. S'ils sont suffisamment explicites, cela permet de corriger l'erreur. Sinon, l'ajout de messages supplémentaires puis une nouvelle exécution du programme permettront d'être plus précis quant à la cause de l'erreur.

Outre ces trois méthodes de débuggage, il est possible qu'un module doivent retourner un code d'erreur ou générer une exception qui soit attrapée par l'interpréteur *Python* puis afficher. C'est l'objet des lignes qui suivent :

```
if (<condition>)
    throw std::runtime_error ("message d'erreur") ;
```

Toute exception dérivant de `runtime_error` sera interceptée par l'interpréteur *Python*. La librairie *Boost Python* génère également des exceptions notamment lorsque les paramètres envoyés à une fonction du module ne correspondent pas avec sa signature.

En terme d'architecture, il est conseillé de bien scinder la partie utilisant *Boost Python* des algorithmes. *Boost Python* n'est qu'une passerelle vers *Python*. Il pourrait être nécessaire de construire une autre passerelle vers d'autres langages tel que le *Visual Basic pour Applications (VBA)*. De même, outre la passerelle vers *Python*, il est utile de construire un programme exécutable permettant de tester les algorithmes écrits en *C++* avant de les exporter. Cela réduit beaucoup le temps de débuggage qui peut alors gérer séparément la partie *C++* de la partie interfaçage entre *Python* et *C++*.

### 7.6.8 Pour aller plus loin

La librairie *Boost Python*[19] permet de faire beaucoup plus de choses que ce qui est présenté dans ces paragraphes. Ceux-ci contiennent néanmoins l'essentiel pour exporter vers *Python* les objets les plus simples. Il est également possible de :

– surcharger une méthode *C++* dans un programme *Python*,

19. http://www.boost.org/libs/python/doc/

– exécuter un programme écrit en *Python* depuis un programme *C++*.

Il faut parfois une certaine dose de volonté et de ténacité pour parvenir à maîtriser un outil tel que *Boost Python*, surtout lorsqu'une erreur se déclare dans le code de librairie après une modification en dehors de cette même librairie.

Fichier libSample/mafonction.h

```
#ifndef LIB_MAFONCTION_H
#define LIB_MAFONCTION_H

#include <vector>

void somme_vecteur (
  const std::vector<double> &v1,
  const std::vector<double> &v2,
  std::vector<double> &res) ;

#endif
```

Fichier libSample/mafonction.cpp

```
#include "mafonction.h"
void somme_vecteur (
  const std::vector<double> &v1,
  const std::vector<double> &v2,
  std::vector<double> &res)
{
  if (v1.size () != v2.size ())
    throw std::runtime_error (
         "dimensions différentes") ;
  res.resize (v1.size ()) ;
  std::vector<double>::
          const_iterator it1,it2 ;
  std::vector<double>::iterator it3 ;
  for (it1 = v1.begin (),
       it2 = v2.begin (),
       it3 = res.begin () ;
       it1 != v1.end () ;
       ++it1, ++it2, ++it3)
       *it3 = *it1 + *it2 ;
}
```

Fichier PythonSample/mafonction.h

```
#ifndef PYTHON_MAFONCTION_H
#define PYTHON_MAFONCTION_H

#include "python/definition.h"

boost::python::list
     python_somme_vecteur (
        boost::python::list v1,
        boost::python::list v2) ;

#endif
```

Fichier PythonSample/mafonction.cpp

```
#include "python/definition.h"
#include "mafonction.h"
#include "../libsample/mafonction.h"
#include "python/conversion.h"
#include "python/conversion.hpp"
boost::python::list
     python_somme_vecteur (
        boost::python::list v1,
        boost::python::list v2)
{
  std::vector<double> cv1,cv2,cres ;
  PythonConvert (v1, cv1) ;
  PythonConvert (v2, cv2) ;
  somme_vecteur (cv1, cv2, cres) ;
  boost::python::list res ;
  PythonConvert (cres, res) ;
  return res ;
}
```

Fichier PythonSample/python/main.cpp

```
// ...
#include "../mafonction.h"
BOOST_PYTHON_MODULE(PythonSample)
{
  // ...
  def ("somme_vecteur",
        &python_somme_vecteur,
        (boost::python::arg ("v1"),
         boost::python::arg ("v2")),
 "retourne la somme de deux vecteurs");
  // ...
} ;
```

Fichier PythonSample/test/test.py

```
import PythonSample as PS
v1 = [1.0, 2.0]
v2 = [6.5, 7.8]
v3 = PS.somme_vecteur (v1, v2)
```

**Figure 7.4 :** *Six fichiers modifiés pour ajouter une fonction dans un module Python écrit en C++ à partir de l'exemple accessible depuis l'adresse http://www.xavierdupre.fr/ .*

# Chapitre 8

# Fichiers, expressions régulières, dates

---

Lorsqu'un programme termine son exécution, toutes les informations stockées dans des variables sont perdues. Le seul moyen de les conserver est de les enregistrer dans un fichier sur disque dur. A l'intérieur de celui-ci, ces informations peuvent apparaître sous un format texte qui est lisible par n'importe quel éditeur de texte, dans un format compressé, ou sous un autre format connu par le concepteur du programme. On appelle ce dernier type un format binaire, il est adéquat lorsque les données à conserver sont très nombreuses ou lorsqu'on désire que celles-ci ne puissent pas être lues par une autre application que le programme lui-même. En d'autres termes, le format binaire est illisible excepté pour celui qui l'a conçu.

Ce chapitre abordera pour commencer les formats texte, binaire et compressé (*zip*) directement manipulable depuis *Python*. Les manipulations de fichiers suivront pour terminer sur les expressions régulières qui sont très utilisées pour effectuer des recherches textuelles. A l'issue de ce chapitre, on peut envisager la recherche à l'intérieur de tous les documents textes présents sur l'ordinateur, de dates particulières, de tous les numéros de téléphones commençant par 06... En utilisant des modules tels que `pypdf` [1] ou encore `win32com` [2], il serait possible d'étendre cette fonctionnalité aux fichiers de type *pdf* et aux fichiers *Microsoft Word, Excel*.

## 8.1 Format texte

Les fichiers texte sont les plus simples : ce sont des suites de caractères. Le format *HTML* et *XML* font partie de cette catégorie. Ils servent autant à conserver des informations qu'à en échanger comme par exemple transmettre une matrice à *Microsoft Excel*.

Ce format, même s'il est simple, implique une certaine organisation dans la façon de conserver les données afin de pouvoir les récupérer. Le cas le plus fréquent est l'enregistrement d'une matrice : on choisira d'écrire les nombres les uns à la suite des autres en choisissant un séparateur de colonnes et un séparateur de lignes. Ce point sera abordé à la fin de cette section.

### 8.1.1 Ecriture

La première étape est l'écriture. Les informations sont toujours écrites sous forme de chaînes de caractères et toujours ajoutées à la fin du fichier qui s'allonge jusqu'à ce que toutes les informations y soient écrites. L'écriture s'effectue toujours selon le même schéma.

1. création ou ouverture du fichier,
2. écriture,
3. fermeture.

Le fichier dans lequel seront écrites les informations est créé s'il n'existe pas ou nettoyé s'il existe déjà. La fermeture permet à d'autres programmes de lire ce que vous avez placé dans ce fichier. Sans cette dernière étape, il sera impossible d'y accéder à nouveau pour le lire ou y écrire à nouveau. A l'intérieur d'un programme informatique, écrire dans un fichier suit toujours le même schéma :

1. http://pybrary.net/pyPdf/
2. http://python.net/crew/mhammond/win32/Downloads.html

```
f = open ("nom-fichier", "w")    # ouverture

f.write ( s )    # écriture de la chaîne de caractères  s
f.write ( s2 )   # écriture de la chaîne de caractères  s2
...

f.close ()  # fermeture
```

Les étapes d'ouverture et de fermeture sont toujours présentes en ce qui concerne les fichiers. Il s'agit d'indiquer au système d'exploitation que le programme souhaite accéder à un fichier et interdire à tout autre programme l'accès à ce fichier. Un autre programme qui souhaiterait créer un fichier du même nom ne le pourrait pas tant que l'étape de fermeture n'est pas exécutée. En revanche, il pourrait tout à fait le lire car la lecture ne perturbe pas l'écriture.

**Remarque 8.1 : fichiers ouverts et non fermés**
Lorsque que le programme se termine, même s'il reste des fichiers "ouverts" pour lesquels la méthode `close` n'a pas été appelée, ils seront automatiquement fermés.

Certains caractères sont fort utiles lors de l'écriture de fichiers texte afin d'organiser les données. Le symbole ; est très utilisé comme séparateur de colonnes pour une matrice, on utilise également le passage à la ligne ou la tabulation. Comme ce ne sont pas des caractères "visibles", ils ont des codes :

- `\n` passage à la ligne
- `\t` insertion d'une tabulation, indique un passage à la colonne suivante dans le logiciel *Microsoft Excel*

Il existe peu de manières différentes de conserver une matrice dans un fichier, le programme ressemble dans presque tous les cas à celui qui suit :

```
mat =  ... # matrice de type liste de listes
f = open ("mat.txt", "w")
for i in range (0,len (mat)) :
    for j in range (0, len (mat [i])) :
        f.write ( str (mat [i][j]) + "\t")
    f.write ("\n")
f.close ()
```

**Remarque 8.2 : fonction `open`**
La fonction `open` accepte deux paramètres, le premier est le nom du fichier, le second définit le mode d'ouverture : "`w`" pour écrire (**w**rite), "a" pour écrire et ajouter (**a**ppend), "r" pour lire (**r**ead). Ceci signifie que la fonction `open` sert à ouvrir un fichier quelque soit l'utilisation qu'on en fait.

**Remarque 8.3 : autre syntaxe avec `print`**
Il existe une autre syntaxe qui permet de remplacer la méthode `write`. Elle utilise la même syntaxe que celle de l'instruction `print` et fonctionne de la même manière à ceci près qu'elle ajoute `>> f`, entre le mot-clé `print` et le texte à écrire dans le fichier `f`.

```
mat = ... # matrice de type liste de listes
f = open ("mat.txt", "w")
for i in range (0,len (mat)) :
    for j in range (0, len (mat [i])) :
        print >> f, str (mat [i][j]), "\t",  # ligne changée
    print >> f                               # ligne changée
f.close ()
```

**Remarque 8.4 : écriture différée**
A la première écriture dans un fichier, lorsqu'on vérifie immédiatement après la taille du fichier créée, elle est souvent nulle. L'écriture dans un fichier n'est pas immédiate, le langage *Python* attend d'avoir reçu

beaucoup d'informations avant de les écrire physiquement sur le disque dur. Les informations sont placées dans un tampon ou *buffer*. Lorsque le tampon est plein, il est écrit sur disque dur. Pour éviter ce délai, il faut soit fermer puis réouvrir le fichier soit appeler la méthode `fflush` qui ne prend aucun paramètre.

### 8.1.2 Ecriture en mode "ajout"

Lorsqu'on écrit des informations dans un fichier, deux cas se présentent. Le premier consiste à ne pas tenir compte du précédent contenu de ce fichier lors de son ouverture pour écriture et à l'écraser. C'est le cas traité par le précédent paragraphe. Le second cas consiste à ajouter toute nouvelle information à celles déjà présentes lors de l'ouverture du fichier. Ce second cas est presque identique au suivant hormis la première ligne qui change :

```
f = open ("nom-fichier", "a")    # ouverture en mode ajout, mode "a"
...
```

Pour comprendre la différence entre ces deux modes d'ouverture, voici un premier programme qui n'utilise pas le mode ajout :

```
f = open ("essai.txt", "w")
f.write (" premiere fois ")
f.close ()
f = open ("essai.txt", "w")
f.write (" seconde fois ")
f.close ()
```

Ce programme crée un fichier "`essai.txt`" qui ne contient que les informations écrites lors de la seconde phase d'écriture, soit `seconde fois`. Le suivant utilise le mode ajout lors de la seconde ouverture.

```
f = open ("essai.txt", "w")
f.write (" premiere fois ")
f.close ()
f = open ("essai.txt", "a")
f.write (" seconde fois ")
f.close ()
```

Au final, le fichier "`essai.txt`", même s'il existait avant l'exécution de ce programme, est effacé puis rempli avec l'information `premiere fois`. Lors de la seconde ouverture, en mode ajout, une seconde chaîne de caractères est ajoutée. le fichier "`essai.txt`", après l'exécution du programme contient donc le message : `premiere fois seconde fois`.

**Remarque 8.5 : fichier de traces**
Un des moyens pour comprendre ou suivre l'évolution d'un programme est d'écrire des informations dans un fichier ouvert en mode ajout qui est ouvert et fermé sans cesse. Ce sont des fichiers de *traces* ou de *log*. Ils sont souvent utilisés pour vérifier des calculs complexes. Ils permettent par exemple de comparer deux versions différentes d'un programme pour trouver à quel endroit ils commencent à diverger.

### 8.1.3 Lecture

La lecture d'un fichier permet de retrouver les informations stockées grâce à une étape préalable d'écriture. Elle se déroule selon le même principe, à savoir :

1. ouverture du fichier en mode lecture,
2. lecture,
3. fermeture.

Une différence apparaît cependant lors de la lecture d'un fichier : celle-ci s'effectue ligne par ligne alors que l'écriture ne suit pas forcément un découpage en ligne. Les instructions à écrire pour lire un fichier diffèrent rarement du schéma qui suit où seule la ligne indiquée par $(*)$ change en fonction ce qu'il faut faire avec les informations lues.

```
f = open ("essai.txt", "r")  # ouverture du fichier en mode lecture
for ligne in f :             # pour toutes les lignes du fichier
    print ligne              # on affiche la ligne (*)
f.close ()                   # on ferme le fichier
```

Pour des fichiers qui ne sont pas trop gros ($< 100000$ lignes), il est possible d'utiliser la méthode `readlines` qui récupère toutes les lignes d'un fichier texte en une seule fois. Le programme suivant donne le même résultat que le précédent.

```
f = open ("essai.txt", "r") # ouverture du fichier en mode lecture
l = f.readlines ()          # lecture de toutes les lignes, placées dans une liste
f.close ()                  # fermeture du fichier

for s in l : print s        # on affiche les lignes à l'écran (*)
```

**Remarque 8.6 : code de fin de ligne**
Lorsque le programme précédent lit une ligne dans un fichier, le résultat lu inclut le ou les caractères (`\n` `\r`) qui marquent la fin d'une ligne. C'est pour cela que la lecture est parfois suivie d'une étape de nettoyage.

```
f = open ("essai.txt", "r") # ouverture du fichier en mode lecture
l = f.readlines ()          # lecture de toutes les lignes, placées dans une liste
f.close ()                  # fermeture du fichier

     # contiendra la liste des lignes nettoyées
l_net = [ s.strip ("\n\r") for s in l ]
```

Les informations peuvent être structurées de façon plus élaborée dans un fichier texte, c'est le cas des formats *HTML* ou *XML*. Pour ce type de format plus complexe, il est déconseillé de concevoir soi-même un programme capable de les lire, il existe presque toujours un module qui permette de le faire. C'est le cas du module `HTMLParser` pour le format HTML ou `xml.sax` pour le format XML. De plus, les modules sont régulièrement mis à jour et suivent l'évolution des formats qu'ils décryptent.

Un fichier texte est le moyen le plus simple d'échanger des matrices avec un tableur et il n'est pas besoin de modules dans ce cas. Lorsqu'on enregistre une feuille de calcul sous format texte, le fichier obtenu est organisé en colonnes : sur une même ligne, les informations sont disposées en colonne délimitées par un séparateur qui est souvent une tabulation (`\t`) ou un point virgule comme dans l'exemple suivant :

```
nom  ; prénom ; livre
Hugo  ; Victor  ; Les misérables
Kessel ; Joseph  ; Le lion
Woolf ; Virginia  ; Mrs Dalloway
Calvino ; Italo  ; Le baron perché
```

Pour lire ce fichier, il est nécessaire de scinder chaque ligne en une liste de chaînes de caractères, on utilise pour cela la méthode `split` des chaînes de caractères.

```
mat = []                     # création d'une liste vide,
f = open ("essai.txt", "r")  # ouverture du fichier en mode lecture
for li in f :                # pour toutes les lignes du fichier
    s = li.strip ("\n\r")    # on enlève les caractères de fin de ligne
    l = s.split (";")        # on découpe en colonnes
    mat.append (l)           # on ajoute la ligne à la matrice
f.close ()                   # fermeture du fichier
```

Ce format de fichier texte est appelé *CSV*[3], il peut être relu depuis un programme *Python* comme le montre l'exemple précédent ou être chargé depuis *Microsoft Excel* en précisant que le format du fichier est

3. Comma Separated Value

le format *CSV*. Pour les valeurs numériques, il ne faut oublier de convertir en caractères lors de l'écriture et de convertir en nombres lors de la lecture.

**Remarque 8.7 : nombres français et anglais**
Les nombres réels s'écrivent en anglais avec un point pour séparer la partie entière de la partie décimale. En français, il s'agit d'une virgule. Il est possible que lors de la conversion d'une matrice, il faille remplacer les points par des virgules et réciproquement pour éviter les problèmes de conversion.

## 8.2 Fichiers zip

Les fichiers *zip* sont très répandus de nos jours et constituent un standard de compression facile d'accès quelque soit l'ordinateur et son système d'exploitation. Le langage *Python* propose quelques fonctions pour compresser et décompresser ces fichiers par l'intermédiaire du module `zipfile`. Le format de compression *zip* est un des plus répandus bien qu'il ne soit pas le plus performant [4]. Ce format n'est pas seulement utilisé pour compresser mais aussi pour regrouper plusieurs fichiers en un seul.

### 8.2.1 Lecture

L'exemple suivant permet par exemple d'obtenir la liste des fichiers inclus dans un fichier *zip* :

```
import zipfile
file = zipfile.ZipFile ("exemplezip.zip", "r")
for info in file.infolist () :
    print info.filename, info.date_time, info.file_size
file.close ()
```

Les fichiers compressés ne sont pas forcément des fichiers textes mais de tout format. Le programme suivant extrait un fichier parmi ceux qui ont été compressés puis affiche son contenu (on suppose que le fichier lu est au format texte donc lisible).

```
import zipfile
file = zipfile.ZipFile ("exemplezip.zip", "r")
data = file.read ("informatique/testzip.py")
file.close ()
print data
```

On retrouve dans ce cas les étapes d'ouverture et de fermeture même si la première est implicitement inclus dans le constructeur de la classe `ZipFile`.

### 8.2.2 Ecriture

Pour créer un fichier *zip*, le procédé ressemble à la création de n'importe quel fichier. La seule différence provient du fait qu'il est possible de stocker le fichier à compresser sous un autre nom à l'intérieur du fichier *zip*, ce qui explique les deux premiers arguments de la méthode `write`. Le troisième paramètre indique si le fichier doit être compressé (`zipfile.ZIP_DEFLATED`) ou non (`zipfile.ZIP_STORED`).

```
import zipfile
file = zipfile.ZipFile ("test.zip", "w")
file.write ("fichier.txt", "nom_fichier_dans_zip.txt", zipfile.ZIP_DEFLATED)
file.close ()
```

4. Le format *rar* offre souvent des taux de compression plus élevés, voir http://www.rarlab.com/.

Une utilisation possible de ce procédé serait l'envoi automatique d'un mail contenant un fichier `zip` en pièce jointe. Une requête comme *python* précédent le nom de votre serveur de mail permettra, via un moteur de recherche, de trouver des exemples sur Internet.

Selon les serveurs de mails, le programme permettant d'envoyer automatiquement un mail en *Python* peut varier. L'exemple suivant permet d'envoyer un email automatiquement via un serveur de mails, il montre aussi comment attacher des pièces jointes. Il faut bien sûr être autorisé à se connecter. De plus, il est possible que l'exécution de ce programme ne soit pas toujours couronnée de succès si le mail est envoyé plusieurs fois à répétition, ce comportement est en effet proche de celui d'un spammeur.

```
import smtplib
from email.MIMEMultipart    import MIMEMultipart
from email.MIMEBase         import MIMEBase
from email.MIMEText         import MIMEText
from email.Utils            import formatdate
from email                  import Encoders
import os

def envoyer_mail (aqui, sujet, contenu, files = []):
    de              = "email de l'auteur"
    msg             = MIMEMultipart()
    msg ['From']    = de
    msg ['To']      = aqui
    msg ['Date']    = formatdate (localtime = True)
    msg ['Subject'] = sujet

    msg.attach(  MIMEText (contenu) )
    for file in files:
        part = MIMEBase('application', 'octet-stream')
        part.set_payload ( open(file,'rb').read () )
        Encoders.encode_base64 (part)
        part.add_header ('Content-Disposition',  \
                         'attachment; filename="%s"' % os.path.basename (file))
        msg.attach (part)

    smtp = smtplib.SMTP ("smtp.gmail.com", 587)
    smtp.ehlo ()
    smtp.starttls ()
    smtp.ehlo ()
    smtp.login ("login", "mot_de_passe")

    smtp.sendmail (de, aqui, msg.as_string () )
    smtp.close()

envoyer_mail ( "destinataire", "sujet","contenu", ["mail.py"] )
```

## 8.3 Manipulation de fichiers

Il arrive fréquemment de copier, recopier, déplacer, effacer des fichiers. Lorsqu'il s'agit de quelques fichiers, le faire manuellement ne pose pas de problème. Lorsqu'il s'agit de traiter plusieurs centaines de fichiers, il est préférable d'écrire un programme qui s'occupe de le faire automatiquement. Cela peut être la création automatique d'un fichier `zip` incluant tous les fichiers modifiés durant la journée ou la réorganisation de fichiers musicaux au format `mp3` à l'aide de modules complémentaires tel que `mutagen`[5] ou `pymedia`[6].

**Remarque 8.8 : majuscules et minuscules**
Pour ceux qui ne sont pas familiers des systèmes d'exploitation, il faut noter que *Windows* ne fait pas de différences entre les majuscules et les minuscules à l'intérieur d'un nom de fichier. Les systèmes *Linux* et

5. http://code.google.com/p/quodlibet/wiki/Development/Mutagen
6. http://pymedia.org/

| | |
|---|---|
| abspath(path) | Retourne le chemin absolu d'un fichier ou d'un répertoire. |
| commonprefix(list) | Retourne le plus grand préfixe commun à un ensemble de chemins. |
| dirname(path) | Retourne le nom du répertoire. |
| exists(path) | Dit si un chemin est valide ou non. |
| getatime(path)<br>getmtime(path)<br>getctime(path) | Retourne diverses dates concernant un chemin, date du dernier accès (getatime), date de la dernière modification (getmtime), date de création (getctime). |
| getsize(file) | Retourne la taille d'un fichier. |
| isabs(path) | Retourne True si le chemin est un chemin absolu. |
| isfile(path) | Retourne True si le chemin fait référence à un fichier. |
| isdir(path) | Retourne True si le chemin fait référence à un répertoire. |
| join(p1, p2, ...) | Construit un nom de chemin étant donné une liste de répertoires. |
| split(path) | Découpe un chemin, isole le nom du fichier ou le dernier répertoire des autres répertoires. |
| splitext(path) | Découpe un chemin en nom + extension. |

**Table 8.1 :** *Liste non exhaustive des fonctionnalités offertes par le module* os.path*. La page http://docs.python.org/library/os.path.html référence toutes les fonctions de ce module.*

*Mac OS X* font cette différence[7].

### 8.3.1 Gestion des noms de chemins

Le module os.path propose plusieurs fonctions très utiles qui permettent entre autres de tester l'existence d'un fichier, d'un répertoire, de récupérer diverses informations comme sa date de création, sa taille... Cette liste présentée par la table 8.1 est loin d'être exhaustive mais elle donne une idée de ce qu'il est possible de faire.

### 8.3.2 Copie, suppression

Le module shutil permet de copier des fichiers, le module os permet de les supprimer ou de les renommer, il permet également de créer ou de supprimer des répertoires. Ces opérations sont regroupées dans la table 8.2.

### 8.3.3 Liste de fichiers

La fonction listdir du module os permet de retourner les listes des éléments inclus dans un répertoire (fichiers et sous-répertoires). Toutefois, le module glob propose une fonction plus intéressante qui permet de retourner la liste des éléments d'un répertoire en appliquant un filtre. Le programme suivant permet par exemple de retourner la liste des fichiers et des répertoires inclus dans un répertoire.

```
# -*- coding: cp1252 -*-
import glob
import os.path

def liste_fichier_repertoire (folder, filter) :
    # résultats
    file,fold = [], []

    # recherche des fichiers obéissant au filtre
```

7. Ceci explique que certains programmes aient des comportements différents selon le système d'exploitation sur lequel ils sont exécutés ou encore que certains liens Internet vers des fichiers ne débouchent sur rien car ils ont été saisis avec des différences au niveau des minuscules majuscules.

| | |
|---|---|
| copy(f1,f2) | (module shutil) Copie le fichier f1 vers f2 |
| chdir(p) | (module os) Change le répertoire courant, cette fonction peut être importante lorsqu'on utilise la fonction system du module os pour lancer une instruction en ligne de commande ou lorsqu'on écrit un fichier sans préciser le nom du répertoire, le fichier sera écrit dans ce répertoire courant qui est par défaut le répertoire où est situé le programme *Python*. C'est à partir du répertoire courant que sont définis les chemins relatifs. |
| getcwd() | (module os) Retourne le répertoire courant, voir la fonction chdir. |
| mkdir(p) | (module os) Crée le répertoire p. |
| makedirs(p) | (module os) Crée le répertoire p et tous les répertoires des niveaux supérieurs s'ils n'existent pas. Dans le cas du répertoire d : /base/repfinal, crée d'abord d : /base s'il n'existe pas, puis d : /base/repfinal. |
| remove(f) | (module os) Supprime un fichier. |
| rename(f1,f2) | (module os) Renomme un fichier |
| rmdir(p) | (module os) Supprime un répertoire |

**Table 8.2 :** *Liste non exhaustive des fonctionnalités offertes par les modules* shutil *et* os. *Les page* http://docs.python.org/library/shutil.html *et* http://docs.python.org/library/os.html *référencent toutes les fonctions de ces modules.*

```
    res = glob.glob (folder + "\\" + filter)

    # on inclut les sous-répertoires qui n'auraient pas été
    # sélectionnés par le filtre
    rep = glob.glob (folder + "\\*")
    for r in rep :
        if r not in res and os.path.isdir (r) :
            res.append (r)

    # on ajoute fichiers et répertoires aux résultats
    for r in res :
        path = r
        if os.path.isfile (path) :
            # un fichier, rien à faire à part l'ajouter
            file.append (path)
        else :
            # sous-répertoire : on appelle à nouveau la fonction
            # pour retourner la liste des fichiers inclus
            fold.append (path)
            fi,fo = liste_fichier_repertoire (path, filter)
            file.extend (fi)  # on étend la liste des fichiers
            fold.extend (fo)  # on étend la liste des répertoires
    # fin
    return file,fold

folder = r"D:\Dupre\_data\informatique"
filter = "*.tex"
file,fold = liste_fichier_repertoire (folder, filter)

for f in file : print "fichier ", f
for f in fold : print "répertoire ", f
```

Le programme repose sur l'utilisation d'une fonction récursive qui explore d'abord le premier répertoire. Elle se contente d'ajouter à une liste les fichiers qu'elle découvre puis cette fonction s'appelle elle-même sur le premier sous-répertoire qu'elle rencontre. La fonction walk permet d'obtenir la liste des fichiers et des sous-répertoire. Cette fonction parcourt automatiquement les sous-répertoires inclus, le programme est plus court mais elle ne prend pas en compte le filtre.

```
# -*- coding: cp1252 -*-
import os
```

```
def liste_fichier_repertoire (folder) :
    file, rep = [], []
    for r, d, f in os.walk (folder) :
        for a in d : rep.append (r + "/" + a)
        for a in f : file.append (r + "/" + a)
    return file, rep

folder = r"D:\Dupre\_data\informatique"
file,fold = liste_fichier_repertoire (folder)

for f in file : print "fichier ", f
for f in fold : print "répertoire ", f
```

## 8.4 Format binaire

Ecrire et lire des informations au travers d'un fichier texte revient à convertir les informations quel que soit leur type dans un format lisible pour tout utilisateur. Un entier est écrit sous forme de caractères décimaux alors que sa représentation en mémoire est binaire. Cette conversion dans un sens puis dans l'autre est parfois jugée coûteuse en temps de traitement et souvent plus gourmande en terme de taille de fichiers [8] même si elle permet de relire les informations écrites grâce à n'importe quel éditeur de texte. Il est parfois plus judicieux pour une grande masse d'informations d'utiliser directement le format binaire, c'est-à-dire celui dans lequel elles sont stockées en mémoire. Les informations apparaissent dans leur forme la plus simple pour l'ordinateur : une suite d'octets (bytes en anglais). Deux étapes vont intervenir que ce soit pour l'écriture :

1. On récupère les informations dans une suite d'octets (fonction `pack` du module `struct`).
2. On les écrit dans un fichier (méthode `write` affiliée aux fichiers).

Ou la lecture :

1. On lit une suite d'octets depuis un fichier (méthode `read` affiliée aux fichiers).
2. On transforme cette suite d'octets pour retrouver l'information qu'elle formait initialement (fonction `unpack` du module `struct`).

L'utilisation de fichiers binaires est moins évidente qu'il n'y paraît et il faut faire appel à des modules spécialisés alors que la gestion des fichiers texte ne pose aucun problème. Cela vient du fait que *Python* ne donne pas directement accès à la manière dont sont stockées les informations en mémoire contrairement à des langages tels que le *C++*. L'intérêt de ces fichiers réside dans le fait que l'information qu'ils contiennent prend moins de place stockée en binaire plutôt que convertie en chaînes de caractères au format texte [9].

L'écriture et la lecture d'un fichier binaire soulèvent les mêmes problèmes que pour un fichier texte : il faut organiser les données avant de les enregistrer pour savoir comment les retrouver. Les types immuables (réel, entier, caractère) sont assez simples à gérer dans ce format. Pour les objets complexes, *Python* propose une solution grâce au module `pickle`.

### 8.4.1 Ecriture dans un fichier binaire

L'écriture d'un fichier binaire commence par l'ouverture du fichier en mode écriture par l'instruction `file` = `open`(" < `nom_fichier` > ","`wb`"). C'est le code "`wb`" qui est important (*w* pour *write*, *b* pour *binary*), il spécifie le mode d'ouverture "`w`" et le format "`b`". La fermeture est la même que pour un fichier texte.

8. Un fichier texte compressé, au format `zip` par exemple, est une alternative aux fichiers binaires en terme de taille mais il allonge la lecture et l'écriture par des étapes de compression et de décompression.

9. Par exemple, un réel est toujours équivalent à huit caractères en format binaire alors que sa conversion au format texte va souvent jusqu'à quinze caractères.

Le module `struct` et la fonction `pack` permet de convertir les informations sous forme de chaîne de caractères avant de les enregistrer au format binaire. La fonction `pack` construit une chaîne de caractères égale au contenu de la mémoire. Son affichage avec la fonction `print` produit quelque chose d'illisible le plus souvent. Le tableau suivant montre les principaux formats de conversion [10].

| code | type correspondant |
|---|---|
| `c` | caractère |
| `B` | caractère non signé (octet) |
| `i` | entier (4 octets) |
| `I` | entier non signé (4 octets) |
| `d` | double (8 octets) |

L'utilisation de ces codes est illustrée au paragraphe 8.4.3.

### 8.4.2 Lecture d'un fichier binaire

Le code associé à l'ouverture d'un fichier binaire en mode lecture est "`rb`", cela donne : `file` = `open`(" < `nom_fichier` > ", La lecture utilise la fonction `unpack` pour effectuer la conversion inverse, celle d'une chaîne de caractères en entiers, réels, ... Le paragraphe suivant illustre la lecture et l'écriture au format binaire.

### 8.4.3 Exemple complet

Cet exemple crée un fichier "`info.bin`" puis écrit des informations à l'intérieur. Il ne sera pas possible d'afficher le contenu du fichier à l'aide d'un éditeur de texte. La méthode `pack` accepte un nombre de paramètres variable [11], ceci explique la syntaxe `struct.pack`("`cccc`", ∗`s`).

```
# -*- coding: cp1252 -*-
import struct
# on enregistre un entier, un réel et 4 caractères
i = 10
x = 3.1415692
s = "ABCD"

# écriture
file = open ("info.bin", "wb")
file.write ( struct.pack ("i" , i) )
file.write ( struct.pack ("d" , x) )
file.write ( struct.pack ("cccc" , *s) )
file.close ()

# lecture
file = open ("info.bin", "rb")
i = struct.unpack ("i",      file.read (4))
x = struct.unpack ("d",      file.read (8))
s = struct.unpack ("cccc",   file.read (4))
file.close ()

# affichage pour vérifier que les données ont été bien lues
print i  # affiche (10,)
print x  # affiche (3.1415692000000002,)
print s  # affiche ('A', 'B', 'C', 'D')
```

Les résultats de la méthode `unpack` apparaissent dans un tuple mais les données sont correctement récupérées. Ce programme fait aussi apparaître une des particularité du format binaire. On suppose ici que la chaîne de caractères est toujours de longueur 4. En fait, pour stocker une information de dimension

10. La liste complète figure à l'adresse http://docs.python.org/library/struct.html.
11. voir également paragraphe 4.4.10, page 72

variable, il faut d'abord enregistrer cette dimension puis s'en servir lors de la relecture pour connaître le nombre d'octets à lire. On modifie le programme précédent pour sauvegarder une chaîne de caractères de longueur variable.

```
# -*- coding: cp1252 -*-
import struct
# on enregistre un entier, un réel et n caractères
i = 10
x = 3.1415692
s = "ABCDEDF"

# écriture
file = open ("info.bin", "wb")
file.write ( struct.pack ("i" , i) )
file.write ( struct.pack ("d" , x) )
file.write ( struct.pack ("i" , len(s)) )  # on sauve la dimension de s
file.write ( struct.pack ("c" * len(s) , *s) )
file.close ()

# lecture
file = open ("info.bin", "rb")
i = struct.unpack ("i",      file.read (4))
x = struct.unpack ("d",      file.read (8))
l = struct.unpack ("i",      file.read (4)) # on récupère la dimension de s
l = l [0]  # l est un tuple, on s'intéresse à son unique élément
s = struct.unpack ("c" * l,  file.read (l))
file.close ()

# affichage pour contrôler
print i  # affiche (10,)
print x  # affiche (3.1415692000000002,)
print s  # affiche ('A', 'B', 'C', 'D', 'E', 'D', 'F')
```

Cette méthode utilisée pour les chaînes de caractères est applicable aux listes et aux dictionnaires de longueur variable : il faut d'abord stocker leur dimension. Il faut retenir également que la taille d'un réel est de huit octets, celle d'un entier de quatre octets et celle d'un caractère d'un octet [12]. Cette taille doit être passée en argument à la méthode `read`.

### 8.4.4 Objets plus complexes

Il existe un moyen de sauvegarder dans un fichier des objets plus complexes à l'aide du module `pickle`. Celui-ci permet de stocker dans un fichier le contenu d'un dictionnaire à partir du moment où celui-ci contient des objets standard du langage *Python*. Le principe pour l'écriture est le suivant :

```
import pickle

dico = {'a': [1, 2.0, 3, "e"], 'b': ('string', 2), 'c': None}
lis  = [1, 2, 3]

f = open ('data.bin', 'wb')
pickle.dump (dico, f)
pickle.dump (lis, f)
f.close()
```

La lecture est aussi simple :

12. Cette règle est toujours vrai sur des ordinateurs *32 bits*. Cette taille varie sur les ordinateurs 64 bits qui commencent à se répandre. Le programme suivant donnera la bonne réponse.

```
from struct import pack
print len(pack('i',0)) # longueur d'un entier
print len(pack('d',0)) # longueur d'un réel
print len(pack('c',"e")) # longueur d'un caractère
```

```
f = open ('data.bin', 'rb')
dico = pickle.load (f)
lis  = pickle.load (f)
f.close()
```

Un des avantages du module `pickle` est de pouvoir gérer les références circulaires : il est capable d'enregistrer et de relire une liste qui se contient elle-même, ce peut être également une liste qui en contient un autre qui contient la première...

Le module `pickle` peut aussi gérer les classes définies par un programmeur à condition qu'elles puissent convertir leur contenu en un dictionnaire dans un sens et dans l'autre, ce qui correspond à la plupart des cas.

```
import pickle
import copy

class Test :
    def __init__ (self) :
        self.chaine = "a"
        self.entier = 5
        self.tuple  = { "h":1, 5:"j" }

t = Test ()

f = open('data.bin', 'wb')  # lecture
pickle.dump (t, f)
f.close()

f = open('data.bin', 'rb')  # écriture
t = pickle.load (f)
f.close()
```

Lorsque la conversion nécessite un traitement spécial, il faut surcharger les opérateurs `__getstate__` et `__setstate__`. Ce cas se produit par exemple lorsqu'il n'est pas nécessaire d'enregistrer tous les attributs de la classe car certains sont calculés ainsi que le montre l'exemple suivant :

```
import pickle
import copy

class Test :
    def __init__ (self) :
        self.x = 5
        self.y = 3
        self.calcule_norme ()   # attribut calculé
    def calcule_norme (self) :
        self.n = (self.x ** 2 + self.y ** 2) ** 0.5
    def __getstate__ (self) :
        """conversion de Test en un dictionnaire"""
        d = copy.copy (self.__dict__)
        del d ["n"]  # attribut calculé, on le sauve pas
        return d
    def __setstate__ (self,dic) :
        """conversion d'un dictionnaire dic en Test"""
        self.__dict__.update (dic)
        self.calcule_norme ()  # attribut calculé

t = Test ()

f = open('data.bin', 'wb')  # lecture
pickle.dump (t, f)
f.close()

f = open('data.bin', 'rb')  # écriture
```

```
t = pickle.load (f)
f.close()
```

## 8.5 Paramètres en ligne de commande

Certains développements nécessitent de pouvoir exécuter un programme *Python* en ligne de commande sans intervention extérieure. Ce peut être un programme de synchronisation de deux répertoires ou la récupération du contenu de pages HTML, des tâches souvent exécutées la nuit à une heure où personne n'est présent pour le faire. Prenons par exemple le cas d'un programme très simple de synchronisation qui recopie tous les fichiers d'un répertoire vers un autre. La fonctionne `copie_repertoire` effectue cette tâche.

```
# -*- coding: cp1252 -*-
import glob
import shutil
def copie_repertoire (rep1, rep2) :
    """copie tous les fichiers d'un répertoire rep1 vers un autre rep2"""
    li = glob.glob (rep1 + "/*.*")   # récupère dans une liste fichiers et
                                     # répertoires qui respectent le filtre
    for l in li :
        to = l.replace (rep1, rep2)  # nom du fichier copié
                                     # (on remplace rep1 par rep2)
        shutil.copy (l, to)
copie_repertoire ("c:/original", "c:/backup")
```

Cette tâche est en plus exécutée sur deux répertoires et on ne voudrait pas avoir deux programmes différents alors que la tâche est la même quelque soit le couple de répertoire choisi. On souhaite pouvoir lancer le programme *Python* et lui spécifier les deux répertoires, c'est-à-dire être capable de lancer le programme comme suit (voir également figure 8.1) :

```
c:\python25\python.exe synchro.py c:/original c:/backup
```

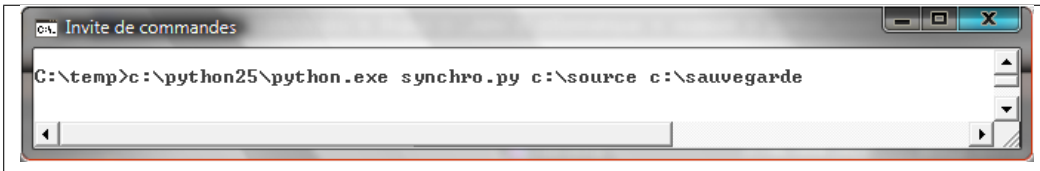



**Figure 8.1 :** *Un programme Python lancé en ligne de commande.*

Pour lancer un programme *Python* en ligne de commande, il faut écrire dans une fenêtre de commande ou un programme d'extension *bat* les instructions suivantes séparées par des espaces :

```
interpréteur_python programme_python argument1 argument2 ...
```

Il faut maintenant récupérer ces paramètres au sein du programme *Python* grâce au module `sys` qui contient une variable `argv` contenant la liste des paramètres de la ligne de commande. S'il y a deux paramètres, cette liste `argv` contiendra trois éléments :

1. nom du programme
2. paramètre 1
3. paramètre 2

Le programme suivante utilise ce mécanisme avec le programme `synchro` contenant la fonction `copie_repertoire` :

```
# -*- coding: cp1252 -*-
import glob
import shutil
def copie_repertoire (rep1, rep2) :
    """copie tous les fichiers d'un répertoire rep1 vers un autre rep2"""
    li = glob.glob (rep1 + "/*.*")
    for l in li :
        to = l.replace (rep1, rep2)  # nom du fichier copié
                                     # (on remplace rep1 par rep2)
        shutil.copy (l, to)

import sys
                    # sys.argv [0] --> nom du programme (ici, synchro.py)
rep1 = sys.argv [1]   # récupération du premier paramètre
rep2 = sys.argv [2]   # récupération du second paramètre
copie_repertoire (rep1, rep2)
```

**Remarque 8.9 : paramètres contenant des espaces**
Lorsque les paramètres écrits sur la ligne de commande contiennent eux-mêmes des espaces, il faut les entourer avec des guillemets.

```
c:\python25\python.exe c:\batch\synchro.py
         "c:\Program Files\source" "c:\Program Files\backup"
```

Pour des besoins plus complexes, le module `getopt` propose des fonctions plus élaborées pour traiter les paramètres de la ligne de commande. Il est également possible de lancer un programme en ligne de commande depuis *Python* grâce à la fonction `system` du module `os` comme ceci :

```
import os
os.system ("c:\python25\python.exe c:\batch\synchro.py " \
           "\"c:\Program Files\source\" \"c:\Program Files\backup\"")
```

Le programme suivant est un autre exemple de l'utilité de la fonction `system`. Il écrit une matrice au format *HTML*[13] puis utilise la fonction `system` pour lancer le navigateur *Firefox* ou *Internet Explorer* afin de pouvoir visualiser la page créée.

```
mat = ["Victor Hugo 6".split (), "Marcel Proust 3".split () ]
f = open ("tableau.html", "w")
f.write ("<body><html>\n")
f.write ("<table border=\"1\">\n")
for m in mat :
    f.write ("<tr>")
    for c in m :
        f.write ("<td>" + c + "</td>")
    f.write ("</tr>\n")
f.write ("</table>")
f.close ()

import os
os.system ("\"C:\\Program Files\\Mozilla Firefox\\firefox.exe\" tableau.html")
os.system ("\"C:\\Program Files\\Internet Explorer\\iexplore.exe\"" \
           " d:\\temp\\tableau.html")
```

13. La description du langage *HTML* sort du cadre de ce livre mais la requête `HTML syntaxe` effectuée sur n'importe quel moteur de recherche Internet retournera des résultats intéressants.

## 8.6 Expressions régulières

### 8.6.1 Premiers pas

Chercher un mot dans un texte est une tâche facile, c'est l'objectif de la méthode `find` attachée aux chaînes de caractères, elle suffit encore lorsqu'on cherche un mot au pluriel ou au singulier mais il faut l'appeler au moins deux fois pour chercher ces deux formes. Pour des expressions plus compliquées, il est conseillé d'utiliser les *expressions régulières*. C'est une fonctionnalité qu'on retrouve dans beaucoup de langages. C'est une forme de grammaire qui permet de rechercher des expressions.

Lorsqu'on remplit un formulaire, on voit souvent le format "MM/JJ/AAAA" qui précise sous quelle forme on s'attend à ce qu'une date soit écrite. Les expressions régulières permettent de définir également ce format et de chercher dans un texte toutes les chaînes de caractères qui sont conformes à ce format.

La liste qui suit contient des dates de naissance. On cherche à obtenir toutes les dates de cet exemple sachant que les jours ou les mois contiennent un ou deux chiffres, les années deux ou quatre.

```
s = """date 0 : 14/9/2000
date 1 : 20/04/1971     date 2 : 14/09/1913     date 3 : 2/3/1978
date 4 : 1/7/1986     date 5 : 7/3/47     date 6 : 15/10/1914
date 7 : 08/03/1941     date 8 : 8/1/1980     date 9 : 30/6/1976"""
```

Le premier chiffre du jour est soit 0, 1, 2, ou 3 ; ceci se traduit par $[0-3]$. Le second chiffre est compris entre 0 et 9, soit $[0-9]$. Le format des jours est traduit par $[0-3][0-9]$. Mais le premier jour est facultatif, ce qu'on précise avec le symbole ? : $[0-3]?[0-9]$. Les mois suivent le même principe : $[0-1]?[0-9]$. Pour les années, ce sont les deux premiers chiffres qui sont facultatifs, le symbole ? s'appliquent sur les deux premiers chiffres, ce qu'on précise avec des parenthèses : ([0-2][0-9]) ?[0-9][0-9]. Le format final d'une date devient :

```
[0-3]?[0-9]/[0-1]?[0-9]/([0-2][0-9])?[0-9][0-9]
```

Le module `re` gère les expressions régulières, celui-ci traite différemment les parties de l'expression régulière qui sont entre parenthèses de celles qui ne le sont pas : c'est un moyen de dire au module `re` que nous nous intéressons à telle partie de l'expression qui est signalée entre parenthèses. Comme la partie qui nous intéresse - une date - concerne l'intégralité de l'expression régulière, il faut insérer celle-ci entre parenthèses.

La première étape consiste à construire l'expression régulière, la seconde à rechercher toutes les fois qu'un morceau de la chaîne `s` définie plus haut correspond à l'expression régulière.

```
import re
# première étape : construction
expression = re.compile ("([0-3]?[0-9]/[0-1]?[0-9]/([0-2][0-9])?[0-9][0-9])")
# seconde étape  : recherche
res = expression.findall (s)
print res
```

Le résultat de ce programme est le suivant :

```
[('14/9/2000', '20'), ('20/04/1971', '19'),  ('14/09/1913', '19'),
 ('2/3/1978', '19'),  ('1/7/1986', '19'),    ('7/3/47', ''),
 ('15/10/1914', '19'), ('08/03/1941', '19'), ('8/1/1980', '19'),
 ('30/6/1976', '19')]
```

C'est une liste de couples dont chaque élément correspond aux parties comprises entre parenthèses qu'on appelle des *groupes*. Lorsque les expressions régulières sont utilisées, on doit d'abord se demander comment définir ce qu'on cherche puis quelles fonctions utiliser pour obtenir les résultats de cette recherche. Les deux paragraphes qui suivent y répondent.

### 8.6.2 Syntaxe

La syntaxe des expressions régulières est décrite sur le site officiel de *Python*[14]. Comme toute grammaire, celle des expressions régulières est susceptible d'évoluer au fur et à mesure des versions du langage *Python*.

#### 8.6.2.1 Les ensembles de caractères

Lors d'une recherche, on s'intéresse aux caractères et souvent aux classes de caractères : on cherche un chiffre, une lettre, un caractère dans un ensemble précis ou un caractère qui n'appartient pas à un ensemble précis. Certains ensembles sont prédéfinis, d'autres doivent être définis à l'aide de crochets.

Pour définir un ensemble de caractères, il faut écrire cet ensemble entre crochets : [0123456789] désigne un chiffre. Comme c'est une séquence de caractères consécutifs, on peut résumer cette écriture en $[0-9]$. Pour inclure les symboles −, +, il suffit d'écrire : $[-0-9+]$. Il faut penser à mettre le symbole − au début pour éviter qu'il ne désigne une séquence.

Le caractère ^ inséré au début du groupe signifie que le caractère cherche ne doit pas être un de ceux qui suivent. Le tableau suivant décrit les ensembles prédéfinis[15] et leur équivalent en terme d'ensemble de caractères :

| | |
|---|---|
| . | désigne tout caractère non spécial quel qu'il soit |
| \d | désigne tout chiffre, est équivalent à $[0-9]$ |
| \D | désigne tout caractère différent d'un chiffre, est équivalent à $[^0-9]$ |
| \s | désigne tout espace ou caractère approché, est équivalent à [ \t\n\r\f\v][16] |
| \S | désigne tout caractère différent d'un espace, est équivalent à [^\t\n\r\f\v] |
| \w | désigne tout lettre ou chiffre, est équivalent à $[a-zA-Z0-9_]$ |
| \W | désigne tout caractère différent d'une lettre ou d'un chiffre, est équivalent à $[^a-zA-Z0-9_]$ |
| ^ | désigne le début d'un mot sauf s'il est placé entre crochets |
| $ | désigne la fin d'un mot sauf s'il est placé entre crochets |

A l'instar des chaînes de caractères, comme le caractère \ est un caractère spécial, il faut le doubler : [\\]. Avec ce système, le mot "taxinomie" qui accepte deux orthographes s'écrira : tax[io]nomie.

**Remarque 8.10 : caractères spéciaux et expressions régulières**
Le caractère \ est déjà un caractère spécial pour les chaînes de caractères en *Python*, il faut donc le quadrupler pour l'insérer dans un expression régulière. L'expression suivante filtre toutes les images dont l'extension est png et qui sont enregistrées dans un répertoire image.

```
s = r"D:\Dupre\_data\informatique\support\vba\image/vbatd1_4.png"
print re.compile ( "[\\\\/]image[\\\\/].*[.]png").search(s) # résultat positif
print re.compile (r"[\\/]image[\\/].*[.]png").search(s)     # même ligne
```

#### 8.6.2.2 Les multiplicateurs

Les multiplicateurs permettent de définir des expressions régulières comme : un mot entre six et huit lettres qu'on écrira $[\w]\{6,8\}$. Le tableau suivant donne la liste des multiplicateurs principaux[17] :

14. La page http://docs.python.org/library/re.html#regular-expression-syntax décrit la syntaxe, la page http://docs.python.org/howto/regex.html#regex-howto décrit comment se servir des expressions régulières, les deux pages sont en anglais.
15. La page http://docs.python.org/library/re.html#regular-expression-syntax les recense tous.
16. Ces caractères sont spéciaux, les plus utilisés sont \t qui est une tabulation, \n qui est une fin de ligne et qui \r qui est un retour à la ligne.
17. La page http://docs.python.org/library/re.html#regular-expression-syntax les recense tous.

| | |
|---|---|
| $*$ | présence de l'ensemble de caractères qui précède entre 0 fois et l'infini |
| $+$ | présence de l'ensemble de caractères qui précède entre 1 fois et l'infini |
| ? | présence de l'ensemble de caractères qui précède entre 0 et 1 fois |
| {m,n} | présence de l'ensemble de caractères qui précède entre m et n fois, si m = n, cette expression peut être résumée par {n}. |
| (?!(...)) | absence du groupe désigné par les points de suspensions (voir paragraphe 8.6.4). |

L'algorithme des expressions régulières essaye toujours de faire correspondre le plus grand morceau à l'expression régulière. Par exemple, dans la chaîne $<$ h1 $>$ mot $<$ /h1 $>$, $<.*>$ correspond avec $<$ h1 $>$, $<$ /h1 $>$ ou encore $<$ h1 $>$ mot $<$ /h1 $>$. Par conséquent, l'expression régulière correspond à trois morceaux. Par défaut, il prendra le plus grand. Pour choisir les plus petits, il faudra écrire les multiplicateurs comme ceci : $*?$, $+?$, $??$.

```
import re
s = "<h1>mot</h1>"
print re.compile ("(<.*>)") .match (s).groups () # ('<h1>mot</h1>',)
print re.compile ("(<.*?>)").match (s).groups () # ('<h1>',)
```

#### 8.6.2.3 Groupes

Lorsqu'un multiplicateur s'applique sur plus d'un caractère, il faut définir un groupe à l'aide de parenthèses. Par exemple, le mot yoyo s'écrira : (yo){2}. Les parenthèses jouent un rôle similaire à celui qu'elles jouent dans une expression numérique. Tout ce qui est compris entre deux parenthèses est considéré comme un groupe.

#### 8.6.2.4 Assembler les caractères

On peut assembler les groupes de caractères les uns à la suite des autres. Dans ce cas, il suffit de les juxtaposer comme pour trouver les mots commençant par s : s[a − z]$*$. On peut aussi chercher une chaîne ou une autre grâce au symbole |. Chercher dans un texte l'expression *Xavier Dupont* ou *M. Dupont* s'écrira : (Xavier)|(M[.])Dupont.

### 8.6.3 Fonctions

La fonction compile du module re permet de construire un objet "expression régulière". A partir de cet objet, on peut vérifier la correspondance entre une expression régulière et une chaîne de caractères (méthode match). On peut chercher une expression régulière (méthode search). On peut aussi remplacer une expression régulière par une chaîne de caractères (méthode sub). La table 8.3 récapitule ces méthodes.

Les méthodes search et match retournent toutes des objets Match dont les méthodes sont présentées par la table 8.4. Appliquées à l'exemple décrit page 167 concernant les dates, cela donne :

```
expression = re.compile ("([0-3]?[0-9]/[0-1]?[0-9]/([0-2][0-9])?[0-9][0-9])[^\d]")
print expression.search (s).group(1,2) # affiche ('14/9/2000', '20')
c = expression.search (s).span(1)      # affiche (9, 18)
print s [c[0]:c[1]]                    # affiche 14/9/2000
```

### 8.6.4 Exemple plus complet

L'exemple suivant présente trois cas d'utilisation des expressions régulières. On s'intéresse aux titres de chansons *MP3* stockées dans un répertoire. Le module mutagen [18] permet de récupérer certaines informations concernant un fichier *MP3* dont le titre, l'auteur et la durée.

18. http://code.google.com/p/quodlibet/wiki/Development/Mutagen

| | |
|---|---|
| `match(s[, pos[, end]])` | Vérifie la correspondance entre l'expression régulière et la chaîne `s`. Il est possible de n'effectuer cette vérification qu'entre les caractères dont les positions sont `pos` et `end`. La fonction retourne `None` s'il n'y a pas de correspondance et sinon un objet de type `Match`. |
| `search(s[, pos[, end]])` | Fonction semblable à `match`, au lieu de vérifier la correspondance entre toute la chaîne et l'expression régulière, cherche la première chaîne de caractères extraite correspondant à l'expression régulière. |
| `split(s [, maxsplit = 0])` | Recherche toutes les chaînes de caractères extraites qui vérifient l'expression régulière puis découpe cette chaîne en fonction des expressions trouvées. La méthode `split` d'une chaîne de caractère permet de découper selon un séparateur. La méthode `split` d'une expression régulière permet de découper selon plusieurs séparateurs. C'est pratique pour découper une chaîne de caractères en colonnes séparées par ; ou une tabulation. `re.compile("[\t;]").split("a;b\tc;g")` donne `["a","b","c","g"]`. |
| `findall(s[, pos[, end]])` | Identique à `split` mais ne retourne pas les morceaux entre les chaînes extraites qui vérifient l'expression régulière. |
| `sub(repl, s [, count = 0])` | Remplace dans la chaîne `repl` les éléments `\1`, `\2`, ... par les parties de `s` qui valident l'expression régulières. |
| `flags` | Mémorise les options de construction de l'expression régulière. C'est un attribut. |
| `pattern` | Chaîne de caractères associée à l'expression régulière. C'est un attribut. |

**Table 8.3 :** *Liste non exhaustive des méthodes et attributs qui s'appliquent à un objet de type "expression régulière" retourné par la fonction* `compile` *du module* `re`*. La page http://docs.python.org/library/re.html contient la documentation associée au module* `re`*.*

| | |
|---|---|
| `group([g1, ...])` | Retourne la chaîne de caractères validée par les groupes `g1`... |
| `groups([default])` | Retourne la liste des chaînes de caractères qui ont été validées par chacun des groupes. |
| `span([group])` | Retourne les positions dans la chaîne originale des chaînes extraites validées par chacun des groupes. |

**Table 8.4 :** *Liste non exhaustive des méthodes qui s'appliquent à un objet de type* `Match` *qui est le résultat des méthodes* `search` *et* `match`*. Les groupes sont des sous-parties de l'expression régulière, chacune d'entre elles incluses entre parenthèses. Le* $n^{ème}$ *correspond au groupe qui suit la* $n^{ème}$ *parenthèse ouvrante. Le premier groupe a pour indice 1. La page http://docs.python.org/library/re.html contient la documentation associée au module* `re`*.*

Le premier problème consiste à retrouver les chansons sans titre ou dont le titre contient seulement son numéro : *track03*, *track - 03*, *audiotrack 03*, *track 03*, *piste 03*, *piste - 03*, *audiopiste 03*, ... Ce titre indésirable doit valider l'expression régulière suivante : $\gamma((\texttt{audio})?\texttt{track}()?(-)?[0-9]1,2)|(\texttt{piste}[0-9]1,2))\$$.

Le second problème consiste à retrouver toutes les chansons dont le titre contient le mot *heart* mais ni *heartland* ni *heartache*. Ce titre doit valider l'expression régulière : $((\texttt{heart})(?!((\texttt{ache})|(\texttt{land}))))$.

Le troisième problème consiste à compter le nombre de mots d'un titre. Les mots sont séparés par l'ensemble de caractères [–, ; !'.?& :]. On utilise la méthode `split` pour découper en mots. Le résultat est illustré par le programme suivant.

```
# -*- coding: latin-1 -*-
import mutagen.mp3, mutagen.easyid3, os, re

def infoMP3 (file, tags) :
    """retourne des informations sur un fichier MP3 sous forme de
    dictionnaire (durée, titre, artiste, ...)"""
    a = mutagen.mp3.MP3(file)
    b = mutagen.easyid3.EasyID3(file)
    info = { "minutes":a.info.length/60, "nom":file }
```

```
    for k in tags :
        try : info [k] = str (b [k][0])
        except : continue
    return info

def all_files (repertoire, tags, ext = re.compile (".mp3$")) :
    """retourne les informations pour chaque fichier d'un répertoire"""
    all = []
    for r, d, f in os.walk (repertoire) :
        for a in f :
            if not ext.search (a) : continue
            t = infoMP3 (r + "/" + a, tags)
            if len (t) > 0 : all.append (t)
    return all

def heart_notitle_mots (all, avoid,sep,heart) :
    """retourne trois résultats
    - les chansons dont le titre valide l'expression régulière heart
    - les chansons dont le titre valide l'expression régulière avoid
    - le nombre moyen de mots dans le titre d'une chanson"""
    liheart, notitle  = [], []
    nbmot,nbsong      = 0,0
    for a in all :
        if "title" not in a :
            notitle.append (a)
            continue
        ti = a ["title"].lower ()
        if avoid.match (ti) :
            notitle.append (a)
            continue
        if heart.search (ti) : liheart.append (a)
        nbsong += 1
        nbmot  += len ([ m for m in sep.split (ti) if len (m) > 0 ])
    return liheart, notitle, float (nbmot)/nbsong

tags  = "title album artist genre tracknumber".split ()
all = all_files (r"C:\musique", tags)

avoid = re.compile ("^(((audio)?track( )?( - )?[0-9]{1,2})|(piste [0-9]{1,2}))$")
sep   = re.compile ("[- ,;!'.?&:]")
heart = re.compile ("((heart)(?!((ache)|(land))))")
liheart, notitle, moymot = heart_notitle_mots (all, avoid, sep, heart)

print "nombre de mots moyen par titre ", moymot
print "somme des durée contenant heart ", sum ( [ s ["minutes"] for s in liheart] )
print "chanson sans titre ", len (notitle)
print "liste des titres "
for s in liheart : print "   ",s ["title"]
```

Le programme suivant est un exemple d'utilisation des expressions régulières dont l'objectif est de détecter les fonctions définies dans un programme mais jamais utilisées. Les expressions servent à détecter les définitions de fonctions (d'après le mot-clé `def`) puis à détecter les appels. On recoupe ensuite les informations en cherchant les fonctions définies mais jamais appelées.

Il n'est pas toujours évident de construire une expression régulière qui correspondent précisément à tous les cas qu'on veut détecter. Une stratégie possible est de construire une expression régulière plus permissive puis d'éliminer les cas indésirables à l'aide d'une seconde expression régulière, c'est le cas ici pour détecter les appels.

```
# -*- coding: cp1252 -*-
"""ce programme détermine toutes les fonctions définies dans
un programme et jamais appelées"""
import glob, os, re
```

```
def trouve_toute_fonction (s, exp, gr, expm = "^$") :
    """ à partir d'une chaîne de caractères correspondant
    à un programme Python, cette fonction retourne
    une liste de 3-uples, chacun contient :
        - le nom de la fonction
        - (debut,fin) de l'expression dans la chaîne
        - la ligne où elle a été trouvée

    Paramètres:
       - s    : chaîne de caractères
       - exp  : chaîne de caractères correspond à l'expression
       - gr   : numéro de groupe correspondant au nom de la fonction
       - expm : expression négative
    """
    exp = re.compile (exp)
    res = []
    pos = 0
    r = exp.search (s, pos)   # première recherche
    while r != None :
        temp = (r.groups () [gr], r.span (gr), r.group (gr))
        x    = re.compile ( expm.replace ("function", temp [0]) )
        if not x.match (temp [2]) :
            # l'expression négative n'est pas trouvé, on peut ajouter ce résultat
            res.append (temp)
        r    = exp.search (s, r.end (gr) )     # recherche suivante
    return res

def get_function_list_definition (s) :
    """trouve toutes les définitions de fonctions"""
    return trouve_toute_fonction (s, \
              "\ndef[ ]+([a-zA-Z_][a-zA-Z_0-9]*)[ ]*[(].*[)][ ]*[:]", 0)

def get_function_list_call (s) :
    """trouve tous les appels de fonctions"""
    return trouve_toute_fonction (s, \
              "\n.*[=(,[{ .]([a-zA-Z_][a-zA-Z_0-9]*)(?![ ]?:)[ ]*[(].*[)]?", 0, \
              "^\\n[ ]*(class|def)[ ]+function.*$")

def detection_fonction_pas_appelee (file) :
    """retourne les couples de fonctions jamais appelées suivies
    du numéro de la ligne où elles sont définies"""

    f       = open (file, "r")
    li      = f.readlines ()
    f.close ()
    sfile   = "".join (li)

    funcdef = get_function_list_definition (sfile)
    funccal = get_function_list_call (sfile)
    f2 = [ p [0] for p in funccal ]
    res = []
    for f in funcdef :
        if f [0] not in f2 :
            ligne = sfile [:f [1][0]].count ("\n")
            res.append ( (f [0], ligne+2))
    return res

def fonction_inutile () :  # ligne 63
    pass

if __name__ == "__main__" :
    file    = "fonction.py"
    print detection_fonction_pas_appelee (file)
            # affiche [('fonction_inutile', 63)]
```

## 8.7 Dates

Le module `datetime` [19] fournit une classe `datetime` qui permet de faire des opérations et des comparaisons sur les dates et les heures. L'exemple suivant calcule l'âge d'une personne née le 11 août 1975.

```
import datetime
naissance = datetime.datetime (1975,11,8,10,0,0)
jour = d.now ()        # obtient l'heure et la date actuelle
print naissance        # affiche 2008-08-31 20:36:40.772000
age = jour - naissance # calcule une différence
print naissance        # affiche 12074 days, 10:48:45.056000
```

L'objet `datetime` autorise les soustractions et les comparaisons entre deux dates. Une soustraction retourne un objet de type `timedelta` qui correspond à une durée. qu'on peut multiplier par un réel ou ajouter à un objet de même type ou à un objet de type `datetime`. L'utilisation de ce type d'objet évite de se pencher sur tous les problèmes de conversion.

Le module `calendar` est assez pratique pour construire des calendriers. Le programme ci-dessous affiche une liste de t-uples incluant le jour et le jour de la semaine du mois d'août 1975. Dans cette liste, on y trouve le t-uple $(11, 0)$ qui signifie que le 11 août 1975 était un lundi. Cela permet de récupérer le jour de la semaine d'une date de naissance.

```
import calendar
c = calendar.Calendar ()
for d in c.itermonthdays2 (1975,8) : print d
```

## 8.8 Problème de jeux de caractères

La langue anglaise est la langue dominante en ce qui concerne l'informatique mais cela n'empêche pas que des programmes anglais manipulent du japonais même si le nombre de caractères est beaucoup plus grand. Les jeux de caractères proposent une solution à ce problème : un jeu de caractères définit la façon de décoder une suite d'octets [20]. Les langues latines n'ont besoin que d'un octet pour coder un caractère, les langues asiatiques en ont besoin de plusieurs. Il n'existe pas qu'un seul jeu de caractères lorsqu'on programme. Ils interviennent à cinq endroits différents :

1. Le jeu de caractères utilisé par l'éditeur de texte pour afficher le programme.
2. Le jeu de caractères du programme, par défaut `ascii` mais il peut être changé en insérant une première ligne de commentaire (voir paragraphe 2.5.2, page 21). Les chaînes de caractères du programme sont codées avec ce jeu de caractères. Ce jeu devrait être identique à celui utilisé par l'éditeur de texte afin d'éviter les erreurs.
3. Le jeu de caractères de la sortie, utilisé pour chaque instruction `print`, il est désigné par le code `cp1252` sur un système `Windows`.
4. Le jeu de caractères dans lequel les chaînes de caractères sont manipulées. Un jeu standard qui permet de représenter toutes les langues est le jeu de caractères `utf` $-$ `8`. Il peut être différent pour chaque variable.
5. Le jeu de caractères d'un fichier texte. Il peut être différent pour chaque fichier.

Le langage *Python* offre deux classes pour représenter les chaînes de caractères. La classe `str` qui est adaptée aux langues latines, le jeu de caractères n'est pas précisé. La classe `unicode` représente un caractère sur un à quatre octets avec un jeu de caractères désigné par l'appellation `unicode`. Il est impératif de savoir quel jeu est utilisé à quel endroit et il faut faire des conversions de jeux de caractères pour passer l'information d'un endroit à un autre. Il existe deux méthodes pour cela présentées par la table 8.5.

19. Voir également la page http://docs.python.org/library/datetime.html.

20. On confond souvent jeu de caractères et *encodage*. Le jeu de caractère désigne l'ensemble de caractères dont le programme a besoin, l'encodage décrit la manière dont on passe d'une séquence de caractères français, japonais, anglais à une séquence d'octets qui est la seule information manipulée par un ordinateur.

```
# -*- coding: latin-1 -*-
st =  "eé"
su = u"eé"  # raccourci pour su = unicode ("eé", "latin-1")
            # ou encore      su = unicode ("eé".decode ("latin-1"))

print type (st)                                    # affiche <type 'str'>
print type (su)                                    # affiche <type 'unicode'>
print len (st),        ";", st                     # affiche  2 ; eé
print len (repr (st)), ";", repr (st)              # affiche  7 ; 'e\xe9'
print len (su),        ";", su.encode ("latin-1")  # affiche  2 ; eé
print len (repr (su)), ";", repr (su)              # affiche  8 ; u'e\xe9'
```

Lorsqu'on manipule plusieurs jeux de caractères, il est préférable de conserver un unique jeu de référence pour le programme par l'intermédiaire de la classe `unicode`. Il "suffit" de gérer les conversions depuis ces chaînes de caractères vers les entrées sorties du programme comme les fichiers texte. Par défaut, ceux-ci sont écrits avec le jeu de caractères du système d'exploitation. Dans ce cas, la fonction `open` suffit. En revanche, si le jeu de caractères est différent, il convient de le préciser lors de l'ouverture du fichier. On utilise la fonction `open` du module `codecs` qui prend comme paramètre supplémentaire le jeu de caractères du fichier. Toutes les chaînes de caractères seront lues et converties au format `unicode`.

```
import codecs
f = codecs.open ("essai.txt", "r", "cp1252")  # jeu Windows
s = "".join (f.readlines ())
f.close ()
print type (s)              # affiche <type 'unicode'>
print s.encode ("cp1252")   # pour l'afficher,
                            # il faut convertir l'unicode en "cp1252"
```

| | |
|---|---|
| encode( [enc[, err]]) | Cette fonction permet de passer d'un jeu de caractères, celui de la variable, au jeu de caractères précisé par `enc` à moins que ce ne soit le jeu de caractères par défaut. Le paramètre `err` permet de préciser comment gérer les erreurs, doit-on interrompre le programme (valeur `'strict'`) ou les ignorer (valeur `'ignore'`). La documentation *Python* recense toutes les valeurs possibles pour ces deux paramètres aux adresses http://docs.python.org/library/codecs.html#id3 et http://docs.python.org/library/stdtypes.html#id4. Cette fonction retourne un résultat de type `str`. |
| decode( [enc[, err]]) | Cette fonction est la fonction inverse de la fonction `encode`. Avec les mêmes paramètres, elle effectue la transformation inverse. |

**Table 8.5 :** *Conversion de jeux de caractères, ce sont deux méthodes qui fonctionnent de façons identiques pour les deux classes de chaînes de caractères disponibles en Python :* `str` *et* `unicode`.

Le programme suivant permet d'obtenir le jeu de caractères par défaut et celui du système d'exploitation.

```
import sys
import locale
            # retourne le jeu de caractères par défaut
print sys.getdefaultencoding ()    # affiche ascii
            # retourne le jeu de caractères du système d'exploitation
print locale.getdefaultlocale()    # affiche ('fr_FR', 'cp1252')
```

Les problèmes de jeux de caractères peuvent devenir vite compliqués lorsqu'on manipule des informations provenant de plusieurs langues différentes. Il est rare d'avoir à s'en soucier tant que le programme gère les langues anglaise et française. Dans le cas contraire, il est préférable d'utiliser le type `unicode` pour toutes les chaînes de caractères. Il est conseillé de n'utiliser qu'un seul jeu de caractères pour enregistrer les informations dans des fichiers et le jeu de caractères `utf − 8` est le plus indiqué.

# Chapitre 9

# Interface graphique

---

Les interfaces graphiques servent à rendre les programmes plus conviviaux. Elles sont pratiques à utiliser mais elles demandent un peu de temps pour les concevoir. Un programme sans interface exécute des instructions les unes à la suite des autres, le programme a un début - un point d'entrée - et une fin. Avec une interface, le programme fonctionne de manière différente. Il n'exécute plus successivement les instructions mais attend un événement - pression d'une touche du clavier, clic de souris - pour exécuter une fonction. C'est comme si le programme avait une multitude de points d'entrée.

Il existe plusieurs modules permettant d'exploiter les interfaces graphiques. Le plus simple est le module `Tkinter` présent lors de l'installation du langage *Python*. Ce module est simple mais limité. Le module `wxPython` est plus complet mais un peu plus compliqué dans son utilisation [1]. Toutefois, le fonctionnement des interfaces graphiques sous un module ou un autre est identique. C'est pourquoi ce chapitre n'en présentera qu'un seul, le module `Tkinter`. Pour d'autres modules, les noms de classes changent mais la logique reste la même : il s'agit d'associer des événements à des parties du programme *Python*.

Les interfaces graphiques évoluent sans doute plus vite que les autres modules, des composantes de plus en plus complexes apparaissent régulièrement. Un module comme `wxPython` change de version plusieurs fois par an. Il est possible de trouver sur Internet des liens [2] qui donne des exemples de programme. Une excellente source de documentation sont les forums de discussion qui sont un lieu où des programmeurs échangent questions et réponses. Un message d'erreur entré sur un moteur de recherche Internet permet souvent de tomber sur des échanges de ce type, sur des problèmes résolus par d'autres.

## 9.1 Introduction

Un programme muni d'une interface graphique fonctionne différemment d'un programme classique. Un programme classique est une succession presque linéaire d'instructions. Il y a un début ou *point d'entrée* du programme et aucun événement extérieur ne vient troubler son déroulement. Avec une interface graphique, le point d'entrée du programme est masqué : il est pris en compte automatiquement. Du point de vue du programmeur, le programme a plusieurs points d'entrée : une simple fenêtre avec deux boutons (voir figure 9.1) propose deux façons de commencer et il faut prévoir une actions associée à chaque bouton.

La conception d'une interface graphique se déroule généralement selon deux étapes. La première consiste à dessiner l'interface, c'est-à-dire choisir une position pour les objets de la fenêtre (boutons, zone de saisie, liste déroulante, ...). La seconde étape définit le fonctionnement de la fenêtre, c'est-à-dire associer à chaque objet des fonctions qui seront exécutées si un tel événement se réalise (pression d'un bouton, pression d'une touche, ...).

Pour le moment, nous allons supposer que ces deux étapes sont scindées même si elles sont parfois entremêlées lorsqu'un événement implique la modification de l'apparence de la fenêtre. La section qui suit décrit tous les objets que propose le module `Tkinter`. La section suivante présente la manière de les disposer dans une fenêtre. La section d'après décrit les événements et le moyen de les relier à des fonctions du programme. Ce chapitre se termine par quelques constructions courantes à propos des interfaces graphiques.

1. Le paragraphe 2.6.5 page 24 présente d'autres alternatives.
2. L'adresse http://gnuprog.info/prog/python/pwidget.php illustre chaque objet de `Tkinter`, on peut citer également http://effbot.org/tkinterbook/.

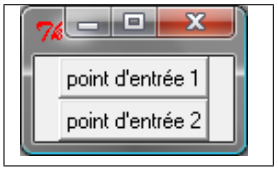



**Figure 9.1 :** *Une fenêtre contenant deux boutons : ce sont deux points d'entrée du programme.*

## 9.2 Les objets

Les interfaces graphiques sont composées d'*objets* ou *widgets*[3]. Ce paragraphe décrit les principales méthodes qui permettent de modifier le contenu et l'apparence des objets. Il faut se reporter à la documentation du langage[4] pour avoir la liste de toutes les options disponibles.

Les exemples de codes des paragraphes qui suivent permettent de disposer les objets dans une fenêtre qui ne s'affichera pas quelques lignes de code supplémentaires présentées au paragraphe 9.4.1 et l'utilisation d'une méthode `pack` (paragraphe 9.3). L'exemple suivant crée un objet :

```
zone_texte = Tkinter.Label (text = "zone de texte")
```

Et pour l'afficher, il faut l'enrober :

```
import Tkinter          # import de Tkinter
root = Tkinter.Tk ()    # création de la fenêtre principale
# ...
obj = Tkinter.Label (text = "zone de texte")
# ...
obj.pack ()             # on ajoute l'objet à la fenêtre principale
root.mainloop ()        # on affiche enfin la fenêtre principal et on attend
                        # les événements (souris, clic, clavier)
```

### 9.2.1 Zone de texte

Une zone de texte sert à insérer dans une fenêtre graphique une légende indiquant ce qu'il faut insérer dans une zone de saisie voisine comme le montre la figure 9.2. Une zone de texte correspond à la classe `Label` du module `Tkinter`. Pour créer une zone de texte, il suffit d'écrire la ligne suivante :

```
zone_texte = Tkinter.Label (text = "zone de texte")
```

Il est possible que le texte de cette zone de texte doive changer après quelques temps. Dans ce cas, il faut appeler la méthode `config` comme suit :

```
zone_texte = Tkinter.Label (text = "premier texte")
# ...
# pour changer de texte
zone_texte.config (text = "second texte")
```

La figure 9.2 montre deux zones de texte. La seconde est grisée par rapport à la première. Pour obtenir cet état, il suffit d'utiliser l'instruction suivante :

```
zone_texte.config (state = Tkinter.DISABLED)
```

3. On les appelle aussi *contrôle*. Comme ils reçoivent des événements, en un sens, ce sont ces objets qui pilotent un programme ou qui le contrôlent.

4. La page http://wiki.python.org/moin/TkInter recense quelques liens utiles autour de `Tkinter` dont la documentation officielle.




**Figure 9.2 :** *Exemple de zone de texte ou* Label *associée à une zone de saisie. La seconde image montre une zone de texte dans l'état* `DISABLED`.

Et pour revenir à un état normal :

```
zone_texte.config (state = Tkinter.NORMAL)
```

Ces deux dernières options sont communes à tous les objets d'une interface graphique. Cette option sera rappelée au paragraphe 9.2.11.

### 9.2.2 Bouton

Un bouton a pour but de faire le lien entre une fonction et un clic de souris. Un bouton correspond à la classe `Button` du module `Tkinter`. Pour créer un bouton, il suffit d'écrire la ligne suivante :

```
bouton = Tkinter.Button (text = "zone de texte")
```

Il est possible que le texte de ce bouton doive changer après quelques temps. Dans ce cas, il faut appeler la méthode `config` comme suit :

```
bouton = Tkinter.Button (text = "premier texte")
# ...
# pour changer de texte
bouton.config (text = "second texte")
```

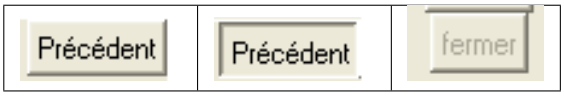



**Figure 9.3 :** *Exemple de boutons, non pressé, pressé, grisé. Le bouton grisé ne peut être pressé.*

La figure 9.3 montre trois boutons. Le troisième est grisé par rapport au premier. Les boutons grisés ne peuvent pas être pressés. Pour obtenir cet état, il suffit d'utiliser l'instruction suivante :

```
bouton.config (state = Tkinter.DISABLED)
```

Et pour revenir à un état normal :

```
bouton.config (state = Tkinter.NORMAL)
```

C'est pour cet objet que cette option est la plus intéressante car elle permet d'interdire la possibilité pour l'utilisateur de presser le bouton tout en le laissant visible.

Il est possible également d'associer une image à un bouton. Par exemple, les trois lignes suivantes créent un bouton, charge une image au format `gif` puis l'associe au bouton `b`. Lors de l'affichage de la fenêtre, le bouton `b` ne contient pas de texte mais une image.

```
b = Tkinter Button ()
im = Tkinter.PhotoImage (file = "chameau.gif")
b.config (image = im)
```



Les images qu'il est possible de charger sont nécessairement au format `GIF`, le seul que le module `Tkinter` puisse lire.

### 9.2.3 Zone de saisie

Une zone de saisie a pour but de recevoir une information entrée par l'utilisateur. Une zone de saisie correspond à la classe `Entry` du module `Tkinter` ; pour en créer une, il suffit d'écrire la ligne suivante :

```
saisie = Tkinter.Entry ()
```

Pour modifier le contenu de la zone de saisie, il faut utiliser la méthode `insert` qui insère un texte à une position donnée.

```
# le premier paramètre est la position
# où insérer le texte (second paramètre)
saisie.insert (pos, "contenu")
```

Pour obtenir le contenu de la zone de saisie, il faut utiliser la méthode `get` :

```
contenu = saisie.get ()
```

Pour supprimer le contenu de la zone de saisie, il faut utiliser la méthode `delete`. Cette méthode supprime le texte entre deux positions.

```
# supprime le texte entre les positions pos1, pos2
saisie.delete (pos1, pos2)
```

Par exemple, pour supprimer le contenu d'une zone de saisie, on peut utiliser l'instruction suivante :

```
saisie.delete (0, len (saisie.get ()))
```




**Figure 9.4 :** *Exemple de zones de saisie, normale et grisée, la zone grisée ne peut être modifiée.*

La figure 9.4 montre deux zones de saisie. La seconde est grisée par rapport à la première. Les zones de saisie grisées ne peuvent pas être modifiées. Pour obtenir cet état, il suffit d'utiliser la méthode `config` comme pour les précédents objets. Cette option sera rappelée au paragraphe 9.2.11.

### 9.2.4 Zone de saisie à plusieurs lignes

Une zone de saisie à plusieurs lignes est identique à la précédente à ceci près qu'elle autorise la saisie d'un texte sur plusieurs lignes. Cette zone correspond à la classe `Text` du module `Tkinter`. Pour créer une telle zone, il suffit d'écrire la ligne suivante :

```
saisie = Tkinter.Text ()
```

Pour modifier le contenu de la zone de saisie, il faut utiliser la méthode `insert` qui insère un texte à une position donnée. La méthode diffère de celle de la classe `Entry` puisque la position d'insertion est maintenant une chaîne de caractères contenant deux nombres séparés par un point : le premier nombre désigne la ligne, le second la position sur cette ligne.

```
# le premier paramètre est la position
# où insérer le texte (second paramètre)
pos = "0.0"
saisie.insert (pos, "première ligne\nseconde ligne")
```

Pour obtenir le contenu de la zone de saisie, il faut utiliser la méthode `get` qui retourne le texte entre deux positions. La position de fin n'est pas connue, on utilise la chaîne de caractères `"end"` pour désigner la fin de la zone de saisie.

```
# retourne le texte entre deux positions
pos1 = "0.0"
pos2 = "end"  # ou Tkinter.END
contenu = saisie.get (pos1, pos2)
```

Pour supprimer le contenu de la zone de saisie, il faut utiliser la méthode `delete`. Cette méthode supprime le texte entre deux positions.

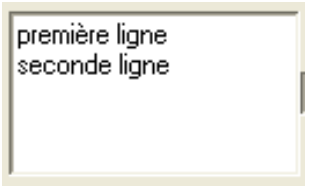
```
# supprime le texte entre les positions pos1, pos2
saisie.delete (pos1, pos2)
```

Par exemple, pour supprimer le contenu d'une zone de saisie à plusieurs lignes, on peut utiliser l'instruction suivante :

```
saisie.delete ("0.0", "end")
# on peut aussi utiliser
# saisie.delete ("0.0", Tkinter.END)
```

Pour modifier les dimensions de la zone de saisie à plusieurs lignes, on utilise l'instruction suivante :

```
# modifie les dimensions de la zone
# width <--> largeur
# height <--> hauteur en lignes
saisie.config (width = 10, height = 5)
```




L'image précédente montre une zone de saisie à plusieurs lignes. Pour griser cette zone, il suffit d'utiliser la méthode `config` rappelée au paragraphe 9.2.11.

### 9.2.5 Case à cocher

Une case à cocher correspond à la classe `Checkbutton` du module `Tkinter`. Pour créer une case à cocher, il suffit d'écrire la ligne suivante :

```
# crée un objet entier pour récupérer la valeur de la case à cocher,
# 0 pour non cochée, 1 pour cochée
v    = Tkinter.IntVar ()
case = Tkinter.Checkbutton (variable = v)
```

En fait, ce sont deux objets qui sont créés. Le premier, de type `IntVar`, mémorise la valeur de la case à cocher. Le second objet, de type `CheckButton`, gère l'apparence au niveau de l'interface graphique. La raison de ces deux objets est plus évidente dans le cas de l'objet `RadioButton` décrit au paragraphe suivant. Pour savoir si la case est cochée ou non, il suffit d'exécuter l'instruction :

```
v.get ()  # égal à 1 si la case est cochée, 0 sinon
```

Pour cocher et décocher la case, il faut utiliser les instructions suivantes :

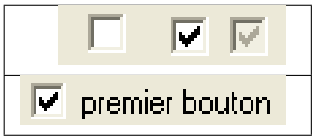
```
case.select ()      # pour cocher
case.deselect ()    # pour décocher
```

Il est possible d'associer du texte à l'objet case à cocher :

```
case.config (text = "case à cocher")
```

La figure 9.5 montre trois cases. La troisième est grisée par rapport à la première. Les cases grisées ne peuvent pas être cochées. Pour obtenir cet état, il suffit d'utiliser la méthode `config` rappelée au paragraphe 9.2.11.

**Figure 9.5 :** *Exemples de cases à cocher, non cochée, cochée, grisée. Lorsqu'elle est grisée, son état ne peut être modifié. La dernière image montre une case à cocher associée à un texte.*




### 9.2.6 Case ronde ou bouton radio

Une case ronde ou *bouton radio* correspond à la classe `Radiobutton` du module `Tkinter`. Elles fonctionnent de manière semblable à des cases à cocher excepté le fait qu'elles n'apparaissent jamais seules : elles fonctionnent en groupe. Pour créer un groupe de trois cases rondes, il suffit d'écrire la ligne suivante :

```
# crée un objet entier partagé pour récupérer le numéro du bouton radio activé
v     = Tkinter.IntVar ()
case1 = Tkinter.Radiobutton (variable = v, value = 10)
case2 = Tkinter.Radiobutton (variable = v, value = 20)
case3 = Tkinter.Radiobutton (variable = v, value = 30)
```

La variable `v` est partagée par les trois cases rondes. L'option `value` du constructeur permet d'associer un bouton radio à une valeur de `v`. Si `v` == `10`, seul le premier bouton radio sera sélectionné. Si `v` == `20`, seul le second bouton radio le sera. Si deux valeurs sont identiques pour deux boutons radio, il seront cochés et décochés en même temps. Et pour savoir quel bouton radio est coché ou non, il suffit d'exécuter l'instruction :

```
v.get ()  #  retourne le numéro du bouton radio coché (ici, 10, 20 ou 30)
```

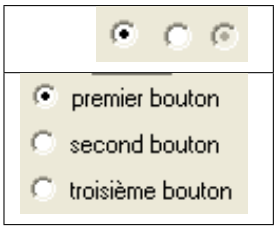
Pour cocher un des boutons radio, il faut utiliser l'instruction suivante :

```
v.set (numero)  # numéro du bouton radio à cocher
                # pour cet exemple, 10, 20 ou 30
```

Il est possible d'associer du texte à un bouton radio (voir figure 9.6) :

```
case1.config (text = "premier bouton")
case2.config (text = "second bouton")
case3.config (text = "troisième bouton")
```

**Figure 9.6 :** *Exemples de cases rondes ou boutons radio, non cochée, cochée, grisée. Lorsqu'elle est grisée, son état ne peut être modifiée. La seconde image présente un groupe de bouton radio. Un seul peut être sélectionné à la fois à moins que deux boutons ne soient associés à la même valeur. Dans ce cas, ils agiront de pair.*




La figure 9.6 montre trois cases. La troisième est grisée par rapport à la première. Si un des boutons radio est grisé parmi les trois, le choix du bouton à cocher s'effectue parmi les deux restants ou aucun si le bouton radio grisé est coché.

### 9.2.7 Liste

Un objet liste contient une liste d'intitulés qu'il est possible de sélectionner. Une liste correspond à la classe `ListBox` du module `Tkinter`. Pour la créer, il suffit d'écrire la ligne suivante :

```
li  = Tkinter.Listbox ()
```

Pour modifier les dimensions de la zone de saisie à plusieurs lignes, on utilise l'instruction suivante :

```
# modifie les dimensions de la liste
# width <--> largeur
# height <--> hauteur en lignes
li.config (width = 10, height = 5)
```

On peut insérer un élément dans la liste avec la méthode `insert` :

```
pos = 0   # un entier, "end" ou Tkinter.END pour insérer ce mot à la fin
li.insert (pos, "première ligne")
```

On peut supprimer des intitulés de cette liste avec la méthode `delete`.

```
pos1 = 0    # un entier
pos2 = None # un entier, "end" ou Tkinter.END pour supprimer tous les éléments
            # de pos1 jusqu'au dernier
li.delete (pos1, pos2 = None)
```

Les intitulés de cette liste peuvent ou non être sélectionnés. Cliquer sur un intitulé le sélectionne mais la méthode `select_set` permet aussi de le faire.

```
pos1 = 0
li.select_set (pos1, pos2 = None)
# sélectionne tous les éléments entre les indices pos1 et
# pos2 inclus ou seulement celui d'indice pos1 si pos2 == None
```

Réciproquement, il est possible d'enlever un intitulé de la sélection à l'aide de la méthode `select_clear`.

```
pos1 = 0
li.select_clear (pos1, pos2 = None)
# retire la sélection de tous les éléments entre les indices
# pos1 et pos2 inclus ou seulement celui d'indice pos1 si pos2 == None
```

La méthode `curselection` permet d'obtenir la liste des indices des éléments sélectionnés.

```
sel = li.curselection ()
```

La méthode `get` permet récupérer un élément de la liste tandis que la méthode `size` retourne le nombre d'éléments :

```
for i in range (0,li.size ()) :
    print li.get (i)
```




**Figure 9.7 :** *Exemple de liste. La seconde liste est grisée et ne peut être modifiée.*

La figure 9.7 montre deux listes. La seconde est grisée par rapport à la première. Elle ne peut être modifiée. Pour obtenir cet état, il faut appeler la méthode `config` rappelée au paragraphe 9.2.11.

**Remarque 9.1 : liste et barre de défilement**
Il est possible d'adjoindre une barre de défilement verticale. Il faut pour cela inclure l'objet dans une sous-fenêtre `Frame` qui est définie au paragraphe 9.3.2 comme dans l'exemple suivant :

```
    frame      = Tkinter.Frame (parent)
    scrollbar  = Tkinter.Scrollbar (frame)
    li         = Tkinter.Listbox (frame, width = 88, height = 6, \
                                  yscrollcommand = scrollbar.set)
    scrollbar.config (command = li.yview)
    li.pack (side = Tkinter.LEFT)
    scrollbar.pack (side = Tkinter.RIGHT, fill = Tkinter.Y)
```

Il suffit de transposer cet exemple pour ajouter une barre de défilement horizontale. Toutefois, il est préférable d'utiliser un objet prédéfini présent dans le module `Tix` qui est une extension du module `Tkinter`. Elle est présentée au paragraphe 9.2.8.

**Remarque 9.2 : plusieurs Listbox**
Lorsqu'on insère plusieurs objets `Listbox` dans une seule fenêtre, ces objets partagent par défaut la même sélection. Autrement dit, lorsqu'on clique sur un élément de la seconde `Listbox`, l'élément sélectionné dans la première ne l'est plus. Afin de pouvoir sélectionner un élément dans chaque `Listbox`, il faut ajouter dans les paramètres du constructeur l'option `exportselection` = `0` comme l'illustre l'exemple suivant :

```
    li = Tkinter.Listbox (frame, width = 88, height = 6, exportselection=0)
```

Il existe des méthodes plus avancées qui permettent de modifier l'aspect graphique d'un élément comme la méthode `itemconfigure`. Son utilisation est peu fréquente à moins de vouloir réaliser une belle interface graphique. Le paragraphe 9.6.2 montre l'utilisation qu'on peut en faire.

### 9.2.8 Liste avec barre de défilement, Combobox

Il n'existe pas dans le module `Tkinter` d'objets `List` avec une barre de défilement incluse mais il existe un autre module interne `Tix` qui étend la liste des objets proposés par `Tkinter`. Ce module propose notamment une liste avec une barre de défilement intégrée :

```
# -*- coding: cp1252 -*-
import Tix as Tk
root = Tk.Tk ()

o = Tk.ScrolledListBox (root)                                                    5
for k in range (0,100) : o.listbox.insert (Tk.END, "ligne " + str (k))
o.pack ()

def print_file () :                     # voir chapitre sur les événements
    print o.listbox.selection_get ()    # idem                                   10

b = Tk.Button (root, text = "print")
b.config (command = print_file)         # idem
b.pack ()
                                                                                 15
root.mainloop ()                        # idem
```

Le module `Tix` quoique assez riche puisqu'il propose des fenêtres permettant de sélectionner un fichier ou de remplir un tableau est mal documenté sur *Python*. Il existe une documentation officielle [5] et des exemples [6]. La plupart du temps, on trouve un exemple approché. Dans l'exemple précédent comme dans le suivant avec une `"ComboBox"`, la pression du bouton `print` écrit la zone sélectionnée. La figure 9.8 illustre graphique ce qu'est une `Combobox`.

```
# -*- coding: cp1252 -*-
import Tix as Tk
```

5. http://pydoc.org/2.1/Tix.html ou encore http://docs.python.org/library/tix.html
6. http://coverage.livinglogic.de/index.html

```
root = Tk.Tk ()

o = Tk.ComboBox (root, label = "label")
o.insert (Tk.END, "ligne 1")
o.insert (Tk.END, "ligne 2")
o.insert (Tk.END, "ligne 3")
o.insert (Tk.END, "ligne 4")
o.pack ()

def print_file () :                     # voir le chapitre sur les événéments
    print o.cget ("value")              # idem

b = Tk.Button (root, text = "print")
b.config (command = print_file)         # idem
b.pack ()

root.mainloop ()                        # idem
```

L'avantage du module `Tix` est d'être installé automatiquement avec le langage *Python*. Il reste malgré tout très peu documenté et arriver à ses fins peut nécessiter quelques heures de recherche.




**Figure 9.8 :** *Exemple de fenêtre* `Combobox` *définie par le module interne* `Tix` *qui est une extension du module* `Tkinter`*. La seconde image est la même* `Combobox` *mais dépliée.*

### 9.2.9 Canevas

Pour dessiner, il faut utiliser un objet canevas, correspondant à la classe `Canvas` du module `Tkinter`. Pour la créer, il suffit d'écrire la ligne suivante :

```
    ca  = Tkinter.Canvas ()
```

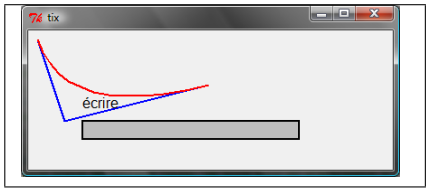
Pour modifier les dimensions de la zone de saisie à plusieurs lignes, on utilise l'instruction suivante :

```
    # modifie les dimensions du canevas
    # width <--> largeur en pixels
    # height <--> hauteur en pixels
    ca.config (width = 10, height = 5)
```

Cet objet permet de dessiner des lignes, des courbes, d'écrire du texte grâce aux méthodes `create_line`, `create_rectangle`, `create_text`. La figure 9.9 illustre ce que donnent les quelques lignes qui suivent.

```
      # dessine deux lignes du point 10,10 au point 40,100 et au point 200,60
      # de couleur bleue, d'épaisseur 2
    ca.create_line (10,10,40,100, 200,60, fill = "blue", width = 2)
      # dessine une courbe du point 10,10 au point 200,60
      # de couleur rouge, d'épaisseur 2, c'est une courbe de Bézier
      # pour laquelle le point  40,100 sert d'assise
    ca.create_line (10,10, 40,100, 200,60, smooth=1, fill = "red", width = 2)
      # dessine un rectangle plein de couleur jaune, de bord noir et d'épaisseur 2
    ca.create_rectangle (300,100,60,120, fill = "gray", width = 2)
      # écrit du texte de couleur noire au point 80,80 et avec la police arial
    ca.create_text (80,80, text = "écrire", fill = "black", font = "arial")
```

**Figure 9.9 :** *Exemple de canevas.*




### 9.2.10 Menus

Les menus apparaissent en haut des fenêtres. La plupart des applications arborent un menu commençant par *Fichier Edition Affichage...* Le paragraphe 9.4.5 page 192 les décrit en détail.

### 9.2.11 Méthodes communes

Nous avons vu que tous les objets présentés dans ce paragraphe possèdent une méthode `config` qui permet de définir l'état du widget (grisé ou normal) [7].

```
widget.config (state = Tkinter.DISABLED) # grisé
widget.config (state = Tkinter.NORMAL)   # aspect normal
```

Elle permet également de modifier le texte d'un objet, sa position, ... De nombreuses options sont communes à tous les objets et certaines sont spécifiques. L'aide associée à cette méthode [8] ne fournit aucun renseignement. En fait, le constructeur et cette méthode ont les mêmes paramètres optionnels. Il est équivalent de préciser ces options lors de l'appel au constructeur :

```
l = Tkinter.Label (text = "légende")
```

Ou de les modifier à l'aide de la méthode `config` :

```
l = Tkinter.Label ()
l.config (text = "légende")
```

L'aide associée à la méthode `config` est la suivante :

```
Help on method configure in module Tkinter:

configure(self, cnf=None, **kw) unbound Tkinter.Label method
    Configure resources of a widget.

    The values for resources are specified as keyword
    arguments. To get an overview about
    the allowed keyword arguments call the method keys.
```

Tandis que l'aide associée au constructeur de la classe `Label` [9] donne plus d'informations :

```
__init__(self, master=None, cnf={}, **kw) unbound Tkinter.Label method
    Construct a label widget with the parent MASTER.

    STANDARD OPTIONS

        activebackground, activeforeground, anchor,
        background, bitmap, borderwidth, cursor,
        disabledforeground, font, foreground,
```

7. La "disparition" d'un objet est évoquée au paragraphe 9.3.
8. Par exemple pour la classe `Label`, cette aide est affichée grâce à l'instruction `help(Tkinter.Label.config)`.
9. Affichée avec l'instruction `help(Tkinter.Label.__init__)`.

```
            highlightbackground, highlightcolor,
            highlightthickness, image, justify,
            padx, pady, relief, takefocus, text,
            textvariable, underline, wraplength

        WIDGET-SPECIFIC OPTIONS

            height, state, width
```

Cette aide mentionne les options communes à tous les objets (ou widgets) et les options spécifiques à cet objet, ici de type `Label`. Toutes ont une valeur par défaut qu'il est possible de changer soit dans le constructeur, soit par la méthode `config`. Quelques-unes ont été décrites, d'autres permettent de modifier entre autres la police avec laquelle est affiché le texte de l'objet (option `font`), la couleur du fond (option `background`), l'alignement du texte, à gauche, à droite, centré (option `justify`), l'épaisseur du bord (option `borderwidth`), le fait qu'un objet reçoive le *focus*[10] après la pression de la touche tabulation (`takefocus`)...

## 9.3 Disposition des objets dans une fenêtre

### 9.3.1 Emplacements

Chacun des objets (ou widgets) présentés au paragraphe précédent possède trois méthodes qui permettent de déterminer sa position dans une fenêtre : `pack`, `grid`, `place`. Les deux premières permettent de disposer les objets sans se soucier ni de leur dimension ni de leur position. La fenêtre gère cela automatiquement. La dernière place les objets dans une fenêtre à l'aide de coordonnées sans utiliser l'aide d'aucune grille. Dans une fenêtre, tous les objets doivent être placés avec la même méthode. Dans le cas contraire, les résultats risquent ne pas être ceux attendus.

#### 9.3.1.1 Méthode `pack`

Cette méthode empile les objets les uns à la suite des autres. Par défaut, elle les empile les uns en dessous des autres. Par exemple, l'exemple suivant produit l'empilement des objets de la figure 9.10.

```
l = Tkinter.Label (text = "première ligne")
l.pack ()
s = Tkinter.Entry ()
s.pack ()
e = Tkinter.Label (text = "seconde ligne")
e.pack ()
```

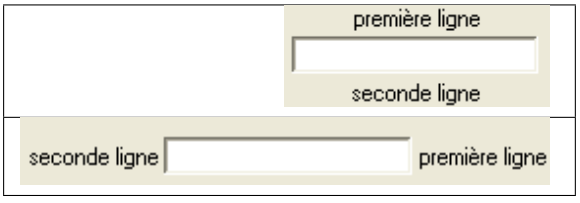



**Figure 9.10 :** *Les objets sont empilés à l'aide de la méthode* `pack` *les uns en dessous des autres pour la première image, les uns à droite des autres pour la seconde image.*

On peut aussi les empiler les uns à droite des autres grâce à l'option `side`.

```
l = Tkinter.Label (text = "première ligne")
l.pack (side = Tkinter.RIGHT)
s = Tkinter.Entry ()
```

10. voir paragraphe 9.4.2

```
s.pack (side = Tkinter.RIGHT)
e = Tkinter.Label (text = "seconde ligne")
e.pack (side = Tkinter.RIGHT)
```

La méthode `pack` possède trois options :

1. `side` : à choisir entre `Tkinter.TOP` (valeur par défaut), `Tkinter.LEFT`, `Tkinter.BOTTOM`, `Tkinter.RIGHT`.
2. `expand` : égale à `True` ou `False` (valeur par défaut), si cette option est vraie, l'objet occupe tout l'espace.
3. `fill` : égale à `None` (valeur par défaut), `X`, `Y`, `BOTH`, l'objet s'étend selon un axe (X ou Y ou les deux).

Il n'est pas toujours évident d'obtenir du premier coup le positionnement des objets souhaités au départ et il faut tâtonner pour y arriver. Lorsque un objet n'est plus nécessaire, il est possible de le faire disparaître en appelant la méthode `pack_forget`. Le rappel de la méthode `pack` le fera réapparaître mais rarement au même endroit.

```
s.pack_forget ()  # disparition
s.pack ()         # insertion à un autre endroit
```

#### 9.3.1.2 Méthode `grid`

La méthode `grid` suppose que la fenêtre qui les contient est organisée selon une grille dont chaque case peut recevoir un objet. L'exemple suivant place trois objets dans les cases de coordonnées $(0, 0)$, $(1, 0)$ et $(0, 1)$. Le résultat apparaît dans la figure 9.11.

```
l = Tkinter.Label (text = "première ligne")
l.grid (column = 0, row = 0)
s = Tkinter.Entry ()
s.grid (column = 0, row = 1)
e = Tkinter.Label (text = "seconde ligne")
e.grid (column = 1, row = 0)
```




**Figure 9.11 :** *Les objets sont placés dans une grille à l'aide de la méthode* `grid`. *Une fois que chaque objet a reçu une position, à l'affichage, il ne sera pas tenu compte des lignes et colonnes vides.*

La méthode `grid` possède plusieurs options, en voici cinq :

1. `column` : colonne dans laquelle sera placée l'objet.
2. `columnspan` : nombre de colonnes que doit occuper l'objet.
3. `row` : ligne dans laquelle sera placée l'objet.
4. `rowspan` : nombre de lignes que doit occuper l'objet.
5. `sticky` : indique ce qu'il faut faire lorsque la case est plus grande que l'objet qu'elle doit contenir. Par défaut, l'objet est centré mais il est possible d'aligner l'objet sur un ou plusieurs bords en précisant que `sticky` = "`N`" ou "`S`" ou "`W`" ou "`E`". Pour aligner l'objet sur un angle, il suffit de concaténer les deux lettres correspondant aux deux bords concernés. Il est aussi possible d'étendre l'objet d'un bord à l'autre en écrivant `sticky` = "`N` + `S`" ou `sticky` = "`E` + `W`".

Enfin, comme pour la méthode `pack`, il existe une méthode `grid_forget` qui permet de faire disparaître les objets.

```
s.grid_forget ()  # disparition
```

#### 9.3.1.3 Méthode `place`

La méthode `place` est sans doute la plus simple à comprendre puisqu'elle permet de placer chaque objet à une position définie par des coordonnées. Elle peut être utilisée en parallèle avec les méthodes `pack` et `grid`.

```
l = Tkinter.Label (text = "première ligne")
l.place (x=10,y=50)
```

La méthode `place_forget` permet de faire disparaître un objet placer avec cette méthode. L'inconvénient de cette méthode survient lorsqu'on cherche à modifier l'emplacement d'un objet : il faut en général revoir les positions de tous les autres éléments de la fenêtre. On procède souvent par tâtonnement pour construire une fenêtre et disposer les objets. Ce travail est beaucoup plus long avec la méthode `place`.

### 9.3.2 Sous-fenêtre

Les trois méthodes précédentes ne permettent pas toujours de placer les éléments comme on le désire. On souhaite parfois regrouper les objets dans des boîtes et placer celles-ci les unes par rapport aux autres. La figure 9.12 montre deux objets regroupés dans un rectangle avec à sa gauche une zone de texte. Les boîtes sont des instances de la classe `Frame`. Ce sont des objets comme les autres excepté le fait qu'une boîte contient d'autres objets y compris de type `Frame`. Pour créer une boîte, il suffit d'écrire la ligne suivante :

```
f = Tkinter.Frame ()
```

Ensuite, il faut pouvoir affecter un objet à cette boîte `f`. Pour cela, il suffit que `f` soit le premier paramètre du constructeur de l'objet créé :

```
l = Tkinter.Label (f, text = "première ligne")
```

L'exemple qui suit correspond au code qui permet d'afficher la fenêtre de la figure 9.12 :

```
f = Tkinter.Frame ()
l = Tkinter.Label (f, text = "première ligne")
l.pack ()                     # positionne l à l'intérieur de f
s = Tkinter.Entry (f)
s.pack ()                     # positionne s à l'intérieur de f
f.pack (side = Tkinter.LEFT)  # positionne f à l'intérieur
                              #   de la fenêtre principale
e = Tkinter.Label (text = "seconde ligne")
e.pack_forget ()
e.pack (side = Tkinter.RIGHT) # positionne e à l'intérieur
                              #   de la fenêtre principale
```

L'utilisation de ces blocs `Frame` est pratique lorsque le même ensemble de contrôles apparaît dans plusieurs fenêtres différentes ou au sein de la même fenêtre. Cette possibilité est envisagée au paragraphe 9.6.3.

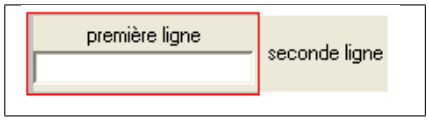



**Figure 9.12 :** *Les deux premiers objets - une zone de texte au-dessus d'une zone de saisie - sont regroupés dans une boîte - rectangle rouge, invisible à l'écran. A droite et centrée, une dernière zone de texte. Cet alignement est plus simple à réaliser en regrouper les deux premiers objets dans un rectangle (objet* `Frame`*).*

## 9.4 Evénements

### 9.4.1 Fenêtre principale

Tous les exemples des paragraphes précédents décrivent les différents objets disponibles et comment les disposer dans une fenêtre. Pour afficher cette fenêtre, il suffit d'ajouter au programme les deux lignes suivantes :

```
root = Tkinter.Tk ()
#  ici, on trouve le code qui définit les objets
#  et leur position dans la position
root.mainloop ()
```

La première ligne permet d'obtenir un identificateur relié à la fenêtre principale. La seconde ligne, outre le fait qu'elle affiche cette fenêtre, lance ce qu'on appelle une *boucle de messages*. Cette fonction récupère - intercepte - les événements comme un clic de souris, la pression d'une touche. Elle parcourt ensuite tous les objets qu'elle contient et regarde si l'un de ces objets est intéressé par cet événement. S'il est intéressé, cet objet prend l'événement et le traite. On peut revenir ensuite à la fonction `mainloop` qui attend à nouveau un événement. Cette fonction est définie par `Tkinter`, il reste à lui indiquer quels événements un objet désire intercepter et ce qu'il est prévu de faire au cas où cet événement se produit.

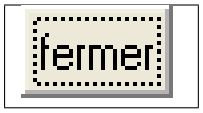
**Remarque 9.3 : instruction** `root = Tkinter.Tk()`
Cette première instruction doit se trouver au début du programme. Si elle intervient alors qu'une méthode de positionnement (`pack`, `grid`, ou `place`) a déjà été appelée, le programme affichera deux fenêtres dont une vide.

### 9.4.2 Focus

Une fenêtre peut contenir plusieurs zones de saisie, toutes capables d'intercepter la pression d'une touche du clavier et d'ajouter la lettre correspondante à la zone de saisie. Or la seule qui ajoute effectivement une lettre à son contenu est celle qui a le *focus*. La pression de la touche tabulation fait passer le focus d'un objet à l'autre. La figure 9.13 montre un bouton qui a le focus. Lorsqu'on désire qu'un objet en particulier ait le focus, il suffit d'appeler la méthode `focus_set`.

```
e = Tkinter.Entry ()
e.pack ()
e.focus_set ()
```

**Figure 9.13 :** *Ce bouton est entouré d'un cercle noir en pointillé, il a le* focus.




### 9.4.3 Lancer une fonction lorsqu'un bouton est pressé

La plupart de temps, le seul événement qu'on souhaite attraper est la pression d'un bouton. Le code suivant permet de créer un bouton dont l'identificateur est `b`. Il a pour intitulé `fonction change_legende`. On définit ensuite une fonction `change_legende` qui change la légende de ce bouton. L'avant-dernière ligne permet d'associer au bouton `b` la fonction `change_legende` qui est alors appelée lorsque le bouton est pressé. La dernière ligne affiche la fenêtre principale et lance l'application.

```
# -*- coding: cp1252 -*-
# la première ligne autorise les accents
```

```
import Tkinter
root = Tkinter.Tk ()
b = Tkinter.Button (text = "fonction change_legende")
b.pack ()

def change_legende () :
    global b
    b.config (text = "nouvelle légende")

b.config (command = change_legende)
root.mainloop ()
```

Lorsque le bouton `b` est pressé, on vérifie qu'il change bien de légende (voir figure 9.14).




**Figure 9.14 :** *La première fenêtre est celle qui apparaît lorsque le programme de la page 188 est lancé. Comme le bouton change de légende la première fois qu'il est pressé, l'apparence de la fenêtre change aussi, ce que montre la seconde image.*

**Remarque 9.4 : événement associé à une méthode**
L'exemple précédent associe une fonction au bouton. Lorsque l'interface devient conséquente, la lisibilité du programme en est réduite car le nombre de fonctions associées à des boutons augmentent rapidement. Pour éviter cela, il est possible d'associer au bouton une méthode de classe comme le suggère l'exemple du paragraphe 9.6.3.

### 9.4.4 Associer n'importe quel événement à un objet

Le paragraphe précédent s'est intéressé à l'association entre une fonction et la pression d'un bouton mais il est possible de faire en sorte que le programme exécute une fonction au moindre déplacement de la souris, à la pression d'une touche. Il est possible d'associer une fonction au moindre événement susceptible d'être intercepté par l'interface graphique.

On peut regrouper les événements en deux classes. La première classe regroupe les événements provenant du clavier ou de la souris. Ce sont des événements en quelque sorte *bruts*. La seconde classe regroupe des événements produit par des objets tels qu'un bouton. En effet, lorsque celui-ci détecte le clic du bouton droit de la souris, il construit un événement "`pressiondubouton`" et c'est celui-ci qui va déclencher l'exécution d'une fonction. Il n'est pas souvent nécessaire de revenir aux événements *bruts* car les objets proposent d'eux-mêmes de pouvoir attacher des fonctions à des événements liés à leur apparence comme l'a montré le paragraphe précédent.

Toutefois, pour un jeu par exemple, il est parfois nécessaire d'avoir accès au mouvement de la souris et il faut revenir aux événements *bruts*. Un événement est décrit par la classe `Event` dont les attributs listés par la table 9.1 décrivent l'événement qui sera la plupart du temps la pression d'une touche du clavier ou le mouvement de la souris.

La méthode `bind` permet d'exécuter une fonction lorsqu'un certain événement donné est intercepté par un objet donné. La fonction exécutée accepte un seul paramètre de type `Event` qui est l'événement qui l'a

| | |
|---|---|
| `char` | Lorsqu'une touche a été pressée, cet attribut contient son code, il ne tient pas compte des touches dites muettes comme les touches `shift`, `ctrl`, `alt`. Il tient pas compte non plus des touches `return` ou `suppr`. |
| `keysym` | Lorsqu'une touche a été pressée, cet attribut contient son code, quelque soit la touche, muette ou non. |
| `num` | Contient un identificateur de l'objet ayant reçu l'événement. |
| `x, y` | Coordonnées relatives de la souris par rapport au coin supérieur gauche de l'objet ayant reçu l'événement. |
| `x_root, y_root` | Coordonnées absolues de la souris par rapport au coin supérieur gauche de l'écran. |
| `widget` | Identifiant permettant d'accéder à l'objet ayant reçu l'événement. |

**Table 9.1 :** *Attributs principaux de la classe* `Event`*, ils décrivent les événements liés au clavier et à la souris. La liste complète est accessible en utilisant l'instruction* `help(Tkinter.Event)`.

déclenchée. Cette méthode a pour syntaxe :

**syntaxe 9.5 : méthode** `bind`

```
w.bind (ev, fonction)
```

`w` est l'identificateur de l'objet devant intercepter l'événement désigné par la chaîne de caractères `ev` (voir table 9.2). `fonction` est la fonction qui est appelée lorsque l'événement survient. Cette fonction ne prend qu'un paramètre de type `Event`.

| | |
|---|---|
| $<$ `Key` $>$ | Intercepter la pression de n'importe quelle touche du clavier. |
| $<$ `Button` $-$ `i` $>$ | Intercepter la pression d'un bouton de la souris. `i` doit être remplacé par 1,2,3. |
| $<$ `ButtonRelease` $-$ `i` $>$ | Intercepter le relâchement d'un bouton de la souris. `i` doit être remplacé par 1,2,3. |
| $<$ `Double` $-$ `Button` $-$ `i` $>$ | Intercepter la double pression d'un bouton de la souris. `i` doit être remplacé par 1,2,3. |
| $<$ `Motion` $>$ | Intercepter le mouvement de la souris, dès que le curseur bouge, la fonction liée à l'événement est appelée. |
| $<$ `Enter` $>$ | Intercepter un événement correspondant au fait que le curseur de la souris entre la zone graphique de l'objet. |
| $<$ `Leave` $>$ | Intercepter un événement correspondant au fait que le curseur de la souris sorte la zone graphique de l'objet. |

**Table 9.2 :** *Chaînes de caractères correspondant aux principaux types d'événements qu'il est possible d'intercepter avec le module* `Tkinter`*. La liste complète est accessible en utilisant l'instruction* `help(Tkinter.Label.bind)`.

L'exemple suivant utilise la méthode `bind` pour que le seul bouton de la fenêtre intercepte toute pression d'une touche, tout mouvement et toute pression du premier bouton de la souris lorsque le curseur est au dessus de la zone graphique du bouton. La fenêtre crée par ce programme ainsi que l'affichage qui en résulte apparaissent dans la figure 9.15.

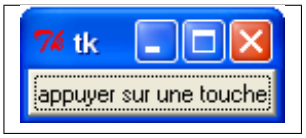
```
import Tkinter
root = Tkinter.Tk ()
b = Tkinter.Button (text = "appuyer sur une touche")
b.pack ()

def affiche_touche_pressee (evt) :
    print "--------------------------- touche pressee"
    print "evt.char = ", evt.char
    print "evt.keysym = ", evt.keysym
```

```
    print "evt.num = ", evt.num
    print "evt.x,evt.y = ", evt.x, ",", evt.y
    print "evt.x_root,evt.y_root = ", evt.x_root, ",", evt.y_root
    print "evt.widget = ", evt.widget

b.bind ("<Key>", affiche_touche_pressee)
b.bind ("<Button-1>", affiche_touche_pressee)
b.bind ("<Motion>", affiche_touche_pressee)
b.focus_set ()

root.mainloop ()
```




```
evt.char =  ??
evt.keysym =  ??
evt.num =  1
evt.x,evt.y =  105 , 13
evt.x_root,evt.y_root =
                292 , 239
evt.widget =  .9261224
```

```
evt.char =
evt.keysym =  Return
evt.num =  ??
evt.x,evt.y =  105 , 13
evt.x_root,evt.y_root =
                292 , 239
evt.widget =  .9261224
```

**Figure 9.15 :** *Fenêtre affichée par le programme du paragraphe 9.4.4. La pression d'une touche déclenche l'affichage des caractéristiques de l'événement. La seconde colonne correspond à la pression du premier bouton de la souris. La dernière colonne correspond à la pression de la touche* `Return`.

**Remarque 9.6 : focus**
L'avant dernière ligne du programme fait intervenir la méthode `focus_set`. Elle stipule que le bouton doit recevoir le `focus`. C'est-à-dire que cet objet est celui qui peut intercepter les événements liés au clavier. Sans cette instruction, cet objet n'y a pas accès, ces événements sont dirigés vers la fenêtre principale qui ne s'en soucie pas.

**Remarque 9.7 : mauvais événement**
Les messages d'erreur liés aux événements ne sont pas forcément très explicites. Ainsi l'instruction suivante adresse un événement inexistant.

```
b.bind ("<button-1>", affiche_touche_pressee)
```

Lors de l'exécution, le programme déclenche la succession d'exceptions suivantes qui signifie que l'événement $<$ `button` $-$ `1` $>$ n'existe pas.

```
Traceback (most recent call last):
  File "exemple_bind.py", line 17, in ?
    b.bind ("<button-1>", affiche_touche_pressee)
  File "c:\python25\lib\lib-tk\Tkinter.py", line 933, in bind
    return self._bind(('bind', self._w), sequence, func, add)
  File "c:\python25\lib\lib-tk\Tkinter.py", line 888, in _bind
    self.tk.call(what + (sequence, cmd))
_tkinter.TclError: bad event type or keysym "button"
```

**Remarque 9.8 : associer un événement à tous les objets**
Il arrive parfois qu'un événement ne doive pas être associé à un seul objet mais à tous ceux que la fenêtre contient. C'est l'objectif de la méthode `bind_all`. Sa syntaxe est exactement la même que la méthode `bind`.

```
b.bind_all ("<Button-1>", affiche_touche_pressee)
```

On utilise peu cette fonction, on préfère construire des objets propres à un programme comme suggéré au paragraphe 9.6.2.

**Remarque 9.9 : désactiver un événement**
De la même manière qu'il est possible d'associer un événement à un objet d'une fenêtre, il est possible

d'effectuer l'opération inverse qui consiste à supprimer cette association. La méthode `unbind` désactive un événement associé à un objet. La méthode `unbind_all` désactive un événement associé pour tous les objets d'une fenêtre.

**syntaxe 9.10 : méthode `unbind`**

```
w.unbind (ev)
w.unbind_all (ev)
```

`w` est l'identificateur de l'objet interceptant l'événement désigné par la chaîne de caractères `ev` (voir table 9.2). Après l'appel à la méthode `unbind`, l'événement n'est plus intercepté par l'objet `w`. Après l'appel à la méthode `unbind_all`, l'événement n'est plus intercepté par aucun objet.

**Remarque 9.11 : événement spécifique**
Il est possible de définir des événements propres aux programmes. Ceux-ci ne sont générés par aucun périphérique mais explicitement par le programme lui-même. Ce mécanisme est presque toujours couplé à l'utilisation de threads. Le paragraphe 10.3 (page 207) illustre ce principe à l'aide d'un exemple à base de thread. Le paragraphe 9.6.5 page 200 propose un exemple plus simple.

### 9.4.5 Menu

Les menus fonctionnent de la même manière que les boutons. Chaque intitulé du menu est relié à une fonction qui sera exécutée à la condition que l'utilisateur sélectionne cet intitulé. L'objet `Menu` ne désigne pas le menu dans son ensemble mais seulement un niveau. Par exemple, le menu présenté par la figure 9.17 est en fait un assemblage de trois menus auquel on pourrait ajouter d'autres sous-menus.

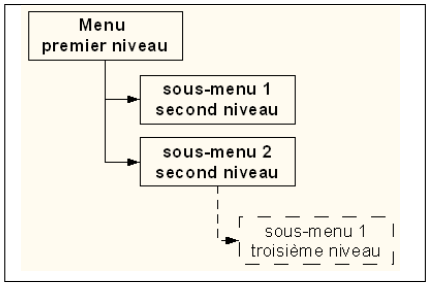



**Figure 9.16 :** *La représentation d'un menu tient plus d'un graphe que d'une liste. Chaque intitulé du menu peut être connecté à une fonction ou être le point d'entrée d'un nouveau sous-menu.*

Pour créer un menu ou un sous-menu, il suffit de créer un objet de type `Menu` :

```
m = Tkinter.Menu ()
```

Ce menu peut être le menu principal d'une fenêtre auquel cas, il suffit de préciser à la fenêtre en question que son menu est le suivant :

```
root.config (menu = m)
```

`root` est ici la fenêtre principale mais ce pourrait être également une fenêtre de type `TopLevel` (voir paragraphe 9.5.1). Ce menu peut aussi être le sous-menu associé à un intitulé d'un menu existant. La méthode `add_cascade` permet d'ajouter un sous-menu associé à un label :

```
mainmenu  = Tkinter.Menu ()
msousmenu = Tkinter.Menu ()
mainmenu.add_cascade (label = "sous-menu 1", menu = msousmenu)
```

En revanche, si on souhaite affecter une fonction à un menu, on utilisera la méthode `add_command` :

```
def fonction1 () :
    ....
m = Tkinter.Menu ()
mainmenu.add_command (label = "fonction 1", command = fonction1)
```

L'exemple suivant regroupe les fonctionnalités présentées ci-dessus.

```
import Tkinter
root = Tkinter.Tk ()

e = Tkinter.Text (width = 50, height = 10)
e.pack ()

m = Tkinter.Menu ()

sm1 = Tkinter.Menu ()
sm2 = Tkinter.Menu ()

m.add_cascade (label = "sous-menu 1", menu = sm1)
m.add_cascade (label = "sous-menu 2", menu = sm2)

nb = 0

def affiche () : print "fonction affiche"
def calcul () : print "fonction calcul ", 3 * 4
def ajoute_bouton () :
    global nb
    nb += 1
    b = Tkinter.Button (text = "bouton " + str (nb))
    b.pack ()

sm1.add_command (label = "affiche",       command = affiche)
sm1.add_command (label = "calcul",        command = calcul)
sm2.add_command (label = "ajoute_bouton", command = ajoute_bouton)
sm2.add_command (label = "fin",           command = root.destroy)

root.config (menu = m, width = 200)
root.titl ("essai de menu")
#help (Tkinter.Tk)
root.mainloop ()
```




**Figure 9.17 :** *Résultat de l'exemple du paragraphe 9.4.5, la sélection de différents intitulés du menu permet d'afficher des choses avec l'instruction* `print` *ou d'ajouter des boutons.*

Chaque intitulé d'un menu est ajouté en fin de liste, il est possible d'en supprimer certains à partir de leur position avec la méthode `delete` :

```
m = Tkinter.Menu ()
m.add_command (...)
m.delete (1, 2) # supprime le second intitulé
                # supprime les intitulés compris entre 1 et 2 exclu
```

### 9.4.6 Fonctions prédéfinies

Il est possible de détruire la fenêtre principale, ce qui mettra fin au programme si celui-ci ne prévoit rien après la fonction `mainloop`. La destruction de la fenêtre s'effectue par la méthode `destroy`. Le programme suivant crée une fenêtre avec un seul bouton qui, s'il est pressé, mettra fin à l'application.

```
import Tkinter
root = Tkinter.Tk ()
Tkinter.Button (text = "fin", command = root.destroy).pack ()
root.mainloop ()
```

La table 9.3 regroupe les fonctions les plus utilisées. Celles-ci s'applique à une fenêtre de type `Toplevel` qui est aussi le type de la fenêtre principale.

| | |
|---|---|
| `destroy`() | Détruit la fenêtre. |
| `deiconify`() | La fenêtre reprend une taille normale. |
| `geometry`(`s`) | Modifie la taille de la fenêtre. `s` est une chaîne de caractères de type "`wxhxy`". `w` et `h` sont la largeur et la hauteur. `x` et `y` sont la position du coin supérieur haut à l'écran. |
| `iconify`() | La fenêtre se réduit à un icône. |
| `resizable`(`w`,`h`) | Spécifie si la fenêtre peut changer de taille. `w` et `h` sont des booléens. |
| `title`(`s`) | Change le titre de la fenêtre, `s` est une chaîne de caractères. |
| `withdraw`() | Fait disparaître la fenêtre. |

**Table 9.3 :** *Liste non exhaustives des méthodes de la classe* `Toplevel` *qui le type de la fenêtre principale.*

## 9.5 D'autres fenêtres

### 9.5.1 Créer une seconde boîte de dialogues

Lorsqu'un programme doit utiliser plusieurs fenêtres et non pas une seule, l'emploi de l'objet `Toplevel` est inévitable. L'instruction `root` = `Tkinter.Tk`() crée la fenêtre principale, l'instruction `win` = `Tkinter.Toplevel`() crée une seconde fenêtre qui fonctionne exactement comme la fenêtre principale puisqu'elle dispose elle aussi d'une boucle de messages via la méthode `mainloop`.

```
import Tkinter
win = Tkinter.Toplevel ()
win.mainloop ()
```

Un cas d'utilisation simple est par exemple un bouton pressé qui fait apparaître une fenêtre permettant de sélectionner un fichier, cette seconde fenêtre sera un objet `Toplevel`. Il n'est pas nécessaire de s'étendre plus sur cet objet, son comportement est identique à celui de la fenêtre principale, les fonctions décrites au paragraphe 9.4.6 s'appliquent également aux objets `Toplevel`. Il reste néanmoins à préciser un dernier point. Tous les objets précédemment décrits au paragraphe 9.2 doivent inclure un paramètre supplémentaire dans leur constructeur pour signifier qu'ils appartiennent à un objet `Toplevel` et non à la fenêtre principale. Par exemple, pour créer une zone de texte, la syntaxe est la suivante :

```
# zone_texte appartient à la fenêtre principale
zone_texte = Tkinter.Label (text = "premier texte")
```

Pour l'inclure à une fenêtre `Toplevel`, cette syntaxe devient :

```
# zone_texte appartient à la fenêtre top
top = Tkinter.Toplevel ()
zone_texte = Tkinter.Label (top, text = "premier texte")
```

Lors de la définition de chaque objet ou *widget*, si le premier paramètre est de type `Toplevel`, alors ce paramètre sera affecté à la fenêtre passée en premier argument et non à la fenêtre principale. Ce principe est le même que celui de la sous-fenêtre `Frame` (voir paragraphe 9.3.2, page 187). La seule différence provient du fait que l'objet `Toplevel` est une fenêtre autonome qui peut attendre un message grâce à la méthode `mainloop`, ce n'est pas le cas de l'objet `Frame`.

Toutefois, il est possible d'afficher plusieurs fenêtre `Toplevel` simultanément. Le programme suivant en est un exemple :

```
#-*- coding: latin-1 -*-
import Tkinter

class nouvelle_fenetre :
    resultat = []
    def top (self) :
        sec = Tkinter.Toplevel ()
        Tkinter.Label (sec, text="entrer quelque chose").pack ()
        saisie = Tkinter.Entry (sec)
        saisie.pack()
        Tkinter.Button (sec, text = "valider", command = sec.quit).pack ()
        sec.mainloop ()
        nouvelle_fenetre.resultat.append ( saisie.get () )
        sec.destroy ()

root = Tkinter.Tk() #fenetre principale
a = Tkinter.Button (text    = "fenêtre Toplevel",
                    command = nouvelle_fenetre ().top)
a.pack()
root.mainloop()

for a in nouvelle_fenetre.resultat :
    print "contenu ", a
```

### 9.5.2 Fenêtres standard

Le module `Tix` propose une fenêtre de sélection de fichiers identique à celle de la figure 9.18. `Tkinter` a l'avantage d'être simple et ne nécessite pas un long apprentissage pour le maîtriser mais il est limité. Pour ce type de fenêtres qu'on retrouve dans la plupart des programmes, il existe presque toujours des solutions toutes faites, via le module `Tix` ou `Pmw`[11]. On trouve également de nombreux programmes sur Internet par le biais de moteurs de recherche. Le programme ci-dessous affiche une fenêtre qui permet de sélectionner un fichier.

```
import Tix as Tk
root = Tk.Tk ()

def command_print () : print box.cget("value")

box = Tk.FileSelectBox (root)
box.config (directory="c:\\")
box.pack ()
Tk.Button (root, text = "print", command = command_print).pack ()

root.mainloop ()
```

11. http://pmw.sourceforge.net/

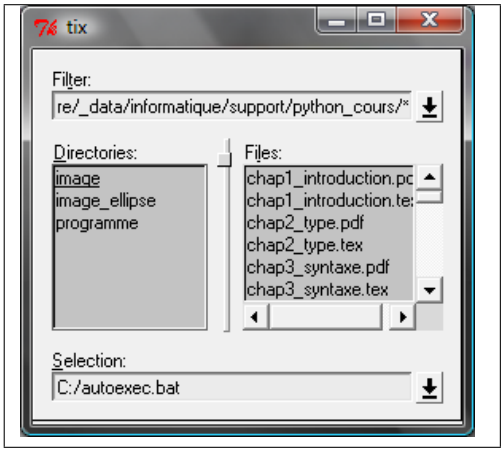



**Figure 9.18 :** *Exemple d'une fenêtre graphique. Celle-ci permet de sélectionner un fichier. C'est une fenêtre prédéfinie du module* `Tix` *qui est une extension de* `Tkinter`.

## 9.6 Constructions classiques

L'objectif des paragraphes qui suivent est d'introduire quelques schémas de construction d'interfaces qui reviennent fréquemment. La première règle de programmation qu'il est préférable de suivre est d'isoler la partie interface du reste du programme. La gestion événementielle a pour défaut parfois de disséminer un traitement, un calcul à plusieurs endroits de l'interface. C'est le cas par exemple de longs calculs dont on souhaite connaître l'avancée. Le calcul est lancé par la pression d'un bouton puis son déroulement est "espionné" par un événement régulier comme un compte à rebours.

Le principal problème des interfaces survient lors du traitement d'un événement : pendant ce temps, l'interface n'est pas réactive et ne réagit plus aux autres événements jusqu'à ce que le traitement de l'événement en cours soit terminé. Pour contourner ce problème, il est possible soit de découper un calcul en petites fonctions chacune très rapide, cela suppose que ce calcul sera mené à bien par une succession d'événements. Il est également possible de lancer un thread, principe décrit au paragraphe 10.3 (page 207).

C'est pourquoi la première règle est de bien scinder interface et calculs scientifiques de façon à pouvoir rendre le programme plus lisible et ainsi être en mesure d'isoler plus rapidement la source d'une erreur. Les paragraphes qui suivent présentent quelques aspects récurrents qu'il est parfois utile d'avoir en tête avant de se lancer.

### 9.6.1 Compte à rebours

Il est possible de demander à un objet d'appeler une fonction après un certains laps de temps exprimé un millisecondes. Le programme suivant crée un objet de type `Label`. Il contient une fonction qui change son contenu et lui affecte un compte à rebours qui impose à l'objet de rappeler cette fonction 1000 millisecondes plus tard. Le résultat est un programme qui crée la fenêtre 9.19 et change son contenu toutes les secondes.

```
import Tkinter
root = Tkinter.Tk ()
l = Tkinter.Label (text = "0 secondes")
l.pack ()
sec = 0
id = None

def change_legende() :
    global l
    global sec
    global id
    sec += 1
    l.config (text = "%d secondes" % sec)
    id = l.after (1000, change_legende)
```

```
l.after (1000, change_legende)

root.mainloop ()
```




**Figure 9.19 :** *Résultat de l'exemple du paragraphe 9.6.1, l'intitulé de l'objet* `Label` *change toutes les secondes.*

La méthode `after` retourne un entier permettant d'identifier le compte à rebours qu'il est possible d'interrompre en utilisant la méthode `after_cancel`. Dans l'exemple précédent, il faudrait utiliser l'instruction suivante :

```
    l.after_cancel (id)
```

### 9.6.2 Contrôles personnalisés : utilisation des classes

On peut personnifier un contrôle. Par exemple, on peut mettre en évidence l'intitulé d'une liste sous le curseur de la souris. Le moyen le plus simple est de créer une nouvelle classe qui se substituera au classique `Listbox`. Il suffit que cette nouvelle classe hérite de `Listbox` en prenant soin de lui donner un constructeur reprenant les mêmes paramètres que celui de la classe `Listbox`. De cette façon, il suffit de remplacer `Listbox` par `MaListbox` pour changer l'apparence d'une liste.

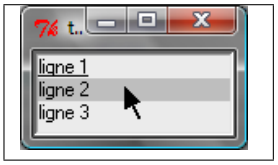
```
# -*- coding: cp1252 -*-
import Tkinter

class MaListbox (Tkinter.Listbox) :
    def __init__ (self, master = None, cnf={}, **kw) :
        Tkinter.Listbox.__init__ (self, master, cnf, **kw)
        self.bind ("<Motion>", self.mouvement)
        self.pos = None  # mémoire l'ancienne position du curseur
    def mouvement (self, ev) :
        pos = self.nearest (ev.y)  # nouvelle position du curseur
        if pos < 0 or pos >= self.size () : return
        if self.pos != pos :
            if self.pos != None : self.itemconfigure(self.pos, bg='')
            self.itemconfigure (pos, bg='gray')
            self.pos = pos

root = Tkinter.Tk ()
b = MaListbox ()
b.insert ("end", "ligne 1")
b.insert ("end", "ligne 2")
b.insert ("end", "ligne 3")
b.pack ()
b.focus_set ()
root.mainloop ()
```

Dans ce cas précis, on fait en sorte que le contrôle intercepte le mouvement du curseur. Lorsque celui-ci bouge, la méthode `mouvement` est appelée comme le constructeur de `MaListbox` l'a spécifié. La méthode `nearest` permet de définir l'intitulé le plus proche du curseur. La méthode `itemconfigure` permet de changer le fond de cet intitulé en gris après avoir modifié le fond de l'intitulé précédent pour qu'il retrouve sa couleur d'avant. Le résultat est illustré la figure 9.20.

**Figure 9.20 :** *Aspect de la classe* `MaListbox` *définie par le programme 197. L'intitulé sous le curseur de la souris a un fond gris.*




### 9.6.3 Fenêtres personnalisées : utilisation des classes

Cet exemple prolonge l'idée du paragraphe précédent. Lorsque l'interface devient complexe, il peut être utile de créer ses propres fenêtres. Jusqu'à présent, seules des fonctions ont été attachées à événement comme la pression d'un bouton mais il est possible d'attacher la méthode d'une classe ce que développe l'exemple qui suit.

```
# -*- coding: cp1252 -*-
import Tkinter as Tk

class MaFenetre :
    def __init__ (self, win) :
        self.win = win
        self.creation ()

    def creation (self) :
        b1 = Tk.Button (self.win, text="bouton 1", command=self.commande_bouton1)
        b2 = Tk.Button (self.win, text="bouton 2", command=self.commande_bouton2)
        b3 = Tk.Button (self.win, text="disparition", command=self.disparition)
        b1.grid (row=0, column=0)
        b2.grid (row=0, column=1)
        b3.grid (row=0, column=2)
        self.lab = Tk.Label (self.win, text = "-")

    def commande_bouton1 (self) :
        # on déplace l'objet lab de type Label
        self.lab.configure (text = "bouton 1 appuyé")
        self.lab.grid (row = 1, column = 0)

    def commande_bouton2 (self) :
        # on déplace l'objet lab de type Label
        self.lab.configure (text = "bouton 2 appuyé")
        self.lab.grid (row = 1, column = 1)

    def disparition (self) :
        # on fait disparaître l'objet lab de type Label
        self.lab.grid_forget ()

if __name__ == "__main__" :
    root = Tk.Tk ()
    f = MaFenetre (root)
    root.mainloop ()
```

Ce programme crée trois boutons et attache à chacun d'entre eux une méthode de la classe `MaFenetre`. Le constructeur de la classe prend comme unique paramètre un pointeur sur un objet qui peut être la fenêtre principale, un objet de type `Frame` ou `Toplevel`. Cette construction permet de considérer cet ensemble de trois boutons comme un objet à part entière ; de ce fait il peut être inséré plusieurs fois comme le montre l'exemple suivant illustré par la figure 9.21.

```
    root = Tk.Tk ()
    f = Tk.Frame ()
    f.pack ()
    MaFenetre (f)      # première instance
    g = Tk.Frame ()
```

```
    g.pack ()
    MaFenetre (g)      # seconde instance
    root.mainloop ()
```

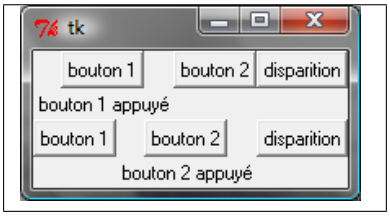



**Figure 9.21 :** *Aspect de la classe* `MaFenetre` *définie au paragraphe 9.6.3. La fenêtre est en fait composée de deux instances de* `MaFenetre`.

### 9.6.4 Séquence d'événements

Il est facile de faire réagir le programme en fonction d'un événement, il suffit d'attacher cet événement à une méthode ou une fonction. En revanche, faire réagir le programme en fonction d'une séquence d'événements est plus complexe. En effet, le premier d'événement de la séquence active une fonction, il n'est pas possible d'attendre le second événement dans cette même fonction, ce dernier ne sera observable que si on sort de cette première fonction pour revenir à la fonction `mainloop`, la seule capable de saisir le prochain événement.

La figure 9.22 précise la gestion des messages. `Tkinter` se charge de la réception des messages puis de l'appel au traitement correspondant indiqué par la méthode ou la fonction attachée à l'événement. Le programmeur peut définir les traitements associés à chaque événement. Ces deux parties sont scindées et à moins de reprogrammer sa boucle de message, il n'est pas évident de consulter les événements intervenus depuis le début du traitement de l'un d'eux.

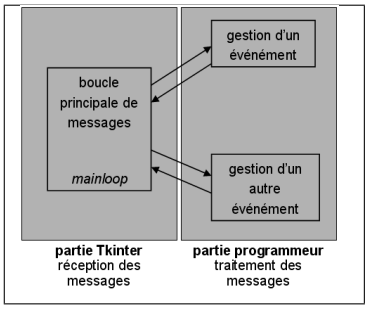



**Figure 9.22 :** *La réception des événements est assurée par la fonction* `mainloop` *qui consiste à attendre le premier événement puis à appeler la fonction ou la méthode qui lui est associée si elle existe.*

Les classes offrent un moyen simple de gérer les séquences d'événements au sein d'une fenêtre. Celle-ci fera l'objet d'une classe qui mémorise les séquences d'événements. Tous les événements feront appel à des méthodes différentes, chacune d'elles ajoutera l'événement à une liste. Après cette ajout, une autre méthode sera appelée pour rechercher une séquence d'événements particulière. Le résultat est également illustré par la figure 9.23.

```
# -*- coding: cp1252 -*-
import Tkinter as Tk

class MaFenetreSeq :
    def __init__ (self, win) :
        self.win = win
        self.creation ()
        self.sequence = []

    def creation (self) :
        b1 = Tk.Button (self.win, text="bouton 1", command=self.commande_bouton1)
        b2 = Tk.Button (self.win, text="bouton 2", command=self.commande_bouton2)
        b3 = Tk.Button (self.win, text="remise à zéro", command=self.zero)
```

```
        b1.grid (row=0, column=0)
        b2.grid (row=0, column=1)
        b3.grid (row=0, column=2)
        self.lab = Tk.Label (self.win, text = "-")

    def commande_bouton1 (self) :
        # ajoute 1 à la liste self.sequence
        self.sequence.append (1)
        self.controle ()

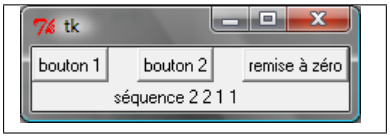
    def commande_bouton2 (self) :
        # ajoute 2 à la liste self.sequence
        self.sequence.append (2)
        self.controle ()

    def zero (self) :
        # on vide la liste self.sequence
        self.sequence = []
        self.lab.grid_forget ()

    def controle (self) :
        # on compare la liste sequence à [1,2,1] et [2,2,1,1]
        # dans ce cas, on fait apparaître l'objet lab
        l = len (self.sequence)
        if l >= 3 and self.sequence [l-3:] == [1,2,1] :
            self.lab.configure (text = "séquence 1 2 1")
            self.lab.grid (row = 1, column = 0)
        elif l >= 4 and self.sequence [l-4:] == [2,2,1,1] :
            self.lab.configure (text = "séquence 2 2 1 1")
            self.lab.grid (row = 1, column = 1)

if __name__ == "__main__" :
    root = Tk.Tk ()
    f = MaFenetreSeq (root)
    root.mainloop ()
```

**Figure 9.23 :** *Aspect de la classe* `MaFenetreSeq` *définie au paragraphe 9.6.4.*




### 9.6.5 Communiquer un résultat par message

Le module `Tkinter` permet de définir ses propres messages qui peuvent servir à communiquer des informations. Une fonction est par exemple appelée lorsqu'un bouton est pressé. Celle-ci, une fois terminée, retourne son résultat sous forme de message envoyé à l'interface graphique. Ce message sera ensuite traité comme tout autre message et pourra être intercepté ou non.

Le programme suivant utilise ce concept. La pression d'un bouton appelle une fonction `event_generate` qui génère un message personnalisé << `perso` >> avec comme paramètre `rooty` $= -5$. A son tour, celui-ci est attrapé et dirigé vers la fonction `perso` qui affiche l'attribut `y_root` de la classe `Event` qui a reçu la valeur $-5$ lors de l'appel de la fonction `event_generate`. Ce procédé ne permet toutefois que de renvoyer que quelques résultats entiers.

```
# -*- coding: cp1252 -*-
import Tkinter
def affiche_touche_pressee () : root.event_generate ("<<perso>>", rooty = -5)
def perso (evt) : print "perso", evt.y_root
root = Tkinter.Tk ()
```

```
b = Tkinter.Button (text = "appuyer sur une touche", \
                    command = affiche_touche_pressee)
b.pack ()
root.bind ("<<perso>>", perso)  # on intercepte un événement personnalisé
root.mainloop ()
```

Ce principe est plus utilisé lorsque l'interface graphique est couplée avec les threads, l'ensemble est présenté au paragraphe 10.3.

# Chapitre 10

# Threads

---

Jusqu'aux années 2003-2004, l'évolution des microprocesseurs était une course vers une augmentation de la puissance, autant en terme de nombre de transistors qu'en fréquence de fonctionnement. Arrivant aux limites de la technologie actuelle, cette évolution s'est tournée maintenant vers la construction de processeurs multicœurs, c'est-à-dire des machines capables d'exécuter des programmes simultanément, de maintenir plusieurs *fils d'exécution* en parallèle.

Les threads ou fils d'exécution ont trois usages principaux. Le premier est relié au calcul distribué ou calcul parallèle. Par exemple, le calcul d'une intégrale sur un intervalle peut être effectué sur deux intervalles disjoints. Le résultat final est la somme des deux résultats sur chacun des intervalles. De plus, ces deux calculs sont indépendants et peuvent être menés de front. Le calcul intégral sera donc deux fois plus rapide puisque les deux intervalles seront traités en même temps. C'est la parallélisation des calculs : les deux calculs sur chaque intervalle seront affectés à deux threads simultanés.

Le second usage est couplé aux interfaces graphiques. Lorsque l'utilisateur entame un processus long après avoir cliqué sur un bouton, l'interface graphique ne réagit plus jusqu'à ce que ce processus s'achève. Afin d'éviter cet inconvénient, l'interface graphique va commencer un thread qui va exécuter ce processus. L'interface graphique n'a plus qu'à attendre la fin du thread, et pendant tout ce temps, elle sera également capable de traiter tout autre événement provenant de l'utilisateur.

Le dernier usage concerne la communication entre ordinateurs ou plus généralement la communication Internet. C'est une communication asynchrone : l'ordinateur effectue des tâches en même temps qu'il écoute un *port* par lequel d'autres ordinateurs communiquent avec lui. Plus précisément, le programme suit deux fils d'exécution : le fil principal et un thread qui ne fait qu'attendre et traiter les messages qu'il reçoit via un port.

La synchronisation est un point commun à ces trois usages. Ce terme désigne la dépendance entre les threads. Lors d'un calcul distribué, le résultat final dépend des résultats retournés par chaque thread, il faut donc attendre que les deux fils d'exécution aient produit le résultat attendu : il faut que les deux fils d'exécution se synchronisent.

Ce document ne s'étendra pas longuement sur les threads bien qu'ils soient amenés à devenir un élément incontournable de tout programme désirant tirer parti des derniers processeurs. Il faudra sans doute attendre encore quelques années pour que les langages de programmation incorpore des schémas plus élaborés que ceux actuellement disponibles et sommairement présentés dans les paragraphes qui suivent.

## 10.1 Premier thread

Le premier exemple consiste à exécuter un thread uniquement destiné à faire des affichages. Deux fils d'exécution vont être lancés en parallèle affichant chacun un message différent. Les affichages vont s'entremêler. Il existe plusieurs manières d'exécuter un thread, une seule sera présentée en utilisant la classe `Thread` du module `threading`. Pour créer un thread, il suffit de surcharger la méthode `run` de la la classe `Thread`. Si le thread a besoin de données lors de son exécution, il faut surcharger son constructeur sans oublier d'appeler le constructeur de la classe mère. L'exécution de thread commence par la création d'une instance et l'appel à la méthode `start`. En résumé, il faut retenir les éléments suivants :

1. surcharger la classe `threading.Thread`,

2. surcharger le constructeur sans oublier d'appeler le constructeur `threading.Thread.__init__`,
3. surcharger la méthode `run`, c'est le code que devra exécuter le thread,
4. créer une instance de la nouvelle classe et appeler la méthode `start` pour lancer le thread secondaire qui formera le second fil d'exécution.

Le programme principal est appelé le thread principal. Voici ce que cela donne dans un exemple :

```
# -*- coding: cp1252 -*-
import threading, time

class MonThread (threading.Thread) :
    def __init__ (self, jusqua) :      # jusqua = donnée supplémentaire
        threading.Thread.__init__(self)# ne pas oublier cette ligne
                                       # (appel au constructeur de la classe mère)
        self.jusqua = jusqua           # donnée supplémentaire ajoutée à la classe

    def run (self) :
        for i in range (0, self.jusqua) :
            print "thread ", i
            time.sleep (0.1)    # attend 100 millisecondes sans rien faire
                                # facilite la lecture de l'affichage

m = MonThread (1000)        # crée le thread
m.start ()                  # démarre le thread,
                            # l'instruction est exécutée en quelques millisecondes
                            # quelque soit la durée du thread

for i in range (0,1000) :
    print "programme ", i
    time.sleep (0.1)            # attend 100 millisecondes sans rien faire
                                # facilite la lecture de l'affichage
```

Le programme affiche des lignes qui proviennent du thread principal et du thread secondaire dont les affichages diffèrent.

```
programme  0
thread  0
thread  1
programme  1
thread  2
programme  2
programme  3
thread  3
programme  4
thread  4
...
```

Le précédent programme a été adapté pour lancer deux threads secondaires en plus du thread principal. Les lignes modifiées par rapport au programme précédent sont commentées.

```
# -*- coding: cp1252 -*-
import threading, time

class MonThread (threading.Thread) :
    def __init__ (self, jusqua, s) :      # jusqua,s = données supplémentaires
        threading.Thread.__init__ (self)
        self.jusqua = jusqua              # données supplémentaires
        self.s = s                        # ajoutées à la classe

    def run (self) :
        for i in range (0, self.jusqua) :
            print "thread ", self.s, " : ", i
            time.sleep (0.1)
```

```
m = MonThread (1000, "A")   # crée un thread
m.start ()                  # démarre le thread,
                            # l'instruction est exécutée en quelques millisecondes
                            # quelque soit la durée du thread

m2 = MonThread (1000, "B")  # crée un second thread
m2.start ()                 # démarre le thread,
                            # l'instruction est exécutée en quelques millisecondes
                            # quelque soit la durée du thread

for i in range (0,1000) :
    print "programme ", i
    time.sleep (0.1)
```

```
thread  A  :  0
programme  0
thread  B  :  0
thread  A  :  1
thread  B  :  1
programme  1
thread  B  :  2
thread  A  :  2
...
```

**Remarque 10.1 : utilisation le la fonction `sleep`**
Tous les exemples présentés dans ce chapitre font souvent intervenir l'instruction `time.sleep(...)`. A moins que ce ne soit explicitement précisé, elle sert la plupart du temps à ralentir l'exécution du programme cité en exemple afin que celle-ci soit humainement observable ou pour exagérer un défaut de synchronisation. Cette fonction est d'ordinaire beaucoup moins fréquente.

## 10.2 Synchronisation

### 10.2.1 Attente

La première situation dans laquelle on a besoin de synchroniser deux threads est l'attente d'un thread secondaire par le thread principal. Et pour ce faire, on a besoin de l'accès par les deux fils d'exécution à une même variable qui indiquera l'état du thread. Dans le programme suivant, on ajoute l'attribut `etat` à la classe `MonThread` qui va indiquer l'état du thread :

- `True` pour en marche
- `False` pour à l'arrêt

Le thread principal va simplement vérifier l'état du thread de temps en temps. Le premier point important est tout d'abord d'attendre que le thread se lance car sans la première boucle, le thread pourrait passer à l'état `True` après être passé dans la seconde boucle d'attente. Le second point important est de ne pas oublier d'insérer la fonction `sleep` du module `time` afin de permettre au thread principal de temporiser. Dans le cas contraire, le thread principal passe l'essentiel de son temps à vérifier l'état du thread secondaire, ce faisant, il ralentit l'ordinateur par la répétition inutile de la même action un trop grand nombre de fois. Ici, le thread principal vérifie l'état du thread secondaire tous les 100 millisecondes. Cette durée dépend de ce que fait le thread secondaire.

```
# -*- coding: cp1252 -*-
import threading, time

class MonThread (threading.Thread) :
    def __init__ (self, jusqua) :
        threading.Thread.__init__ (self)
        self.jusqua = jusqua
        self.etat = False       # l'état du thread est soit False (à l'arrêt)
                                # soit True (en marche)

    def run (self) :
        self.etat = True                        # on passe en mode marche
        for i in range (0, self.jusqua) :
```

```
            print "thread itération ", i
            time.sleep (0.1)
        self.etat = False                       # on revient en mode arrêt

m = MonThread (10)          # crée un thread
m.start ()                  # démarre le thread,

print "début"

while m.etat == False :
    # on attend que le thread démarre
    time.sleep (0.1)  # voir remarque ci-dessous

while m.etat == True :
    # on attend que le thread s'arrête
    # il faut introduire l'instruction time.sleep pour temporiser, il n'est pas
    # nécessaire de vérifier sans cesse que le thread est toujours en marche
    # il suffit de le vérifier tous les 100 millisecondes
    # dans le cas contraire, la machine passe son temps à vérifier au lieu
    # de se consacrer à l'exécution du thread
    time.sleep (0.1)

print "fin"
```

Ce mécanisme d'attente peut également être codé en utilisation les objets `Condition` et `Event` du module `threading`. Ces deux objets permettent d'éviter l'utilisation de la méthode `sleep`.

```
# -*- coding: cp1252 -*-
import threading, time

class MonThread (threading.Thread) :
    def __init__ (self, jusqua, event) :    # event = objet Event
        threading.Thread.__init__ (self)    #       = donnée supplémentaire
        self.jusqua = jusqua
        self.event  = event                 # on garde un accès à l'objet Event

    def run (self) :
        for i in range (0, self.jusqua) :
            print "thread itération ", i
            time.sleep (0.1)
        self.event.set ()                   # on indique qu'on a fini :
                                            # on active l'object self.event
print "début"

event = threading.Event ()          # on crée un objet de type Event
event.clear ()                      # on désactive l'ojet Event
m = MonThread (10, event)           # crée un thread
m.start ()                          # démarre le thread,
event.wait ()                       # on attend jusqu'à l'objet soit activé
                                    # event.wait (0.1) : n'attend qu'un
print "fin"                         #          seulement 1 dizième de seconde
```

La méthode `wait` de l'objet `Event` attend que l'objet soit activé. Elle peut attendre indéfiniment ou attendre pendant une durée donnée seulement. Pour afficher la durée d'attente, on pourrait utiliser une boucle comme la suivante :

```
m.start ()
while not event.isSet ():
    print "j'attends"
    event.wait (0.1)
print "fin"
```

La méthode `isSet` permet de savoir si l'événement est bloquant ou non. Le programme affiche "`j'attends`" puis attend le thread un dixième de secondes. Au delà de cette durée, il vérifie l'état de l'événement puis recommence si le thread n'est pas fini.

Ces objets de synchronisation sont plus efficaces que le mécanisme décrit dans le premier programme car il fait appel aux fonctions du système d'exploitation.

### 10.2.2 Partage d'informations

La seconde situation dans laquelle on a besoin de synchroniser est l'accès par deux fils d'exécution aux mêmes informations ou plutôt aux mêmes variables. Un problème survient quand parfois un thread lit ou modifie en même temps qu'un autre modifie la même variable. Le second cas de synchronisation est l'ajout de verrous qui permettent de protéger une partie du code d'un programme contre plusieurs accès simultanés. Ce verrou est également un objet du module `threading`.

Dans cet exemple, l'information partagée est la chaîne de caractères `message`, le verrou sert à protéger la fonction `ajoute` contre des ajouts simultanés. Si les deux threads veulent modifier `message` en même temps, un thread va entrer dans la fonction `ajoute` alors que l'autre n'en est pas encore sorti. Les résultats seraient imprévisibles car cette fonction modifie la variable qu'ils utilisent. On aboutit à l'exemple suivant :

```
# -*- coding: cp1252 -*-
import threading, time

message = ""
verrou  = threading.Lock ()

def ajoute (c) :
    global message     # message et verrou sont des variables gloables
    global verrou      # pour ne pas qu'elle disparaisse dès la fin de la fonction
    verrou.acquire ()  # on protège ce qui suit  (*)

    s = message + c    # instructions jamais exécutée simultanément par 2 threads
    time.sleep (0.001) # time.sleep : pour exagérer le défaut de synchronisation
    message = s        # si verrou n'est pas utilisé

    verrou.release ()  # on quitte la section protégée  (*)

class MonThread (threading.Thread) :
    def __init__ (self, jusqua, event, s) :
        threading.Thread.__init__ (self)
        self.jusqua = jusqua
        self.s      = s
        self.event  = event

    def run (self) :
        for i in range (0, self.jusqua) :
            ajoute (self.s)
            time.sleep (0.003)
        self.event.set ()

print "début"

# synchronisation attente
e1 = threading.Event ()
e2 = threading.Event ()
e1.clear ()
e2.clear ()

m1 = MonThread (10, e1, "1")     # crée un thread
m1.start ()                      # démarre le thread,
m2 = MonThread (10, e2, "2")     # crée un second thread
m2.start ()                      # démarre le second thread,

e1.wait ()
e2.wait ()

print "longueur ", len(message) # affiche 20
print "message = ", message     # affiche quelque chose comme 12212112211212121221
```

Les trois instructions protégées pourraient être résumées en une seule : `message+ = c` ; le résultat resterait

inchangé. En revanche, en commentant les instructions `verrou.acquire()` et `verrou.release()` de ce programme [1], la longueur du résultat final `message` est variable alors qu'elle devrait être de 20 puisque les deux threads appellent chacun 10 fois dans la fonction `ajoute`. Le tableau suivant montre l'évolution des variables `message`, `c`, `s` durant deux premiers appels qui s'entremêlent. Le résultat devrait être "12" pour message mais un caractère a été perdu. Il faut retenir que si la variable `message` est globale, les deux autres `c`, `s` sont locales et donc différentes pour les deux threads.

| ordre | thread 1 | thread 2 | message | c | s |
|---|---|---|---|---|---|
| 1 | s = message + c | | "" | "1" | "1" |
| 2 | | s = message + c | "" | "2" | "2" |
| 3 | time.sleep(0.001) | | "" | "1" | "1" |
| 4 | | time.sleep(0.001) | "" | "2" | "2" |
| 5 | message = s | | "1" | "1" | "1" |
| 6 | | message = s | "2" | "2" | "2" |

Le verrou empêche d'exécuter une même portion de code en même temps, un code qui modifie des données partagées. C'est pourquoi le verrou est souvent déclaré au même endroit que les données qu'il protège. Le verrou de type `Lock` n'autorise qu'un seul thread à la fois à l'intérieur de la portion de code protégée ce qui aboutit au schéma suivant :

| ordre | thread 1 | thread 2 | message | c | s |
|---|---|---|---|---|---|
| 1 | s = message + c | | "" | "1" | "1" |
| 2 | time.sleep(0.001) | | "" | "1" | "1" |
| 3 | message = s | | "1" | "1" | "1" |
| 4 | | s = message + c | "1" | "2" | "12" |
| 5 | | time.sleep(0.001) | "1" | "2" | "12" |
| 6 | | message = s | "12" | "2" | "12" |

**Remarque 10.2 : réduction des accès à quelques threads**
Le verrou de type `Semaphore` autorise un nombre maximal de thread à parcourir le même code. Ce procédé est parfois utile si le code en question permet d'imprimer un document. Cela permet de limiter sans interdire les accès simultanés aux ressources de l'ordinateur.

**Remarque 10.3 : blocage d'un programme**
Ce mécanisme de verrou peut aboutir à des blocages avec deux threads et deux portions de code protégées. Chaque thread est "coincé" dans une section attendant que l'autre libère la sienne. Dans ce cas de figure, il est conseillé d'utiliser le même verrou pour protéger les deux sections. Ainsi, chaque thread ne pourra pas entrer dans l'une ou l'autre des portions de code protégées tant que l'une d'entre elles est visitée par l'autre thread.

## 10.3 Interface graphique

Un programme bâti autour d'une interface graphique inclut nécessairement une boucle de message. Celle-ci attend les messages en provenance de l'interface. Lorsqu'un de ceux-ci lui commande de lancer un traitement long, l'interface graphique n'est plus en mesure de réagir aux événements qui lui viennent pendant ce temps. Afin de remédier cela, il suffit d'insérer le traitement dans un thread. A la fin de ce dernier, un événement sera envoyé à l'interface afin de lui signifier la fin du traitement.

Le paragraphe 9.4.4 a montré comment associer un événement particulier à une fenêtre. La différence ici est que l'événement accroché à la fenêtre n'est pas prédéfini par le module `Tkinter` mais par le programme lui-même - dans cet exemple << `thread_fini` >> [2] -. La méthode `event_generate` permet d'insérer un

1. Celles marquées d'une étoile (∗).
2. Les symboles << et >> au début et à la fin du nom de l'événement sont la seule contrainte.

message dans la boucle de messages de façon à ce que celui-ci soit traité au même titre qu'un clic de souris, la pression d'une touche, ...

```
# -*- coding: cp1252 -*-
import threading, time, random, copy

# définition du thread
class MonThread (threading.Thread) :
    def __init__ (self, win, res) :
        threading.Thread.__init__ (self)
        self.win = win  # on mémorise une référence sur la fenêtre
        self.res = res

    def run (self) :
        for i in range (0, 10) :
            print "thread ", i
            time.sleep (0.1)

          # afin que le thread retourne un résultat
          # self.res désigne thread_resultat qui reçoit un nombre de plus
        h = random.randint (0,100)
        self.res.append (h)

          # on lance un événement <<thread_fini>> à la fenêtre principale
          # pour lui dire que le thread est fini, l'événement est ensuite
          # géré par la boucle principale de messages
          # on peut transmettre également le résultat lors de l'envoi du message
          # en utilisant un attribut de la classe Event pour son propre compte
        self.win.event_generate ("<<thread_fini>>", x = h)

thread_resultat = []

def lance_thread () :
    global thread_resultat
      # fonction appelée lors de la pression du bouton
      # on change la légnde de la zone de texte
    text .config (text = "thread démarré")
    text2.config (text = "thread démarré")
      # on désactive le bouton pour éviter de lancer deux threads en même temps
    bouton.config (state = TK.DISABLED)
      # on lance le thread
    m = MonThread (root, thread_resultat)
    m.start ()

def thread_fini_fonction (e) :
    global thread_resultat
      # fonction appelée lorsque le thread est fini
    print "la fenêtre sait que le thread est fini"
      # on change la légende de la zone de texte
    text .config (text = "thread fini + résultat " + str (thread_resultat))
    text2.config (text = "thread fini + résultat (e.x) " + str (e.x))
      # on réactive le bouton de façon à pouvoir lancer un autre thread
    bouton.config (state = TK.NORMAL)

import Tkinter as TK

# on crée la fenêtre
root   = TK.Tk ()
bouton = TK.Button (root, text = "thread départ", command = lance_thread)
text   = TK.Label (root, text = "rien")
text2  = TK.Label (root, text = "rien")
bouton.pack ()
text.pack ()
text2.pack ()

# on associe une fonction à un événement <<thread_fini>> propre au programme
```

```
root.bind ("<<thread_fini>>", thread_fini_fonction)

# on active la boucle principale de message
root.mainloop ()
```

La figure 10.1 contient la fenêtre affichée par le programme lorsqu'elle attend la pression du bouton qui lance le thread et lorsqu'elle attend la fin de l'exécution de ce thread.

**Remarque 10.4 : méthode `event_generate`**
Le programme précédent utilise une astuce pour retourner un résultat autrement qu'un utilisant un paramètre global. On peut adjoindre lors de l'appel à la méthode `event_generate` quelques informations supplémentaires attachées à l'événement en utilisant les attributs prédéfinis de la classe `Event`. Dans cet exemple, on utilise l'attribut `x` pour retourner le dernier entier tiré aléatoirement.

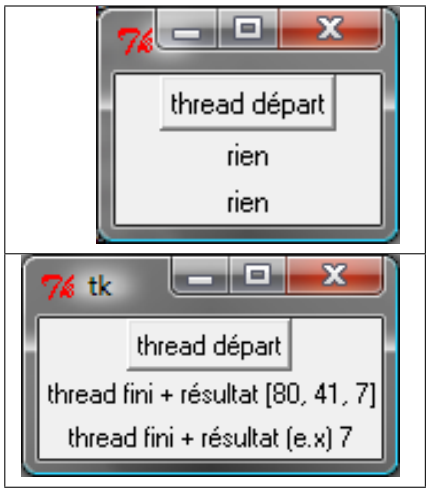



**Figure 10.1 :** *La première image est la fenêtre après trois exécutions du thread. La liste* `thread_resultat` *contient trois nombres. Une fois l'unique bouton pressé, la fenêtre change d'aspect pour devenir comme la seconde image. Cette transition est assurée par la fonction* `lance_thread` *reliée au bouton. La transition inverse est assurée par la fonction* `thread_fini_fonction` *qui est relié à l'événement que génère le thread lorsqu'il a terminé.*

## 10.4 Files de messages

Les trois usages principaux des threads sont le calcul distribué, la conception d'une interface graphique réactive et l'attente permanente d'événements. En ce qui concernent les deux premiers usages, on peut considérer qu'il existe un thread principal qui lance et attend l'exécution de threads secondaires. Les échanges d'informations ont lieu au début et à la fin de chaque thread. Il n'est pas toujours nécessaire de partager des variables en cours d'exécution : l'usage de verrous est peu fréquent pour ces deux schémas sauf pour partager des informations en cours d'exécution.

En ce qui concerne le troisième usage, c'est un cas où tout au long du programme, il y a constamment au moins deux threads actifs : un thread principal et un thread qui attend. Dans ce cas, l'échange et la synchronisation d'informations est inévitable et il est souvent fastidieux de concevoir la synchronisation. C'est pourquoi on la conçoit presque toujours sous forme de messages que les threads s'envoient.

Il existe un objet `Queue` du module `Queue` qui facilite cet aspect. C'est une liste qui possède son propre verrou de sorte que n'importe quel thread peut ajouter ou retirer des éléments de cette liste. Elle est utilisée principalement via quatre méthodes. Deux méthodes `get` sont utilisées au sein du thread qui possède la pile. Deux méthodes `put` sont appelées par des threads étrangers.

| | |
|---|---|
| get([timeout = ...]) | Retourne un élément de la liste ou attend qu'il y en ait un, le supprime si elle en trouve un. Si timeout est renseigné, la fonction attend au plus timeout secondes, sinon, elle déclenche l'exception Queue.Empty. |
| get_nowait() | Retourne un élément de la liste s'il y a en un, dans ce cas, cet élément est supprimé. Dans le cas contraire, la méthode déclenche l'exception Queue.Empty. |
| put(e[, timeout = ...]) | Ajoute l'élément e à la liste ou attend qu'une place se libère si la liste est pleine. Si timeout est renseigné, la fonction attend au plus timeout secondes, sinon, elle déclenche l'exception Queue.Full. |
| put_nowait(e) | Ajoute l'élément e à la liste ou déclenche l'exception Queue.Full si la liste est pleine. |
| qsize() | Retourne la taille de la pile. |

Cette pile est utilisée dans l'exemple qui suit pour simuler deux joueurs qui essaye de découvrir le nombre que l'autre joueur a tiré au hasard. A chaque essai, un joueur envoie un message de type ("essai", n) à l'autre joueur pour dire qu'il joue le nombre n. Ce joueur lui répond avec des messages de type ("dessous", n), ("dessus", n), ("gagne", n).

```
# -*- coding: cp1252 -*-
import threading, time, Queue, random

class Joueur (threading.Thread) :

    # initialisation
    def __init__ (self, nom, e, nb = 1000, temps = 0.1) :
        threading.Thread.__init__(self)
        self.nb    = nb
        self.queue = Queue.Queue ()
        self.nom   = nom
        self.event = e
        self.temps = temps  # temps de réflexion
    def Joueur (self, autre_joueur) : self.autre = autre_joueur

    # méthodes : l'adversaire m'envoie un message
    def Joue    (self, nombre) : self.queue.put_nowait ( ("essai", nombre) )
    def Dessus  (self, nombre) : self.queue.put_nowait ( ("dessus", nombre) )
    def Dessous (self, nombre) : self.queue.put_nowait ( ("dessous", nombre) )
    def Gagne   (self, nombre) :
        while not self.queue.empty () :
            try :self.queue.get ()
            except : pass
        self.queue.put ( ("gagne", nombre) )

    # je joue
    def run (self) :
        x = random.randint (0,self.nb)
        print self.nom, " : je joue (", x, ")"
        i = 0
        a = 0
        b = self.nb
        while True :
            time.sleep (self.temps)

            try :
                m,n = self.queue.get_nowait ()       # désynchronisé
                #m,n = self.queue.get (timeout = 0.5)# l'un après l'autre
            except Queue.Empty : m,n = None,None

            # traitement du message --> réponse à l'adversaire
            if m == "essai" :
                if n == x :
                    self.autre.Gagne (n)
                    print self.nom, " : j'ai perdu après ", i, " essais"
```

```
                    break
                elif n < x : self.autre.Dessus  (n)
                else       : self.autre.Dessous (n)
            elif m == "dessus" :
                a = max (a, n+1)
                continue  # assure l'équité en mode l'un après l'autre
            elif m == "dessous" :
                b = min (b, n-1)
                continue  # assure l'équité en mode l'un après l'autre
            elif m == "gagne" :
                print self.nom, " : j'ai gagné en ", i, " essais, solution ", n
                break

            # on fait une tentative
            if a == b : n = a
            else : n = random.randint (a,b)
            self.autre.Joue (n)
            i += 1
            print self.nom, " : je tente ", n, " écart ", b-a, \
                  " à traiter ", self.queue.qsize ()

        # fini
        print self.nom, " : j'arrête"
        self.event.set ()

# on crée des verrous pour attendre la fin de la partie
e1 = threading.Event ()
e2 = threading.Event ()
e1.clear ()
e2.clear ()

# création des joueurs
A = Joueur ("A", e1, 1000, temps = 0.1)
B = Joueur ("B", e2, 1000, temps = 0.3)

# chaque joueur sait qui est l'autre
A.Joueur (B)
B.Joueur (A)

# le jeu commence
A.start ()
B.start ()

# on attend la fin de la partie
e1.wait ()
e2.wait ()
```

Si la méthode `get` est choisie, les joueurs doivent attendre une tentative de l'adversaire avant de proposer la leur. Dans l'autre cas, la méthode `get_nowait` permet de ne pas attendre sa réponse et d'envoyer plusieurs propositions à l'adversaire qui ne répondra pas plus vite pour autant. Dans cette configuration, le joueur `A` est trois fois plus réactif ce qui explique les résultats qui suivent.

```
A  : je joue ( 8 )
B  : je joue ( 569 )
A  : je tente  42  écart  1000  à traiter  0
A  : je tente  791  écart  1000  à traiter  0
...
A  : je tente  528  écart  62  à traiter  0
B  : je tente  20  écart  43  à traiter  57
A  : je tente  508  écart  62  à traiter  0
A  : je tente  548  écart  62  à traiter  0
B  : je tente  8  écart  43  à traiter  59
A  : j'ai perdu après  67  essais
A  : j'arrête
```

```
B  : j'ai gagné en  23  essais, solution  8
B  : j'arrête
```

Les affichages se chevauchent parfois, il faudrait pour éviter cela synchroniser l'affichage à l'aide d'un verrou.

Deuxième partie

# ENONCÉS PRATIQUES, EXERCICES

Cette partie regroupe quelques énoncés que les élèves de l'ENSAE ont étudié ou à partir desquels ils ont été évalués entre les années 2005 et 2008. On retient souvent que le boulot d'un informaticien n'est pas compliqué, qu'il est plutôt plutôt ennuyeux et redondant. On oublie parfois que sans l'informatique, certaines tâches seraient tout autant longues et fastidieuses. Cette partie présente certaines tournures d'esprits qui permettent de gagner du temps lors de la conception de programmes et peut-être derendre la programmation moins répétitive, le travail moins fastidieux.

Les premiers énoncés illustrent quelques schémas récurrents de programmation. Les langages de programmation qui nécessitent de reprogrammer un tri sont très rares, la plupart proposent des fonctions qui exécutent un tri rapide - ou quicksort -. Mais connaître certains algorithmes classiques permet de programmer plus rapidement qu'avec la seule connaissance des principes de base - boucle, test -. A l'instar de certains jeux de réflexion - les ouvertures avec les échecs -, il n'est pas nécessaire de réinventer certains algorithmes ou raisonnements fréquemment utilisés.

Cette partie se termine par trois années d'énoncés d'examens d'informatique. Les exercices proposés illustrent parfois le côté ennuyeux d'un programme informatique : sa mise en œuvre dépasse souvent de loin sa conception. La correction des erreurs ou *bugs* est la partie la plus longue et la plus rébarbative. Ces exercices ont pour objectif d'accélérer cette étape en proposant des situations qui reviennent fréquemment. C'est aussi pour cela que lors d'entretiens d'embauche, des exercices similaires sont présentés aux candidats pour vérifier leur expérience en programmation.

# Chapitre 11

# Exercices pratiques pour s'entraîner

---

## 11.1 Montant numérique, montant littéral

### 11.1.1 Enoncé

L'objectif de cet exercice est d'écrire un programme qui convertit en entier un nombre écrit sous forme littérale. On suppose que tous les nombres sont écrits en minuscules et correctement orthographiés. Cet exercice tourne autour des manipulations de chaînes de caractères et principalement les deux fonctions suivantes :

| **Fonctions utiles :** | |
|---|---|
| `replace(s, old, new)` | A l'intérieur de la chaîne `s`, remplace toutes les occurrences de la chaîne `old` par la chaîne `new`. Pour l'utiliser, il faut ajouter au début du programme la ligne `import string` puis écrire `l = string . replace ("dix - neuf", " - ", " ")`, par exemple. |
| `split(s[, sep])` | Retourne une liste contenant tous les mots de la chaîne de caractères `s`. Si le paramètre facultatif `sep` est renseigné, c'est cette chaîne qui fait office de séparateur entre mot et non les espaces. Pour l'utiliser, il faut ajouter au début du programme la ligne `import string` puis écrire `l = string . split ("vingt et un", " ")`, par exemple. |

L'appel à la fonction `replace` peut se faire en la considérant comme une méthode de la classe `str` ou une fonction du module `string`.

```
import string
s = "dix-neuf"

l = s.replace ("-", " ")
l = string.replace ("-", " ")
```

**1)** Ecrire une fonction qui prend un nombre littéral et retourne une liste contenant tous ses mots.

**2)** Ecrire une fonction qui reçoit une chaîne de caractères correspondant à un nombre compris entre 0 et 16 inclus ou à un nombre parmi 20, 30, 40, 50, 60. La fonction doit retourner un entier correspondant à sa valeur.

**3)** A l'aide de la fonction précédente, écrire une fonction qui reçoit une chaîne de caractères contenant un nombre compris entre 0 et 99 inclus et qui retourne sa valeur.

**4)** Pour vérifier que la fonction précédente marche bien dans tous les cas, écrire une fonction qui écrit un nombre sous sa forme littérale puis vérifier que ces deux fonctions marchent correctement pour les nombres compris entre 0 et 99.

### 11.1.2 Correction

A chaque question correspond une fonction utilisant celles décrites dans les questions précédentes. Le programme complet est obtenu en juxtaposant chaque morceau.

**1)**

```
# -*- coding: cp1252 -*-
def lire_separation(s):
    """divise un nombre littéral en mots"""
    s = string.replace (s, "-", " ")
    return string.split (s)
```

**2)** Il n'existe pas qu'une seule manière de rédiger cette fonction. La première, la plus simple, est sans doute la suivante :

```
def valeur_mot (s) :
    """convertit numériquement les nombres inclus entre 0 et 16 inclus,
    20, 30, 40, 50, 60, s est une chaîne de caractères, le résultat est entier"""
    if   s == "zéro"      : return 0
    elif s == "un"        : return 1
    elif s == "deux"      : return 2
    elif s == "trois"     : return 3
    elif s == "quatre"    : return 4
    elif s == "cinq"      : return 5
    elif s == "six"       : return 6
    elif s == "sept"      : return 7
    elif s == "huit"      : return 8
    elif s == "neuf"      : return 9
    elif s == "dix"       : return 10
    elif s == "onze"      : return 11
    elif s == "douze"     : return 12
    elif s == "treize"    : return 13
    elif s == "quatorze"  : return 14
    elif s == "quinze"    : return 15
    elif s == "seize"     : return 16
    elif s == "vingt"     : return 20
    elif s == "trente"    : return 30
    elif s == "quarante"  : return 40
    elif s == "cinquante" : return 50
    elif s == "soixante"  : return 60
    else                  : return 0      # ce cas ne doit normalement pas
                                          # se produire
```

La suivante utilise un dictionnaire. Cette écriture peut éviter un copier-coller long pour la dernière question.

```
def valeur_mot (s) :
    dico = {'cinquante': 50, 'quarante': 40, 'onze': 11, 'huit': 8, 'six': 6, \
            'quinze': 15, 'trente': 30, 'douze': 12, 'cinq': 5, 'deux': 2, \
            'quatorze': 14, 'neuf': 9, 'soixante': 60, 'quatre': 4, \
            'zéro': 0, 'treize': 13, 'trois': 3, 'seize': 16, \
            'vingt': 20, 'un': 1, 'dix': 10, 'sept': 7}
    if s not in dico : return 0   # cas imprévu
    else : return dico [s]
```

**3)** La réponse à cette question est divisée en deux fonctions. La première traduit une liste de mots en un nombre numérique. Ce nombre est la somme des valeurs retournées par la fonction `valeur_mot` pour chacun des mots : soixante dix neuf $= 60 + 10 + 9$. Cette règle s'applique la plupart du temps sauf pour *quatre vingt*.

```
def lire_dizaine_liste(s):
    """convertit une liste de chaînes de caractères dont
```

```
    juxtaposition forme un nombre littéral compris entre 0 et 99"""
    r       = 0         # contient le résultat final
    dizaine = False     # a-t-on terminé le traitement des dizaines ?
    for mot in s:
        n = lire_unite (mot)
        if n == 20 :
            if not dizaine and r > 0 and r != 60 :
                r       *= n     # cas 80
                dizaine  = True
            else : r += n
        else : r += n
    return r
```

La seconde fonction remplace tous les traits d'union par des espaces puis divise un nombre littéral en liste de mots séparés par des espaces.

```
def lire_dizaine(s):
    s2 = s.replace ("-", " ")
    li = string.split (s2)
    return lire_dizaine_liste (li)
```

On pourrait réduire encore cette fonction en utilisant les expressions régulières en utilisant la fonction `split` décrite au paragraphe 8.6.3 page 169.

```
import re
def lire_separation(s):
    """divise un nombre littéral en mots avec les
    expressions régulières"""
    return re.compile ("[- ]").split (s)
```

**4)** Tout d'abord, les deux fonctions réciproques de celles présentées ci-dessus.

```
def ecrit_unite (x):
    """convertit un nombre compris inclus entre 0 et 16 inclus,
    20, 30, 40, 50, 60 en une chaîne de caractères"""
    if   x == 0: return "zéro"
    elif x == 1: return "un"
    elif x == 2: return "deux"
    elif x == 3: return "trois"
    elif x == 4: return "quatre"
    elif x == 5: return "cinq"
    elif x == 6: return "six"
    elif x == 7: return "sept"
    elif x == 8: return "huit"
    elif x == 9: return "neuf"
    elif x == 10: return "dix"
    elif x == 11: return "onze"
    elif x == 12: return "douze"
    elif x == 13: return "treize"
    elif x == 14: return "quatorze"
    elif x == 15: return "quinze"
    elif x == 16: return "seize"
    elif x == 20: return "vingt"
    elif x == 30: return "trente"
    elif x == 40: return "quarante"
    elif x == 50: return "cinquante"
    elif x == 60: return "soixante"
    elif x == 70: return "soixante-dix"
    elif x == 80: return "quatre-vingt"
    elif x == 90: return "quatre-vingt-dix"
    else        : return "zéro"
```

Cette fonction pourrait être simplifiée en utilisant le même dictionnaire que pour la fonction `valeur_mot` mais inversé : la valeur devient la clé et réciproquement.

```
def mot_valeur (x):
    """convertit un nombre compris inclus entre 0 et 16 inclus,
    20, 30, 40, 50, 60 en une chaîne de caractères"""
    dico = {'cinquante': 50, 'quarante': 40, 'onze': 11, 'huit': 8, 'six': 6, \
            'quinze': 15, 'trente': 30, 'douze': 12, 'cinq': 5, 'deux': 2, \
            'quatorze': 14, 'neuf': 9, 'soixante': 60, 'quatre': 4, \
            'zéro': 0, 'treize': 13, 'trois': 3, 'seize': 16, \
            'vingt': 20, 'un': 1, 'dix': 10, 'sept': 7}
    inv = {}
    for k,v in dico.iteritems () : inv [v] = k
    inv [70] = "soixante-dix"
    inv [80] = "quatre-vingt"
    inv [90] = "quatre-vingt-dix"
    return inv [x]
```

Le programme continue avec l'une ou l'autre des fonctions précédente. La fonction suivante construit le montant littéral pour un nombre à deux chiffres. Elle traite le chiffre des dizaines puis des unités.

```
def ecrit_dizaine(x):
    """convertit un nombre entre 0 et 99 sous sa forme littérale"""

    if x <= 16 : return ecrit_unite(x)

    s       = ""
    dizaine = x / 10
    unite   = x % 10
    s = mot_valeur (dizaine*10)
    s += " "
    s += mot_valeur (unite)
    return s
```

Le morceau de code ci-dessous permet de vérifier que le passage d'un nombre sous sa forme littérale et réciproquement fonctionne correctement pour tous les nombres compris entre 0 et 99 inclus.

```
for i in xrange(0,100):
    s = ecrit_dizaine (i)
    j = lire_dizaine (s)
    if i != j : print "erreur ", i, " != ", j, " : ", s
```

**fin correction TD 11.1.1** □

## 11.2 Représentation des données, partie de dames

### 11.2.1 Enoncé

Une partie de dames met en jeu quarante pions, vingt noirs, vingt blancs, chacun sur des cases différentes. L'objectif est de savoir si un pion est en mesure d'en prendre un autre. On ne traitera pas le cas des dames. Chaque pion est défini par :

– deux coordonnées entières, chacune comprise entre 1 et 10
– une couleur, noir ou blanc

**1)** On propose deux représentations de l'ensemble de pions :

1. Un tableau de 40 pions indicés de 0 à 39 inclus, chaque pion étant défini par :

  - deux coordonnées comprises entre 1 et 10, ou (0,0) si le pion n'est plus sur le damier
  - un entier qui vaut 1 pour blanc, 2 pour noir
2. Un tableau d'entiers à deux dimensions, chaque case contient :
  - soit 0 s'il n'y a pas de pion
  - soit 1 si la case contient un pion blanc
  - soit 2 si la case contient un pion noir

Y a-t-il d'autres représentations de ces informations ? Si on considère que l'efficacité d'une méthode est reliée à sa vitesse - autrement dit aux coûts des algorithmes qu'elles utilisent -, parmi ces deux représentations, quelle est celle qui semble la plus efficace pour savoir si un pion donné du damier est en mesure d'en prendre un autre ?

**2)** Comment représenter un tableau d'entiers à deux dimensions en langage *Python* à l'aide des types standards qu'il propose, à savoir t-uple, liste ou dictionnaire ?

**3)** On cherche à écrire l'algorithme qui permet de savoir si un pion donné est un mesure de prendre un pion. Quels sont les paramètres d'entrées et les résultats de cet algorithme ?

**4)** Il ne reste plus qu'à écrire cet algorithme.

### 11.2.2 Correction

**1)** La seconde représentation sous forme de tableau à deux dimensions est plus pratique pour effectuer les tests de voisinages. Chaque case a quatre voisines aux quatre coins, il est ensuite facile de déterminer si ces quatre voisines sont libres ou si elles contiennent un pion.

Avec la première représentation - le tableau des pions - pour savoir s'il existe un pion dans une case voisine, il faut passer en revue tous les pions pour savoir si l'un d'eux occupe ou non cette case. Avec la seconde représentation - le tableau à deux dimensions - on accède directement à cette information sans avoir à la rechercher. On évite une boucle sur les pions avec la seconde représentation.

**2)** Pour représenter le tableau en deux dimensions, il existe trois solutions :

1. Une liste de listes, chaque ligne est représentée par une liste. Toutes ces listes sont elles-mêmes assemblées dans une liste globale.
2. Une seule liste, il suffit de numéroter les cases du damier de 0 à 99, en utilisant comme indice pour la case $(i, j)$ : $k = 10 * i + j$. Réciproquement, la case d'indice $k$ aura pour coordonnées ($k/10$, $k\%10$).
3. Un dictionnaire dont la clé est un couple d'entiers.

**3)** On désire savoir si le pion de la case $(i, j)$ peut en prendre un autre. On suppose que le tableau à deux dimensions est une liste de dix listes appelée `damier`. `damier`[`i`][`j`] est donc la couleur du pion de la case $(i, j)$, à savoir 0 si la case est vide, 1 si le pion est blanc, 2 si le pion est noir. Pour ces deux derniers cas, la couleur des pions de l'adversaire sera donc `3 - damier`[`i`][`j`]. Les entrées de la fonctions sont donc les indices `i`, `j` et le damier `damier`. La sortie est une variable booléenne qui indique la possibilité ou non de prendre. On ne souhaite pas déplacer les pions.

**4)**

```
def pion_prendre(i,j,damier):
   c = damier [i][j]
   if c == 0: return False  # case vide, impossible de prendre
   c = 3 - c                # couleur de l'adversaire

   if damier [i-1][j-1] == c :     # s'il y a un pion adverse en haut à gauche
       if damier [i-2][j-2] == 0 : # si la case d'après en diagonale est vide
           return True             # on peut prendre

   # on répète ce test pour les trois autres cases
   if (damier [i-1][j+1] == c) and (damier [i-2][j+2] == 0): return True
   if (damier [i+1][j-1] == c) and (damier [i+2][j-2] == 0): return True
   if (damier [i+1][j+1] == c) and (damier [i+2][j+2] == 0): return True
```

```
   # si tous les tests ont échoué, on ne peut pas prendre
   return False
```

Voici une fonction équivalente lorsque le damier est un dictionnaire dont la clé est un couple d'entiers.

```
def pion_prendre(i,j,damier):
   c = damier [(i,j)]
   if c == 0: return False  # case vide, impossible de prendre
   c = 3 - c                # couleur de l'adversaire

   # test pour une prise du pion dans les quatre cases voisines
   if (damier [(i-1,j-1)] == c) and (damier [(i-2,j-2)] == 0): return True
   if (damier [(i-1,j+1)] == c) and (damier [(i-2,j+2)] == 0): return True
   if (damier [(i+1,j-1)] == c) and (damier [(i+2,j-2)] == 0): return True
   if (damier [(i+1,j+1)] == c) and (damier [(i+2,j+2)] == 0): return True

   # si tous les tests ont échoué, on ne peut pas prendre
   return False
```

La même fonction lorsque le damier est représenté par une seule liste.

```
def pion_prendre(i,j,damier):
   c = damier [10*i+j]
   if c == 0: return False  # case vide, impossible de prendre
   c = 3 - c                # couleur de l'adversaire

   # test pour une prise du pion dans les quatre cases voisines
   if (damier [10*(i-1)+j-1] == c) and (damier [10*(i-2)+j-2] == 0): return True
   if (damier [10*(i-1)+j+1] == c) and (damier [10*(i-2)+j+2] == 0): return True
   if (damier [10*(i+1)+j-1] == c) and (damier [10*(i+2)+j-2] == 0): return True
   if (damier [10*(i+1)+j+1] == c) and (damier [10*(i+2)+j+2] == 0): return True

   # si tous les tests ont échoué, on ne peut pas prendre
   return False
```

Pour ces trois cas, aucun effet de bord n'a été envisagé. Si la case est trop près d'un des bords, un des indices $i$, $j$, $i-1$, $j-1$, $i+1$, $j+1$, $i-2$, $j-2$, $i+2$, $j+2$ désignera une case hors du damier. Le code de la fonction `pion_prendre` devra donc vérifier que chaque case dont elle vérifie le contenu appartient au damier.

```
def pion_prendre(i,j,damier):
   c = damier [i][j]
   if c == 0: return False  # case vide, impossible de prendre
   c = 3 - c                # couleur de l'adversaire

   # on répète ce test pour les trois autres cases
   if (i >= 2)                and (j >= 2) and \
      (damier [i-1][j-1] == c) and (damier [i-2][j-2] == 0): return True
   if (i >= 2)                and (j < len (damier)-2) and \
      (damier [i-1][j+1] == c) and (damier [i-2][j+2] == 0): return True

   if (i < len (damier)-2)    and (j >= 2) and \
      (damier [i+1][j-1] == c) and (damier [i+2][j-2] == 0): return True
   if (i < len (damier)-2)    and (j < len (damier)-2) and \
      (damier [i+1][j+1] == c) and (damier [i+2][j+2] == 0): return True

   # si tous les tests ont échoué, on ne peut pas prendre
   return False
```

Une autre option consiste à entourer le damier d'un ensemble de cases dont le contenu sera égal à une constante différente de `0`, `-1` par exemple. Dans ce cas, si le damier est représenté par une liste de listes, la première case du damier aura pour coordonnées $(1,1)$ au lieu de $(0,0)$ car les listes n'acceptent pas les

indices négatifs. Ce n'est pas le cas lorsque le damier est représenté par un dictionnaire car une case peut tout à fait avoir pour coordonnées $(-1, -1)$.

Avec cette convention, les tests introduits dans le dernier programme ne seront plus nécessaires. Il faudra juste réécrire la seconde ligne de chaque fonction `pion_prendre` par :

```
    if c <= 0 : return False  # au lieu de if c == 0 : return False
```

**Remarque 11.1 : matrice et dictionnaire**
Les structures représentées ici sont des tableaux à deux dimensions : ce sont des matrices. Les dictionnaires sont particulièrement indiqués dans le cas où ces matrices sont *creuses* : elles contiennent beaucoup de valeurs nulles. Pour une liste ou une liste de listes, toutes les cases sont représentées. Dans le cas d'un dictionnaire, il est possible d'adopter comme convention que si un couple de coordonnées n'existe pas en tant que clé, cela veut dire que la case associée est nulle. Il en résulte un gain de place équivalent à la proportion de cases nulles.

**Remarque 11.2 : dictionnaire et clé de type tuple**
Le second exemple utilise un dictionnaire avec comme clé un `tuple`. Dans ce cas, les parenthèses sont superflues.

```
    damier [(i-1,j-1)]   # est équivalent à
    damier [ i-1,j-1 ]   # cette ligne
```

## 11.3 Reconnaître la langue d'un texte

### 11.3.1 Enoncé

L'objectif est de distinguer un texte anglais d'un texte français sans avoir à le lire. Le premier réflexe consisterait à chercher la présence de mots typiquement anglais ou français. Cette direction est sans doute un bon choix lorsque le texte considéré est une œuvre littéraire. Mais sur Internet, les contenus mélangent fréquemment les deux langues : la présence de tel mot anglais n'est plus aussi discriminante. Il n'est plus aussi évident d'étiqueter un document de langue anglaise lorsque les mots anglais sont présents partout.

On ne cherche plus à déterminer la langue d'un texte mais plutôt la langue majoritaire. Il serait encore possible de compter les mots de chacune des langues à l'aide d'un dictionnaire réduit de mots anglais et français. La langue majoritaire correspondrait à celle dont les mots sont les plus fréquents. Mais construire un dictionnaire est d'abord fastidieux. Ensuite, il faudrait que celui-ci contienne des mots présents dans la plupart des textes. Il faudrait aussi étudier le problème des mots communs aux deux langues.

Pour ces trois raisons, il paraît préférable d'étudier une direction plus simple quitte à revenir aux dictionnaires plus tard. Dans un premier temps, on cherchera donc à distinguer un texte anglais d'un texte français à partir de la fréquence de certaines lettres dont l'usage est différent dans les deux langues comme la lettre *w*.

La première étape consiste à calculer la fréquence des lettres dans différents textes anglais et français qu'on peut se procurer depuis le site *Gutenberg*[1] par exemple. Il suffit alors de télécharger quelques textes et de les enregistrer dans des fichiers texte.

**1)** Il faut tout d'abord récupérer dans un programme *Python* le contenu de ces textes dont on suppose qu'ils ont été téléchargés. On pourra s'inspirer des exemples du chapitre 8.

**2)** On veut construire une fonction qui compte l'occurrence de chaque lettre dans une chaîne de caractères. Cette fonction prend en entrée une chaîne de caractères et retourne un dictionnaire. Chaque élément contiendra le nombre de fois qu'un caractère apparaît.

**3)** Afin d'obtenir des probabilités, on divise chaque nombre par le nombre total de lettres (attention aux

1. http://www.gutenberg.net/

divisions entières).

**4)** Ecrire une fonction qui retourne la probabilité de chaque lettre dans un fichier texte.

**5)** En observant les résultats obtenus sur un texte français et un texte anglais, pensez-vous qu'il soit possible d'écrire une fonction qui détermine automatiquement la langue d'un texte à condition que celle-ci soit l'anglais ou le français.

### 11.3.2 Correction

**1)** La fonction répondant à la première question revient régulièrement dans beaucoup de programmes.

```
def lit_fichier (nom) :
    f = open (nom, "r")                  # ouverture du fichier
    l = f.readlines ()                   # on récupère toutes les lignes
    f.close ()                           # on ferme le fichier
    return "".join (l)                   # on retourne la somme des lignes
```

On pourrait également souhaiter récupérer un texte directement depuis Internet à condition d'en connaître l'adresse, ce que fait la fonction suivante. La première fonction doit recevoir un nom de fichier, la seconde une adresse Internet.

```
import urllib                        # import du module urllib
def lit_url (nom) :
    f = urllib.urlopen (nom)         # on ouvre l'url
    res = f.read ()                  # on lit son contenu
    f.close ()                       # on termine la lecture
    return res                       # on retourne le résultat
```

On peut regrouper ces deux fonctions et appeler soit l'une soit l'autre selon que le nom du texte est un fichier texte ou une adresse Internet :

```
def lit (texte) :
    if "http" in texte : s = lit_url (texte)  # Internet
    else : s = lit_fichier (texte)            # fichier texte
    return s

s = lit ("hugo.txt")
```

Une autre option consiste à utiliser les exceptions : on essaye d'abord d'ouvrir un fichier avec la fonction `open`. Si cela ne fonctionne pas, on peut supposer que le nom du fichier fait référence à une adresse Internet.

```
def lit (texte) :
    try :
        s = lit_fichier (texte)          # fichier texte
        return s
    except :                             # si cela ne marche pas,
        s = lit_url (texte)              # on suppose que texte
        return s                         # est une adresse Internet

s = lit ("hugo.txt")
```

**2)** La fonction suivante compte les occurrences de chaque lettre dans la chaîne de caractères `texte`. Une première solution consiste à appeler 26 fois la méthode `count` des chaînes de caractères.

```
def compte_lettre_count (texte) :
    texte = texte.upper ()             # pour éviter les accents
    res = { }                          # résultat, vide pour le moment
    for c in "ABCDEFGHIJKLMNOPQRSTUVWXYZ" :  # pour tout l'alphabet
        res [c] = texte.count (c)      # on compte les occurrences de c
    return res
```

Une autre solution consiste à parcourir le texte puis à compter les lettres au fur et à mesure qu'elles apparaissent.

```
def compte_lettre (texte) :
    texte = texte.upper ()              # pour éviter les accents
    res = { }                           # résultat, vide pour le moment
    for c in texte :                    # pour tous les caractères du texte
        if not("A" <= c <= "Z") : continue # si ce n'est pas une lettre, on passe
        if c not in res : res [c] = 1   # si elle n'est pas là, on lui affecte 1
        else : res [c] += 1        # sinon, on augmente son nombre d'apparitions
    return res
```

Il n'est pas évident de choisir l'une ou l'autre des méthodes. Calculer le coût algorithmique de chacune des fonctions n'est pas toujours évident car la première utilise la fonction `count` dont le fonctionnement est inconnu. Il paraît plus facile dans ce cas de comparer la vitesse d'exécution de chaque fonction.

Dans l'exemple suivant, la fonction `comparaison` appelle les deux méthodes sur le même texte. Le module `profile` va ensuite mesurer le temps passé dans chacune des fonctions.

```
def comparaison () :
    s = lit_fichier ("hugo.txt")
    compte_lettre_count (s)      # on ne mémorise pas les résultats
    compte_lettre (s)            # car on souhaite mesurer le temps passé

import profile                   # import du module profile
profile.run ("comparaison()")    # mesure du temps passé dans la fonction
                                 # comparaison et les fonctions qu'elle appelle
```

A la fin du programme, le temps passé dans chaque fonction est affiché par le module `profile`. On peut y lire le nombre de fois qu'une fonction a été appelée, le temps moyen passé dans cette fonction.

```
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.160    0.160    0.174    0.174 langue.py:19(compte_lettre)
        1    0.000    0.000    0.017    0.017 langue.py:31(compte_lettre_count)
        1    0.000    0.000    0.190    0.190 langue.py:43(comparaison)
```

| | |
|---|---|
| ncalls | Nombre d'appels à la fonction |
| tottime | Temps passé uniquement dans la fonction, le temps passé dans une fonction appelée par celle-ci n'est pas compté. |
| percall | Temps moyen passé uniquement dans la fonction. |
| cumtime | Temps cumulé passé dans la fonction, y compris celui passé dans une fonction appelée. |
| percall | Temps cumulé moyen. |
| filename | nom du fichier et nom de la fonction entre parenthèse |

La première solution est clairement la plus rapide. Ce résultat s'explique par la rapidité de la méthode `count`. Mais cela ne permet de pas de conclure dans tous les cas. Il faudrait connaître le coût algorithmique de chaque fonction pour cela. Par exemple, lorsqu'il s'agit de calculer les fréquences de tous les caractères et non plus des seules 26 lettres, on s'aperçoit que la seconde solution n'est plus que 1,5 fois plus lente et non 10 fois.

```
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.120    0.120    0.133    0.133 langue.py:19(compte_lettre)
        1    0.002    0.002    0.082    0.082 langue.py:31(compte_lettre_count)
```

Lorsque le nombre d'éléments dont il faut calculer la fréquence est assez grand, la seconde solution est préférable. La première solution est efficace dans des cas particuliers comme celui-ci. On s'aperçoit également que la fonction `compte_lettre` est plus rapide pour cette seconde expérience ($0.133 < 0.174$) mais cette observation est peu pertinente car cette fonction n'a été exécutée qu'une fois. Il y a en effet peu de chance pour que la fonction s'exécute à chaque fois à la même vitesse car l'ordinateur exécute un grand nombre de tâches simultanément.

Pour obtenir un temps d'exécution fiable, il faut appeler la fonction un grand nombre de fois. Le tableau suivant montre le temps moyen total passé dans chaque fonction après 100 appels à la fonction `comparaison`.

| **fonction** | **26 lettres** | **tous les caractères** |
|---|---|---|
| `compte_lettre` | 0.087 | 0.061 |
| `compte_lettre_count` | 0.017 | 0.073 |

La fonction `compte_lettre` est effectivement plus rapide dans le second cas. Comme on s'intéresse maintenant à tous les caractères, le test `if not` ("`A`" <= `c` <= "`Z`") n'est plus nécessaire : la fonction est d'autant plus rapide que cette instruction était exécutée un grand nombre de fois. On remarque également que la fonction `compte_lettre` ne paraît pas dépendre du nombre de caractères considérés. La fonction `compte_lettre` est résolument un meilleur choix.

**3)** Il suffit de diviser chaque valeur du dictionnaire `res` par la somme de ses valeurs. On insère donc les trois lignes suivantes juste avant l'instruction `return` de la fonction précédente.

```
def compte_lettre (texte) :
    # ...
    s = sum (res.values ())
    for k in res :
        res [k] = float (res [k]) / s
    return res
```

**4)** Le tableau suivant montre les probabilités obtenues pour les quatre lettres H,U,W,Y pour lesquelles les résultats montrent des différences significatives. Les deux textes sélectionnés sont *Le dernier jour d'un condamné* de Victor Hugo et sa traduction anglaise *The Man Who Laughs*.

| **lettre** | Le dernier jour d'un condamné | The Man Who Laughs |
|---|---|---|
| H | 1,22 % | 6,90 % |
| U | 6,79 % | 2,57 % |
| W | 0,11 % | 2,46 % |
| Y | 0,42 % | 1,34 % |

Ces chiffres ont été obtenus avec le programme suivant qui télécharge ces deux textes directement depuis le site *Gutenberg*[2]. Les fréquences de ces quatre lettres montrent des différences significatives, ce que montre de façon plus visuelle la figure 11.1 pour les lettres `W` et `H`.

```
# le dernier jour d'un condamné
c1 = compte_lettre (lit_url ("http://www.gutenberg.org/dirs/etext04/8ldrj10.txt"))
# the man who laughs
c2 = compte_lettre (lit_url ("http://www.gutenberg.org/files/12587/12587-8.txt"))

car = c1.keys ()
car.sort ()
for k in car :
    print k, " : ", "% 2.2f" % (c1 [k] * 100), "%", " % 2.2f" % (c2 [k] * 100), "%"
```

**5)** La fréquence de la lettre `W` ou `H` devrait suffire à départager un texte français d'un texte anglais. Pour vérifier cette hypothèse, on décrit chaque texte par deux coordonnées correspondant aux fréquences des lettres `H` et `W`, ce que calcule la fonction suivante :

2. Il est conseillé de vérifier ces adresses depuis le site http://www.gutenberg.net/, ces liens peuvent changer.

```
def langue_lettre (texte) :
    if "http" in texte : s = lit_url (texte)   # cas URL
    else : s = lit_fichier (texte)             # cas fichier
    c = compte_lettre (s)                      # on compte les lettres
    return c ["W"], c ["H"]                    # on retourne deux fréquences
```

La dernière instruction de cette fonction suppose que tous les textes incluent au moins un `W` et un `H`. Dans le cas contraire, la fonction produirait une erreur car elle ne pourrait pas trouver la clé "`W`" ou "`H`" dans le dictionnaire `c`. Pour remédier à cela, on utilise la méthode `get` des dictionnaires qui permet de retourner une valeur lorsqu'une clé est introuvable.

```
def langue_lettre (texte) :
    ...                                        # lignes inchangées
    return c.get ("W", 0.0), c.get ("H", 0.0)  # on retourne deux fréquences
```

Après cette correction, la fonction retourne une valeur nulle lorsque les lettres `W` ou `H` n'apparaissent pas. La fonction suivante permet de construire deux listes `cx` et `cy` qui contiennent les fréquences des lettres `W` et `H` pour une liste de textes.

```
def curve (li) :
    cx,cy = [], []
    for l in li :                 # pour tous les textes de la liste
        x,y = langue_lettre (l)   # coordonnées d'un texte, fréquence W et H
        cx.append (x)             # on ajoute x à la liste des abscisses
        cy.append (y)             # on ajoute y à la liste des ordonnées
    return cx,cy
```

On utilise cette dernière fonction pour faire apparaître sur un graphique plusieurs textes anglais et français. Chaque texte est représenté par deux coordonnées : la fréquence des lettres `W` et `H`. On obtient un nuage de points dans un plan ce que montre le graphe de la figure 11.1. Ce dernier a été tracé à l'aide du module `matplotlib`[3]. Il s'inspire du logiciel *Matlab* dans la manière de constuire des graphes[4].

```
frcx,frcy = curve (fr)          # on récupère les coordonnées des textes français
encx,ency = curve (en)          # on récupère les coordonnées des textes anglais

import pylab                        # import du module matplotlib
pylab.plot (frcx, frcy, "rx",ms=10,\
                            mew=2.5) # on trace la courbe des textes français
pylab.plot (encx, ency, "bv")       # on trace la courbe des textes anglais
pylab.legend (("francais", "anglais"), loc=2)  # légende (sans accent)
pylab.title ("langue")              # titre
pylab.xlabel ("frequence de W")     # légende de l'axe des abscisses
pylab.ylabel ("frequence de H")     # légende de l'axe des ordonnées
pylab.savefig ("graphe.png")        # enregistrement sous forme d'image
pylab.show ()                       # on fait apparaître le graphique
```

La figure 11.1 montre qu'au delà d'une fréquence de 1% pour la lettre `W`, le texte est anglais. On en déduit la fonction qui permet de déterminer la langue d'un texte.

```
def est_anglais (texte) :
    w,h = langue_lettre (texte)
    return w > 0.01
```

3. http://matplotlib.sourceforge.net/

4. Il a l'inconvénient de ne pas supporter les accents français mais il est possible de contourner cet obstacle en associant *MatPlotLib* et *Latex* qui permet également d'afficher des formules mathématiques sur le graphique. L'utilisation de ce genre de modules est plutôt facile, de nombreux exemples de graphiques sont proposés avec le programme *Python* qui a permis de les réaliser. Il suffit souvent des recopier et de les adapter. Une autre solution consiste à écrire les courbes dans un fichier texte puis à les récupérer avec un tableur tel que *OpenOffice* pour les représenter sous forme de graphe (voir l'exemple page 154).

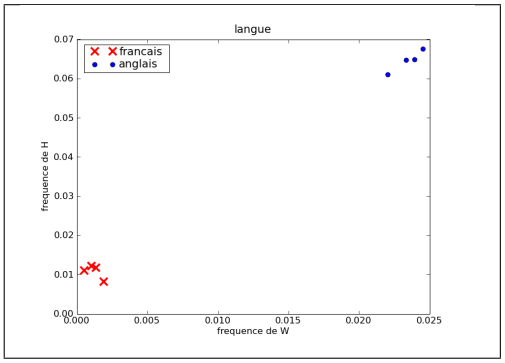



**Figure 11.1 :** *Les textes anglais et français sont cantonnés chacun dans une zone précise du graphe : il apparaît que les fréquences des lettres* `H` *et* `W` *permettent toutes deux de distinguer des textes anglais et français. Une fréquence inférieure à 1% pour la lettre W signifie un texte français.*

La réponse à la dernière question pourrait s'arrêter ici. Toutefois, les paragraphes qui suivent s'intéressent à la fiabilité de la fonction de la fonction `est_anglais` dans un autre contexte : Internet.

**Interprétation**

En effet, ce dernier résultat peut être critiqué : ce seuil s'appuie sur seulement quatre textes. Ces derniers sont longs : la fréquence des lettres en est d'autant plus fiable. Sur un texte d'une centaine de lettres, la fréquence de la lettre $W$ est soit nulle, soit supérieure à 1%. Au delà de ce cas difficile et peu fréquent, on vérifie la fiabilité de la fonction `est_anglais` sur des articles de journaux français et anglais.

Internet va nous aider à construire une base de textes : il est facile des récupérer automatiquement de nombreux articles de journaux sur des sites tels que celui du journal *Le Monde* ou le *New-York Times*. On pourra ensuite calculer la fréquence des lettres `W` et `H` pour chacun d'entre eux.

**Récupération de pages HTML**

Une page Internet contient du texte dont la présentation est décrite par le format HTML. On y trouve tout type de contenu, des images, de la vidéo, du texte, et des hyperliens. Le site du journal *Le Monde*[5] contient beaucoup de liens vers des articles qui eux-mêmes contiennent des liens vers d'autres articles (voir figure 11.2). A partir d'une seule page, on récupére une myriade de liens Internet. L'objectif est ici d'explorer quelques pages pour récupérer des liens vers des articles de ce journal. La récupération des pages Internet d'un site s'inspire de la récupération de la liste de tous les fichiers inclus dans un répertoire[6].

La principale différence provient du fait que les liens ne désignent pas toujours une page nouvelle mais parfois la première page du site ou une autre page déjà vue dont il ne faut pas tenir compte. L'autre différence vient du faite que le nombre de pages sur un site comme *Le Monde* peut s'avérer très grand : il faut bien souvent se limiter aux premières pages et à celles qui appartiennent à ce site.

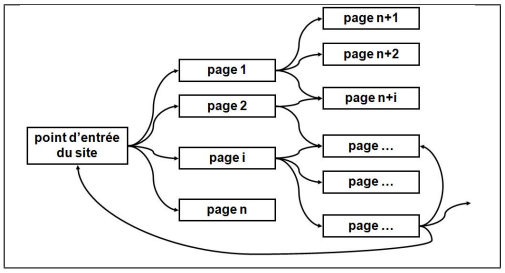



**Figure 11.2 :** *Un site Internet contient généralement une page qui est le moyen d'accéder à toutes les pages du site. C'est un point d'entrée qui mène directement ou indirectement à l'ensemble des pages du site. Il suffit de suivre les liens pour en établir une liste. Certains liens mènent à des pages nouvelles, d'autres à des pages déjà vue : ce graphe est cyclique.*

Le format HTML fonctionne par balises : une information est encadrée par deux balises : < `balise` > `texte`... < /`balise` >
Les hyperliens utilisent une syntaxe à peine plus compliquée puisqu'une balise peut aussi recevoir quelques paramètres : < `balise param1` = "`valeur1`" `param2` = "`valeur2`" > `texte`... < /`balise` >. Selon cette syntaxe, un lien vers une autre page est indiqué comme suit :

5. http://www.lemonde.fr/
6. voir paragraphe 8.3.3 page 159

```
<a href="http://...."> texte qui apparaît à l'écran </a>
```

Les expressions régulières [7] permettent de retrouver aisément tous les liens insérés dans une page sans avoir à se soucier du format HTML. L'algorithme de recherche consiste à recenser tous les liens de la première page. La syntaxe HTML nous apprend que ces liens sont compris entre guillemets. On considère ensuite chacune des pages recensées et on procède de même. On s'arrête après avoir obtenu le nombre de pages souhaité.

```
import urllib       # pour accéder une adresse internet
import re           # pour traiter les expressions régulières

def list_url (site, root, nbpage = 100) :

    # expression régulières
    # tous les liens commençant par root et entre guillemets
    # exemple :  "http://www.lemonde.fr/index,0.26.html"
    # on place entre parenthèses la partie intéressante :
    # c'est-à-dire tout ce qu'il y a entre les guillemets
    s = "\"(" + root + "[-_~a-zA-Z0-9/.?,]*?)\""
    exp  = re.compile (s, re.IGNORECASE)

    res    = [ ]         # résultat
    pile   = [ site ]    # page à explorer

    while len (pile) > 0 and len (res) < nbpage :

        # on bascule toutes les pages de pile vers res
        for u in pile :
            if u not in res : res.append (u)

        u = pile.pop ()  # on s'intéresse à la prochaine page

        try :
            f       = urllib.urlopen (u)      # accès à l'url
            text    = f.read ()               # on lit son contenu
            f.close ()                        # fin de l'accès

            liens = exp.findall (text)        # recherche de tous les liens
            for u in liens :
                if u in pile or u in res :    # on passe au suivant si
                    continue                  # déjà vu

                # on enlève les images et autres fichiers indésirables
                if ".gif" in u or ".png" in u or ".jpg" in u : continue
                if ".cs" in u or ".css" in u or ".js" in u : continue

                # on ajoute le liens à la liste des liens à explorer
                pile.append (u)

        except IOError, exc:
            print "problème avec url ", u
            continue

    return res
```

Il ne reste plus qu'à appeler cette fonction pour récupérer la liste des URLs :

```
url  = "http://www.lemonde.fr/"   # un journal français
res  = list_url (url, url)
for r in res : print r
```

7. voir paragraphe 8.6 page 167

```
url  = "http://www.nytimes.com/"  # un journal américain
res  = list_url (url, url)
for r in res : print r
```

### Résultats

Les résultats obtenus sont ceux de la figure 11.3. La frontière est nettement plus floue et montre à première vue qu'il n'est pas toujours possible de distinguer une page française d'une page anglaise. Toutefois, la plupart des pages mal classées par cette méthode contiennent peu de texte et plus d'images : les fréquences portent plus sur la syntaxe HTML que sur un contenu littéraire. Il faudrait pour améliorer la qualité des résultats filtrer les pages pour ne conserver que leur contenu textuel. Cela montre aussi que cette méthode est plutôt robuste puisqu'on voit deux classes se dessiner nettement malgré les caractères parasites HTML.

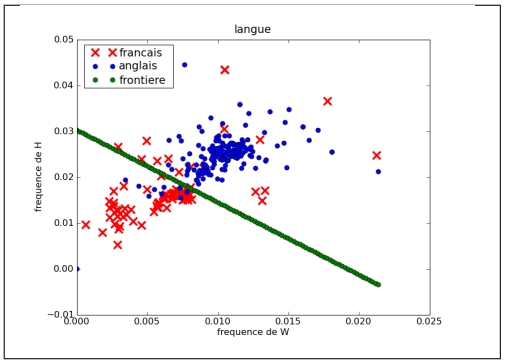



**Figure 11.3 :** *La frontière entre texte anglais et français paraît plus floue et ne dépend plus de la fréquence d'une seule lettre.*

### Pour aller plus loin

Jusqu'à présent, seules les lettres `W` et `H` ont été utilisées. Les deux autres lettres intéressantes `U` et `Y` ont été laissées de côté. On peut se demander quelle est la meilleure façon de procéder avec quatre lettres. Chaque texte est maintenant représenté par quatre fréquences, soit quatre coordonnées et il n'est possible de n'en représenter que deux.

Dans ce nuage de points à quatre dimension, l'analyse en composantes principales (ACP) [8] est une méthode qui permet de calculer un plan de projection, plus exactement le meilleur plan de projection : celui qui minimise les distances de chaque point au plan. On utilise pour cela le module `mdp` [9] qui implémente cet algorithme.

On suppose dans l'exemple suivant que `fr4` et `en4` sont deux matrices de quatre colonnes représentant les coordonnées de chaque document. La figure 11.4 montre que l'utilisation des quatre lettres `H`, `U`, `W`, `Y` permet de construire une frontière plus efficiente qu'avec l'utilisation de `W`, `H` seulement.

```
import mdp,copy,numpy
nbfr,nben = len (fr), len (en)
all = numpy.array (fr + en)    # construction du nuage de points
all = copy.deepcopy (fr4)      # incluant les points français et anglais

node = mdp.nodes.PCANode()     # ACP
y    = node (all)              # on peut aussi écrire y = mdp.pca (all)

# obtention des coordonnées des points dans le plan de projection
frcx = [ y [i,0] for i in range (0, nbfr) ]
frcy = [ y [i,1] for i in range (0, nbfr) ]
encx = [ y [i,0] for i in range (nbfr, nbfr + nben) ]
ency = [ y [i,1] for i in range (nbfr, nbfr + nben) ]

# dessin des points dans le plan de projection
```

8. voir le livre *Probabilités, analyse des données et statistique* de Gilbert Saporta (éditions Technip).
9. http://mdp-toolkit.sourceforge.net/

```
# c'est le même code que précédemment
# ...
```

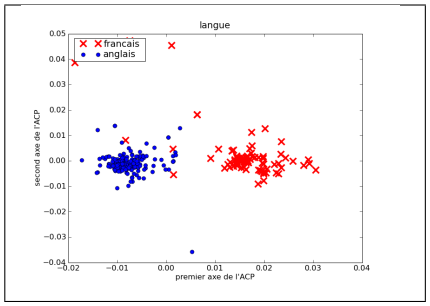



**Figure 11.4 :** *La frontière entre les textes anglais et français est beaucoup plus nette dans ce plan de projection issu de l'analyse en composantes principales.*

**fin correction TD 11.3.1** □

## 11.4 Carrés magiques

### 11.4.1 Enoncé

Un carré magique est un carré 3x3 dont chaque case contient un nombre entier et pour lequel les sommes sur chacune des lignes, chacune des colonnes, chacune des diagonales sont égales, soit huit sommes égales. Le plus simple des carrés magiques est celui dont tous les chiffres sont égaux :

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

**1)** Un carré magique n'est a priori qu'une matrice de trois lignes et trois colonnes. Dans un premier temps, on va donc construire une classe `CarreMagique` qui contiendra une liste de trois listes, chacune de trois colonnes. Compléter le programme pour créer un carré magique `cm`.

```
class CarreMagique :
    def __init__ (self, nb) :
        self.nb = nb

m = [ [9, 3, 3], [ 4,5,6] , [3,8,2] ]
```

**2)** On désire afficher le carré magique. Que donne l'instruction `print cm` ? On ajoute la méthode `__str__`.Que produit l'instruction `print cm` maintenant ? Que faudrait-il changer pour que le programme affiche le carré en entier ?

```
    def __str__ (self) :
        s = ""
        s += str (self.nb [0][0])
        return s
```

**3)** Ecrire une méthode qui calcule la somme des chiffres sur la ligne `i`.

**4)** Ajouter une ligne dans la méthode `__str__` afin d'afficher aussi la somme des chiffres sur la ligne 0. Cette information correspond à la somme des chiffres sur chaque ligne ou chaque colonne si le carré est

magique.

**5)** Ecrire une méthode qui calcule la somme des chiffres sur la colonne `j`.

**6)** Sans tenir compte des diagonales, écrire une méthode qui détermine si un carré est magique.

**7)** Il ne reste plus qu'à inclure les diagonales.

**8)** Ecrire une fonction qui détermine si tous les nombres du carré magique sont différents.

**9)** Terminer le programme pour déterminer tous les carrés magiques 3x3 dont les nombres sont tous différents.

**Remarque 11.3 : différence entre `print` et `return`**
A la fin d'un calcul, afin de voir son résultat, on utilise souvent l'instruction `print`. On peut se demander alors si à la fin de chaque fonction, il ne faudrait pas utiliser l'instruction `print`. A quoi servirait alors l'instruction `return` ? On suppose qu'un calcul est en fait le résultat de trois calculs à la suite :

```
a = calcul1 (3)
b = calcul2 (a)
c = calcul3 (b) # c résultat souhaité et affiché
```

Chaque terme `calculx` cache une fonction or seul le résultat de la dernière nous intéresse et doit être affiché. Pour les deux premières, la seule chose importante est que leur résultat soit transmis à la fonction suivante et ceci ne peut se faire que grâce à l'instruction `return`. L'instruction `print` insérée dans le code de la fonction `calcul1` ou `calcul2` permettra d'afficher le résultat mais ne le transmettra pas : l'instruction `return` est donc indispensable, `print` facultative.

En revanche, dans la dernière fonction `calcul3`, il est possible de se passer de `return` et de se contenter uniquement d'un `print`. Cependant, il est conseillé d'utiliser quand même `return` au cas où le résultat de la fonction `calcul3` serait utilisé par une autre fonction.

### 11.4.2 Correction

**1)** La réponse demandée correspond à la création d'une instance de type `CarreMagique`.

```
cm = CarreMagique (m)
```

**2)** Le message affiché par l'instruction `print cm` avant d'ajouter la méthode `__str__` correspond au suivant :

```
<__main__.CarreMagique instance at 0x01A254E0>
```

La réponse à cette question et aux suivantes sont présentées d'un seul tenant. Des commentaires permettent de retrouver les réponses aux questions de l'énoncé excepté pour la dernière question.

```
# -*- coding: iso-8859-15 -*-

class CarreMagique :

    def __init__ (self, nb) :                                                    5
        """on place la matrice nb (liste de listes)
        dans la classe accessible par le mot-clé self"""
        self.nb = nb

    # réponse à la question 2                                                    10
    def __str__ (self) :
        """méthode appelée lorsque on cherche à afficher
        un carré magique avec l'instruction print"""
        s = ""
```

```
        for i in range (0, len (self.nb)) :
            for j in range (0, len (self.nb)) : s += str ( self.nb [i][j]) + " "
            s += "\n" # pour passer à la ligne
        # réponse à la question 4
        s += "somme " + str (self.somme_ligne (0))
        return s

    # réponse à la question 3
    def somme_ligne (self, i) :
        s = 0
        for j in range (0, len (self.nb)) : s += self.nb [i][j]
        return s

    # réponse à la question 5
    def somme_colonne (self, j) :
        s = 0
        for i in range (0, len (self.nb)) : s += self.nb [i][j]
        return s

    # réponse à la question 6
    def est_magique (self) :
        # on stocke toutes les sommes
        l = []
        for i in range (0, len (self.nb)) : l.append ( self.somme_ligne (i) )
        for j in range (0, len (self.nb)) : l.append ( self.somme_colonne (j) )

        # réponse à la question 7
        l.append ( self.somme_diagonale (0))
        l.append ( self.somme_diagonale (1))

        # on trie la liste
        l.sort ()

        # on compare le plus petit et le plus grand, s'il sont égaux,
        # le carré est magique
        return l [0] == l [ len(l)-1 ]

    # réponse à la question 7
    def somme_diagonale (self, d) :
        """d vaut 0 ou 1, première ou seconde diagonale"""
        s = 0
        if d == 0 :
            for i in range (0, len (self.nb)) : s += self.nb [i][i]
        else :
            for i in range (0, len (self.nb)) :
                s += self.nb [i][len(self.nb)-i-1]
        return s

    # réponse à la question 8
    def nombre_differents (self) :
        """retourne True si tous les nombres sont différents,
        on place les nombres un par un dans un dictionnaire,
        dès que l'un d'eux s'y trouve déjà,
        on sait que deux nombres sont identiques, le résultat est False"""
        k = { }
        for i in range (0, len (self.nb)) :
            for j in range (0, len (self.nb)) :
                c = self.nb [i][j]
                if c in k : return False     # pas besoin d'aller plus loin
                                             # il y a deux nombres identiques
                else : k [c] = 0
        return True


m = [ [9, 3, 7],  [ 4,5,6] , [1,8,2] ]
```

```
cm = CarreMagique (m)           # réponse à la question 1
print cm                        # affiche 15
print cm.est_magique ()         # affiche False
print cm.nombre_differents ()   # affiche True

m = [ [9, 9, 9],  [9, 9, 9], [9, 9, 9] ]
cm = CarreMagique (m)
print cm                        # affiche 15
print cm.est_magique ()         # affiche True
print cm.nombre_differents ()   # affiche False
```

La dernière question consiste à trouver tous les carrés magiques dont les nombres sont tous différents. Il faut donc passer en revue tous les carrés 3x3 et ne conserver que ceux pour lesquels les méthodes `est_magique` et `nombre_differents` retournent un résultat positif. On suppose que le programme précédent contenant la classe `CarreMagique` porte le nom de `carre_magique.py`. Ceci explique la ligne `from`....

```
from carre_magique import CarreMagique
res = []
for a in range (1,10) :
    for b in range (1,10) :
        for c in range (1,10) :
            for d in range (1,10) :
                for e in range (1,10) :
                    for f in range (1,10) :
                        for g in range (1,10) :
                            for h in range (1,10) :
                                for i in range (1,10) :
                                    l  = [ [a,b,c], [d,e,f], [g,h,i] ]
                                    cm = CarreMagique (l)
                                    if cm.nombre_differents () and \
                                                cm.est_magique () :
                                        res.append (cm)

print len (res)
for r in res :
    print r
```

L'inconvénient de cette solution est qu'elle est inadaptée pour des carrés d'une autre dimension que 3x3. Il faudrait pouvoir inclure un nombre de boucles variable ce qui est impossible. L'autre solution est de programmer un compteur en reproduisant l'algorithme d'une addition. C'est l'objectif du programme suivant.

```
# -*- coding: iso-8859-15 -*-
from carre_magique import CarreMagique

dim = 3
nb  = dim*dim
res = []         # contiendra la liste des carrés magiques

# le compteur : neuf nombres 1
ind = [1 for i in range (0,nb) ]

while ind [0] <= nb :

    # transformation d'une liste en une liste de listes
    # [1,2,3,4,5,6,7,8,9] --> [[1,2,3],[4,5,6],[7,8,9]]
    l = []
    for i in range (0, dim) :
        l.append ( ind [i*dim:(i+1)*dim] )

    # on vérifie que le carré est magique et
    # a des nombres tous différents
    cm = CarreMagique (l)
```

```
    if cm.nombre_differents () and cm.est_magique () :
        res.append (cm)

    # on passe au carré suivant :
    i        = nb-1      # dernier indice (9 ici)
    ind [i] += 1         # addition de 1 au dernier indice
    while i > 0 and ind [i] > nb :
        ind [i-1] += 1   # un des indices est supérieur à nb (9 ici)
        ind [i]    = 1   # on le remet à 1
        i         -= 1   # et on propage l'information à l'indice inférieur

# résultat final
print len (res)
for r in res : print r
```

Il existe une autre manière de parcourir les carrés magiques en décomposant un entier compris entre 1 et $9^9$ en une succession de neuf entiers. On utilise pour cela la décomposition d'un nombre en base neuf : $n-1 = \sum_{k=1}^{9} \alpha_k (i_k - 1)$ où tous les nombres $i_k$ sont dans l'ensemble $\{1, \ldots, 9\}$.

```
# -*- coding: iso-8859-15 -*-
from carre_magique import CarreMagique

dim = 3
nb  = dim*dim
M   = 9 ** nb    # on va tester 9^9 carrés possibles
res = []         # contiendra la liste des carrés magiques

for n in xrange (0,M) :

    # on décompose n en liste
    ind = []
    k   = n
    for t in range (0,nb) :
        dec = k % nb
        k   = k / nb
        ind.append (dec+1)

    # transformation d'une liste en une liste de listes
    # [1,2,3,4,5,6,7,8,9] --> [[1,2,3],[4,5,6],[7,8,9]]
    l = []
    for i in range (0, dim) :
        l.append ( ind [i*dim:(i+1)*dim] )

    # on vérifie que le carré est magique et
    # a des nombres tous différents
    cm = CarreMagique (l)
    if cm.nombre_differents () and cm.est_magique () :
        res.append (cm)

# résultat final
print len (res)
for r in res : print r
```

Il faut noter l'utilisation de la fonction `xrange` au lieu de la fonction `range` lors de la première boucle. La fonction `xrange` ne crée pas de liste, elle simule un itérateur qui parcourt une liste. Dans notre cas, la fonction `range` crée une liste de $9^9$ éléments qui ne tient pas en mémoire, l'erreur suivante est déclenchée :

```
Traceback (most recent call last):
  File "carre_magique_tous5.py", line 9, in <module>
    for n in range (0,M) :
MemoryError
```

Ces trois solutions sont très lentes car elles explorent $9^9$ solutions. Comme on ne cherche que les carrés magiques dont les éléments sont tous différents, on pourrait n'explorer que les permutations soit 9! configurations. On crée une fonction `permutation` qui parcourt toutes les permutations possibles par récurrence (voir figure 11.5).

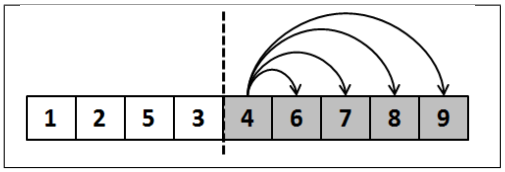



**Figure 11.5 :** *C'est une illustration de l'idée permettant de parcourir les permutations. On suppose que la partie du côté gauche du trait en pointillé (relié à la variable* `pos`*) est figée. On échange tour à tour le premier chiffre à droite (*$\texttt{pos}+1$,$\texttt{pos}+2$*, ...) avec les suivants. A chaque échange, on appelle à nouveau la fonction* `permutation` *récursive en décalant le trait en pointillé vers la droite. Après cet appel, on effectue l'échange inverse.*

```
# -*- coding: iso-8859-15 -*-
from carre_magique import CarreMagique

def echange (p, i, pos) :
    # échange de deux éléments d'indice i et pos du tableau p
    if i != pos :
        e       = p [i]
        p [i]   = p [pos]
        p [pos] = e

def permutation (res, dim, pos = 0, p = None) :
    # parcours des permutations par récurrence
    # pour le premier appel à la fonction permutation
    if p == None : p = range (1, dim*dim+1)

    if pos < len (p) :
        # on organise les permutations de l'élément p [pos], p [pos+1], ...
        for i in range (pos, len (p)) :
            echange (p, i, pos)
            permutation (res, dim, pos+1, p)
            echange (p, i, pos)
    else :
        # pos correspond au dernier élément : il n'y a plus de permutations
        # possibles avec les éléments qui suivent
        # on teste donc le nouveau carré
        l = []
        for i in range (0, len (p)/dim) :
            l.append ( p [i*dim:(i+1)*dim] )
        cm = CarreMagique (l)
        if cm.est_magique () : res.append (cm)

# permutations
res = []
permutation ( res, 3 )
# résultats
print "nombre de carrés ", len (res)
for r in res : print r
```

Le programme indique huit carrés magiques qui sont en fait huit fois le même carré obtenus par rotation ou symétrie. La somme des chiffres est 15.

**fin correction TD 11.4.1** □

## 11.5 Tri rapide ou quicksort

### 11.5.1 Enoncé

Cet énoncé a pour objectif de présenter l'algorithme de tri *quicksort* qui permet de trier par ordre croissant un ensemble d'éléments (ici des chaînes de caractères) avec un coût moyen [10] en $O(n \ln n)$ où $n$ est le nombre d'éléments à classer. Le tri *quicksort* apparaît rarement sous la forme d'un graphe : il est plus simple à programmer sans les graphes mais il est plus simple à appréhender avec les graphes. Dans cette dernière version, l'algorithme insère un à un les éléments d'une liste à trier dans un graphe comme celui de la figure 11.6 (page 237). Chaque nœud de ce graphe est relié à deux autres nœuds :

1. Un nœud `avant` ou " < " qui permet d'accéder à des éléments classés avant celui de ce nœud.
2. Un nœud `apres` ou " > " qui permet d'accéder à des éléments classés après celui de ce nœud.

Les nœuds `avant` et `apres` sont appelés les successeurs. Le terme opposé est prédécesseur. Ces deux nœuds ont nécessairement un prédécesseur mais un nœud n'a pas forcément de successeurs. S'il en avait toujours un, l'arbre serait infini.

**1)** On cherche à construire une classe ayant pour nom `NoeudTri` et qui contient une chaîne de caractères initialisée lors de la création de la classe : `n = NoeudTri("essai")`.

**2)** On écrit la méthode `__str__` de sorte que l'instruction `print n` affiche la chaîne de caractères que contient `n`.

**3)** On cherche maintenant à définir d'autres nœuds, reliés à des attributs `avant` et `apres`. On suppose que les nœuds utilisent l'attribut `mot`, on crée alors une méthode `insere(s)` qui :

– Si `s` < `self.mot`, alors on ajoute l'attribut `avant` = `NoeudTri(s)`.
– Si `s` > `self.mot`, alors on ajoute l'attribut `apres` = `NoeudTri(s)`.

| **Fonction utile :** | |
|---|---|
| `cmp(s1, s2)` | Compare deux chaînes de caractères, retourne -1,0,1 selon que `s1` est classée avant, est égale ou est classée après `s2`. |

**4)** La méthode `__str__` n'affiche pour le moment qu'un mot. Il s'agit maintenant de prendre en compte les attributs `avant` et `apres` afin que l'instruction `print n` affiche `avant.__str__()` et `apres.__str__()`. Il faudra également faire en sorte que la méthode `avant.__str__()` ne soit appelée que si l'attribut `avant` existe. Comme la liste des mots à trier est finie, il faut bien que certains nœuds n'aient pas de successeurs. On pourra s'inspirer du programme page 89 (attribut `__dict__`). Qu'est-ce qu'affiche le programme suivant ?

```
racine = NoeudTri ("un")
racine.insere ("unite")
racine.insere ("deux")
print racine
```

**5)** Est-il possible de trier plus de trois mots avec ce programme ? Que faut-il modifier dans la méthode `insere` afin de pouvoir trier un nombre quelconque de mots ?

**6)** Ajouter le code nécessaire afin que la méthode `insere` génère une exception lorsqu'un mot déjà présent dans l'arbre est à nouveau inséré.

**7)** On se propose de construire une image représentant l'arbre contenant les mots triés par l'algorithme *quicksort*. Cette représentation utilise le module `pydot` qui utilise l'outil *Graphviz* [11] ainsi que le module `pyparsing` [12]. Leur installation est assez facile sous *Windows* puisqu'elle consiste seulement à exécuter un

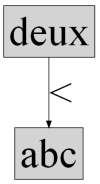
10. Lorsqu'on parle de coût moyen, cela signifie que le coût n'est pas constant en fonction de la dimension du problème. Ici, le coût moyen désigne le coût moyen d'un tri *quicksort* obtenu en faisant la moyenne du coût du même algorithme sur toutes les permutations possibles de l'ensemble de départ.
11. http://www.graphviz.org/
12. http://pyparsing.wikispaces.com/

programme d'installation. Il faut ensuite installer manuellement le module `pydot` [13]. Après avoir décompressé les fichiers de ce module, il faut se placer dans le même répertoire et utiliser la ligne de commande suivante :

```
c:\python25\python setup.py install
```

Le tracé d'un graphe passe par l'écriture d'un fichier dans lequel on indique les nœuds et les arcs. Un nœud est un numéro suivi de [`label` = "`deux`"] qui indique que ce nœud contient le mot `deux`. Si on ajoute , `style` = `filled`, `shape` = `record`, le nœud est représenté par un rectangle au fond grisé. Un arc est un couple de numéros séparés par $->$. Le texte [`label` = " < "] permet d'étiqueter cet arc. Selon cette syntaxe, le texte suivant décrit l'image située à droite :

```
digraph GA {
    2 [label="deux",style=filled,shape=record]
    3 [label="abc" ,style=filled,shape=record]
    2 -> 3 [label="<"]
}
```




| **Fonction utile :** | |
|---|---|
| `id(obj)` | Retourne un identifiant entier unique pour chaque variable `obj`. |

Une fois que cette chaîne de caractères a été construite, il suffit de l'écrire dans un fichier puis d'appeler le module `pydot` pour le convertir en image avec le code suivant :

```
g = open ("graph.txt", "w")  #
g.write (graph)              # partie écriture dans un fichier
g.close ()                   #
dot = pydot.graph_from_dot_file ("graph.txt") # partie graphe
dot.write_png ("graph.png", prog="dot")       # avec pydot
```

L'objectif de cette question est de construire une chaîne de caractères pour l'ensemble du graphe. La correction de cette exercice envisage également de construire une page HTML contenant le graphe et la liste triée ainsi que la création automatique d'un fichier au format PDF par le biais de *Latex* [14].

### 11.5.2 Correction

**1)** La chaîne de caractères que contient `NoeudTri` s'appelle `mot`.

```
class NoeudTri (object):
    def __init__(self,s):
        self.mot = s
```

**2)**

```
class NoeudTri (object):
    def __init__(self,s): self.mot = s
    def __str__(self)   : return self.mot + "\n"  # \n : passage à la ligne
```

**3)**

13. http://code.google.com/p/pydot/

14. *Latex* est un langage adapté aux publications scientifiques, également à l'écriture de livres comme celui-ci. En utilisant sa syntaxe, il permet de créer rapidement un fichier au format PDF.

```
class NoeudTri (object):
    def __init__(self,s): self.mot = s
    def __str__(self)   : return self.mot + "\n"

    def insere (self,s):
        c = cmp (s, self.mot)
        if   c == -1 : self.avant = NoeudTri (s)  # ajout d'un successeur
        elif c ==  1 : self.apres = NoeudTri (s)  # ajout d'un successeur
```

La méthode `insere` prévoit de ne rien faire dans le cas où le mot `s` passé en argument est égal à l'attribut `mot` : cela revient à ignorer les doublons dans la liste de mots à trier.

**4)**

```
class NoeudTri (object):
    def __init__(self,s): self.mot = s

    def __str__(self):
        s = ""
        if "avant" in self.__dict__: s += self.avant.__str__ ()
        s += self.mot + "\n"
        if "apres" in self.__dict__: s += self.apres.__str__()
        return s

    def insere (self,s):
        c = cmp (s, self.mot)
        if   c == -1 : self.avant = NoeudTri (s)
        elif c ==  1 : self.apres = NoeudTri (s)
```

L'insertion des mots donnés dans l'énoncé produit le code suivant :

```
deux
un
unite
```

**5) 6)** Il reste à compléter la fonction `insere` afin qu'elle puisse trouver le bon nœud où insérer un nouveau mot. Cette méthode est récursive : si un nœud contient deux attributs `avant` et `apres`, cela signifie que le nouveau mot doit être inséré plus bas, dans des nœuds reliés soit à `avant` soit à `apres`. La méthode `insere` choisit donc un des attributs et délègue le problème à la méthode `insere` de ce nœud.

```
# -*- coding: cp1252 -*-
import string

class SecondeInserstion (AttributeError):
    "insertion d'un mot déjà inséré"

class NoeudTri :

    def __init__(self,s): self.mot = s

    # la création d'un nouveau noeud a été placée dans une méthode
    def nouveau_noeud (self, s) : return NoeudTri (s)

    def __str__(self):
        s = ""
        if "avant" in self.__dict__: s += self.avant.__str__ ()
        s += self.mot + "\n"
        if "apres" in self.__dict__: s += self.apres.__str__()
        return s

    def insere (self,s):
        c = cmp (s, self.mot)
```

```
        if c == -1:
            if "avant" in self.__dict__ : self.avant.insere (s) # délégation
            else :  self.avant = self.nouveau_noeud (s)         # création
        elif c == 1:
            if "apres" in self.__dict__ : self.apres.insere (s) # délégation
            else: self.apres = self.nouveau_noeud (s)           # création
        else:
            raise SecondeInsertion, "mot : " + s

l = ["un", "deux", "unite", "dizaine", "exception", "dire", \
     "programme", "abc", "xyz", "opera", "quel"]

racine = None
for mot in l :
    if racine == None :
        # premier cas : aucun mot --> on crée le premier noeud
        racine = NoeudTri (mot)
    else :
        # second cas : il y a déjà un mot, on ajoute le mot suivant
        # à l'arbre
        racine.insere (mot)

print racine
```

Chaque nouveau mot va partir du tronc pour s'accrocher à une feuille de l'arbre pour devenir à son tour une feuille. La méthode `nouveau_noeud` crée un nouveau nœud dans le graphe. Son utilité est mise en évidence par le programme suivant.

**7)** La figure 11.6 détient le graphe obtenu par le programme qui suit. Plutôt que de modifier la classe `NoeudTri`, une seconde est créée qui hérite de la première. On lui adjoint la méthode `chaine_graphe` qui convertit un graphe en une chaîne de caractères dont le format reprend celui énoncé plus haut. Cette fonction s'occupe de construire récursivement cette chaîne de caractères. Pour identifier chaque nœud, on utilise la fonction `id` qui retourne un identifiant distinct pour chaque instance de classe.

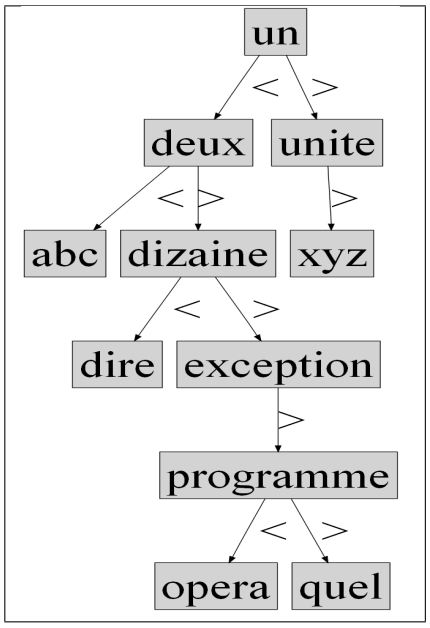



**Figure 11.6 :** *Graphe de tri obtenu lors du tri quicksort. Chaque nœud du graphe inclut un mot. Les symboles "<" et ">" des arcs désignent les membres* `avant` *et* `apres` *de la classe* `NoeudTri`*. Tous les mots attachés à un arc "<" d'un nœud sont classés avant le mot de ce nœud. De même, tous les mots attachés à un arc ">" d'un nœud sont classés après le mot de ce nœud.*

```
# -*- coding: cp1252 -*-
import string
import pydot
import quicksort
```

```
class NoeudTri2 (quicksort.NoeudTri):

    def chaine_graphe (self):
        # le principe est le même que pour la méthode __str__
        # excepté que le format est différent
        g = str (id (self)) + " [label=\"" + self.mot \
                            + "\",style=filled,shape=record,fontsize=60]\n"
        if "avant" in self.__dict__:
            h  = self.avant.chaine_graphe ()
            g += h + str (id (self)) + " -> " + str (id (self.avant)) \
                   + " [label=\"<\",fontsize=60]" + "\n"
        if "apres" in self.__dict__:
            h  = self.apres.chaine_graphe ()
            g += h + str (id (self)) + " -> " + str (id (self.apres)) \
                   + " [label=\">\",fontsize=60]" + "\n"
        return g

    def nouveau_noeud (self, s) : return NoeudTri2 (s)

    def image (self, file, im) :
        # on crée un graphe : on récupère le code du graphe
        graph = self.chaine_graphe ()
        # auquel on ajoute un début et une fin
        graph = "digraph GA {\n" + graph + "}\n"

        # puis on l'écrit dans le fichier file
        g = open (file, "w")
        g.write (graph)
        g.close ()

        # enfin, on convertit ce fichier en image
        dot = pydot.graph_from_dot_file (file)
        dot.write_png (im, prog="dot")

def construit_arbre () :
    # même code que dans le programme précédent
    # mais inclus dans une fonction
    l = ["un", "deux", "unite", "dizaine", "exception", "dire", \
         "programme", "abc", "xyz", "opera", "quel"]
    racine = None
    for mot in l :
        if racine == None : racine = NoeudTri2 (mot)
        else : racine.insere (mot)
    return racine

racine = construit_arbre ()
print racine
racine.image ("graph.txt", "graph.png")
```

La méthode `nouveau_noeud` permet de s'assurer que tous les nœuds insérés lors de la création du graphe seront bien du type `NoeudTri2` qui inclut la méthode `chaine_graphe`. Cette méthode serait inutile s'il n'y avait qu'une seule classe `NoeudTri` contenant toutes les méthodes. Si on la met en commentaire, le message d'erreur suivant apparaît :

```
Traceback (most recent call last):
  File "quicksort2.py", line 53, in <module>
    racine.image ("graph.txt", "graph.png")
  File "quicksort2.py", line 27, in image
    graph = self.chaine_graphe ()
  File "quicksort2.py", line 14, in chaine_graphe
    h  = self.avant.chaine_graphe ()
AttributeError: NoeudTri instance has no attribute 'chaine_graphe'
```

L'erreur signifie que le programmeur cherche à appeler une méthode qui n'existe pas dans la classe `NoeudTri` parce que seul le premier nœud de l'arbre est de type `NoeudTri2` contrairement aux nœuds insérés par la méthode `nouveau_noeud` de la classe `NoeudTri`. En surchargeant cette méthode, on s'assure que tous les nœuds sont du même type `NoeudTri2`.

Au final, la méthode `image` construit l'image du graphe. Le fichier `graphe.txt` doit ressembler à ce qui suit :

```
digraph GA {
18853120 [label="un",style=filled,shape=record]
28505472 [label="deux",style=filled,shape=record]
28505712 [label="abc",style=filled,shape=record]
28505472 -> 28505712 [label="<"]
28505552 [label="dizaine",style=filled,shape=record]
28505592 [label="dire",style=filled,shape=record]
28505552 -> 28505592 [label="<"]
28505632 [label="exception",style=filled,shape=record]
28505672 [label="programme",style=filled,shape=record]
28505792 [label="opera",style=filled,shape=record]
28505672 -> 28505792 [label="<"]
28505832 [label="quel",style=filled,shape=record]
28505672 -> 28505832 [label=">"]
28505632 -> 28505672 [label=">"]
28505552 -> 28505632 [label=">"]
28505472 -> 28505552 [label=">"]
18853120 -> 28505472 [label="<"]
28505512 [label="unite",style=filled,shape=record]
28505752 [label="xyz",style=filled,shape=record]
28505512 -> 28505752 [label=">"]
18853120 -> 28505512 [label=">"]
}
```

La numérotation des nœuds importe peu du moment que chaque nœud reçoit un identifiant unique. C'est pour cela que la fonction `id` est pratique dans ce cas-là. Le programme suivant construit une sortie au format HTML mélangeant image et texte. Il commence par importer le programme `quicksort2` qui n'est autre que celui incluant la classe `NoeudTri2`. Il termine en appelant le navigateur *Mozilla Firefox* afin d'afficher le résultat automatiquement.

```
# -*- coding: cp1252 -*-
import quicksort2

# construction de l'arbre
racine = quicksort2.construit_arbre ()
# construction de l'image du graphe
racine.image ("graph.txt", "graph.png")

# création d'un fichier HTML
f = open ("page.html", "w")
f.write ("<body><html>\n")              # début

f.write ("<H1> liste triée </H1>\n")    # titre pour la liste triée
s = str (racine)                        # on récupère la liste triée
s = s.replace ("\n", "<BR>\n")          # <BR> permet de passer à la ligne
f.write (s)

f.write ("<H1> graphe </H1>\n")         # titre pour l'image
f.write ('<img src="graph.png" width=400/>\n')    # image

f.write ("<H1> code du graphe </H1>\n") # titre pour le code du graphe
s = racine.chaine_graphe ()             # on récupère le code du graphe
f.write ("<pre>\n")                     # on l'affiche tel quel
f.write (s)
f.write ("</pre>\n")
```

```
f.write ("</html></body>\n")            # fin
f.close ()

# on lance le navigateur automatiquement pour afficher la page
import os
os.system (r'"C:\Program Files\Mozilla Firefox\firefox" page.html')
```

Le fichier `page.html` contient les lignes suivantes excepté les points de suspension qui remplacent la partie tronquée qu'on peut aisément deviner.

```
<body><html>
<H1> liste triée </H1>
abc<BR>
deux<BR>
...
unite<BR>
xyz<BR>
<H1> graphe </H1>
<img src="graph.png" width=400/>
<H1> code du graphe </H1>
<pre>
13697312 [label="un",style=filled,shape=record,fontsize=60]
13697192 [label="deux",style=filled,shape=record,fontsize=60]
34692472 [label="abc",style=filled,shape=record,fontsize=60]
...
13697232 -> 34692592 [label=">",fontsize=60]
13697312 -> 13697232 [label=">",fontsize=60]
</pre>
</html></body>
```

On veut modifier ce programme pour que la sortie ne soit plus au format HTML mais au format PDF. Pour cela, on utilise comme intermédiaire le langage *Latex* qui s'occupe de créer ce fichier au format PDF. Les dernières lignes sont propres au système *Windows* où on suppose que la version de *Latex* installée est *Miktex 2.7*[15]. Il faut également le logiciel *Adobe Reader*[16].

```
# -*- coding: cp1252 -*-
import quicksort2

# construction de l'arbre
racine = quicksort2.construit_arbre ()
# construction de l'image du graphe
racine.image ("graph.txt", "graph.png")

# construction du début du fichier tex
package = """a4 amsmath amssymb subfigure float latexsym amsfonts
epic eepic makeidx multido varindex moreverb alltt fancyvrb fancyhdr
color eurosym tabularx placeins url shorttoc""".split ()

header = """\\documentclass[french,11pt]{article}\n\\usepackage[french]{babel}
\\usepackage[usenames]{color}\\usepackage{""" + \
                        "}\n\\usepackage{".join (package) + \
"""}\\usepackage[small,normal]{caption2}\\urlstyle{sf}
\\usepackage[pdftex]{graphicx}\usepackage[T1]{fontenc}
\DefineVerbatimEnvironment{verbatimx}{Verbatim}{frame=single,
framerule=.1pt, framesep=1.5mm, fontsize=\\footnotesize,xleftmargin=0pt}
\\begin{document}\n"""

# création d'un fichier tex
f = open ("page.tex", "w")
f.write (header)
```

15. http://miktex.org/, ce logiciel est volumineux et assez long à télécharger.
16. http://www.adobe.com/fr/products/acrobat/readstep2.html

```
f.write ("\\title{Tri quicksort}\n")    # définit le titre
f.write ("\\maketitle\n")               # écrit le titre
f.write ("\\tableofcontents\n")         # table des matières

f.write ("\\section{liste triée}\n")    # titre pour la liste triée
s = str (racine)                        # on récupère la liste triée
s = s.replace ("\n", "\\\\  \n")        # \\ passe à la ligne
f.write ("\\begin{tabular}{|l|}\n")
f.write (s)
f.write ("\\end{tabular}\n")

f.write ("\\section{graphe}\n")         # titre pour l'image
f.write ('\\includegraphics[height=5cm]{graph.png}\n')    # image

f.write ("\\section{code du graphe}\n") # titre pour le code du graphe
s = racine.chaine_graphe ()             # on récupère le code du graphe
f.write ("\\begin{verbatimx}\n")        # on l'affiche tel quel
f.write (s)
f.write ("\\end{verbatimx}\n")

f.write ("\\end{document}\n")            # fin
f.close ()

# on compile deux fois le fichier pour que la table des matières
# soit bien prise en compte
import os
os.system (r'"C:\Program Files\MiKTeX 2.7\miktex\bin\pdflatex" page.tex')
os.system (r'"C:\Program Files\MiKTeX 2.7\miktex\bin\pdflatex" page.tex')

# on affiche le résultat avec Adobe Reader
os.system (r'"C:\Program Files\Adobe\Reader 9.0\Reader\AcroRd32.exe" page.pdf')
```

La partie `header` est longue dans cet exemple, elle inclut des packages Latex qui ne sont pas utilisés mais qui pourraient l'être dans le cas d'un rapport plus long. Il faut bien sûr connaître quelques rudiments de ce langage pour construire le document PDF. L'avantage de ce système est de pouvoir retravailler manuellement le document final. Il est également indiqué lorsque le rapport mélange tableaux de chiffres récupérés depuis différentes sources et graphiques qui peuvent être générés par *Python* via un module comme `matplotlib` [17].

Sous *Linux*, il suffit de modifier les chemins d'accès aux différentes applications. Une dernière remarque, il existe sous *Windows* un éditeur convenable qui est *TeXnicCenter* [18].

**fin correction TD 11.5.1** □

17. http://matplotlib.sourceforge.net/
18. http://www.toolscenter.org/

# Chapitre 12

# Exercices pratiques pour s'évaluer

---

Les trois énoncés qui suivent sont prévus pour être réalisés en deux heures, ils ne font référence à aucun algorithme ou méthode connue. Il suffit de connaître les types standard du *Python*, les boucles, les tests et les fonctions. Le second énoncé nécessite également quelques notions sur les classes.

## 12.1 Recherche dichotomique (2005-2006)

L'objectif de cet exercice est de programmer une recherche dans une liste triée.

**1)** Il faut d'abord récupérer un fichier texte disponible sur Intranet [1]. Ce fichier contient un mot par ligne. Il faut lire ce fichier et construire une liste avec tous ces mots. (4 points)

**2)** Construire une fonction qui vérifie que la liste chargée à la question précédente est triée. (4 points)

**3)** Construire une fonction qui recherche un mot X dans la liste et qui retourne sa position ou -1 si ce mot n'y est pas. Cette fonction prend deux paramètres : la liste et le mot à chercher. Elle retourne un entier. On précise que pour savoir si deux chaînes de caractères sont égales, il faut utiliser l'opérateur ==. (4 points)

**4)** Quels sont les positions des mots "UN" et "DEUX" ? La réponse doit figurer en commentaire dans le programme. Il faudra écrire aussi le nombre de comparaisons effectuées pour trouver ces deux positions. (2 points)

**5)** Lorsqu'une liste est triée, rechercher un élément est beaucoup plus rapide. Si on cherche le mot X dans la liste, il suffit de le comparer au mot du milieu pour savoir si ce mot est situé dans la partie basse (X inférieur au mot du milieu), la partie haute (X supérieur au mot du milieu). S'il est égal, le mot a été trouvé. Si le mot n'a pas été trouvé, on recommence avec la sous-liste inférieure ou supérieure selon les cas jusqu'à ce qu'on ait trouvé le mot ou qu'on soit sûr que le mot cherché n'y est pas.

Le résultat de la recherche est la position du mot dans la liste ou -1 si ce mot n'a pas été trouvé. Cette recherche s'appelle une recherche dichotomique.

Ecrire la fonction qui effectue la recherche dichotomique d'un mot dans une liste triée de mots. Vérifiez que les deux fonctions retournent bien les mêmes résultats. Cette fonction peut être récursive ou non. Elle prend au moins les deux mêmes paramètres que ceux de la question 3, si elle en a d'autres, il faudra leur donner une valeur par défaut. On précise que les comparaisons entre chaînes de caractères utilisent aussi les opérateurs <, ==, >. (4 points)

**6)** Normalement, les positions des mots "UN" et "DEUX" n'ont pas changé mais il faut de nouveau déterminer le nombre d'itérations effectuées pour trouver ces deux positions avec la recherche dichotomique. (2 points)

**7)** Quel est, au pire [2], le coût d'une recherche non dichotomique ? La réponse doit figurer en commentaire dans le programme. (1 point)

**8)** Quel est, au pire, le coût d'une recherche dichotomique ? La réponse doit figurer en commentaire dans le programme. (1 point)

1. http://www.xavierdupre.fr/enseignement/initiation/td_note_texte.txt
2. On cherche la meilleure majoration du coût de la recherche non dichotomique en fonction de la taille $n$ de la liste.

### Correction

```
# -*- coding: latin-1 -*-
# question 1
def lit_fichier (file) :
    f = open (file, "r")
    mot = []
    for l in f :
        mot.append ( l.replace ("\n", "") )
    f.close ()
    return mot

mot = lit_fichier ("td_note_texte.txt")
print mot

# question 2
def est_trie (mot) :
    for i in range (1, len (mot)) :
        if mot [i-1] > mot [i] :
            return False
    return True

tri = est_trie (mot)
print "liste triée ", tri

# question 3
def cherche (mot, m) :
    for i in range (0, len (mot)) :
        if mot [i] == m :
            return i
    return -1

print "mot ACHATS ", cherche (mot, "ACHATS")
print "mot achats ", cherche (mot, "achats")

# question 4
un   = cherche (mot, "UN")
deux = cherche (mot, "DEUX")
print "recherche normale ", un, deux
print "nombre d'itérations", un + deux

# question 5, 6, nbun et nbdeux contiennent le nombre de comparaisons
def cherche_dicho (mot, m) :
    a = 0
    b = len (mot)-1
    nb = 0
    while a < b :
        nb += 1
        p = (a+b)/2
        if   mot [p] == m :  return p,nb
        elif mot [p] >  m :  b = p-1
        else :               a = p+1
    return -1,nb

un,nbun     = cherche_dicho (mot, "UN")
deux,nbdeux = cherche_dicho (mot, "DEUX")
print "recherche dichotomique ", un, deux
print "nombre d'itérations ", nbun + nbdeux

# question 7
"""
Lors d'une recherche simple, au pire, l'élément cherche sera
en dernière position, ce qui signifie n itérations pour le trouver.
Le coût de la recherche simple est en O(n).
"""
```

```
# question 8
"""
Lors de la recherche dichotomique, à chaque itération, on divise par deux
l'ensemble dans lequel la recherche s'effectue,
au départ n, puis n/2, puis n/4 jusqu'àă ce que n/2^k soit nul
c'est-à-dire k = partie entière de ln n / ln 2
il y a au plus k itérations donc le coût de l'algorithme est en O (ln n).
"""
```

**fin exo 1** □

## 12.2 Ajouter un jour férié (2006-2007)

### 12.2.1 Avec des fonctions uniquement

**Enoncé**

Le gouvernement désire ajouter un jour férié mais il voudrait le faire à une date éloignée des jours fériés existant. On suppose également que ce jour ne sera pas inséré entre Noël et le jour de l'an. On va donc calculer le nombre de jours qui sépare deux jours fériés dont voici la liste pour l'année 2007 :

| | |
|---|---|
| Jour de l'an | 1er janvier 2007 |
| Lundi de Pâques | 9 avril 2007 |
| Fête du travail | 1er mai 2007 |
| Victoire de 1945 | 8 mai 2007 |
| Ascension | 17 mai 2007 |
| Lundi de Pentecôte | 4 juin 2007 |
| Fête nationale | 14 juillet 2007 |
| Assomption | 15 août 2007 |
| Toussaint | 1er novembre 2007 |
| Armistice de 1918 | 11 novembre 2007 |
| Noël | 25 décembre 2007 |

On rappelle que l'année 2007 n'est pas une année bissextile et qu'en conséquence, le mois de février ne comporte que 28 jours.

**1)** Afin de simplifier la tâche, on cherche à attribuer un numéro de jour à chaque jour férié : l'année a 365 jours, pour le numéro du lundi de Pâques, soit 31 (mois de janvier) + 28 (février) + 31 (mars) + 9 = 89. La première question consiste à construire une fonction qui calcule le numéro d'une date étant donné un jour et un mois. Cette fonction prend comme entrée : (4 points)

- un numéro de jour
- un numéro de mois
- une liste de 12 nombres correspondant au nombre de jours dans chacun des douze mois de l'année

**2)** Si on définit la liste des jours fériés comme étant une liste de couples (jour, mois) triée par ordre chronologique, il est facile de convertir cette liste en une liste de nombres correspondant à leur numéro dans l'année. La fonction à écrire ici appelle la précédente et prend une liste de couples en entrée et retourne comme résultat une liste d'entiers. (4 points)

**3)** A partir de cette liste d'entiers, il est facile de calculer l'écart ou le nombre de jours qui séparent deux jours fériés. Il ne reste plus qu'à écrire une fonction qui retourne l'écart maximal entre deux jours fériés, ceux-ci étant définis par la liste de numéros définie par la question précédente. Un affichage du résultat permet de déterminer les deux jours fériés les plus éloignés l'un de l'autre. Quels sont-ils ? (4 points)

### 12.2.2 Programme équivalent avec des classes

#### Enoncé

Le programme précédent n'utilise pas de classe. L'objectif de ce second exercice est de le réécrire avec une classe.

**1)** Une fonction du programme précédent effectue la conversion entre un couple jour-mois et un numéro de jour. Les calculs sont faits avec le numéro mais le résultat désiré est une date : les numéros ne sont que des intermédiaires de calculs qui ne devraient pas apparaître aussi explicitement. La première question consiste à créer une classe `Date` : (2 points)

```
class Date :
    def __init__ (self, jour, mois) :
        ...
```

**2)** A cette classe, on ajoute une méthode qui retourne la conversion du couple jour-mois en un numéro de jour de l'année. (2 points)

**3)** On ajoute maintenant une méthode calculant le nombre de jours séparant deux dates (ou objet de type `Date` et non pas numéros). Cette méthode pourra par exemple s'appeler `difference`. (2 points)

**4)** Il ne reste plus qu'à compléter le programme pour obtenir les mêmes résultats que le programme de l'exercice 1. (2 points)

**5)** Avec ce programme, lors du calcul des écarts entre tous les jours fériés consécutifs, combien de fois effectuez-vous la conversion du couple jour-mois en numéro pour le second jour férié de l'année ? Est-ce le même nombre que pour le programme précédent (en toute logique, la réponse pour le premier programme est 1) ? (2 points)

**6)** La réponse à la question précédente vous suggère-t-elle une modification de ce second programme ? (2 points)

#### Correction

```
# -*- coding: latin-1 -*-
####################################
# exercice 1
####################################

# question 1
def numero (jour, mois, duree = [31, 28, 31,30,31,30,31,31,30,31,30,31] ) :
    s = 0
    for i in range (0,mois-1) :
        s += duree [i]
    s += jour - 1
    return s+1

# question 2
def conversion_liste (li) :
    res = []
    for jour,mois in s : res.append ( numero (jour, mois))
    # pareil que
    # for i in range (0, len (s)) : res.append ( numero (s [i][0], s [i][1]))
    return res

def ecart (num) :
    res = []
    for i in range (1, len (num)) :
        d = num [i] - num [i-1]
        res.append (d)
    return res
```

```
s = [ (1,1), (9,4), (1,5), (8,5), (17,5), (4,6), (14,7), \
        (15,8), (1,11), (11,11), (25,12) ]
r = conversion_liste (s)
ec = ecart (r)

# question 3
pos = ec.index ( max (ec) )
print "position de l'écart le plus grand ", pos
print "jour ", s [pos], " --> ", s [pos+1]

####################################
# exercice 2
####################################

# question 4
class Date :
    def __init__ (self, jour, mois) :
        self.jour = jour
        self.mois = mois
        self.duree = [31, 28, 31,30,31,30,31,31,30,31,30,31]

    # question 5
    def numero (self) :
        s = 0
        for i in range (0,self.mois-1) :
            s += self.duree [i]
        s += self.jour - 1
        return s+1

    # question 6
    def difference (self, autre) :
        return self.numero () - autre.numero ()

def conversion_date (s) :
    res = []
    for jour,mois in s :
        res.append ( Date (jour, mois) )
    return res

def ecart_date (date) :
    ec = []
    for i in range (1, len (date)) :
        ec.append ( date [i].difference ( date [i-1] ) )
    return ec


# question 7
s = [ (1,1), (9,4), (1,5), (8,5), (17,5), (4,6), \
            (14,7), (15,8), (1,11), (11,11), (25,12) ]

r = conversion_date (s)
ec = ecart_date (r)
pos = ec.index ( max (ec) )
print "position de l'ecart le plus grand ", pos
print "jour ", s [pos], " --> ", s [pos+1]

# question 8
"""
La conversion en Date est faite une fois pour les dates (1,1) et (25,12)
et 2 fois pour les autres en effet, la méthode difference effectue
la conversion en numéros des dates self et autre
la fonction ecart_date calcule date [i].difference ( date [i-1] ) et
                                    date [i+1].difference ( date [i] )
            --> la date [i] est convertie 2 fois
```

```
"""

# question 9
"""
On peut par exemple stocker la conversion en numéro
dans le constructeur comme suit :
"""

class Date :
    def __init__ (self, jour, mois) :
        self.jour = jour
        self.mois = mois
        self.duree = [31, 28, 31,30,31,30,31,31,30,31,30,31]
        self.num = self.numero ()

    # question 5
    def numero (self) :
        s = 0
        for i in range (0,self.mois-1) :
            s += self.duree [i]
        s += self.jour - 1
        return s+1

    # question 6
    def difference (self, autre) :
        return self.num - autre.num


r = conversion_date (s)
ec = ecart_date (r)
pos = ec.index ( max (ec) )
print "position de l'écart le plus grand ", pos
print "jour ", s [pos], " --> ", s [pos+1]
```

On pourrait encore améliorer la dernière classe `Date` en créant un attribut statique pour l'attribut `duree` qui est identique pour toutes les instances de la classe `Date`.

**fin exo 12.2.1** □

## 12.3 Fréquentation d'un site Internet (2007-2008)

Lorsqu'on se connecte à un site internet, celui-ci enregistre votre adresse IP, l'heure et la date de connexion ainsi que la page ou le fichier désiré. Ainsi le fichier *logpdf.txt*[3] recense ces quatre informations séparées par des espaces pour les fichiers d'extension *pdf* téléchargés sur le site http://www.xavierdupre.fr/.

**1)** La première étape consiste à charger les informations depuis le fichier texte dans une matrice de 4 colonnes (liste de listes, liste de t-uples). On utilisera pour cela les fonctions `open` et les méthodes `split`, `replace` associées aux chaînes de caractères. On pourra s'aider des corrections des TD précédents. (2 points)

**2)** On souhaite dans un premier temps faire des statistiques sur les dates : on veut compter le nombre de fichiers téléchargés pour chaque date présente dans les données. On pourra pour cela utiliser un dictionnaire dont la clé sera bien choisie. Le résultat devra être inséré dans une fonction prenant comme entrée une matrice (ou liste de listes, liste de t-uples) et retournant un dictionnaire comme résultat. (4 points)

**3)** On s'intéresse au programme suivant :

```
l = [ (1, "un"), (3, "deux",), (2, "deux"), (1, "hun"), (-1, "moinsun") ]
l.sort (reverse = True)
print l
```

3. Il faut télécharger ce fichier depuis l'adresse http://www.xavierdupre.fr/enseignement/initiation/logpdf.txt

Donc le résultat est :

```
[(3, 'deux'), (2, 'deux'), (1, 'un'), (1, 'hun'), (-1, 'moinsun')]
```

Que s'est-il passé ? (2 points)

**4)** On désire maintenant connaître les 10 dates pour lesquelles il y a eu le plus de téléchargements ces jours-là. L'inconvénient est que le dictionnaire élaboré à la question 2 ne retourne pas les réponses dans l'ordre souhaité : il faut classer les dates par nombre de téléchargements croissants. Il faut ici imaginer une fonction qui retourne ces dix meilleures dates et des dix fréquentations correspondantes en se servant de la remarque de la question 3. (4 points) A quels événements correspondent les quatre premières dates ? (Cette petite question ne rapporte pas de point.)

**5)** Effectuez le même travail pour déterminer les dix documents les plus téléchargés. (2 points)

**6)** Ecrire une fonction qui retourne l'heure sous forme d'entier à partir d'une date définie par une chaîne de caractères au format "hh : mm : ss". Par exemple, pour "14 : 55 : 34", la fonction doit retourner 14 sous forme d'entier. L'instruction int("14") convertit une chaîne de caractères en un entier. (2 points)

**7)** Calculer le nombre de documents téléchargés pour chaque heure de la journée. Le site est-il consulté plutôt le matin ou le soir ? Serait-il possible de conclure aussi rapidement pour un site d'audience internationale ? (4 points)

### Correction

```
# -*- coding: latin-1 -*-
# la première ligne autorise les accents dans un programme Python
# la langue anglaise est la langue de l'informatique,
# les mots-clés de tous les langages
# sont écrits dans cette langue.

####################################
# exercice 1
####################################
#

# question 1
def lit_fichier (file) :
    f = open (file, "r")
    li = f.readlines ()           # découpage sous forme de lignes
    f.close ()
    res = []
    for l in li :
        s = l.replace ("\n", "")
        s = s.split (" ")         # le séparateur des colonnes est l'espace
        res.append (s)
    return res


mat = lit_fichier ("logpdf.txt")
for m in mat [0:5] :       # on affiche les 5 premières lignes
    print m                # parce que sinon, c'est trop long

# question 2
def compte_date (mat) :
    d = { }
    for m in mat :
        date = m [1]   # clé
        if date in d : d [date] += 1
        else : d [date] = 1
    return d

dico_date = compte_date (mat)
print dico_date
```

```
# remarque générale : si le fichier logpdf.txt contient des lignes
# vides à la fin, il se produira une erreur à la ligne 34 (date = m [1])
# car la matrice mat contiendra des lignes avec une seule colonne et non quatre.
# Il suffit soit de supprimer les lignes vides du fichier logpdf.txt
# soit de ne pas les prendre en compte lors de la lecture de ce fichier.

# question 3
# La méthode sort trie la liste mais comment ?
# Il est facile de trier une liste de nombres mais une liste de couples de
# nombres ? L'exemple montre que la liste est triée selon le premier élément de
# chaque couple. Pour les cas où deux couples ont un premier élément en commun,
# les éléments semblent triés selon le second élément. L'exemple suivant le monte :

l = [(1, "un"), (3, "deux",), (2, "deux"), (1, "hun"), (1, "un"), (-1, "moinsun")]
l.sort (reverse = True)
print l  # affiche [(3, 'deux'), (2, 'deux'), (1, 'un'),
         #          (1, 'un'),   (1, 'hun'), (-1, 'moinsun')]

# question 4
def dix_meilleures (dico) :
    # dans cette fonction on crée une liste de couples (valeur,clé) ou
    # la clé représente une date et valeur le nombre de téléchargement
    # pour cette date
    li = []
    for d in dico :
        cle = d
        valeur = dico [cle]
        li.append ( ( valeur, cle ) )
    li.sort (reverse = True)
    return li [0:10]

dix = dix_meilleures (dico_date)
print dix # la première date est (283, '26/Sep/2007')

# les quatre premières dates correspondent aux quatre premiers TD en 2007 à l'ENSAE

# question 5
# la date est en colonne 1, le document en colonne 3
# on fait un copier-coller de la fonction compte_date en changeant un paramètre
def compte_document (mat) :
    d = { }
    for m in mat :
        doc = m [3]   # clé, 3 au lieu de 1 à la question 2
        if doc in d : d [doc] += 1
        else : d [doc] = 1
    return d

dix = dix_meilleures ( compte_document (mat) )
print dix   # le premier document est
            # (323, '/mywiki/Enseignements?.....target=python_cours.pdf'),

# question 6
def heure (s) :
    hs = s [0:2]           # on extrait la partie correspondant à l'heure
    return int (hs)        # on retourne la conversion sous forme d'entiers

# question 7
# on recommence avec un copier-coller
def compte_heure (mat) :
    d = { }
    for m in mat :
        h = m [2]   # clé, 2 au lieu de 1 à la question 2
        cle = heure (h)
        if cle in d : d [cle] += 1
```

```
        else : d [cle] = 1
    return d

h = compte_heure (mat)
dix = dix_meilleures ( h )
print dix   # la première heure est  (432, 17), ce qui correspond à l'heure des TD

for i in h :
    print i, "h  ", h [i]

# Il y a beaucoup plus de téléchargement entre 20h et 2h du matin
# que le matin avant 10h.
# Le site est plutôt consulté le soir.
# La conclusion ne serait pas aussi évidente avec un site consulté par des gens
# du monde entier puisque 6h du matin est une heure de l'après midi au Japon.
# Il faudrait croiser l'heure avec la position géographique de la personne
# qui consulte le site.
```

**fin exo 4** □

## 12.4 Ajout d'une pièce de monnaie (2008-2009)

Le président de la république souhaite créer une pièce de monnaie à son effigie à condition que celle-ci facilite la vie des citoyens. Il voudrait que celle-ci soit d'un montant compris entre 1 et 10 euros et qu'elle permette de construire la somme de 99 euros avec moins de pièces qu'actuellement. Les questions qui suivent ont pour but de décomposer en pièces n'importe quel montant avec un jeu de pièces donné.

**8)** On considère que `pieces` est une liste de pièces triées par ordre croissant. Compléter le programme suivant afin d'obtenir la liste triée par ordre décroissant [4]. (2 points)

```
pieces = [1,2,5,10,20,50]
...
```

**9)** On souhaite écrire une fonction qui prend comme argument un montant `m` et une liste `pieces` toujours triée en ordre décroissant. Cette fonction doit retourner la plus grande pièce inférieure ou égale au montant `m`. (3 points)

**10)** En utilisant la fonction précédente, on veut décomposer un montant `m` en une liste de pièces dont la somme est égale au montant. On construit pour cela une seconde fonction qui prend aussi comme argument un montant `m` et une liste de pièces `pieces`. Cette fonction retourne une liste de pièces dont la somme est égale au montant. Une même pièce peut apparaître plusieurs fois. L'algorithme proposé est le suivant :

1. La fonction cherche la plus grande pièce inférieure ou égale au montant.
2. Elle ajoute cette pièce au résultat puis la soustrait au montant.
3. Si la somme restante est toujours positive, on retourne à la première étape.

Cet algorithme fonctionne pour le jeu de pièces donné en exemple et décompose 49 euros en $49 = 20 + 20 + 5 + 2 + 2$. (5 points)

**11)** Prolongez votre programme pour déterminer avec cet algorithme le plus grand montant compris entre 1 et 99 euros inclus et qui nécessite le plus grand nombre de pièces ? (2 points)

**12)** On ajoute la pièce 4 à la liste des pièces :

```
pieces = [1,2,4,5,10,20,50]
```

Que retourne l'algorithme précédent comme réponse à la question 3 avec cette nouvelle liste et pour le montant 98 ? Est-ce la meilleure solution ? (2 points)

4. On peut s'aider de tout type de document. Il est possible d'utiliser les méthodes associées à la classe `list` qu'on peut aussi retrouver via un moteur de recherche internet avec la requête *python list*.

**13)** Comme l'algorithme précédent ne fournit pas la bonne solution pour tous les montants, il faut imaginer une solution qui puisse traiter tous les jeux de pièces. On procède par récurrence. Soit $P = (p_1, \ldots, p_n)$ l'ensemble des pièces. On définit la fonction $f(m, P)$ comme étant le nombre de pièces nécessaire pour décomposer le montant $m$. On vérifie tout d'abord que :

1. La fonction $f(m, P)$ doit retourner 1 si le montant $m$ est déjà une pièce.
2. La fonction $f(m, P)$ vérifie la relation de récurrence suivante :

$$f(m, P) = \min \{f(m - p, P) + 1 \text{ pour tout } p \in P\}$$

Cet algorithme permet de construire la meilleure décomposition en pièces pour tout montant. La dernière expression signifie que si le montant $m$ est décomposé de façon optimale avec les pièces $(p_1, p_2, p_3, p_4)$ alors le montant $m - p_4$ est aussi décomposé de façon optimale avec les mêmes pièces $(p_1, p_2, p_3)$.

Il ne reste plus qu'à implémenter la fonction [5] $f(m, P)$. (5 points)

**14)** Qu'obtient-on avec le jeu de pièces `pieces` = $[1, 2, 4, 5, 10, 20, 50]$ ? Prolongez le programme pour déterminer tous les choix possibles du président parmi les pièces [3,4,6,7,8,9] ? (1 point)

**15)** Quel est le coût de votre algorithme ? (facultatif)

**Correction**

```
# -*- coding: latin-1 -*-

####################################
# question 1 : retourner un tableau
####################################

pieces = [1,2,5,10,20,50]
pieces.reverse ()
print pieces # affiche [50, 20, 10, 5, 2, 1]

# il existait d'autres solutions
pieces.sort (reverse = True)

# ou encore l'utilisation d'une fonction compare modifiée
# qui s'inspire de l'exemple
# http://www.xavierdupre.fr/enseignement/initiation/...
# ...initiation_via_python_ellipse/chap2_type_tex_-_tri-_3.html

# on encore un tri programmé
# http://www.xavierdupre.fr/enseignement/initiation/...
# ...initiation_via_python_ellipse/chap3_syntaxe_tex_-_tri-_27.html

#####################################################################
# question 2 : trouve la plus grande pièce inférieure à un montant
#####################################################################

def plus_grande_piece (montant, pieces) :
    # on suppose que les pièces sont triées par ordre décroissant

    for p in pieces :
        if p <= montant : return p

    # on peut ajouter la ligne
    return 0
    # qui correspond au fait qu'aucune pièce plus petite ou égale au montant
    # n'a été trouvé --> le montant est négatif ou nul

# on vérifie
```

5. Même si l'algorithme est présenté de façon récurrente, il peut être implémenté de façon récurrente ou non. La méthode récurrente est toutefois déconseillée car beaucoup plus lente.

```
print "plus_grande_piece (80, pieces) =", plus_grande_piece (80, pieces)
                  # affiche 50

####################################
# question 3 : décomposer un montant
####################################

def decomposer (montant, pieces) :
    # on suppose que les pièces sont triées par ordre décroissant
    res = [ ]  # contiendra la liste des pièces pour le montant
    while montant > 0 :
        p = plus_grande_piece (montant, pieces)  # on prend la plus grande pièce
                                                 # inférieure ou égale au montant
        res.append (p)                           # on l'ajoute à la solution
        montant -= p                             # on ôté p à montant :
                                      # c'est ce qu'il reste encore à décomposer
    return res

print "decomposer (98, pieces) =", decomposer (98, pieces)
                # affiche [50, 20, 20, 5, 2, 1]

print "decomposer (99, pieces) =", decomposer (99, pieces)
                # affiche [50, 20, 20, 5, 2, 2]

#####################################################
# question 4 : trouver la décomposition la plus grande
#####################################################

def maximum_piece (pieces) :
    # détermine le nombre maximum de pièces à utiliser
    maxi    = 0
    montant = 0
    for m in range (1, 100) :
        r = decomposer (m, pieces)
        if len (r) >= maxi :    # si on remplace cette ligne par if len (r) > maxi :
            maxi    = len (r)   # on trouve le plus petit montant
                                # avec la pire décomposition
            montant = m         # et non le plus grand montant
                                # avec la pire décomposation
    return maxi, montant

print "maximum_piece (pieces) =", maximum_piece (pieces) # affiche (6, 99)

#################################
# question 5 : décomposition de 98
#################################

pieces4 = [1,2,4,5,10,20,50]
pieces4.reverse ()

print "decomposer (98, pieces) = ", decomposer (98, pieces)    # [50, 20, 20, 5, 2, 1]
print "decomposer (98, pieces4) = ", decomposer (98, pieces4)  # [50, 20, 20, 5, 2, 1]

"""
Les deux décompositions sont identiques.
Or il existe une décomposition plus courte avec la pièce 4 :
98 = 50 + 20 + 20 + 4 + 4 = 5 pièces

L'algorithme fait la même erreur lorsqu'il décompose le montant 8.
Il cherche toujours la plus grande pièce inférieure au montant qui est 5.
Il lui est alors impossible d'utiliser la pièce 4  pour décomposer 8.

Cet algorithme ne fournit pas la bonne solution avec ce nouveau jeu de pièces.
"""
```

```
################################
# question 6 : algorithme optimal
################################

# version récursive : très longue
def decomposer_optimal (montant, pieces) :
    if montant in pieces :
        return [ montant ]
    else :
        r = [ 1 for m in range (0, montant) ]
        for p in pieces :
            if montant > p : # si ce test n'est pas fait, la récurrence peut être infinie
                             # car les montants négatifs ne sont pas pris en compte
                             # par le premier test
                dec = decomposer_optimal (montant - p, pieces) + [p]
                if len (dec) < len (r) :
                    r = dec

        return r

# print "decomposer_optimal (98, pieces4) =", decomposer_optimal (98, pieces4)
# trop long

# version non récursive
def decomposer_optimal (montant, pieces) :
    memo = [ [ 1 for l in range (0, m) ] for m in range (0, montant+1) ]
    # memo [i] contient la pire décomposition du montant i (que des pièces de un)

    # pour les pièces de pieces, on sait faire plus court
    for p in pieces :
        if p < len (memo) :
            memo [p] = [ p ]

    for m in range (1, montant+1) :
        for p in pieces :
            if m > p :
                # on calcule la nouvelle décomposition
                dec = [p] + memo [m-p]
                # si elle est plus courte, on la garde
                if len (dec) < len (memo [m] ) :
                    memo [m] = dec

    # on retourne la meilleur décomposition pour montant
    return memo [ montant ]

# beaucoup plus rapide
print "decomposer_optimal (98, pieces4) =",  decomposer_optimal (98, pieces4)
             # affiche [50, 20, 20, 4, 4]


########################
# question 7 : résultat
########################


# pour trouver la décomposition la plus longue avec n'importe quel jeu de pièces
# on reprend la fonction maximum_piece et on remplace decomposer par decomposer optimale

def maximum_piece (pieces) :
    # détermine le nombre maximum de pièces à utiliser
    maxi    = 0
    montant = 0
    for m in range (1, 100) :
        r = decomposer_optimal (m, pieces)
        if len (r) >= maxi :    # si on remplace cette ligne par if len (r) > maxi :
```

```
            maxi    = len (r)   # on trouve le plus petit montant
                                # avec la pire décomposation
            montant = m         # et non le plus grand montant
                                # avec la pire décomposation
    return maxi, montant

print "maximum_piece (pieces) =", maximum_piece (pieces) # affiche (6, 99)
print "maximum_piece (pieces4) =", maximum_piece (pieces4) # affiche (5, 99)


# on teste pour toutes les pièces [3,4,6,7,8,9]
# ajoutées au jeu de pièces standard [1,2,5,10,20,50]
ensemble = [3,4,6,7,8,9]

for ajout in [3,4,6,7,8,9] :
    pieces = [1,2,5,10,20,50] + [ ajout ]
    pieces.sort (reverse = True)
    print "maximum_piece (" + str (pieces) + ") = ", maximum_piece (pieces)

# résultat :
"""
maximum_piece ([50, 20, 10, 5, 3, 2, 1]) =  (6, 99)
maximum_piece ([50, 20, 10, 5, 4, 2, 1]) =  (5, 99)  # 4, ok
maximum_piece ([50, 20, 10, 6, 5, 2, 1]) =  (6, 99)
maximum_piece ([50, 20, 10, 7, 5, 2, 1]) =  (5, 99)  # 7, ok
maximum_piece ([50, 20, 10, 8, 5, 2, 1]) =  (5, 99)  # 8, ok
maximum_piece ([50, 20, 10, 9, 5, 2, 1]) =  (5, 98)  # 9, ok
"""


############################
# question 8 : décomposition
############################

"""
On cherche ici le coût de la fonction decomposer_optimal en fonction du montant.
Il n'y a qu'une seule boucle qui dépend du montant, le coût de la fonction est en O(n).

Dans la version récursive, le coût est le résultat d'une suite récurrente :
u(n) = u(n-1) + ... + u(n-d)
où d est le nombre de pièces.

Le coût est donc en O(a^n) où a est la plus grande des racines du polynôme :

P (x) = x^d - x^(d-1) - ... - 1

P(1) < 0 et lim P(x) = infini lorsque x tend vers infini,
donc la plus grande des racines est supérieure à 1.
Le coût de la fonction decomposer_optimal récursive est en O(a^n) avec 1,96 < a < 1,97.

Pour des explications plus conséquentes, voir la page
http://fr.wikipedia.org/wiki/Suite_r%C3%A9currente_lin%C3%A9aire
sur les suites récurrentes et l'exercice 12.3.3 du livre :
http://www.xavierdupre.fr/enseignement/initiation/...
 ...initiation_via_python_ellipse.pdf
(ou 13.3.3 http://www.xavierdupre.fr/enseignement/...
 ...initiation/initiation_via_python_small.pdf).
"""
```

L'algorithme qui décompose un montant de façon optimale quelque soient le montant et le jeu de pièces s'apparente à la programmation dynamique. C'est le même algorithme que celui qui permet de chercher le plus court chemin dans un graphe où les nœuds serait les montants de 0 à 99 et où il y aurait autant d'arcs que de pièces partant de chaque nœuds. Les arcs seraient tous de même poids : 1. Avec cette description, trouver la meilleure décomposition revient à trouver le chemin incluant le moins d'arcs possible.

**fin exo 5** □

## 12.5 Fusion de fichiers (2008-2009)

Cet exercice propose de fusionner des données provenant de deux sources. La première étape consiste à télécharger deux fichiers disponibles à l'adresse suivante : http://www.xavierdupre.fr/mywiki/InitiationPython :

td_note_2009_cluedo_1.txt et td_note_2009_cluedo_2.txt.

Il est préférable de placer ces pièces jointes dans le même répertoire que le progamme lui-même. Le premier fichier contient une matrice dont chaque ligne contient 4 informations séparées par des tabulations :

un nom un prénom une couleur une arme

Les premières lignes sont les suivantes [6] :

```
Adiceam          Faustin          rouge    poignard
Alik-Lagrange    Arthur           jaune    révolver
Anderhalt        Matthieu         blanc    matraque
```

Le second fichier contient une matrice dont chaque ligne contient 3 informations séparées par des tabulations :

un nom un prénom une pièce

Les premières lignes sont les suivantes [7] :

```
Bouley               Aymeric         cuisine
Bouvier              Alexandre       bureau
Castro du Plessis    José Patricio   salle à manger
```

Certaines élèves sont présents dans les deux fichiers, l'objectif de cet exercice et de construire un troisième fichier regroupant les informations concernant les élèves présents dans les deux fichiers. Ce fichier contiendra donc 5 colonnes.

**1)** Construire une fonction qui prend comme argument un nom de fichier et qui retourne le contenu du fichier dans une matrice (une liste de listes) [8].

```
def convertit_fichier_en_matrice (fichier) :
...
```

**2)** Construire une fonction qui prend comme entrée une matrice et qui construit un dictionnaire dont :
– la clé est un tuple composé des valeurs des deux premières colonnes (nom, prénom)
– la valeur est la liste des valeurs des autres colonnes

```
def convertit_matrice_en_dictionnaire (matrice) :
...
```

**3)** Construire une fonction qui effectue la fusion de deux dictionnaires. Le dictionnaire résultant vérifie :
– chaque clé est présente dans les deux dictionnaires à la fois

6. Ces données sont complètement aléatoires.
7. Ces données sont aussi complètement aléatoires.
8. On pourra s'inspirer du programme présent à l'adresse http://www.xavierdupre.fr/enseignement/initiation/initiation_via_python_all/chap7_fichiers_200_.html.

– la valeur associée à une clé est la somme des deux listes associées à cette même clé dans chacun des deux dictionnaires.

```
def fusion_dictionnaire (dico1, dico2) :
```

**4)** Construire une fonction qui convertit un dictionnaire en matrice. C'est la fonction réciproque de la fonction de la question 2. Les deux premières colonnes sont celles de la clé, les suivantes sont celles des valeurs.

```
def convertit_dictionnaire_en_matrice (dico) :
```

**5)** Ecrire une fonction qui écrit une matrice dans un fichier texte.

```
def convertit_matrice_en_fichier (mat, fichier) :
```

**6)** En utilisant toutes les fonctions précédentes, écrire une fonction qui effectue la fusion de deux fichiers et qui écrit le résultat dans un autre fichier.

```
def fusion_fichier (fichier1, fichier2, fichier_resultat) :
```

**7)** En vous inspirant du schéma proposé dans les précédentes questions, il s'agit ici de produire un troisième fichier qui contient tous les couples (nom, prénom) qui sont présents dans un fichier mais pas dans les deux à la fois.

```
def union_moins_intersection_fichier (fichier1, fichier2, fichier_resultat) :
```

**8)** Serait-il compliqué d'écrire une fonction qui permette de fusionner 3 fichiers ? $n$ fichiers ? Justifier.

**A quoi ça sert :** cette opération de fusion peut être effectuée sous Excel lorsque la taille des fichiers n'excède pas 65000 lignes. Au delà, cet énoncé propose un moyen de fusionner des fichiers de plusieurs millions de lignes.

### Correction

```
# -*- coding: latin-1 -*-
# Question 1

def convertit_fichier_en_matrice (file):
    f = open (file, "r")
    l = f.readlines ()
    f.close ()

    mat = list()
    for s in l :
        l = s.strip ("\n\r").split ("\t")
        mat.append (l)
    return mat

# Question 2

def convertit_matrice_en_dictionnaire (matrice) :
    d={}
    for line in matrice :
        d[line[0],line[1]] = line [2:]
    return d

# Question 3
```

```
def fusion_dictionnaire (dico1, dico2) :
    dico={}
    for k in dico1 :
        if k in dico2 : dico[k] = dico1[k] + dico2[k]
    return dico

# Question 4

def convertit_dictionnaire_en_matrice (dico) :
    m =[]
    for k,v in dico.iteritems () :
        line = list (k) + v
        m.append (line)
    return m

# Question 5

def convertit_matrice_en_fichier (mat,nomfichier):
    f = open (nomfichier, "w")
    for line in mat :
        f.write ( "\t".join ( [ str (x) for x in line ] ) + "\n")
    f.close ()

# Question 6

def fusion_fichier (fichier1, fichier2, fichier_resultat) :
    matrice1    = convertit_fichier_en_matrice(fichier1)
    matrice2    = convertit_fichier_en_matrice(fichier2)
    dico1       = convertit_matrice_en_dictionnaire(matrice1)
    dico2       = convertit_matrice_en_dictionnaire(matrice2)
    dico        = fusion_dictionnaire (dico1,dico2)
    matrice     = convertit_dictionnaire_en_matrice(dico)
    convertit_matrice_en_fichier (matrice,fichier_resultat)

fusion_fichier (    "td_note_2009_cluedo_1.txt",
                    "td_note_2009_cluedo_2.txt",
                    "cluedo.txt")

# Question 7

def fusion_dictionnairebis (dico1,dico2) :
    l1=dico1.keys()
    l2=dico2.keys()
    dico={}
    for k in l1 :
        if k not in l2 : dico[k]=[]
    for k in l2 :
        if k not in l1 : dico[k]=[]
    return dico

def union_moins_intersection_fichier (fichier1, fichier2, fichier_resultat):
    matrice1    = convertit_fichier_en_matrice(fichier1)
    matrice2    = convertit_fichier_en_matrice(fichier2)
    dico1       = convertit_matrice_en_dictionnaire(matrice1)
    dico2       = convertit_matrice_en_dictionnaire(matrice2)

    dico        = fusion_dictionnairebis (dico1,dico2)

    matrice     = convertit_dictionnaire_en_matrice(dico)
    convertit_matrice_en_fichier (matrice,fichier_resultat)

union_moins_intersection_fichier (  "td_note_2009_cluedo_1.txt",
                                    "td_note_2009_cluedo_2.txt",
                                    "cluedo2.txt")
```

```
# Question 8
"""
Il suffit de fusionner les fichiers deux par deux
en procédant par récurrence. On fusionne d'abord les
deux premiers, puis on fusionne le troisième
au résultat de la première fusion...
"""
```

**fin exo 6** □

# Chapitre 13

# Exercices écrits

---

Chacun des énoncés qui suit est une série d'exercices auxquels une personne expérimentée peut répondre en une heure. C'est souvent le temps imparti à un test informatique lors d'un entretien d'embauche pendant lequel le candidat propose des idées plus qu'il ne rédige des solutions. Chaque énoncé est un panel représentatif des problèmes que la programmation amène inévitablement. La difficulté de chaque exercice est notée du plus facile au plus difficile : *, **, ***. Le barème des énoncés est laissé à titre d'indication, il ne reflète pas toujours la difficulté de l'exercice.

## 13.1 Premier énoncé (2005-2006)

### 13.1.1 Programme à deviner *

**Enoncé**

Que fait le programme suivant ? Que contiendra `res` à la fin du programme ? (1 point)

```
l = [ 0,1,2,3,4,6,5,8,9,10]
res = True
for i in range (1,len (l)) :
    if l[i-1] > l[i] :
        res = False
```

**Correction**

Le programme vérifie que la liste `l` est triée, `res` vaut `True` si elle est triée, `False` sinon. Et dans ce cas précis, elle vaut `False` car la liste n'est pas triée (4,6,5). **fin exo 13.1.1** □

### 13.1.2 Somme des chiffres d'un nombre *

**Enoncé**

Ecrire une fonction qui calcule la somme des chiffres d'un entier positif. (2 points)

**Correction**

Tout d'abord, obtenir la liste des chiffres d'un nombre entier `n` n'est pas immédiat bien qu'elle puisse s'écrire en une ligne :

```
def somme (n) :
    return sum ( [ int (c) for c in str (n) ] )
```

Cette écriture résumée cache une conversion en chaîne de caractères. Une boucle est inévitable à moins de procéder par récurrence. Commençons par une version plus étendue et qui passe par les chaînes de caractères. Pour cette version, il ne faut pas oublier de faire la conversion inverse, c'est-à-dire celle d'un caractère en nombre.

```
def somme (n) :
    l = str (n)           # il ne faut pas confondre l=str (n) avec l = "n"
    s = 0
    for c in l :          # ou   for i in range (0, len (c)) :
        s += int (c)      # ou       s += int (c [i])
    return s
```

Une version numérique maintenant, celle qui utilise l'opération % ou modulo pour obtenir le dernier chiffre :

```
def somme (n) :
    s = 0
    while n > 0 :
        s += n % 10
        n /= 10    # ici, c'est une division entière, si vous n'êtes pas sûr :
                   #       n = int (n/10)
    return n
```

Enfin, une autre solution utilisant la récurrence et sans oublier la condition d'arrêt :

```
def somme (n) :
    if n <= 0 : return 0
    else : return (n % 10) + somme ( n / 10 )
```

Parmi les autres solutions, certaines, exotiques, ont utilisé la fonction `log` en base 10 ou encore la fonction `exp`. En voici une :

```
import math
def somme (n) :
    k = int (math.log (n) / math.log (10) + 1)
    s = 0
    for i in range (1,k+1) :
        d = 10 ** i   # ou encore d = int (exp ( k * log (10) ) )
        c = n / d
        e = n - c * d
        f = e / (d / 10)
        s += f
    return s
```

L'idée principale permet de construire une fonction retournant le résultat souhaité mais il faut avouer qu'il est plus facile de faire une erreur dans cette dernière fonction que dans les trois précédentes.

**fin exo 13.1.2** □

### 13.1.3 Calculer le résultat d'un programme **

#### Enoncé

Que vaut `n` à la fin de l'exécution du programme suivant ? Expliquez en quelques mots le raisonnement suivi. (2 points)

```
n = 0
for i in range (0,10) :
    if (n + i) % 3 == 0 :
        n += 1
```

**Correction**

L'ensemble `range(0,10)` est égale à la liste `[0,1,2,3,4,5,6,7,8,9]`. Dans ce programme, `n` est incrémenté lorsque `i+n` est un multiple de 3. La première incrémentation a lieu lors de la première itération puisque 0 est un multiple de 3. On déroule le programme dans le tableau suivant :

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| n avant le test | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 5 |
| i+n | 0 | 2 | 3 | 5 | 6 | 8 | 9 | 11 | 12 | 14 |
| n après le test | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 5 | 5 |

`range(0,10)` contient les nombres de 0 à 10 **exclu**. La réponse cherchée est donc 5. **fin exo 13.1.3** □

### 13.1.4 Suite récurrente (Fibonacci) *

**Enoncé**

Ecrire un programme qui calcule l'élément $u_{13}$ de la suite définie par :

$$\begin{array}{rrcl} & u_1 & = & 1 \\ & u_2 & = & 2 \\ \forall n \geqslant 2, & u_n & = & u_{n-1} + u_{n-2} \end{array}$$

$u_{13}$ est le nombre de manières possibles pour une grenouille de monter en haut d'une échelle de 13 barreaux si elle peut faire des bonds de un ou deux barreaux seulement. Si cette affirmation, qui n'est pas à démontrer, vous semble douteuse, avant d'y revenir, faites d'abord le reste de l'énoncé. (2 points)

**Correction**

Tout d'abord, la grenouille : elle peut faire des bonds de un ou deux barreaux à chaque fois. Donc, lorsqu'elle est sur le barreau $n$, elle a pu venir du barreau $n-1$ et faire un saut de 1 barreau ou venir du barreau $n-2$ et faire un bond de 2 barreaux. Par conséquent, le nombre de manières qu'a la grenouille d'atterrir sur le barreau $n$, c'est la somme des manières possibles de venir jusqu'au barreau $n-1$ et du barreau $n-2$.

Maintenant, comment le programmer ? Tout d'abord les solutions récursives, les plus simples à programmer :

```
def grenouille (n) :
    if   n == 2 : return 2
    elif n == 1 : return 1
    else :        return grenouille (n-1) + grenouille (n-2)
print grenouille (13)
```

Cette solution n'est pas très rapide car si `grenouille(13)` va être appelée une fois, `grenouille(12)` une fois aussi, mais `grenouille(11)` va être appelée deux fois... Cette solution implique un grand nombre de calculs inutiles.

Après la solution récursive descendante (`n` décroît), la solution récursive montante (`n` croît) :

```
def grenouille (fin, n = 2, u1 = 1, u2 = 2) :
    if   fin == 1   : return u1
    elif fin == 2   : return u2
    elif n   == fin : return u2
    u = u1 + u2
    return grenouille (fin, n+1, u2, u)
print grenouille (13)
```

La méthode la plus simple est non récursive mais il ne faut pas se tromper dans les indices :

```
def grenouille (n) :
    if n == 1 : return 1
    u1 = 1
    u2 = 2
    for i in range (3,n+1) :
        u  = u1 + u2   # il est impossible de
        u1 = u2        # résumer ces trois lignes
        u2 = u         # en deux
    return u2
print grenouille (13)
```

Quelques variantes utilisant les listes :

```
def grenouille (n) :
    if n == 1 : return 1
    u = [1,2]
    for i in range (2,n) :
        u.append (u [i-1] + u [i-2])
    return u [n-1]
print grenouille (12)
```

Ou encore :

```
def grenouille (n) :
    if n == 1 : return 1
    u = range (0, n)  # il ne faut pas oublier de créer le tableau
                      # avec autant de cases que nécessaire
                      # ici 13
    u [0] = 1
    u [1] = 2
    for i in range (2,n) :
        u [i] = u [i-1] + u [i-2]
    return u [n-1]
print grenouille (12)
```

**fin exo 13.1.4** □

### 13.1.5 Comprendre une erreur d'exécution *

#### Enoncé

On écrit le programme suivant :

```
a = "abcdefghijklmnopqrstuvwxyz"
print len (a)
d = {}
for i in range (0,len (a)) :
    d [ a [ i ] ] = i

print d ["M"]
```

Une erreur est déclenchée lors de l'exécution :

```
examen.py:14:  KeyError: 'M'
```

Que faudrait-il écrire pour que le programme marche ? Qu'affichera-t-il alors ? (1 point)

**Correction**

L'erreur signifie que la clé "M" n'est pas présente dans le dictionnaire d. Elle n'est présente qu'en minuscule. La modification proposée est la suivante :

```
a = "abcdefghijklmnopqrstuvwxyz"
print len (a)
d = {}
for i in range (0,len (a)) :
    d [ a [ i ] ] = i

print d ["m"]         ######   ligne modifiée
```

Mais il était également possible de procéder comme suit :

```
a = "abcdefghijklmnopqrstuvwxyz"
a = a.upper ()        #######   ligne ajoutée
print len (a)
d = {}
for i in range (0,len (a)) :
    d [ a [ i ] ] = i

print d ["M"]
```

Dans les deux cas, le programme affiche la position de M dans l'alphabet qui est 12 car les indices commencent à 0.

**fin exo 13.1.5** □

### 13.1.6 Copie de variables **

**Enoncé**

```
def somme (tab) :
    l = tab[0]
    for i in range (1, len (tab)) :
        l += tab [i]
    return l
ens = [[0,1],[2,3]]
print somme ( ens )   # affiche [0,1,2,3]
print ens             # affiche [ [0,1,2,3], [2,3] ]
```

La fonction somme est censée faire la concaténation de toutes les listes contenues dans ens. Le résultat retourné est effectivement celui désiré mais la fonction modifie également la liste ens, pourquoi ? (2 points)

**Correction**

Le problème vient du fait qu'une affectation en *Python* (seconde ligne de la fonction somme) ne fait pas une copie mais crée un second identificateur pour désigner la même chose. Ici, l et tab[0] désignent la même liste, modifier l'une modifie l'autre. Ceci explique le résultat. Pour corriger, il fallait faire une copie explicite de tab[0] :

```
import copy                     ###### ligne ajoutée
def somme (tab) :
    l = copy.copy (tab[0])      ###### ligne modifiée
    for i in range (1, len (tab)) :
        l += tab [i]
    return l
ens = [[0,1],[2,3]]
```

```
print somme ( ens )   # affiche [0,1,2,3]
print ens             # affiche [ [0,1,2,3], [2,3] ]
```

Il était possible, dans ce cas, de se passer de copie en écrivant :

```
def somme (tab) :
    l = []                            ###### ligne modifiée
    for i in range (0, len (tab)) :   ###### ligne modifiée
        l += tab [i]
    return l
ens = [[0,1],[2,3]]
print somme ( ens )   # affiche [0,1,2,3]
print ens             # affiche [ [0,1,2,3], [2,3] ]
```

**fin exo 13.1.6** □

### 13.1.7 Comprendre une erreur d'exécution **

#### Enoncé

On écrit le programme suivant :

```
li  = range (0,10)
sup = [0,9]
for i in sup :
    del li [i]
print li
```

Mais il produit une erreur :

```
examen.py:44:  IndexError: list assignment index out of range
```

Que se passe-t-il ? Comment corriger le programme pour qu'il marche correctement ? Qu'affichera-t-il alors ? (2 points)

#### Correction

L'erreur signifie qu'on cherche à accéder à un élément de la liste mais l'indice est soit négatif, soit trop grand. Pour comprendre pourquoi cette erreur se produit, il faut suivre les modifications de la liste `li` dans la boucle `for`. Au départ, `li` vaut `range(0,10)`, soit :

```
[ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Lors du premier passage de la boucle, on supprime l'élément d'indice 0, la liste `li` est donc égale à :

```
[ 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Et elle ne contient plus que 9 éléments au lieu de 10, donc le dernier élément est celui d'indice 8. Or lors du second passage dans la boucle `for`, on cherche à supprimer l'élément d'indice 9, ce qui est impossible. L'idée la plus simple pour corriger l'erreur est de supprimer les éléments en commençant par ordre d'indices décroissant :

```
li  = range (0,10)
sup = [9,0]      ####### ligne modifiée
for i in sup :
    del li [i]
print li
```

Mais on pouvait tout à fait écrire aussi `sup` = [0, 8] même si ce n'est la réponse que je conseillerais. L'objectif du programme n'était pas non plus de supprimer toute la liste `li`, dans ce cas, on aurait plutôt écrit `li` = [] ou `del li`[:].

Une dernière précision, pour supprimer l'élément d'indice `i` de la liste `li`, on peut écrire soit `del li`[i] soit `del li`[i : i + 1].

**fin exo 13.1.7** □

### 13.1.8 Comprendre une erreur de logique **

#### Enoncé

Le programme suivant fonctionne mais le résultat n'est pas celui escompté.

```
l = ["un", "deux", "trois", "quatre", "cinq"]
for i in range (0,len (l)) :
    mi = i
    for j in range (i, len (l)) :
        if l[mi] < l [j] : mi = j
    e      = l [i]
    l [mi] = l [i]
    l [i]  = e
print l
```

Le résultat affiché est :

```
['un', 'deux', 'deux', 'deux', 'cinq']
```

Qu'est censé faire ce programme ? Quelle est l'erreur ? (2 points)

#### Correction

Ce programme est censé effectuer un tri par ordre alphabétique **décroissant**. Le problème intervient lors de la permutation de l'élément `l`[i] avec l'élément `l`[mi]. Il faut donc écrire :

```
l = ["un", "deux", "trois", "quatre", "cinq"]
for i in range (0,len (l)) :
    mi = i
    for j in range (i, len (l)) :
        if l[mi] < l [j] : mi = j
    e      = l [mi]       ######## ligne modifiée
    l [mi] = l [i]
    l [i]  = e
print l
```

**fin exo 13.1.8** □

### 13.1.9 Coût d'un algorithme **

#### Enoncé

Le coût d'un algorithme ou d'un programme est le nombre d'opérations (additions, multiplications, tests, ...) qu'il effectue. Il s'exprime comme un multiple d'une fonction de la dimension des données que le programme manipule. Par exemple : $O(n)$, $O(n^2)$, $O(n \ln n)$, ...

Quel est le coût de la fonction `variance` en fonction de la longueur de la liste `tab` ? N'y a-t-il pas moyen de faire plus rapide ? (2 points)

```
def moyenne (tab) :
    s = 0.0
    for x in tab :
        s += x
    return s / len (tab)

def variance (tab) :
    s = 0.0
    for x in tab :
        t  = x - moyenne (tab)
        s += t * t
    return s / len (tab)

l = [ 0,1,2, 2,3,1,3,0]
print moyenne (l)
print variance (l)
```

### Correction

Tout d'abord, le coût d'un algorithme est très souvent exprimé comme un multiple de la dimension des données qu'il traite. Ici, la dimension est la taille du tableau `tab`. Par exemple, si on note `n = len(tab)`, alors le coût de la fonction `moyenne` s'écrit $O(n)$ car cette fonction fait la somme des $n$ éléments du tableau.

La fonction `variance` contient quant à elle un petit piège. Si elle contient elle aussi une boucle, chacun des $n$ passages dans cette boucle fait appel à la fonction `moyenne`. Le coût de la fonction `variance` est donc $O(n^2)$.

Il est possible d'accélérer le programme car la fonction `moyenne` retourne le même résultat à chaque passage dans la boucle. Il suffit de mémoriser son résultat dans une variable avant d'entrer dans la boucle comme suit :

```
def variance (tab) :
    s = 0.0
    m = moyenne (tab)
    for x in tab :
        t  = x - m
        s += t * t
    return s / len (tab)
```

Le coût de la fonction `variance` est alors $O(n)$.

**Remarque 13.1 : coût d'un algorithme**
Le coût d'un algorithme peut être évalué de manière plus précise et nécessiter un résultat comme $n^2 + 3n + 2$ mais cette exigence est rarement utile pour des langages comme *Python*. L'expression `for x in tab :` cache nécessairement un test qu'il faudrait prendre en compte si plus de précision était exigée. Il faudrait également se tourner vers un autre langage de programmation, plus précis dans sa syntaxe. Par exemple, lorsqu'on conçoit un programme avec le langage C ou C++, à partir du même code informatique, on peut construire deux programmes exécutables. Le premier (ou version *debug*), lent, sert à la mise au point : il inclut des tests supplémentaires permettant de vérifier à chaque étape qu'il n'y a pas eu d'erreur (une division par zéro par exemple). Lorsqu'on est sûr que le programme marche, on construit la seconde version (ou *release*), plus rapide, dont ont été ôtés tous ces tests de conception devenus inutiles.

*Python* aboutit à un programme lent qui inclut une quantité de tests invisibles pour celui qui programme mais qui détecte les erreurs plus vite et favorise une conception rapide. Il n'est pas adapté au traitement d'information en grand nombre et fait une multitude d'opérations cachées.

**fin exo 13.1.9** □

### 13.1.10 Héritage **

#### Enoncé

On a besoin dans un programme de créer une classe `carre` et une classe `rectangle`. Mais on ne sait pas quelle classe doit hériter de l'autre. Dans le premier programme, `rectangle` hérite de `carre`.

```
class carre :
    def __init__ (self, a) :
        self.a = a
    def surface (self) :
        return self.a ** 2

class rectangle (carre) :
    def __init__ (self, a,b) :
        carre.__init__(self,a)
        self.b = b
    def surface (self) :
        return self.a * self.b
```

Dans le second programme, c'est la classe `carre` qui hérite de la classe `rectangle`.

```
class rectangle :
    def __init__ (self, a,b) :
        self.a = a
        self.b = b
    def surface (self) :
        return self.a * self.b

class carre (rectangle) :
    def __init__ (self, a) :
        rectangle.__init__ (self, a,a)
    def surface (self) :
        return self.a ** 2
```

**1)** Dans le second programme, est-il nécessaire de redéfinir la méthode `surface` dans la classe `carre` ? Justifiez. (1 point)

**2)** Quel est le sens d'héritage qui vous paraît le plus censé, `class rectangle(carre)` ou `class carre(rectangle)` ? Justifiez. (1 point)

**3)** On désire ajouter la classe `losange`. Est-il plus simple que `rectangle` hérite de la classe `carre` ou l'inverse pour introduire la classe `losange` ? Quel ou quels attributs supplémentaires faut-il introduire dans la classe `losange` ? (1 point)

#### Correction

**1)** Le principe de l'héritage est qu'une classe `carre` héritant de la classe `rectangle` hérite de ses attributs et méthodes. L'aire d'un carré est égale à celle d'un rectangle dont les côtés sont égaux, par conséquent, la méthode `surface` de la classe retourne la même valeur que celle de la classe `rectangle`. Il n'est donc pas nécessaire de la redéfinir.

**2)** D'après la réponse de la première question, il paraît plus logique de considérer que `carre` hérite de `rectangle`.

**3)** Un losange est défini par un côté et un angle ou un côté et la longueur d'une de ses diagonales, soit dans les deux cas, deux paramètres. Dans la première question, il paraissait plus logique que la classe la plus spécifique hérite de la classe la plus générale afin de bénéficier de ses méthodes. Pour introduire le losange, il paraît plus logique de partir du plus spécifique pour aller au plus général afin que chaque classe ne contienne que les informations qui lui sont nécessaires.

```
class carre :
    def __init__ (self, a) :
```

```
        self.a = a
    def surface (self) :
        return self.a ** 2

class rectangle (carre) :
    def __init__ (self, a,b) :
        carre.__init__(self,a)
        self.b = b
    def surface (self) :
        return self.a * self.b

class losange (carre) :
    def __init__ (self, a,theta) :
        carre.__init__(self,a)
        self.theta = theta
    def surface (self) :
        return self.a * math.cos (self.theta) * self.a * math.sin (self.theta) * 2
```

Le sens de l'héritage dépend de vos besoins. Si l'héritage porte principalement sur les méthodes, il est préférable de partir du plus général pour aller au plus spécifique. La première classe sert d'interface pour toutes ses filles. Si l'héritage porte principalement sur les attributs, il est préférable de partir du plus spécifique au plus général. Dans le cas général, il n'y a pas d'héritage plus sensé qu'un autre mais pour un problème donné, il y a souvent un héritage plus sensé qu'un autre.

**fin exo 13.1.10** □

### 13.1.11 Précision des calculs ***

#### Enoncé

**1)** On exécute le programme suivant :

```
x = 1.0
for i in range (0,15) :
    x = x / 10
    print i, "\t", 1.0 - x, "\t", x, "\t", x **(0.5)
```

Il affiche à l'écran le résultat suivant :

```
0    0.90000000000000002220     0.1              0.316227766017
1    0.98999999999999999112     0.01             0.1
2    0.99899999999999999911     0.001            0.0316227766017
3    0.99990000000000001101     0.0001           0.01
4    0.99999000000000004551     1e-05            0.00316227766017
5    0.99999899999999997124     1e-06            0.001
6    0.99999990000000005264     1e-07            0.000316227766017
7    0.99999998999999994975     1e-08            0.0001
8    0.99999999900000002828     1e-09            3.16227766017e-05
9    0.99999999989999999173     1e-10            1e-05
10   0.99999999998999999917     1e-11            3.16227766017e-06
11   0.99999999999900002212     1e-12            1e-06
12   0.99999999999989996891     1e-13            3.16227766017e-07
13   0.99999999999999000799     1e-14            1e-07
14   0.99999999999999900080     1e-15            3.16227766017e-08
15   0.99999999999999988898     1e-16            1e-08
16   1.00000000000000000000     1e-17            3.16227766017e-09
17   1.00000000000000000000     1e-18            1e-09
18   1.00000000000000000000     1e-19            3.16227766017e-10
19   1.00000000000000000000     1e-20            1e-10
```

Que peut-on en déduire ? (1 point)

**2)** On écrit une classe `matrice_carree_2` qui représente une matrice carrée de dimension 2.

```
class matrice_carree_2 :
    def __init__ (self, a,b,c,d) :
        self.a, self.b, self.c, self.d = a,b,c,d

    def determinant (self) :
        return self.a * self.d - self.b * self.c

m1 = matrice_carree_2 (1.0,1e-6,1e-6,1.0)
m2 = matrice_carree_2 (1.0,1e-9,1e-9,1.0)
print m1.determinant ()
print m2.determinant ()
```

Qu'affichent les deux dernières lignes ? (1 point)

**3)** On considère la matrice $M = \begin{pmatrix} 1 & 10^{-9} \\ 10^{-9} & 1 \end{pmatrix}$.

On pose $D = \det(M) = 1 - 10^{-18}$ et $T = tr(M) = 2$. $\Delta$ est le déterminant de $M$ et $T$ sa trace. On sait que les valeurs propres de $M$ notées $\lambda_1, \lambda_2$ vérifient :

$$\begin{array}{rcl} D &=& \lambda_1 \lambda_2 \\ T &=& \lambda_1 + \lambda_2 \end{array}$$

On vérifie que $(x - \lambda_1)(x - \lambda_2) = x^2 - x(\lambda_1 + \lambda_2) + \lambda_1\lambda_2$. Les valeurs propres de $M$ sont donc solutions de l'équation : $x^2 - Tx + D = 0$.

Le discriminant de ce polynôme est $\Delta = T^2 - 4D$. On peut donc exprimer les valeurs propres de la matrice $M$ par :

$$\begin{array}{rcl} \lambda_1 &=& \frac{T - \sqrt{\Delta}}{2} \\ \lambda_2 &=& \frac{T + \sqrt{\Delta}}{2} \end{array}$$

On ajoute donc la méthode suivante à la classe `matrice_carree_2` :

```
        def valeurs_propres (self) :
            det   = self.determinant ()
            trace = self.a + self.d
            delta = trace ** 2 - 4 * det
            l1    = 0.5 * (trace - (delta ** (0.5)) )
            l2    = 0.5 * (trace + (delta ** (0.5)) )
            return l1,l2
```

D'après la précédente question, que retourne cette méthode pour la matrice $M$ ? (à justifier) (1 point)

**4)** On décompose la matrice $M = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 10^{-9} \\ 10^{-9} & 0 \end{pmatrix} = I + M'$.

On peut démontrer que si $\lambda$ est une valeur propre de $M'$, alors $1+\lambda$ est une valeur propre de $M$. Que donne le calcul des valeurs propres de $M'$ si on utilise la méthode `valeurs_propres` pour ces deux matrices ? (1 point)

**5)** On considère maintenant la matrice $M'' = \begin{pmatrix} 1 & 10^{-9} \\ -10^{-9} & 1 \end{pmatrix}$. En décomposant la matrice $M''$ de la même manière qu'à la question 4, quelles sont les valeurs propres retournées par le programme pour la matrice $M''$ ? Quelles sont ses vraies valeurs propres ? (1 point)

**Correction**

L'exercice a pour but de montrer que l'ordinateur ne fait que des calculs approchés et que la précision du résultat dépend de la méthode numérique employée.

**1)** Le programme montre que l'ordinateur affiche `1` lorsqu'il calcule $1 - 10^{-17}$. Cela signifie que la précision

des calculs en *Python* est au mieux de $10^{-16}$.

**2)** Il s'agit ici de donner le résultat que calcule l'ordinateur et non le résultat théorique qui sera toujours exact. On cherche à calculer ici le déterminant des matrices $M_1$ et $M_2$ définies par :

$$M_1 = \begin{pmatrix} 1 & 10^{-6} \\ 10^{-6} & 1 \end{pmatrix} \text{ et } M_2 = \begin{pmatrix} 1 & 10^{-9} \\ 10^{-9} & 1 \end{pmatrix}$$

Or, le programme proposé calcule les déterminants comme suit :

$$\det M_1 = 1 - 10^{-12} \text{ et } \det M_2 = 1 - 10^{-18}$$

D'après les affichages du programme de la question 1, le programme de la question 2 donnera comme réponse :

```
0.99999999999900002212
1.00000000000000000000
```

La seconde valeur est donc fausse.

**3)** Le déterminant est utilisé pour calculer les valeurs propres d'une matrice. Cette question s'intéresse à la répercussion de l'approximation faite pour le déterminant sur les valeurs propres. D'après l'énoncé, les deux valeurs propres sont calculés comme étant :

```
l1    = 0.5 * (trace - ((trace ** 2 - 4 * det) ** (0.5)) )
l2    = 0.5 * (trace + ((trace ** 2 - 4 * det) ** (0.5)) )
```

Pour la matrice $M_1$, on obtient donc en remplaçant le déterminant par 0.99999999999900002212, on obtient les réponses données par le petit programme de la question 1 :

```
l1 = 1,000001
l2 = 0.99999899999999997124  # égale à 1 - 1e-6
```

Pour la matrice $M_2$, le déterminant vaut 1. En remplaçant `trace` par 2 et `det` par 1, on obtient :

```
l1 = 1
l2 = 1
```

**4)** On change la méthode de calcul pour la matrice $M_2$, on écrit que $M_2 = I + \begin{pmatrix} 0 & 10^{-9} \\ 10^{-9} & 0 \end{pmatrix} = I + M_2'$.

Cette fois-ci le déterminant calculé par *Python* est bien $\texttt{1e} - \texttt{18}$. La trace de la matrice $M_2'$ est nulle, on applique les formules suivantes à la matrice $M_2'$ pour trouver :

```
l1    = 0.5 * (trace - ((trace ** 2 - 4 * det) ** (0.5)) ) = - det ** 0.5 = -1e-9
l2    = 0.5 * (trace + ((trace ** 2 - 4 * det) ** (0.5)) ) = det ** 0.5 = 1e-9
```

D'après l'énoncé, les valeurs propres de la matrice $M_2$ sont les sommes de celles de la matrice $I$ et de la matrice $M_2'$. Par conséquent, ce second calcul mène au résultat suivant :

```
l1    = 1-1e-9  = 0.99999999900000002828
l2    = 1+ 1e-9 = 1.000000001
```

**5)** La matrice $M''$ n'est en fait pas diagonalisable, c'est-à-dire que $tr\,(M'')^2 - 4 * \det M'' = 4 - 4(1+10^{-18}) < 0$. Or le calcul proposé par la question 3 aboutit au même résultat faux que pour la matrice $M_2$, les deux valeurs propres trouvées seront égales à 1. Si on applique la décomposition de la question 4 :

$$M'' = I + \begin{pmatrix} 0 & -10^{-9} \\ 10^{-9} & 0 \end{pmatrix} = I + N''$$

Le programme calcule sans erreur le discriminant négatif de la matrice $N''$ qui n'est pas diagonalisable. Il est donc impossible d'obtenir des valeurs propres réelles pour la matrice $M''$ avec cette seconde méthode.

Cette question montre qu'une erreur d'approximation peut rendre une matrice diagonalisable alors qu'elle ne l'est pas. Il est possible d'accroître la précision des calculs mais il faut faire appel à des modules externes [1].

**fin exo 13.1.11** □

## 13.2 Second énoncé (2006-2007)

### 13.2.1 Logarithme en base deux **

#### Enoncé

On veut écrire une fonction qui retourne $n$ tel que $n$ soit le premier entier qui vérifie $2^n \geqslant k$. Cette fonction prend comme paramètre d'entrée $k$ et retourne également un entier. Elle ne devra utiliser ni logarithme ni exponentielle. On précise qu'en langage *Python*, `2**n` signifie 2 à la puissance $n$. Une boucle `for` ne semble pas indiquée dans ce cas. (3 points)

#### Correction

L'indication qui préconisait d'utiliser autre chose qu'une boucle `for` ne voulait pas dire ne pas utiliser de boucle du tout mais une boucle `while`. La majorité des élèves ont réussi à trouver la fonction suivante :

```
def fonction_log2 (k) :
    n = 0
    while 2**n < k :
        n += 1
    return n
```

Même s'il est possible d'utiliser malgré tout une boucle `for` :

```
def fonction_log2 (k) :
    for i in range (0,1000) :
        if 2**i >= k :
            return i
```

Voici un exemple de fonction récursive :

```
def fonction_log2 (k) :
    if k <= 1 : return 0
    else : return fonction_log2 ((k+1)/2)+1
```

**fin exo 13.2.1** □

### 13.2.2 Calculer le résultat d'un programme **

#### Enoncé

On définit la fonction suivante :

```
def parcours (n) :
    i = 1
    j = 1
```

1. comme le module `GMPy`, http://code.google.com/p/gmpy/

```
    while i+j < n :
        print (i,j)
        i += 1
        j -= 1
        if j < 1 :
            j = i+j
            i = 1
```

Quelles sont les 6 lignes qu'affiche cette fonction si `n = 5` ? (2 points)

**Correction**

La solution est :

```
(1,1)
(1,2)
(2,1)
(1,3)
(2,2)
(3,1)
```

Et si on continue :

```
(1,4)
(2,3)
(3,2)
(4,1)
(1,5)
(2,4)
(3,3)
...
```

Il fallait voir deux choses importantes dans l'énoncé de l'exercice, tout d'abord, les deux lignes suivantes :

```
        i += 1
        j -= 1
```

Elles signifient que lorsque $i$ augmente de 1, $j$ diminue de 1. L'autre information est que $j$ ne devient jamais inférieur à 1 :

```
        if j < 1 :
            j = i+j
            i = 1
```

Et lorsque ce cas arrive, $i$ devient égal à 1, $j$ devient égale à $i+j$, soit $i+1$ puisque la condition est vérifiée lorsque $j == 1$.

**fin exo 13.2.2** □

### 13.2.3 Calculer le résultat d'un programme **

**Enoncé**

Que vaut `n` à la fin du programme suivant ? Il n'est en principe pas nécessaire d'aller jusqu'à `i = 10`. (2 points)

```
def suite (n) :
    n = n ** 2 / 2 + 1
    return n

n = 3
for i in range (0,10) :
    n = suite (n) % 17
print n
```

**Correction**

Il est difficile d'arriver au bout des calculs correctement lors de cet exercice. La première chose à faire est d'identifier la relation entre $n$ et $n+1$. On rappelle que le symbole % désigne le reste d'une division entière et que / désigne une division entière car tous les nombres manipulés dans ce programme sont entiers. La fonction `suite` aboutit à $f(n) = E\left[\frac{n^2}{2}\right] + 1$ où $E[x]$ désigne la partie entière de $x$. Ajouté aux trois dernières lignes, on obtient :

$$u_{n+1} = f(u_n)\%17 = \left(E\left[\frac{n^2}{2}\right] + 1\right)\%17$$

Même si $i$ n'est pas utilisé dans la boucle `for`, celle-ci s'exécute quand même 10 fois. Pour trouver la valeur finale de $n$, on calcule donc les premiers termes comme suit :

| $i$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $u_i = n$ | 3 | 5 | 13 | 0 | 1 |
| $n^2$ | 9 | 25 | 169 | 0 | 1 |
| $y = E\left[\frac{n^2}{2}\right] + 1$ | 4 | 12 | 85 | **1** | 1 |
| $u_{i+1} = y\%17$ | 5 | 13 | 0 | 1 | 1 |

A partir de $i = 3$, la suite reste constante et égale à 1. Il n'est pas nécessaire d'aller plus loin.

**fin exo 13.2.3** □

### 13.2.4 Comprendre une erreur d'exécution *

**Enoncé**

Le programme suivant est incorrect.

```
def compte_lettre (s) :
    nombre = {}
    for c in s :
        nombre [c] += 1
    return nombre

print compte_lettre ( "mysteres" )
```

Il retourne l'erreur suivante :

```
KeyError: 'm'
```

**1)** Quelle est la cause de l'erreur ? (1 point)

**2)** Quelle est la correction à apporter pour que cette fonction compte les lettres d'un mot ? (1 point)

**3)** Qu'afficherait le programme une fois corrigé ? (1 point)

**4)** La fonction `compte_lettre` est-elle réservée aux chaînes de caractères ? Voyez-vous d'autres types de données auxquels elle pourrait s'appliquer ? (1 point)

**Correction**

**1)** L'erreur provient de la ligne `nombre[c]+ = 1` pour `c =' m'`, cette ligne équivaut à `nombre[c] = nombre[c] + 1`. Cela signifie que `nombre[c]` doit exister avant l'exécution de la ligne, c'est-à-dire que le dictionnaire `nombre` doit avoir une valeur associée à la clé `c =' m'`, ce qui n'est pas le cas ici.

Dans ce programme, `nombre` est un dictionnaire et non une liste (ou un tableau) et les dictionnaires acceptent des indices autres que des entiers, ils acceptent comme indice ou clé toute variable de type immuable : les nombres (entiers, réels), les caractères, les t-uples.

**2) 3)** La solution compte le nombre d'occurrences de chaque lettre dans le mot `s`.

```
def compteur (s) :
    nombre = {}
    for c in s : nombre [c] = 0        # ligne ajoutée
    for c in s :
        nombre [c] += 1
    return nombre
```

ou encore :

```
def compteur (s) :
    nombre = {}
    for c in s :
        if c not in nombre : nombre [c] = 0       # ligne ajoutée
        nombre [c] += 1
    return nombre
```

Ces deux programmes retournent :

```
{'e': 2, 'm': 1, 's': 2, 'r': 1, 't': 1, 'y': 1}
```

La dernière solution pour ceux qui n'aiment pas les dictionnaires :

```
def compteur (s) :
    alpha = "abcdefghijklmnopqrstuvwxyz"
    nombre = [ 0 for c in alpha ]
    for c in s :
        i = alpha.index (c)
        nombre [i] += 1
    return nombre
```

Le programme retourne :

```
[0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0, 0, 1, 0]
```

**4)** La fonction `compteur` accepte d'autres types de données à condition que les deux lignes `for c in s :` et `nombre[c]+=1` aient un sens et elles ne sont valables que si le paramètre `s` contient des éléments susceptibles de servir de clé pour le dictionnaire `s`, c'est-à-dire un autre dictionnaire, un tuple, une liste d'éléments de type immuables. Autrement dit, les lignes suivantes sont correctes :

```
print compteur ( "mysteres" )
print compteur ( compteur ("mysteres") )
print compteur ( [0,1,1,4,-1, (6,0), 5.5, "ch"] )
print compteur ( { 1:1, 2:2, 1:[] } )
```

Mais pas celle-ci :

```
print compteur ( [0, [0,0] ] )
```

Toutefois, cette dernière ligne est valide si la fonction `compteur` se contente seulement de compter le nombre d'éléments, c'est-à-dire la première solution citée aux questions 2) et 3).

**fin exo 13.2.4** □

### 13.2.5 Comprendre une erreur de logique ***

#### Enoncé

On précise que l'instruction `random.randint(0,1)` retourne un nombre aléatoire choisi dans l'ensemble $\{0,1\}$ avec des probabilités équivalentes ($\mathbb{P}(X=0)=0,5$ et $\mathbb{P}(X=1)=0,5$). La fonction `ligne_nulle` doit compter le nombre de lignes nulles de la matrice `mat` donnée comme paramètre.

```
def ligne_nulle (mat) :
    nb = 0
    for i in range (0, len (mat)) :
        lig = 0
        for j in range (0, len (mat [i])) :
            if mat [i][j] > 0 : lig += 1
            if lig == 0 : nb += 1
    return nb

matri = [ [ random.randint (0,1) for i in range (0,4) ] for j in range (0,20) ]
print ligne_nulle (matri)
```

Après avoir exécuté le programme trois fois de suite, les résultats affichés sont successivement 15, 19, 17. Bien que l'exécution du programme ne provoque aucune erreur, le concepteur de la fonction s'interroge quand même sur ces résultats.

**1)** Sachant que chaque case de la matrice reçoit aléatoirement 0 ou 1, quelle est la probabilité qu'une ligne soit nulle ? Quelle est la probabilité d'avoir 15 lignes nulles dans la matrice sachant que cette matrice a 20 lignes ? Ces deux réponses peuvent être littérales. Donnez-vous raison à celui qui a écrit le programme (il pense s'être trompé) ? (1 point)

**2)** Si vous lui donnez raison, ce qui est fort probable, où serait son erreur ? (1 point)

**Correction**

**1)** La probabilité d'avoir une ligne nulle est la probabilité d'avoir 4 zéros, c'est donc :

$$\mathbb{P}\left(\textit{ligne nulle}\right) = \left(\frac{1}{2}\right)^4 = \frac{1}{16}$$

Comment calculer la probabilité d'avoir 15 lignes nulles parmi 20 ? Cela revient à estimer la probabilité de tirer 15 fois sur 20 une boule blanche lors d'un tirage à remise sachant que dans l'urne, il y a 1 boule blanche et 15 boules noires. Le nombre de boules blanches tirées suit une loi binomiale de paramètre $p = \frac{1}{16}$. On en déduit que :

$$\mathbb{P}\left(15 \textit{ lignes nulles sur } 20\right) = C_{20}^{15} \left(\frac{1}{16}\right)^{15} \left(1 - \frac{1}{16}\right)^5$$

Cette probabilité est très faible, il est donc presque impossible d'obtenir trois fois de suite un nombre de lignes supérieur à 15. Le programme est sans aucun doute faux.

**2)** La construction de la matrice est manifestement correcte, c'est donc le comptage des lignes nulles qui est faux. Cette erreur intervient lors de la ligne `if lig == 0 : nb+ = 1`. Celle-ci est incluse dans la seconde boucle `for` ce qui a pour effet d'incrémenter `lig` dès qu'un premier zéro est rencontré sur une ligne. Au final, la fonction retourne le nombre de lignes contenant au moins un zéro. Voici donc la correction à apporter :

```
def ligne_nulle (mat) :
    nb = 0
    for i in range (0, len (mat)) :
        lig = 0
        for j in range (0, len (mat [i])) :
            if mat [i][j] > 0 : lig += 1
        if lig == 0 : nb += 1                    # ligne décalée vers la gauche
    return nb
```

**fin exo 13.2.5** □

### 13.2.6 Récursivité *

#### Enoncé

Le programme suivant provoque une erreur dont le message paraît sans fin.

```
class erreur :
    def __init__ (self) :
        self.e = erreur ()
e = erreur ()
```

Auriez-vous une explication pour ce qui suit ? (1 point)

```
Traceback (most recent call last):
  File "examen2007.py", line 4, in ?
    e = erreur ()
  File "examen2007.py", line 2, in __init__
    self.e = erreur ()
  File "examen2007.py", line 2, in __init__
    self.e = erreur ()
  File "examen2007.py", line 2, in __init__
    self.e = erreur ()
  ...
```

#### Correction

L'erreur retournée est une erreur d'exécution et non une erreur de syntaxe. Cela veut dire que le programme de l'exercice est syntaxiquement correct. Il n'est donc pas possible de dire que l'attribut `e` n'est pas défini et de corriger le programme comme suit pour expliquer l'erreur :

```
class erreur :
    def __init__ (self,e) :
        self.e = e
```

En fait, la classe `erreur` définit un attribut qui est également de type `erreur`. Cet attribut va lui aussi définir un attribut de type `erreur`. Ce schéma va se reproduire à l'infini puisqu'à aucun moment, le code du programme ne prévoit la possibilité de s'arrêter. Le programme crée donc des instances de la classe `erreur` à l'infini jusqu'à atteindre une certaine limite dépendant du langage *Python*. C'est à ce moment-là que se produit l'erreur citée dans l'énoncé.

**fin exo 13.2.6** □

### 13.2.7 Compléter un programme **

#### Enoncé

11 (base décimale) s'écrit 102 en base 3 puisque $11 = 1 * 3^2 + 0 * 3^1 + 2 * 3^0$. L'objectif de cet exercice est d'écrire une fonction qui écrit un nombre entier en base 3. Cette fonction prend comme paramètre d'entrée un entier et retourne une chaîne de caractères. On précise que l'opérateur % calcule le reste d'une division entière et que `11/3` → `3` car c'est une division entière dont le résultat est un entier égal au quotient de la division. La fonction `str` permet de convertir un nombre en une chaîne de caractères. Il ne reste plus qu'à compléter les deux lignes manquantes du programme suivant : (2 points)

```
def base3 (n) :
    s = ""
    while n > 0 :
        r = n % 3
        # ....... à compléter
        # ....... à compléter
    return s
```

### Correction

Il y a effectivement deux lignes à corriger. `r` désigne le reste de la division de `n` par 3. Il faut le convertir en chaîne de caractères et l'ajouter à gauche à la chaîne `s`. On passe au chiffre suivant en division `n` par 3 ce qui est rassurant puisque `n` va tendre vers zéro et le programme s'arrêter nécessairement au bout d'un moment. Pour vérifier que ce programme est correct, il suffit de l'appliquer à un nombre, voire appliquer le même algorithme mais en base 10 pour être vraiment sûr.

```
def base3 (n) :
    s = ""
    while n > 0 :
        r = n % 3
        s = str (r) + s
        n = n / 3            # équivalent à n = (n-r) / 3
                             # puisque / est une division entière
    return s
```

**fin exo 13.2.7** □

## 13.2.8 Comprendre une erreur de logique ***

### Enoncé

Un professeur désireux de tester une répartition aléatoire des notes à un examen décide de tirer au hasard les notes de ses élèves selon une loi normale de moyenne 15 et d'écart-type 3. Il arrondit ses notes à l'entier le plus proche en n'omettant pas de vérifier que ses notes sont bien dans l'intervalle $[0, 20]$. On précise que l'instruction `float(i)` convertit un nombre `i` en nombre réel, cette conversion est utilisée pour être sûr que le résultat final sera bien réel et non le résultat d'opérations sur des entiers. Cette conversion intervient le plus souvent lors de divisions.

```
import copy
import math
import random

class Eleve :
    def __init__ (self, note) :
        self.note = note

e = Eleve (0)
l = []
for i in range (0,81) :
    e.note = int (random.gauss (15, 3) + 0.5)  # tirage aélatoire et arrondi
    if e.note >= 20 : e.note = 20              # pas de note au-dessus de 20
    if e.note < 0 : e.note = 0                 # pas de note négative
    l.append (e)

moy = 0
var = 0

for e in l :
    moy += e.note
moy = float (moy) / len (l)  # les notes sont entières,
                             # il faut convertir avant de diviser
                             # pour obtenir la moyenne
for e in l :
    var += (e.note - moy) ** 2
var = math.sqrt ( float (var) ) / len (l)

print "moyenne ", moy
print "écart-type ", var
```

Il songe à vérifier néanmoins que la moyenne de ses notes arrondies est bien conforme à ce qu'il a échafaudé.

```
moyenne  16.0
écart-type  0.0
```

La moyenne égale à 16 ne le perturbe guère, il se dit que l'arrondi a été plutôt généreux. Toutefois l'écart-type nul le laisse perplexe.

**1)** Que signifie un écart-type nul ? Quelle est l'erreur du professeur ? (Elle est située à l'intérieur de la boucle `for i in range(0,81)` :, ce n'est pas une erreur lors du calcul de l'écart-type ni une erreur de définition de la classe `Eleve`.) (1 point)

**2)** Proposer deux solutions pour corriger ce problème, chacune d'elles revient à remplacer la ligne `l.append(e)`. (2 points)

**3)** On sait que la variance d'une variable aléatoire $X$ vérifie : $\mathbb{V}(X) = \mathbb{E}\left(X^2\right) - [\mathbb{E}(X)]^2$. Cette astuce mathématique permet-elle de réduire le nombre de boucles du programme, si oui, comment ? (1 point)

### Correction

**1)** Un écart-type nul signifie que toutes les notes sont identiques et égales à la moyenne. Selon le programme, tous les élèves ont donc 16. L'erreur provient du fait que l'instruction `l.append(e)` ajoute à chaque fois la même variable de type `Eleve`. A la fin de la boucle, la liste `l` contient 81 fois le même objet `Eleve` ou plus exactement 81 fois la même instance de la classe `Eleve`. On modifie la note de cet unique objet en écrivant : `e.note = int(random.gauss(15,3) + 0.5)`. 16 est donc la note attribuée au dernier élève, la dernière note tirée aléatoirement.

**2)** Les deux corrections possibles consistent à créer à chaque itération une nouvelle instance de la classe `Eleve`.

1. `l.append(e)` devient `l.append(Eleve(e.note))`
2. `l.append(e)` devient `l.append(copy.copy(e))`

La variable `e` dans la boucle est un `Eleve` temporaire, il est ensuite recréé ou recopié avant l'ajout dans la liste. Ceci veut dire que l'instance ajoutée dans la liste n'est pas la même que celle utilisée dans la boucle.

Une troisième solution est envisageable même si elle introduit des modifications dans la suite du programme, elle est logiquement correcte : `l.append(e)` devient `l.append(e.note)`. La liste `l` n'est plus une liste de `Eleve` mais une liste d'entiers. Le programme devient :

```
#...
e = Eleve (0)
l = []
for i in range (0,81) :
    e.note = int (random.gauss (15, 3) + 0.5)
    if e.note >= 20 : e.note = 20
    if e.note < 0 : e.note = 0
    l.append (e.note)                           # ligne modifiée

moy = 0
var = 0

for note in l :                                 # ligne modifiée
    moy += note                                 # ligne modifiée
moy = float (moy) / len (l)
for note in l :                                 # ligne modifiée
    var += (note - moy) ** 2                    # ligne modifiée
var = math.sqrt ( float (var) ) / len (l)

print "moyenne ", moy
print "écart-type ", var
```

**3)** La formule mathématique permet de réduire le nombre de boucles à deux. Lors de la version initiale du programme, la première sert à créer la liste `l`. La seconde à calculer la moyenne, la troisième à calculer la variance. On regroupe les deux dernières boucles en une seule.

```
moy = 0
var = 0

for note in l :
    moy += note
    var += note * note
moy = float (moy) / len (l)
var = float (var) / len (l)
var = var - moy * moy
var = math.sqrt ( float (var) )

print "moyenne ", moy
print "écart-type ", var
```

**fin exo 13.2.8** □

## 13.3 Troisème énoncé (2007-2008)

### 13.3.1 Calcul d'un arrondi *

**Enoncé**

Ecrire une fonction qui arrondit un nombre réel à 0.5 près ? Ecrire une autre fonction qui arrondit à 0.125 près ? Ecrire une fonction qui arrondit à $r$ près où $r$ est un réel ? Répondre uniquement à cette dernière question suffit pour répondre aux trois questions. (1 point)

**Correction**

Pour arrondi à 1 près, il suffit de prendre la partie entière de $x+0.5$, arrondir $x$ à 0.5 près revient à arrondi $2x$ à 1 près.

```
    def arrondi_05 (x) :
        return float (int (x * 2 + 0.5)) / 2

    def arrondi_0125 (x) :
        return float (int (x * 8 + 0.5)) / 8

    def arrondi (x, p) :
        return float (int (x / p + 0.5)) * p
```

**fin exo 13.3.1** □

### 13.3.2 Même programme avec 1,2,3 boucles **

**Enoncé**

Le programme suivant affiche toutes les listes de trois entiers, chaque entier étant compris entre 0 et 9.

```
for a in range (0, 10) :
    for b in range (0, 10) :
        for c in range (0, 10) :
             print [a,b,c]
```

**1)** Proposez une solution avec une boucle `while` et deux tests `if`. (1 point)

**2)** Proposez une solution avec deux boucles `while` et un test `if`. (1 point)

### Correction

**1)** Une seule boucle contrôle les indices $a$, $b$, $c$. Quand un indice atteint sa limite, on incrémente le suivant et on remet l'indice à 0.

```
a,b,c = 0,0,0
while c < 10 :
    print [a,b,c]
    a += 1
    if a == 10 :
        b += 1
        a = 0
        if b == 10 :
            c += 1
            b = 1
```

**2)** L'avantage de cette dernière solution est qu'elle ne dépend pas du nombre d'indices. C'est cette solution qu'il faut préconiser pour écrire une fonction dont le code est adapté quelque soit la valeur de $n$.

```
l = [0,0,0]
while l [2] < 10 :
    print l
    l [0] += 1
    i = 0
    while i < len (l)-1 and l [i] == 10 :
        l [i] = 0
        l [i+1] += 1
        i += 1
```

**fin exo 13.3.2** □

## 13.3.3 Suite récurrente (Fibonacci) **

### Enoncé

La fonction `fibo` retourne la valeur de la suite de Fibonacci pour tout entier `n`. Quel est son coût en fonction de $O(n)$ ? (1 point)

```
def fibo (n) :
    if n <= 2 : return 2
    else : return fibo (n-1) + fibo (n-2)
```

### Correction

Lorsqu'on cherche à calculer `fibo(n)`, on calcule `fibo(n-1)` et `fibo(n-2)` : le coût du calcul `fibo(n)` est égal à la somme des coûts des calculs de `fibo(n-1)` et `fibo(n-2)` plus une addition et un test. Le coût de la fonction `fibo(n)` est plus facile à définir par récurrence. Le coût $c_n$ du calcul de `fibo(n)` vérifie donc :

$$c_0 = c_1 = c_2 = 1 \tag{13.1}$$
$$c_n = c_{n-1} + c_{n-2} + 2 \tag{13.2}$$

Le terme 1 dans (13.1) correspond au premier test. Le terme 2 dans (13.2) correspond au test et à l'addition. Le coût du calcul de `fibo(n)` est égal à une constante près à une suite de Fibonacci. Le code suivant permet de le vérifier en introduisant une variable globale. On modifie également la fonction `fibo` de façon à pouvoir changer $u_0$ et $u_1$.

```
nb = 0  # variable globale
def fibo (n,p) :
    global nb
    if n <= 2 :
        nb += 1
        return p # plus de récurrence
    else :
        nb += 2
        return fibo (n-1,p) + fibo (n-2,p)

for n in range (1, 20) :
    nb = 0    # remis à zéro, à chaque fois
              # nb est la mesure du coût
    print fibo(n,3)-2, nb  # nombres identiques
    # nb vérifie la récurrence de la suite c(n)
    #          c(n) = c(n-1) + c(n-2) + 2
```

suites de la forme (13.3) :

$$
\begin{aligned}
u_0 &= p \\
u_1 &= p \\
u_2 &= u_0 + u_1 + d = 2p + d \\
u_3 &= u_1 + u_2 + d = 3p + 2d \\
u_4 &= u_2 + u_3 + d = 5p + 4d \\
u_5 &= u_3 + u_4 + d = 8p + 7d \\
u_6 &= u_4 + u_5 + d = 13p + 12d \\
u_7 &= u_5 + u_6 + d = 21p + 20d \\
u_8 &= u_6 + u_7 + d = 34p + 33d \\
&\dots
\end{aligned}
$$

Dans cet exemple, on s'aperçoit que la suite est égale à son coût. Pour le démontrer d'une façon plus théorique, on s'intéresse aux suites de la forme suivante dont la récurrence est développée ci-dessus à droite :

$$
\begin{aligned}
u_0 &= u_1 = u_2 = p \\
u_n &= u_{n-1} + u_{n-2} + d
\end{aligned}
\tag{13.3}
$$

Par une astuce de calcul, on peut réduire l'écriture de la suite $(u_n)$ à une somme linéaire de deux suites $(U_n)$ et $(V_n)$. La suite $(U_n)$ est définie par $U_0 = 1$, $U_1 = 1$, $U_n = U_{n-1} + U_{n-2}$ et la suite $(V_n)$ définie par $V_0 = 0$, $V_1 = 0$, $V_n = V_{n-1} + V_{n-2} + 1$. On en déduit que :

$$u_n = U_n p + V_n d$$

Si on arrive à montrer que $U_n = V_n + 1$, cela montrera que la fonction `fibo` citée dans la correction est bien égale à son coût. Or :

$$V_n + 1 = V_{n-1} + V_{n-2} + 1 + 1 = (V_{n-1} + 1) + (V_{n-2} + 1)$$

La suite $V_n^* = V_n + 1$ suit bien la récurrence de la suite $U_n$. On vérifie que les premières valeurs sont identiques et cela montre que $U_n = V_n + 1$. On en déduit que :

$$u_n = U_n p + (U_n - 1)d = U_n(p + d) - d$$

Pour conclure, une suite de Fibonacci vérifie la récurrence $u_n = u_{n-1} + u_{n-2}$. Si on pose $\lambda_1 = \frac{1+\sqrt{5}}{2}$ et $\lambda_2 = \frac{1-\sqrt{5}}{2}$. Cette suite peut s'écrire sous la forme $u_n = A\lambda_1^n + B\lambda_2^n$. Le second terme étant négligeable par rapport au premier, le coût de la fonction `fibo` est donc en $O(\lambda_1^n)$. En d'autres termes, si on calcule le coût du calcul récursif d'une suite de Fibonacci, ce dernier est équivalent à la suite elle-même.

**fin exo 13.3.3** □

### 13.3.4 Calculer le résultat d'un programme *

**Enoncé**

**1)** Qu'affiche le code suivant : (1 point)

```
l = [0,1,2,3,4,5]
g = l
```

```
for i in range (0, len (l)-1) :
    g [i] = g [i+1]
print l
print g
```

**2)** Et celui-ci : (1 point)

```
l = [0,1,2,3,4,5]
g = [0,1,2,3,4,5]
for i in range (0, len (l)-1) :
    g [i] = g [i+1]
print l
print g
```

**3)** Et encore celui-là : (1 point)

```
l = [0,1,2,3,4,5]
g = [0,1,2,3,4,5]
for i in range (0, len (l)) :
    g [i] = g [(i+1)%len (l)]
print l
print g
```

**4)** A votre avis, quel est l'objectif du programme et que suggérez-vous d'écrire ? (1 point)

**5)** Voyez-vous un moyen d'écrire plus simplement la seconde ligne `g = [0, 1, 2, 3, 4, 5]` tout en laissant inchangé le résultat ? (1 point)

**Correction**

**1)** L'instruction `g = l` implique que ces deux variables désignent la même liste. La boucle décale les nombres vers la gauche.

```
[1, 2, 3, 4, 5, 5]
[1, 2, 3, 4, 5, 5]
```

**2)** L'instruction `g = [0, 1, 2, 3, 4, 5]` implique que ces deux variables ne désignent plus la même liste. L'instruction `print l` affiche le contenu du début.

```
[0, 1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 5]
```

**3)** La boucle s'intéresse cette fois-ci au déplacement du premier élément en première position. Le programmeur a pris soin d'utiliser le modulo, $n\%n = 0$ mais le premier élément de la liste $g$ est devenu le second. Le résultat est donc :

```
[0, 1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 1]
```

**4)** Le programme souhaite décaler les éléments d'une liste vers la gauche, le premier élément devenant le dernier.

```
l = [0,1,2,3,4,5]
g = [0,1,2,3,4,5]
for i in range (0, len (l)) :
    g [i] = l [(i+1)%len (l)] # ligne modifiée, g devient l
print l
print g
```

**5)** La seconde ligne impose de répéter le contenu de la liste. Il existe une fonction qui permet de le faire :

```
import copy
l = [0,1,2,3,4,5]
g = copy.copy (l) # on pourrait aussi écrire g = list (l)
                  # ou encore g = [ i for i in l ]
for i in range (0, len (l)) :
    g [i] = l [(i+1)%len (l)]
print l
print g
```

**fin exo 13.3.4** □

### 13.3.5 Comprendre une erreur de logique **

#### Enoncé

**1)** Un individu a écrit la fonction suivante, il inclut un seul commentaire et il faut deviner ce qu'elle fait. Après exécution, le programme affiche 4. (1 point)

```
def mystere (l) :
    """cette fonction s'applique à des listes de nombres"""
    l.sort ()
    nb = 0
    for i in range (1,len (l)) :
        if l [i-1] != l [i] : nb += 1
    return nb+1

l = [4,3,1,2,3,4]
print mystere (l)  # affiche 4
```

**2)** Après avoir écrit l'instruction `print l`, on s'aperçoit que la liste `l` a été modifiée par la fonction `mystere`. Qu'affiche cette instruction ? (1 point)

**3)** Comment corriger le programme pour que la liste ne soit pas modifiée ? (1 point)

#### Correction

**1)** La fonction travaille sur une liste de nombres triés. La fonction parcourt cette liste et compte le nombre de fois que la liste triée contient deux éléments successifs différents. La fonction retourne donc le nombre d'éléments distincts d'une liste.

**2)** La fonction trie la liste, elle ressort donc triée de la fonction car le passage des listes à une fonction s'effectue par adresse. Le résultat de l'instruction `print l` est donc :

```
[1, 2, 3, 3, 4, 4]
```

**3)** Il suffit d'utiliser le module `copy`.

```
import copy
def mystere (l) :
    """cette fonction s'applique à des listes de nombres"""
    l = copy.copy (l) # ligne insérée
                      # on peut écrire aussi l = list (l)
    l.sort ()
    nb = 0
    for i in range (1,len (l)) :
        if l [i-1] != l [i] : nb += 1
    return nb+1
```

**fin exo 13.3.5** □

### 13.3.6 Héritage **

**Enoncé**

```
class Personne :
    def __init__ (self, nom) :
        self.nom = nom
    def entete (self) :
        return ""
    def __str__ (self) :
        s = self.entete () + self.nom
        return s

class Homme (Personne) :
    def __init__ (self, nom) :
        Personne.__init__ (self, nom)
    def entete (self) :
        return "M. "

class Femme (Personne) :
    def __init__ (self, nom) :
        Personne.__init__ (self, nom)
    def entete (self) :
        return "Melle "

h = Homme ("Hector")
f = Femme ("Gertrude")
print h
print f
```

*Programme A*

```
class Personne :
    def __init__ (self, nom, entete) :
        self.nom = nom
        self.entete = entete
    def __str__ (self) :
        s = self.entete + self.nom
        return s

h = Personne ("Hector", "M. ")
f = Personne ("Gertrude", "Melle ")
print h
print f
```

*Programme B*

**1)** Les deux programmes précédents affichent-ils les mêmes choses ? Qu'affichent-ils ? (1 point)

**2)** On souhaite ajouter une personne hermaphrodite, comment modifier chacun des deux programmes pour prendre en compte ce cas ? (1 point)

**3)** Quel programme conseilleriez-vous à quelqu'un qui doit manipuler cent millions de personnes et qui a peur de manquer de mémoire ? Justifiez. (1 point)

**Correction**

**1)** Les programmes affichent la même chose et ils affichent :

```
M. Hector
Melle Gertrude
```

**2)**

```
class Personne :
    def __init__ (self, nom) :
        self.nom = nom
    def entete (self) :
        return ""
    def __str__ (self) :
        s = self.entete () + self.nom
        return s

class Homme (Personne) :
    def __init__ (self, nom) :
        Personne.__init__ (self, nom)
    def entete (self) :
        return "M. "

class Femme (Personne) :
    def __init__ (self, nom) :
        Personne.__init__ (self, nom)
    def entete (self) :
        return "Melle "

class Hermaphrodite (Personne) :
    def __init__ (self, nom) :
        Personne.__init__ (self, nom)
    def entete (self) :
        return "Melle et M. "

h = Homme ("Hector")
f = Femme ("Gertrude")
g = Hermaphrodite ("Marie-Jean")
print h
print f
print g
```

*Programme A*

```
class Personne :
    def __init__ (self, nom, entete) :
        self.nom = nom
        self.entete = entete
    def __str__ (self) :
        s = self.entete + self.nom
        return s

h = Personne ("Hector", "M. ")
f = Personne ("Gertrude", "Melle ")
g = Personne ("Marie-Jean", \
                    "Melle et M. ")
print h
print f
print g
```

*Programme B*

**3)** Les deux programmes définissent une personne avec un prénom et un en-tête. Dans le second programme, l'en-tête est un attribut de la classe et cette classe permet de modéliser les hommes et les femmes. Dans le premier programme, l'en-tête est défini par le type de la classe utilisée. Il y a moins d'information mémorisée, en contre partie, il n'est pas possible d'avoir d'autres en-têtes que `M.` et `Melle`. Pour manipuler 100 millions de personnes, il vaut mieux utiliser le premier programme avec deux classes, il sera moins gourmant en mémoire.

**fin exo 13.3.6** □

### 13.3.7 Analyser un programme ***

#### Enoncé

On considère la fonction suivante qui prend comme entrée une liste d'entiers. Elle ne retourne pas de résultat car elle modifie la liste. Appeler cette fonction modifie la liste donnée comme paramètre. Elle est composée de quatre groupes d'instructions.

```
def tri_entiers(l):
    """cette fonction s'applique à une liste d'entiers"""

    # groupe 1
    m = l [0]
    M = l [0]
    for k in range(1,len(l)):
        if l [k] < m : m = l [k]
        if l [k] > M : M = l [k]
```

```
# groupe 2
p = [0 for i in range (m,M+1) ]
for i in range (0, len (l)) :
    p [ l [i] - m ] += 1

# groupe 3
R     = [0 for i in range (m,M+1) ]
R [0] = p [0]
for k in range (1, len (p)) :
    R [k] = R [k-1] + p [k]

# groupe 4
pos = 0
for i in range (1, len (l)) :
    while R [pos] < i : pos += 1
    l [i-1] = pos + m
l [len (l)-1] = M
```

**1)** Que fait le groupe 1 ? (0,5 point)

**2)** Que fait le groupe 2 ? (0,5 point)

**3)** Que fait le groupe 3 ? (0,5 point)

**4)** Que fait le groupe 4 ? (1 point)

**5)** Quel est la boucle contenant le plus grand nombre d'itérations ? (0,5 point)

### Correction

**1)** Le groupe 1 détermine les minimum et maximum de la liste `l`.

**2)** Dans le groupe 2, la liste `p` compte le nombre d'occurrences d'un élément dans la liste `l`. `p[0]` correspond au nombre de fois que le minimum est présent dans la liste, `p[len(l) - 1]` correspond au nombre de fois que le maximum est présent.

**3)** Le groupe 3 construit la fonction de répartition de la liste `l`. Si `m` est le minimum de la liste `l`, `R[i]` est le nombre d'éléments inférieurs ou égaux à `i + m` que la liste contient.

**4)** Le dernier groupe trie la liste `l`. Si $X$ est une variable aléatoire de fonction de répartition $F$ alors $F(X)$ suit une loi uniforme sur $[0, 1]$. C'est le même principe ici. La figure 13.1 illustre la fonction de répartition. A chaque marche correspond un élément de la liste et à chaque marche correspond une case du tableau `R`. En parcourant les marches du tableau `R`, on retrouve les éléments de la liste `l` triés. La marche plus haute traite le cas de plusieurs éléments égaux.

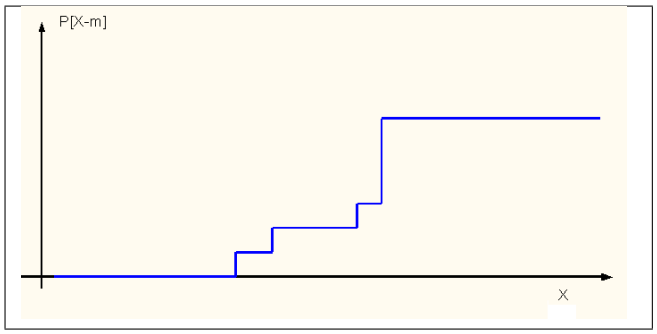



**Figure 13.1 :** *Construction de la fonction de répartition.*

**5)** Le groupe 1 inclut une boucle de $n$ itérations où $n$ est le nombre d'éléments de la liste `l`. Les groupes 2 et 3 incluent une boucle de $M - n$ itérations. Le groupe 4 inclut deux boucles, la première inclut $n$ itérations, la seconde implique une variable `pos` qui passe de $m$ à $M$. Le coût de ce bloc est en $O(n + M - n)$. C'est ce dernier bloc le plus long.

**fin exo 13.3.7** □

### 13.3.8 Comprendre une erreur de logique ***

**Enoncé**

Chaque année, une société reçoit des cadeaux de ses clients et son patron souhaite organiser une tombola pour attribuer ces cadeaux de sorte que les plus bas salaires aient plus de chance de recevoir un cadeau. Il a donc choisi la méthode suivante :

- Chaque employé considère son salaire net par mois et le divise par 1000, il obtient un nombre entier $n_i$.
- Dans une urne, on place des boules numérotées que l'on tire avec remise.
- Le premier employé dont la boule est sortie autant de fois que $n_i$ gagne le lot.

La société a distribué plus de 100 cadeaux en quelques années mais jamais le patron n'en a gagné un seul, son salaire n'est pourtant que cinq fois celui de sa sécrétaire. Son système est-il équitable ? Il fait donc une simulation sur 1000 cadeaux en *Python* que voici. Sa société emploie quatre personnes dont les salaires sont $[2000, 3000, 5000, 10000]$.

```
import random

def tirage (poids) :
    nb = [ 0 for p in poids ]
    while True :
        i = random.randint (0, len (poids)-1)
        nb [i] += 1
        if nb [i] == poids [i] :
            return i

salaire = [ 10000, 5000, 3000, 2000 ]
poids   = [ s / 1000 for s in salaire]

nombre = [ 0 for s in salaire ]

for n in range (0,1000) :
    p = tirage (poids)
    nombre [p] += 1

for i in range (0, len (poids)) :
    print "salaire ", salaire [i], " : nb : ", nombre [i]
```

Les résultats sont plus que probants :

```
salaire  10000  : nb :  0
salaire  5000  : nb :  49
salaire  3000  : nb :  301
salaire  2000  : nb :  650
```

Il n'a aucune chance de gagner à moins que son programme soit faux. Ne s'en sortant plus, il décide d'engager un expert car il trouve particulièrement injuste que sa secrétaire ait encore gagné cette dernière bouteille de vin.

**1)** Avez-vous suffisamment confiance en vos connaissances en *Python* et en probabilités pour déclarer le programme correct ou incorrect ? Deux ou trois mots d'explications seraient les bienvenus. (1 point)

**2)** Le patron, après une folle nuit de réflexion, décida de se débarrasser de son expert perdu dans des probabilités compliquées et modifia son programme pour obtenir le suivant :

```
import random

def tirage (poids, nb) :
    while True :
        i = random.randint (0, len (poids)-1)
        nb [i] += 1
        if nb [i] % poids [i]  == 0 :
```

```
            return i

salaire = [ 10000, 5000, 3000, 2000 ]
poids   = [int (s) / 1000 for s in salaire]

nombre = [ 0 for s in salaire ]
temp = [ 0 for s in salaire ]

for n in range (0,1000) :
    p = tirage (poids, temp)
    nombre [p] += 1

for i in range (0, len (poids)) :
    print "salaire ", salaire [i], " : nb : ", nombre [i]
```

Et le résultat est beaucoup plus conforme à ses attentes :

```
salaire  10000  : nb :  90
salaire  5000  : nb :  178
salaire  3000  : nb :  303
salaire  2000  : nb :  429
```

Quelle est la modification dans les règles qu'il a apportée ? Vous paraissent-elles justes ? (1 point)

**Correction**

**1)** Le programme est correct, ses résultats aussi. Prenons le cas simple où il n'y a que le patron et sa sécrétaire. Le patron gagne 10000 euros et sa secrétaire seulement 1000 euros. On reformule le problème. On construit ensuite une suite de 10 nombres choisis uniformément au hasard dans l'ensemble $\{0, 1\}$. Le patron gagne si les 10 nombres sont les siens. Ce jeu est équivalent à celui décrit dans l'énoncé à ceci près qu'on continue le tirage même si quelqu'un gagne. Et avec cette nouvelle présentation, on peut conclure qu'il y a exactement $2^{10}$ tirages possibles et qu'il n'y a qu'un seul tirage gagnant pour le patron. La probabilité de gagner est seulement de $2^{-10}$ et non $\frac{1}{10}$ comme le patron le souhaitait.

Si on revient au jeu à quatre personnes et les salaires de l'énoncé, la probabilité de gagner du patron est cette fois de $4^{-10}$, celle de sa sécrétaire est au moins supérieure à $\frac{1}{16}$ qui correspond à sa probabilité de gagner en deux tirages. Si le patron ne gagne pas, c'est que sa probabilité de gagner est beaucoup trop petite par rapport au nombre de tirages. Il en faudrait au moins $4^{-10}$ pour avoir une probabilité non négligeable que le patron gagne au moins une fois.

**2)** $90 * 5 = 450$ qui n'est pas très loin de 429 ou encore $178/2 * 5 = 445$. La secrétaire paraît avoir 5 fois plus de chances de gagner que son patron et 5/2 fois plus que la personne payée 5000 euros.

La fonction `tirage` reçoit un nouvel élément `nb` qui est un compteur. Le programme ne remet plus à zéro ce compteur entre deux tirages. Cela signifie que si la boule du patron est sortie trois fois alors que la secrétaire a gagné. Lors du prochain tirage, le compteur du patron partira de 3 et celui de sa secrétaire de 0.

Si on ne remet jamais les compteurs à zéro, au bout de 1000 tirages (et non 1000 cadeaux), la boule du patron est sortie environ autant de fois que celle de la secrétaire, soit 250 fois puisqu'il y a quatre employés. La secrétaire aura gagné $\frac{250}{2} = 125$ fois, le patron aura gagné $\frac{250}{10} = 25$ soit 5 fois moins [2].

**fin exo 13.3.8** □

2. Je cite ici la plus belle des réponses retournée par un élève : Les patrons qui organisent des tombolas sont toujours mariés à leur secrétaire donc la question ne se pose pas.

## 13.4 Quatrième énoncé (2007-2008)

### 13.4.1 Suite récurrente (Fibonacci) *

**Enoncé**

Réécrire la fonction u de façon à ce qu'elle ne soit plus récurrente. (4 points)

```
def u (n) :
    if n <= 2 : return 1
    else : return u (n-1) + u (n-2) + u (n-3)
```

**Correction**

```
def u_non_recursif (n) :
    if n <= 2 : return 1
    u0 = 1
    u1 = 1
    u2 = 1
    i = 3
    while i <= n :
        u  = u0 + u1 + u2
        u0 = u1
        u1 = u2
        u2 = u
        i += 1
    return u
```

**fin exo 13.4.1** □

### 13.4.2 Calculer le résultat d'un programme *

**Enoncé**

**1)** Qu'affiche le programme suivant : (1 point)

```
def fonction (n) :
    return n + (n % 2)

print fonction (10)
print fonction (11)
```

**2)** Que fait la fonction `fonction` ? (1 point)

**3)** Ecrire une fonction qui retourne le premier multiple de 3 supérieur à `n` ? (2 points)

**Correction**

**1)** La fonction `fonction` ajoute 1 si $n$ est impair, 0 sinon. Le programme affiche :

```
10
12
```

**2)** La fonction retourne le plus petit entier pair supérieur ou égal à $n$.

**3)** On cherche à retourner le plus petit multiple de 3 supérieur ou égal à $n$. Tout d'abord, on remarque que $n + (n\%3)$ n'est pas la réponse cherchée car $4 + 4\%3 = 5$ qui n'est pas divisible par 3. Les fonctions les plus simples sont les suivantes :

```
def fonction3 (n) :
    k = 0
    while k < n : k += 3
    return k
```

Ou encore :

```
def fonction3 (n) :
    if   n % 3 == 0 : return n
    elif n % 3 == 1 : return n + 2
    else : return n + 1
```

Ou encore :

```
def fonction3 (n) :
    if n % 3 == 0 : return n
    else : return n + 3 - (n % 3)
```

**fin exo 13.4.2** □

### 13.4.3 Calculer le résultat d'un programme *

**Enoncé**

**1)** Qu'affiche le programme suivant : (1 point)

```
def division (n) :
    return n / 2

print division (1)
print division (0.9)
```

**2)** Proposez une solution pour que le résultat de la fonction soit 0.5 lors d'une instruction `print division`... ? (1 point)

**Correction**

**1)** 1/2 est égal à zéro en *Python* car c'est une division de deux entiers et le résultat est égal à la partie entière. La seconde division est une division entre un réel et un entier, le résultat est réel. Le programme affiche :

```
0
0.45
```

**2)** Voici deux solutions `print division(1.0)` ou `print division(float(1))`. Il est également possible de convertir `n` à l'intérieur de la fonction `division`. **fin exo 13.4.3** □

### 13.4.4 Ecrire un programme à partir d'un algorithme **

**Enoncé**

On considère deux listes d'entiers. La première est inférieure à la seconde si l'une des deux conditions suivantes est vérifiée :

– les $n$ premiers nombres sont égaux mais la première liste ne contient que $n$ éléments tandis que la seconde est plus longue
– les $n$ premiers nombres sont égaux mais que le $n+1^{\text{ème}}$ de la première liste est inférieur au $n+1^{\text{ème}}$ de la seconde liste

Par conséquent, si $l$ est la longueur de la liste la plus courte, comparer ces deux listes d'entiers revient à parcourir tous les indices depuis 0 jusqu'à $l$ exclu et à s'arrêter sur la première différence qui détermine le résultat. S'il n'y pas de différence, alors la liste la plus courte est la première. Il faut écrire une fonction `compare_liste(p,q)` qui implémente cet algorithme.

(3 points)

**Correction**

Cet algorithme de comparaison est en fait celui utilisé pour comparer deux chaînes de caractères.

```
def compare_liste (p,q) :
    i = 0
    while i < len (p) and i < len (q) :
        if   p [i] < q [i] : return -1  # on peut décider
        elif p [i] > q [i] : return  1  # on peut décider
        i += 1                          # on ne peut pas décider
    # fin de la boucle, il faut décider à partir des longueurs des listes
    if   len (p) < len (q) : return -1
    elif len (p) > len (q) : return  1
    else : return 0
```

On pourrait également écrire cette fonction avec la fonction `cmp` qui permet de comparer deux éléments quels qu'ils soient.

```
def compare_liste (p,q) :
    i = 0
    while i < len (p) and i < len (q) :
        c = cmp (p [i], q [i])
        if c != 0 : return c            # on peut décider
        i += 1                          # on ne peut pas décider
    # fin de la boucle, il faut décider à partir des longueurs des listes
    return cmp (len (p), len (q))
```

**fin exo 13.4.4** □

### 13.4.5 Comprendre une erreur d'exécution *

**Enoncé**

Le programme suivant est erroné.

```
l = [0,1,2,3,4,5,6,7,8,9]
i = 1
while i < len (l) :
    print l [i], l [i+1]
    i += 2
```

Il affiche des résultats puis une erreur.

```
1 2
3 4
5 6
7 8
9
```

```
Traceback (most recent call last):
  File "examen2008_rattrapage.py", line 43, in <module>
    print l [i], l [i+1]
IndexError: list index out of range
```

Pourquoi ? (1 point)

**Correction**

Le programme affiche les nombres par groupes de deux nombres consécutifs. Une itération de la boucle commence si la liste contient un élément suivant et non deux. Le programme est donc contraint à l'erreur car lors de la dernière itération, la liste contient un dixième nombre mais non un onzième. Le programme affiche le dixième élément (9) puis provoque une erreur `list index out of range`.

**fin exo 13.4.5** □

### 13.4.6 Précision des calculs **

**Enoncé**

On cherche à calculer la somme des termes d'une suite géométriques de raison $\frac{1}{2}$. On définit $r = \frac{1}{2}$, on cherche donc à calculer $\sum_{i=0}^{\infty} r^i$ qui une somme convergente mais infinie. Le programme suivant permet d'en calculer une valeur approchée. Il retourne, outre le résultat, le nombre d'itérations qui ont permis d'estimer le résultat.

```
def suite_geometrique_1 (r) :
    x = 1.0
    y = 0.0
    n = 0
    while x > 0 :
        y += x
        x *= r
        n += 1
    return y,n

print suite_geometrique_1 (0.5)  #affiche (2.0, 1075)
```

Un informaticien plus expérimenté a écrit le programme suivant qui retourne le même résultat mais avec un nombre d'itérations beaucoup plus petit.

```
def suite_geometrique_2 (r) :
    x = 1.0
    y = 0.0
    n = 0
    yold = y + 1
    while abs (yold - y) > 0 :
        yold = y
        y += x
        x *= r
        n += 1
    return y,n

print suite_geometrique_2 (0.5)  #affiche (2.0, 55)
```

Expliquez pourquoi le second programme est plus rapide tout en retournant le même résultat. (2 points) $2^{-55} \sim 2,8.10^{-17}$.

**Correction**

Tout d'abord le second programme est plus rapide car il effectue moins d'itérations, 55 au lieu de 1075. Maintenant, il s'agit de savoir pourquoi le second programme retourne le même résultat que le premier mais plus rapidement. L'ordinateur ne peut pas calculer numériquement une somme infinie, il s'agit toujours d'une valeur approchée. L'approximation dépend de la précision des calculs, environ 14 chiffres pour *Python*. Dans le premier programme, on s'arrête lorsque $r^n$ devient nul, autrement dit, on s'arrête lorsque $x$ est si petit que *Python* ne peut plus le représenter autrement que par 0, c'est-à-dire qu'il n'est pas possible de représenter un nombre dans l'intervalle $\left[0, 2^{-1055}\right]$.

Toutefois, il n'est pas indispensable d'aller aussi loin car l'ordinateur n'est de toute façon pas capable d'ajouter un nombre aussi petit à un nombre plus grand que 1. Par exemple, $1+10^{17} = 1,000\,000\,000\,000\,000\,01$. Comme la précision des calculs n'est que de 15 chiffres, pour *Python*, $1 + 10^{17} = 1$. Le second programme s'inspire de cette remarque : le calcul s'arrête lorsque le résultat de la somme n'évolue plus car il additionne des nombres trop petits à un nombre trop grand. L'idée est donc de comparer la somme d'une itération à l'autre et de s'arrêter lorsqu'elle n'évolue plus.

Ce raisonnement n'est pas toujours applicable. Il est valide dans ce cas car la série $s_n = \sum_{i=0}^{n} r^i$ est croissante et positive. Il est valide pour une série $s_n = \sum_{i=0}^{n} u_i$ et une suite $u_n$ de module décroissant.

**fin exo 13.4.6** □

### 13.4.7 Analyser un programme **

**Enoncé**

Un chercheur cherche à vérifier qu'une suite de 0 et de 1 est aléatoire. Pour cela, il souhaite compter le nombre de séquences de $n$ nombres successifs. Par exemple, pour la suite 01100111 et $n = 3$, les triplets sont 011, 110, 100, 001, 011, 111. Le triplet 011 apparaît deux fois, les autres une fois. Si la suite est aléatoire, les occurrences de chaque triplet sont en nombres équivalents.

Le chercheur souhaite également faire varier $n$ et calculer les fréquences des triplets, quadruplets, quintuplets, ... Pour compter ses occurrences, il hésite entre deux structures, la première à base de listes (à déconseiller) :

```
def hyper_cube_liste (n, m = [0,0]) :
    if n > 1 :
        m [0] = [0,0]
        m [1] = [0,0]
        m [0] = hyper_cube_liste (n-1, m [0])
        m [1] = hyper_cube_liste (n-1, m [1])
    return m
h = hyper_cube_liste (3)
print h                     # affiche [[[0, 0], [0, 0]], [[0, 0], [0, 0]]]
```

La seconde à base de dictionnaire (plus facile à manipuler) :

```
def hyper_cube_dico (n) :
    r = { }
    ind = [ 0 for i in range (0,n) ]
    while ind [0] <= 1 :
        cle                 = tuple (  ind ) # conversion d'une liste en tuple
        r [cle]             = 0
        ind [ len (ind)-1] += 1
        k                   = len (ind)-1
        while ind [ k ] == 2 and k > 0 :
            ind [k]    = 0
            ind [k-1] += 1
            k         -= 1
```

```
    return r
h = hyper_cube_dico (3)
print h        # affiche {(0, 1, 1): 0, (1, 1, 0): 0, (1, 0, 0): 0, (0, 0, 1): 0,
               #          (1, 0, 1): 0, (0, 0, 0): 0, (0, 1, 0): 0, (1, 1, 1): 0}
```

Le chercheur a commencé à écrire son programme :

```
def occurrence (l,n) :
    d = ....... # choix d'un hyper_cube (n)
    .....
    return d
suite = [ 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1 ]
h = occurrence (suite, 3)
print h
```

Sur quelle structure se porte votre choix (a priori celle avec dictionnaire), compléter la fonction `occurrence`. (4 points)

**Correction**

Tout d'abord, la structure matricielle est à déconseiller fortement même si un exemple d'utilisation en sera donné. La solution avec dictionnaire est assez simple :

```
def occurrence (l,n) :
    d = hyper_cube_dico (n)
    for i in range (0, len (l)-n) :
        cle = tuple (l [i:i+n])
        d [cle] += 1
    return d
```

Il est même possible de se passer de la fonction `hyper_cube_dico` :

```
def occurrence (l,n) :
    d = { }
    for i in range (0, len (l)-n) :
        cle = tuple (l [i:i+n])
        if cle not in d : d [cle] = 0
        d [cle] += 1
    return d
```

La seule différence apparaît lorsqu'un n-uplet n'apparaît pas dans la liste. Avec la fonction `hyper_cube_dico`, ce n-uplet recevra la fréquence 0, sans cette fonction, ce n-uplet ne sera pas présent dans le dictionnaire `d`. Le même programme avec la structure matricielle est plus une curiosité qu'un cas utile.

```
def occurrence (l,n) :
    d = hyper_cube_liste (n, [0,0])         # * remarque voir plus bas
    for i in range (0, len (l)-n) :
        cle = l [i:i+n]
        t   = d                         #
        for k in range (0,n-1) :        # point clé de la fonction :
            t = t [ cle [k] ]           #                accès à un élément
        t [cle [ n-1] ] += 1
    return d
```

Si on remplace la ligne marquée d'une étoile par `d = hyper_cube_list(n)`, le programme retourne une erreur :

```
Traceback (most recent call last):
  File "examen2008_rattrapage.py", line 166, in <module>
```

```
    h = occurrence (suite, n)
  File "examen2008_rattrapage.py", line 160, in occurrence
    t [cle [ n-1] ] += 1
TypeError: 'int' object is not iterable
```

Cette erreur est assez incompréhensible puisque la modification a consisté à appeler une fonction avec un paramètre de moins qui était de toutes façons égal à la valeur par défaut associée au paramètre. Pour comprendre cette erreur, il faut exécuter le programme suivant :

```
def fonction (l = [0,0]) :
    l [0] += 1
    return l

print fonction ()         # affiche [1,0] : résultat attendu
print fonction ()         # affiche [2,0] : résultat surprenant
print fonction ( [0,0])   # affiche [1,0] : résultat attendu
```

L'explication provient du fait que la valeur par défaut est une liste qui n'est pas recréée à chaque appel mais c'est la même liste à chaque fois que la fonction est appelée sans paramètre. Pour remédier à cela, il faudrait écrire :

```
import copy
def fonction (l = [0,0]) :
    l = copy.copy (l)
    l [0] += 1
    return l
```

Dans le cas de l'hypercube, il faudrait écrire :

```
def hyper_cube_liste (n, m = [0,0]) :
    m = copy.copy (m)
    if n > 1 :
        m [0] = [0,0]
        m [1] = [0,0]
        m [0] = hyper_cube_liste (n-1, m [0])
        m [1] = hyper_cube_liste (n-1, m [1])
    return m
```

**fin exo 13.4.7** □

## 13.5 Cinquième énoncé (2008-2009)

### 13.5.1 Portée des variables

#### Enoncé

**1)** Répondre aux deux questions qui suivent ? (2 points)

```
def f():
    x = 0
    print x


f()

def g():
    print x

g()
```

Le premier programme produit l'erreur qui suit. Pourquoi ?

```
Traceback (most recent call last):
  File "p1.py", line 11, in <module>
    g()
  File "p1.py", line 9, in g
    print x
NameError: global name 'x' is not defined
```

```
def f():
    x = 0
    print x

x = 1
f()

def g():
    print x

g()
```

Qu'affiche le second programme ?

**2)** Le premier des deux programmes suivant s'exécute sans erreur, qu'affiche-t-il ? (1 point)

```
a       = [ None, 0 ]
b       = [ None, 1 ]
a [0]   = b
b [0]   = a
print a [0][0] [0][0] [0][0] [1]
```

```
import copy
a       = [ None, 0 ]
b       = [ None, 1 ]
a [0]   = copy.copy (b)
b [0]   = a
print a [0][0] [0][0] [0][0] [1]
```

**3)** On ne sait rien du second programme qui utilise le module `copy`. S'exécute-t-il sans erreur ? Si la réponse est positive, qu'affiche-t-il ? Si la réponse est négative, pourquoi ? (1 point)

**Correction**

**1)** L'erreur se produit dans la fonction `g` car il n'existe aucune variable globale ou locale portant le nom `x`, l'instruction `print x` ne peut donc être exécutée. La variable `x` définie dans la fonction `f` est une variable locale et son existence est limité à la fonction `f`.

A contrario, le second programme définit une variable globale `x = 1` avant l'appel à la fonction `g`. Le programme affiche donc :

```
0
1
```

**2)** L'instruction `print a[0][0][0][0][0][0][1]` peut surprendre mais elle est tout-à-fait correcte dans le premier programme. Lorsqu'il s'agit d'une liste, une affectation ne copie pas la liste mais donne à cette liste un second identifiant :

```
a       = [ None, 0 ]
b       = [ None, 1 ]
a [0]   = b   # b et a [0] désignent la même liste
b [0]   = a   # b [0] et a désignent la même liste
```

On peut résumer les listes `a` et `b` par l'écriture suivante :

$$a \Longleftrightarrow [b, 0]$$
$$b \Longleftrightarrow [a, 0]$$

On peut écrire que :

$$\begin{array}{rr} & a[0][0][0][0][0][0][1] \\ = & b[0][0][0][0][0][1] \\ = & a[0][0][0][0][1] \\ = & a[0][0][1] \\ = & a[1] \\ = & 0 \end{array}$$

Le programme affiche donc :

```
0
```

**3)** Dans le second programme, l'instruction `copy.copy(b)` créée une copie `c` de la liste `b`. On peut résumer les listes `a` et `b` par l'écriture suivante :

$$\begin{array}{l} a \Longleftrightarrow [c, 0] \Longleftrightarrow [[None, 1], 0] \\ b \Longleftrightarrow [a, 0] \end{array}$$

Par conséquent :

$$\begin{array}{rr} & a[0][0][0][0][0][0][1] \\ = & c[0][0][0][0][0][1] \\ = & None[0][0][0][0][1] \\ \Longleftarrow & erreur \end{array}$$

**fin exo 13.5.1** □

### 13.5.2 Somme d'éléments aléatoires

**Enoncé**

**1)** On considère le programme suivant :

```
import random
def somme_alea (tableau, n) :
    s = 0
    for i in range (0, n) :
        h = random.randint (0, len (tableau))
        s += tableau [h]
    print s / n
x = somme_alea ( range (0,100), 10)
print "x = ", x
```

A sa première exécution, il affiche :

```
44
x =  None
```

Ce n'est pas tout-à-fait le résultat attendu, le terme `None` ne devrait pas apparaître, que faut-il faire pour le corriger ? (1 point)

**2)** On s'aperçoit également que le programme provoque l'erreur suivante environ une fois sur dix :

```
Traceback (most recent call last):
  File "examen2009.py", line 10, in <module>
    print somme_alea ( range (0,100), 10)
  File "examen2009.py", line 8, in somme_alea
    s += tableau [h]
IndexError: list index out of range
```

Pourquoi ? Expliquer également pourquoi le programme ne provoque cette erreur qu'une fois sur dix environ ? Une réponse statistique est attendue. (2 points)

**3)** Enfin, le résultat attendu est réel alors que le résultat du programme est toujours entier, pourquoi ? (1 point)

**Correction**

**1)** Il suffit de remplacer l'instruction `print s/n` par rapport `return s/n`. Le programme affichera :

```
44
```

**2)** L'erreur nous apprend que l'instruction `s+ = tableau[n]` provoque une erreur parce que la variable `h` contient un entier soit négatif, soit supérieur ou égal à `len(tableau)`. C'est ce second cas qui se produit parfois. La fonction `random.randint` retourne un entier aléatoire compris entre `0` et `len(tableau)` inclus. Il faudrait changer l'instruction précédente en :

```
h = random.randint (0, len (tableau)-1)
```

L'erreur ne se produit pas toujours car le programme tire un nombre aléatoire qui peut être égal à `len(tableau)`. En fait, il faut calculer la probabilité que le programme a de tirer le nombre `len(tableau)` que l'on notera $n$ dans la suite. Le programme fait la moyenne de 10 nombres aléatoires tirés au hasard mais uniformément dans l'ensemble $\{0, \ldots, n\}$. La probabilité que le programme fonctionne est la probabilité $p$ de ne jamais tomber sur le nombre $n$ en 10 tirages :

$$p = \left(\frac{n-1}{n}\right)^{10} = \left(1 - \frac{1}{n}\right)^{10} \tag{13.4}$$

Pour le programme en exemple :

$$p = (1 - \frac{1}{101})^{10} \sim 1 - \frac{10}{101} \sim 0,9 \tag{13.5}$$

Si $0,9$ est la probabilité que le programme fonctionne, 0,1 est la probabilité que le programme produise une erreur. On retrouve l'estimation citée dans l'énoncé. Le calcul non approché donne une valeur de 0,0947.

Beaucoup d'élèves ont répondu que le programme produisait une erreur une fois sur dix environ car appeler 10 fois une fonction qui produit une erreur une fois sur 100 revient à faire le calcul : $10 * \frac{1}{101} \sim \frac{1}{10}$. Même si ce raisonnement aboutit à résultat proche de la bonne réponse, il est tout-à-fait faux. Il suffit pour s'en convaincre d'écrire à un programme qui appelle 200 fois la fonction `somme_alea`. D'après ce raisonnement, la probabilité que le programme produit une erreur serait de $200 * \frac{1}{101} \sim 2$ ce qui est impossible.

**3)** Le programme ne fait que manipuler des entiers, l'instruction `print s/n` effectue donc une division entière. Le programme ne peut produire qu'un résultat entier et non réel. Il faudrait écrire pour cela `print float(s)/float(n)`.

**fin exo 13.5.2** □

### 13.5.3 Coût d'une moyenne mobile centrée

**Enoncé**

**1)** Quel est le coût de la fonction `moyenne_mobile` en fonction de la longueur `n` de la suite `suite` et de la largeur de la moyenne mobile `m` à un facteur multiplicatif près ? (1 point)

```
def moyenne_mobile (suite, m) :
    res = []
    for i in range (m, len (suite)-m) :
        s = 0
        for j in range (-m,m+1) : s += suite [i+j]
        res.append (  float (s) / (m*2+1) )
    return res
```

**2)** On peut faire plus rapide en remplaçant les lignes en pointillés dans le programme qui suit. (1 point)

```
def moyenne_mobile (suite, m) :
    res = []
    for i in range (m, len (suite)-m) :
        if i == m :
            # ... ligne à remplacer
        else :
            # ... ligne à remplacer
        res.append (  float (s) / (m*2+1) )
    return res
```

**3)** Quel est le coût du nouveau programme toujours en fonction de la longueur `n` de la suite `suite` et de la largeur de la moyenne mobile `m` ? (1 point)

**Correction**

**1)** La fonction `moyenne_mobile` inclut deux boucles imbriquée. La première est de longueur $n - 2m$, la seconde est de longueur $2m$. Le coût de la fonction est donc en $O((n-2m)2m) = O(2mn - 4m^2)$. On peut supposer que $m$ est plus petit devant $n$ et négliger le second terme. Le coût recherché est : $O(mn)$.

**2)** Lorsqu'on calcule la valeur de la somme `s` pour les itérations $i$ et $i+1$, on ajoute presque les mêmes termes (on cote $\texttt{s} = \texttt{suite}$ pour simplifier) :

$$\begin{array}{l|lllll} \texttt{i} & \texttt{s}[\texttt{i}-\texttt{m}] + & \texttt{s}[\texttt{i}-\texttt{m}+1] + & \ldots & + \texttt{s}[\texttt{i}+\texttt{m}-1] & + \texttt{s}[\texttt{i}-\texttt{m}] \\ \texttt{i}+1 & & \texttt{s}[\texttt{i}-\texttt{m}+1] + & \ldots & + \texttt{s}[\texttt{i}+\texttt{m}-1] & + \texttt{s}[\texttt{i}-\texttt{m}] \quad + \texttt{s}[\texttt{i}-\texttt{m}+1] \end{array}$$

Pour passer d'une somme à la suivante, il suffit d'ajouter un terme et d'en soustraire un. Il faut également calculer la première somme lorsque $\texttt{i} = \texttt{m}$. Ceci donne :

```
def moyenne_mobile (suite, m) :
    res = []
    for i in range (m, len (suite)-m) :
        if i == m :
            s = sum ( suite [i-m : i+m+1] )
        else :
            s = s - suite [i-m-1] + suite [i+m]
        res.append (  float (s) / (m*2+1) )
    return res
```

**3)** La fonction contient toujours deux boucles imbriquées (la fonction `sum` cache une boucle) mais la seconde boucle n'est exécutée qu'une seule fois. Le coût de la fonction n'est plus que $O(n+m)$. Comme $n$ est souvent beaucoup plus grand que $m$, on peut négliger le second terme : $O(n)$.

**fin exo 13.5.3** □

### 13.5.4 Deviner l'objectif d'une fonction

**Enoncé**

**1)** Que fait la fonction suivante ? (2 points)

```
def fonction_mystere (tableau) :
    for i in range (0, len (tableau)) :
        if tableau [i] % 2 != 0 : continue
        pos = i
        for j in range (i+1, len (tableau)) :
            if tableau [j] % 2 != 0 : continue
            if tableau [pos] < tableau [j] : pos = j
        ech           = tableau [i]
        tableau [i]   = tableau [pos]
        tableau [pos] = ech
```

**Correction**

**1)** Pour comprendre l'objectif de la fonction, il suffit d'enlever les deux lignes suivantes :

```
        if tableau [i] % 2 != 0 : continue
            if tableau [pos] < tableau [j] : pos = j
```

La fonction devient :

```
def fonction_mystere (tableau) :
    for i in range (0, len (tableau)) :
        pos = i
        for j in range (i+1, len (tableau)) :
            if tableau [pos] < tableau [j] : pos = j
        ech           = tableau [i]
        tableau [i]   = tableau [pos]
        tableau [pos] = ech
```

Cette fonction correspond à un tri par ordre décroissant. Les deux lignes supprimées ne font qu'éviter l'exécution des itérations pour les éléments impairs. La fonction trie par ordre décroissant les éléments pairs d'un tableau tout en laissant inchangés les éléments impairs. Par exemple :

```
a = [ 0, 1, 4, 3, 5, 2 ]
print a                     # affiche [ 0, 1, 4, 3, 5, 2 ]
b = fonction_mystere (a)
print b                     # affiche [ 4, 1, 2, 3, 5, 0 ]
```

**fin exo 13.5.4** □

### 13.5.5 Calcul du plus court chemin dans un graphe

**Enoncé**

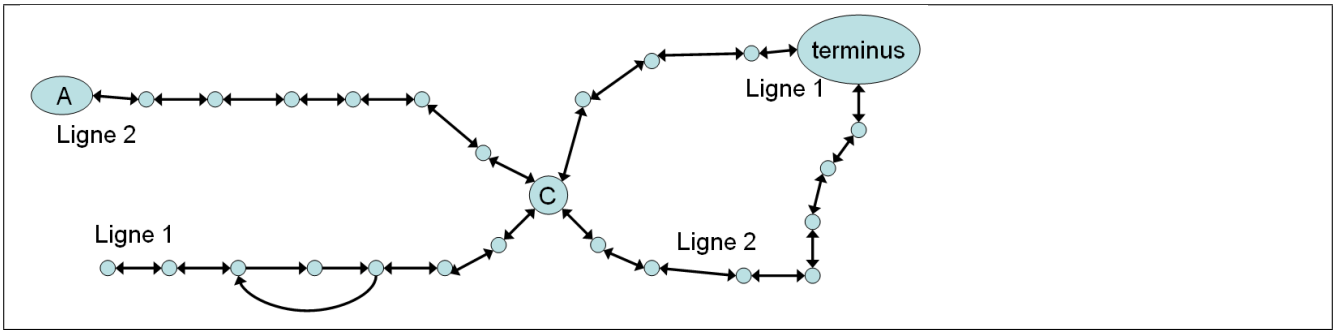



**Figure 13.2 :** *Figure représentant deux lignes de métro qui se croisent en C et qui se rejoignent en terminus. Chaque station est représentée par un point, chaque arc indique que deux stations sont reliées par la même ligne de métro. Tous les arcs sont à double sens sauf trois car dans ce cas, le métro ne suit pas le même chemin dans les deux sens.*

La figure 13.2 représente deux lignes de métro. On représente ce graphe par une matrice carrée $M = (m_{ij})$ dans laquelle il y a autant de lignes que de stations. Chaque élément $m_{ij}$ représente la durée de transport entre les stations $i$ et $j$ dans le sens $i \longrightarrow j$. $m_{ij}$ est positif si la liaison existe, infini (ou très grand) si elle n'existe pas.

**1)** La matrice $M$ est-elle symétrique ? Pourquoi ? (1 point)

**2)** On détermine le chemin le plus court entre les stations $A$ et *terminus* grâce à un algorithme (Bellman, Dikjstra, ...). La solution indique qu'il faut changer de ligne à la station $C$. Toutefois, cette solution ne prend pas en compte la durée du changement de ligne. C'est pourquoi on souhaite modifier la matrice $M$ en y insérant la durée du changement de ligne à la station $C$ de façon à pouvoir utiliser le même algorithme pour déterminer le plus court chemin. Comment faire ? La réponse peut être soit rédigée, soit graphique en s'inspirant de la figure 13.2 sans nécessairement la reproduire en entier. (2 points)

**Correction**

**1)** La matrice n'est pas symétrique sur la ligne 1, il existe trois arcs orientés : le passage du métro ne s'effectue que dans un seul sens. Si on nomme $i$, $j$, $k$ les trois stations concernées, les coefficients $m_{ij}$ et $m_{ji}$ sont différents. L'un des deux sera infini puisque la connexion n'existe pas.

**2)** Il faut nécessairement ajouter des lignes ou des colonnes dans la matrice $M$ car tous les coefficients représente une durée de transport. Pour introduire les durées de changements, on considère que les nœuds du graphe représente un couple $(station, ligne)$. De cette façon, un coefficient $m_{ij}$ de la matrice représente :
– une durée de transport si les nœuds $i$ et $j$ sont deux stations différents de la même ligne
– une durée de changement si les nœuds $i$ et $j$ sont en fait la même station mais sur deux lignes différentes.
La réponse graphique est représentée par la figure 13.3. Il faut noter également que cette solution permet de prendre également en compte un couple de stations reliées par deux lignes différentes ce qui n'était pas possible avec la matrice décrite dans l'énoncé. C'est le cas par exemple des stations Concorde et Madeleine qui sont reliées par les lignes 8 et 12 du métro parisien.

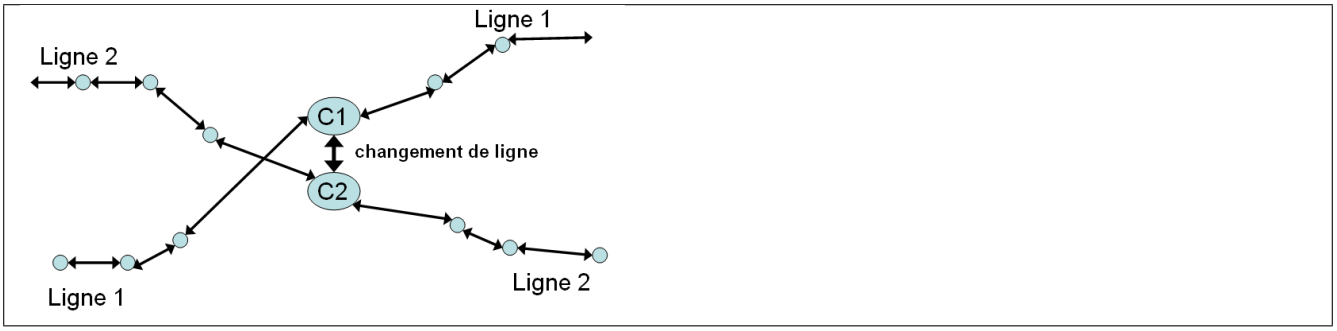



**Figure 13.3 :** *Chaque nœud du graphe représente une station pour une ligne donnée. Cela signifie que si une station est présente sur deux lignes, elle est représentée par deux nœuds. Un arc peut donc relier deux stations différents de la même ligne ou deux stations identiques de deux lignes différentes. Dans ce second cas, c'est la durée d'un changement de ligne.*

Quelques élèves ont ajouté trois stations au lieu d'une seule, la station C étant représentée par quatre stations, ces quatre stations étant reliées par des arcs de longueurs nulles ou égales au temps de changement de ligne. C'est une solution même si elle est inutilement compliquées.

Un élève a proposé une solution qui n'implique pas l'ajout d'une station. Cette solution est envisageable dans le cas de deux lignes de métro mais elle peut devenir compliquée à mettre en place dans le cas de plusieurs lignes avec plusieurs stations consécutives sur la même ligne (voir figure 13.4).

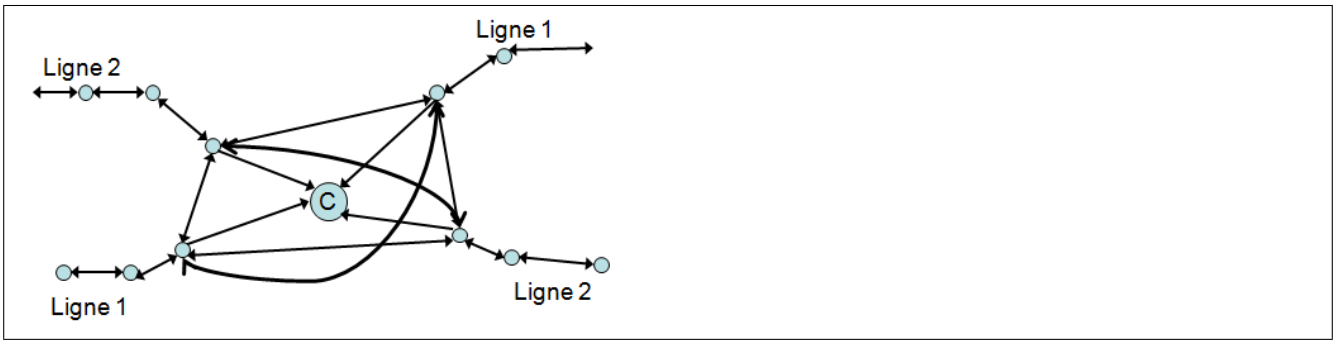



**Figure 13.4 :** *Toutes les connexions menant à la station C sont maintenant à sens unique : elles ne permettent que d'arriver en C mais pas d'en repartir. Pour aller à une autre station voisine de C, il faut passer par l'un des 6 arcs ajoutés qui incluent ou non un temps de transfert selon que ces stations appartiennent à la même ligne. Cette solution n'est pas aussi simple que la première et peut rapidement devenir compliquée si une même station appartient à trois, quatre, lignes.*

La solution impliquait nécessairement la création d'arcs (ou de connexion) entre différentes stations. Il ne suffisait pas d'ajouter le temps de changement à des coefficients de la matrice de transition, cette opération a pour conséquence de modifier les distances d'au moins un voyage sur une même ligne.

**fin exo 13.5.5** □

### 13.5.6 Héritage

#### Enoncé

**1)** On construit une classe dont le but est de faire une boucle.

```
class Boucle :
    def __init__ (self, a, b) :
        self.a = a
```

```
        self.b = b
    def iteration (self, i) :              # contenu d'une itération
        pass
    def boucle (self) :                    # opère la boucle
        for i in range (self.a, self.b) :
            self.iteration (i)
```

On l'utilise pour faire une somme : que fait le programme suivant ? Quel est le résultat numérique affiché ? (1 point)

```
class Gauss (Boucle) :
    def __init__ (self, b) :
        Boucle.__init__ (self, 1, b+1)
        self.x = 0

    def iteration (self,i) :
        self.x += i

g = Gauss (10)
g.boucle ()
print g.x
```

**2)** De façon la plus concise possible, écrire la classe `Carre` qui hérite de la classe `Gauss` et qui fait la somme des `n` premiers entiers au carré. (1 point)

**3)** On souhaite écrire la classe `Boucle2` qui effectue une double boucle : une boucle imbriquée dans une boucle. Le barême de cette question est le suivant :
– 1 point si le code de la classe `Boucle2` contient deux boucles (`for` ou `while`).
– 2 points si le code de la classe `Boucle2` ne contient qu'une boucle (`for` ou `while`).

**Correction**

**1)** Le programme effectue la somme des entiers compris entre 1 et 10 inclus : $10*11/2 = 55$. Le programme affiche 55.

**2)** Le plus court est de repartir de la classe `Gauss` qui redéfinit le constructeur dont on a besoin.

```
class Carre (Gauss) :
    def iteration (self,i) :
        self.x += i*i           # ligne changée

g = Carre (10)
g.boucle ()
print g.x
```

**3)** La version incluant deux boucles est la plus simple à présenter. On modifie la classe `Boucle` de façon à avoir deux boucles. La classe `Gauss2` est un exemple d'utilisation de la classe `Boucle2`.

```
class Boucle2 (Boucle) :
    def __init__ (self, a, b, c, d) :
        Boucle.__init__ (self, a, b)
        self.c = c
        self.d = d
    def iteration (self, i, j) :
        pass
    def boucle (self) :
        for j in range (self.c, self.d) :
            for i in range (self.a, self.b) :
                self.iteration (i,j)

class Gauss2 (Boucle2) :
    def __init__ (self, a, b) :
```

```
        Boucle2.__init__ (self, 1, a+1, 1, b+1)
        self.x = 0
    def iteration (self, i,j) :
        self.x += i+j
```

La méthode `iteration` inclut maintenant deux paramètres `i` et `j` et il paraît difficile de réutiliser la méthode `iteration` de la classe `boucle`. Un moyen consiste à ajouter l'attribut `i` à la classe `Boucle2`. Il faut également modifier la classe `Gauss2`.

```
class Boucle2 (Boucle) :
    def __init__ (self, a, b, c, d) :
        Boucle.__init__ (self, a, b)
        self.c = c
        self.d = d
    def iteration (self, i) :
        pass
    def boucle (self) :
        for j in range (self.c, self.d) :
            self.j = j
            for i in range (self.a, self.b) :
                self.iteration (i)

class Gauss2 (Boucle2) :
    def __init__ (self, a, b) :
        Boucle2.__init__ (self, 1, a+1, 1, b+1)
        self.x = 0
    def iteration (self, i) :
        self.x += self.j+i
```

On peut maintenant supprimer une boucle dans la méthode `boucle` de la classe `Boucle2`. La classe `Gauss2` reste quant à elle inchangée.

```
class Boucle2 (Boucle) :
    def __init__ (self, a, b, c, d) :
        Boucle.__init__ (self, a, b)
        self.c = c
        self.d = d
    def iteration (self, i) :
        pass
    def boucle (self) :
        for j in range (self.c, self.d) :
            self.j = j
            Boucle.boucle (self)  # suppression d'une boucle
                                  # caché dans la classe Boucle
```

La solution suivante est sans doute la plus simple puisqu'on n'utilise pas de subterfuge tel que `self.j = j`. On surcharge cette fois la méthode `iteration` et non plus `boucle`. Mais il faut créer une seconde méthode `iteration2` acceptant `i, j` comme paramètres [3].

```
class Boucle2 (Boucle) :
    def __init__ (self, a, b, c, d) :
        Boucle.__init__ (self, a, b)
        self.c = c
        self.d = d

    def iteration (self, i) :
```

3. Le langage *C++* permettrait d'appeler la méthode `iteration(self, i, j)` car ce langage différencie les méthodes acceptant des nombres différents de paramètres.

```
        for j in range (self.c, self.d) :
            self.iteration2 (i,j)

    def iteration2 (self, i,j) :
        pass


class Gauss2 (Boucle2) :
    def __init__ (self, a, b) :
        Boucle2.__init__ (self, 1, a+1, 1, b+1)
        self.x = 0
    def iteration2 (self, i, j) :
        self.x += j+i
```

Il existait une autre méthode pour faire disparaître une boucle sans doute plus simple que ces manipulations qui utilise une façon simple de manipuler les indices. C'est le même mécanisme que celui utilisé pour représenter une matrice avec une seule liste et non une liste de listes. Ce n'était pas la réponse souhaitée mais elle n'introduit qu'une seule boucle également.

```
class Boucle2 (Boucle) :
    def __init__ (self, a, b, c, d) :
        Boucle.__init__ (self, a, b)
        self.c = c
        self.d = d
    def iteration (self, i, j) :
        pass
    def boucle (self) :
        ab = self.b - self.a
        cd = self.d - self.c
        n  = ab * cd
        for k in range (0, n) :
            i = k % ab + self.a
            j = k / ab + self.c
            self.iteration (i,j)
```

**fin exo 13.5.6** □

## 13.6 Exercices supplémentaires

### 13.6.1 Position initiale d'un élément après un tri **

#### Enoncé

On suppose qu'on peut trier la liste `l = [...]` avec la méthode `sort`. Une fois le tableau trié, comment obtenir la position du plus petit élément dans le tableau initial ?

#### Correction

L'idée est de ne pas trier la liste mais une liste de couples incluant chaque élément avec sa position. On suppose que la liste `l` existe. La méthode `sort` utilisera le premier élément de chaque couple pour tier la liste de couples.

```
l2 = [ ( l [i], i ) for i in range (0, len (l)) ]
l2.sort ()
print l2 [0][1]  # affiche la position du plus petit élément
                 # dans le tableau initial
```

**fin exo 13.6.1** □

### 13.6.2 Comprendre une erreur de logique *

**Enoncé**

**1)** Quel est le résultat affiché par le programme suivant :

```
def ensemble_lettre (s) :
          ens = []
          for i in range (0, len (s)) :
              c = s [i]
              if c in ens :
                  ens.append (c)
          return ens

print lettre ("baaa")
```

Est-ce que ce résultat change si on appelle la fonction `ensemble_lettre` avec un autre mot ?

**2)** Le programme précédent n'est vraisemblablement pas fidèle aux intentions de son auteur. Celui-ci avait pour objectif de déterminer l'ensemble des lettres différentes de la chaîne de caractères passée en entrée. Que faut-il faire pour le corriger ? Que sera le résultat de l'instruction `ensemble_lettre`("`baaa`") en tenant compte de la modification suggérée ?

**Correction**

**1)** L'instruction `if c in ens :` signifie que le caractère `c` est ajouté seulement s'il est déjà présent dans la liste `s` qui est vide au début de la fonction. Elle sera donc toujours vide à la fin de l'exécution de la fonction et sera vide quelque soit le mot `s` fourni en entrée comme paramètre. Le résultat ne dépend donc pas du mot.

**2)** Il suffit de changer `if c in ens :` en `if c not in ens :` pour donner :

```
def ensemble_lettre (s) :
    ens = []
    for i in range (0, len (s)) :
        c = s [i]
        if c not in ens :
            ens.append (c)
    return ens
```

Le résultat pour le mot "`baaa`" est ["`b`","`a`"].

**fin exo 13.6.2** □

### 13.6.3 Comprendre une erreur d'exécution *

**Enoncé**

```
k = [10,14,15,-1,6]
l = []
for i in range (0,len (k)) :
    l.append ( k [ len (k) - i ] )
```

Le programme génère une exception de type `IndexError`, pourquoi ?

**Correction**

Lors du premier passage dans la boucle `for`, le premier ajouté dans la liste `l` est `k`[`len`(`k`)] qui n'existent pas puisque les indices vont de 0 à `len`(`k`). Voici la correction :

```
k = [10,14,15,-1,6]
l = []
for i in range (0,len (k)) :
    l.append ( k [ len (k) - i-1 ] )  # -1 a été ajouté
```

**fin exo 13.6.3** □

### 13.6.4 Précision des calculs ***

#### Enoncé

On cherche à calculer $\ln 2$ grâce à la formule $\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + ....$ Pour $x = 1$, cette suite est convergente et permet de calculer $\ln 2$.

**1)** On propose deux programmes. Le premier utilise une fonction `puiss`. Le second ne l'utilise pas :

```
def puiss (x, n) :
    s = 1.0
    for i in range (0,n) :
        s *= x
    return s

def log_suite (x) :
    x    = float (x-1)
    # ......
    s    = 0
    old  = -1
    n    = 1
    while abs (old - s) > 1e-10 :
        old = s
        po  = puiss (x,n) / n
        if n % 2 == 0 : po = -po
        s  += po
        n  += 1
        # ......
    return s

print log_suite (2)
```

```
# ......
# ......
# ......
# ......
# ......
# ......
def log_suite (x) :
    x    = float (x-1)
    x0   = x
    s    = 0
    old  = -1
    n    = 1
    while abs (old - s) > 1e-10 :
        old = s
        po  = x / n
        if n % 2 == 0 : po = -po
        s  += po
        n  += 1
        x  *= x0
    return s

print log_suite (2)
```

Quel est le programme le plus rapide et pourquoi ? Quels sont les coûts des deux algorithmes ?

**2)** Combien faut-il d'itérations pour que la fonction retourne un résultat ?

**3)** On introduit la fonction `racine_carree` qui calcule $\sqrt{k}$. Celle-ci est issue de la résolution de l'équation $f(x) = 0$ avec $f(x) = x^2 - k$ à l'aide de la méthode de *Newton*[4].

```
def racine_carree (k) :
    x0 = float (k)+1
    x  = float (k)
    while abs (x-x0) > 1e-10 :
        x0 = x
        x  = (k-x*x) / (2 * x) + x
    return x
```

Cette fonction retourne un résultat pour $\sqrt{2}$ en 6 itérations. On décompose ensuite $\ln 2 = \ln\left((\sqrt{2})^2\right) = 2\ln\sqrt{2} = 2\ln\left(1 + \left(\sqrt{2} - 1\right)\right)$. En utilisant cette astuce et la fonction `racine2`, à combien estimez-vous

4. Cette méthode est utilisée pour résoudre l'équation $f(x) = 0$. On construit une suite convergeant vers la solution définie par $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$. Sous certaines conditions, la suite $(x_n)$ converge vers la solution. Il suffit d'appliquer cela à la fonction $f(x) = x^2 - k$ où $k$ est le nombre dont on veut trouver la racine.

grossièrement le nombre d'itérations nécessaires pour calculer $\ln 2$ : 10, 40 ou 100 ? Justifiez. On rappelle que $2^{10n} \sim 10^{3n}$.

**4)** Que proposez-vous pour calculer $\ln 100$ ?

**Correction**

**1)** Le second programme est le plus rapide. A chaque boucle du premier programme, la fonction `puiss` calcule $x^n$ avec $n$ multiplications, ce calcul est remplacé par une seule multiplication dans le second programme.

Les deux programmes exécuteront exactement le même nombre de fois la boucle `while`. Soit $n$ ce nombre d'itérations, le coût du premier programme est $1+2+3+4+...+n \sim O(n^2)$ car la fonction `puiss` fait $i$ passages dans sa boucle `for` pour l'itération $i$. Le coût du second programme est en $O(n)$.

**2)** Si on pose $s_n = \sum_{k=1}^{n} (-1)^{k+1} \frac{x^k}{k}$, la condition d'arrêt de la fonction `log_suite` correspond à :

$$|s_n - s_{n-1}| > 10^{-10} \iff \left|\frac{x^k}{k}\right| > 10^{-10}$$

Comme on calcule $\ln(1+1)$, $x = 1$, la condition est donc équivalente à :

$$\frac{1}{k} > 10^{-10} \iff k < 10^{10}$$

La fonction `log_suite` fait dix milliards de passages dans la boucle `while`, dix milliards d'itérations.

**3)** Cette fois-ci, on cherche à calculer $\ln\sqrt{2} = \ln\left(1+\sqrt{2}-1\right)$, d'où $x = \sqrt{2}-1$. La condition d'arrêt est toujours $\left|\frac{x^k}{k}\right| > 10^{-10}$ et on cherche à majorer le nombre d'itérations nécessaires pour que la fonction converge vers un résultat. $\sqrt{2} \sim 1.414 < \frac{1}{2}$. On majore donc $x$ par $\frac{1}{2}$ :

$$\begin{aligned}
&\implies \left|\frac{\left(\sqrt{2}-1\right)^k}{k}\right| > 10^{-10} \\
&\implies \left|\frac{1}{2^k k}\right| > 10^{-10} \\
&\implies \frac{1}{2^k} > 10^{-10} \implies 2^k < 10^{10}
\end{aligned}$$

On utilise l'approximation donnée par l'énoncé à savoir $2^{10n} \sim 10^{3n}$.

$$2^k < 10^{10} \iff 2^{10k} < 10^{100} \sim 10^{3k} < 10^{100} \iff 3k < 100 \iff k < \frac{100}{3} \iff k < 34$$

Si on ajoute 34 aux 6 itérations nécessaires pour calculer $\sqrt{2}$, on trouve 40 itérations.

**4)** $\ln(1+x)$ n'est calculable que pour $|x| < 1$. Il faut fonc transformer $\ln 100$ pour le calculer à partir d'une racine carrée et de $\ln(1+x)$ pour un $x$ tel que $x < 1$ (dans le cas contraire, la suite ne converge pas) :

$$\begin{aligned}
\ln 100 &= 2\ln 10 = 2\ln 2 + 2\ln 5 = 2\ln 2 + 2\ln\left(4\frac{5}{4}\right) \\
&= 2\ln 2 + 2\ln 4 + 2\ln\frac{5}{4} = 2\ln 2 + 4\ln 2 + 2\ln\left(1+\frac{5}{4}-1\right) \\
&= 6\ln 2 + 2\ln\left(1+\frac{1}{4}\right)
\end{aligned}$$

C'est une décomposition, ce n'est pas la seule possible.

**fin exo 13.6.4** □

Troisième partie

# RÉSUMÉ PARTIQUE, OUTILS D'AIDE, ALGORITHMES À CONNAÎTRE

Cette partie est un résumé de la première, elle reprend sans détailler les éléments essentiels du langage *Python* et ne tient pas compte des extensions telles que les interfaces graphiques. Elle suffit à décrire le langage Python à quelqu'un qui ne le connaît pas mais connaît déjà un autre langage de programmation. Elle est également utile lorsqu'on conçoit ses premiers programmes et limite la recherche à une vingtaine de pages au lieu d'une centaine.

Le second chapitre s'attarde sur des algorithmes de tri qu'il est préférable de connaître. Ils existent dans tous les langages et sont utilisés dans presque tous les programmes. Il est difficile d'écrire un programme ne contenant aucun tri comme il est difficile d'écrire sans utiliser la lettre *e*. Il est également parfois plus rapide de reprogrammer un tri que de chercher une fonction qui correspond à vos besoins comme le fait de trier conjointement plusieurs listes. Les deux sujets qui suivent sont rédigés de telle sorte qu'ils peuvent aussi être lu comme des énoncés d'exercices.

Certaines pages concernent le calcul du coût d'un algorithme [5]. Ces considérations mathématiques n'interviennent que rarement, principalement lorsque l'exigence en terme de rapidité d'exécution n'est pas accessible simplement par l'utilisation d'une machine plus puissante ou d'un langage de programmation plus rapide. Cette exigence passe par une amélioration de l'algorithme. Il existe par exemple des fonctions dont il n'existe pas d'expression du coût à l'aide des opérations standard (addition, multiplication, puissance). C'est le cas de la fonction d'Ackermann.

$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ et } n = 0 \\ A(m-1, A(mn, n-1)) & \textit{sinon} \end{cases} \tag{13.6}$$

5. Le coût d'un algorithme est le nombre d'opérations élémentaires (addition, multiplication, affectation, ...) nécessaires à sa réalisation.

# Chapitre 14

# Instructions fréquentes en *Python*

---

## 14.1 Le langage

Quelques rappels sur le langage :

1. Il n'y a pas de séparateur d'instructions, il faut écrire une instruction par ligne et décaler les lignes dans une boucle, un test, une fonction, une classe. Pour écrire une instruction sur plusieurs lignes, il faut utiliser le caractères \ sur toutes les lignes de l'instruction sauf la dernière. Il ne doit rien y avoir derrière ce caractère, ni espace, ni commentaires.
2. On peut mettre autant d'espaces qu'on veut. Il peut y en avoir aucun sauf derrière une instruction `for`, `if`, ...
3. Les commentaires dans un programme commencent par le symbole # et vont jusqu'à la fin de la ligne.
4. L'instruction `print` permet d'afficher n'importe quelle information. Elle affiche le résultat de la méthode `__str__` pour les classes.
5. L'instruction `help` affiche l'aide associée à une variable, une fonction, une classe, une méthode, un module. Pour une fonction, une classe, une méthode du programme, cette aide correspond à une chaîne de caractères encadrée par trois ". Ce message d'aide peut s'étaler sur plusieurs lignes.

```
def fonction () :
     """fonction de
     démonstration"""
     return 0
help (fonction)    # affiche fonction de
                   # démonstration
```

**En cas de doute sur une partie de code, un calcul, une priorité entre opérateurs, le résultat d'une fonction, il est possible d'utiliser la fonction `print` pour afficher une valeur intermédiaire pendant un calcul. Il ne faut pas non plus hésiter à vérifier sur un petit exemple dans un petit programme que des lignes douteuses font exactement ce pour quoi elles ont été écrites.**

Il est souvent utile de chercher sur Internet des exemples de programmes pour corriger une syntaxe incorrecte, utiliser Google en ajoutant une requête commençant par le mot *Python*. Pour les erreurs, il est parfois intéressant de recopier intégralement le message d'erreur sous Google, les réponses obtenues sont souvent assez claires.

## 14.2 Les variables

Le nom d'une variable commence par une lettre ou un blanc souligné, il peut également inclure par la suite des chiffres. *Python* distingue les minuscules des majuscules. La portée d'une variable, c'est-à-dire la portion de code où elle définie, s'étend depuis sa déclaration (première affectation) jusqu'à la fin du programme ou de la fonction ou de la boucle ou du test où elle est définie. Pour déclarer une variable portant le nom `va`, il suffit d'écrire :

```
va = <valeur>
```

Le type de < valeur > détermine le type de la variable va. Si une variable de même portée portait déjà ce nom-là, son contenu est écrasé (perdu aussi). L'instruction type(x) retourne le type de la variable x. **Un identificateur ne peut être utilisé qu'une seule fois, qu'il désigne une variable, une fonction, une classe ou un module.**

### 14.2.1 Les types immuables

Les variables de type immuable ne peuvent pas être modifiées.

1. None, ce type veut dire rien, il est utilisé comme convention de programmation pour dire qu'un algorithme, un calcul ne s'est pas terminé correctement ou une valeur n'a pas encore été calculée.
2. bool : un booléen (résultat d'un test)
3. int : un entier
4. float : un réel
5. complex : un complexe
6. str : une chaîne de caractères ou string, elle apparaît entre guillemets, entre apostrophes, entre trois guillemets (""") si elle s'étend sur plusieurs lignes. s = "exemple".
7. tuple : un vecteur d'éléments de types identiques ou différents, il apparaît entre parenthèses, on accède à un de ses éléments à l'aide de crochets. Les éléments d'un t-uple t sont indicés de 0 à len(t) − 1 inclus.

```
t = ()  # tuple vide
t = (2, "e") # tuple de deux éléments
print t[0]   # affiche le premier éléments
```

L'affectation d'une valeur de type immuable à une variable est une copie. On peut appliquer sur les types numériques les opérations usuelles (+ * - / % ** += *= -= /= %= **=) [1]. On rappelle que a+ = 10 est équivalent à a = a + 10, ceci signifie que la valeur de a avant le calcul n'a plus besoin d'exister. Le *et* logique et le *ou* logique sont notés and et or. Les priorités sont celles usuellement utilisées en mathématique, en cas de doute, il faut utiliser des parenthèses.

Les opérateurs de comparaison (< > == <= >=) s'appliquent sur tous les types numériques ainsi que sur les chaînes de caractères. Rappel : les minuscules sont classées après les majuscules.

**Fréquente source de bug : une division entière a pour résultat le quotient et non un nombre décimal. Autrement dit : 1/2 = 0 et non 0.5.**

Pour convertir une information d'un type à un autre, il suffit d'utiliser le nom de ce type suivi de la valeur à convertir entre parenthèses : b = float("2.145") équivaut à la conversion d'une chaîne de caractères en réel.

L'addition d'un t-uple et d'une valeur retourne un t-uple incluant cette valeur à la fin (plus long d'un élément). L'addition de deux t-uples concatène les deux t-uples. L'addition de deux chaînes de caractères retourne leur concaténation.

Pour savoir si un élément x fait partie d'un t-uple t, il faut utiliser la syntaxe x in t dont la réciproque est x not in t.

La fonction len retourne la longueur d'un tuple ou d'une chaîne de caractères. Les éléments ou les caractères d'un tuple ou d'une chaîne de caractères t sont indicés de 0 à len(t) − 1 inclus.

1. ** est le symbole pour puissance : $3 * *4 = 3^4$

```
t [i:j]  # correspond à un sous-ensemble allant des indices i à j exclu
t [:j]   #  = t[0:j]
t [i:]   # = t [i: len (t)]
```

Pour les chaînes de caractères, on utilise fréquemment les méthodes de la table 14.1, exemple :

```
st = "langage python"
st = st.upper ()                                                    # mise en lettres majuscules
i  = st.find ("PYTHON")                                  # on cherche "PYTHON" dans st
print i                                                                                          # affiche 8
print st.count ("PYTHON")                # affiche 1
print st.count ("PYTHON", 9)         # affiche 0
```

| | |
|---|---|
| count(sub[, start[, end]]) | Retourne le nombre d'occurences de la chaîne de caractères sub, les paramètres par défaut start et end permettent de réduire la recherche entre les caractères d'indice start et end exclu. Par défaut, start est nul tandis que end correspond à la fin de la chaîne de caractères. |
| find(sub[, start[, end]]) | Recherche une chaîne de caractères sub, les paramètres par défaut start et end ont la même signification que ceux de la fonction count. Cette fonction retourne -1 si la recherche n'a pas abouti. |
| isalpha() | Retourne True si tous les caractères sont des lettres, False sinon. |
| isdigit() | Retourne True si tous les caractères sont des chiffres, False sinon. |
| replace(old, new[, count]) | Retourne une copie de la chaîne de caractères en remplaçant toutes les occurrences de la chaîne sub par new. Si le paramètre optionnel count est renseigné, alors seules les count premières occurrences seront remplacées. |
| split([sep[, maxsplit]]) | Découpe la chaîne de caractères en se servant de la chaîne split comme délimiteur. Si le paramètre maxsplit est renseigné, au plus maxsplit coupures seront effectuées. |
| upper() | Remplace les minuscules par des majuscules. |
| lower() | Remplace les majuscules par des minuscules. |

**Table 14.1 :** *Quelques fonctions s'appliquant aux chaînes de caractères, l'aide associée au langage Python fournira la liste complète. Certains des paramètres sont encadrés par des crochets, ceci signifie qu'ils sont facultatifs (voir également page 35).*

L'affichage de réels nécessite parfois de tronquer la partie décimale ce qui est fait grâce à la syntaxe suivante (voir également page 35) :

```
x = 0.123456789
print "%1.2f" % x    # donne 0.12
s = "%2.2e %s" % (3.14, "est une approximation de pi")
print s
```

### 14.2.2 Les types modifiables

*Python* fournit deux types modifiables : les listes et les dictionnaires. Pour ces deux types, **il faut faire attention à chaque affectation**.

```
a = [1,2]
b = a
```

La seconde ligne ne fait pas une copie de la première liste, elle ne fait que créer un second nom pour nommer la même liste. Pour copier une liste ou un dictionnaire, il faut utiliser :

```
a = [1,2]
import copy
b = copy.copy (a)
```

ou, si la liste inclut également d'autres listes ou dictionnaires :

```
a = [1,2]
import copy
b = copy.deepcopy (a)
```

Cette remarque s'applique à tout type modifiable, liste, dictionnaire ou tout autre classe. La suppression d'une variable n'implique pas la suppression de toutes les variables se référant à une seule et même instance de classe.

#### 14.2.2.1 Liste

Une liste est une sorte de tableau qui permet de mémoriser un ensemble d'éléments de types variés. C'est une sorte de t-uple modifiable. La table 14.2 regroupe les opérations qu'une liste supporte et la table 14.3 les méthodes dont elle dispose.

```
x = [4,5]               # création d'une liste composée de deux entiers
x = ["un",1,"deux",2]   # création d'une liste composée deux chaînes de caractères
                        # et de deux entiers, l'ordre d'écriture est important
x = [3,]                # création d'une liste d'un élément, sans la virgule,
                        # le résultat reste une liste
x = [ ]                 # crée une liste vide
x = list ()             # crée une liste vide
```

Les listes peuvent aussi être définies à partir d'une écriture abrégée :

```
x = range(0,5)                      # liste des entiers de 0 à 5 exclu
y = [ i for i in x if i % 2 == 0]   # sélection des éléments pairs
print y                             # affiche [0,2,4]
z = [ i+j for i in x for j in x]    # construit tous les nombres i+j possibles
print z                             # affiche [0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 2, 3,
                                    # 4, 5, 6, 3, 4, 5, 6, 7, 4, 5, 6, 7, 8]
```

#### 14.2.2.2 Dictionnaire

Un dictionnaire est un tableau pour lequel les indices ou clés ne sont pas uniquement des entiers mais tout type non modifiable (le plus souvent un entier, un réel, une chaîne de caractères, un t-uple).

```
x = { "cle1":"valeur1", "cle2":"valeur2" }
print x ["cle1"]   # affiche valeur1
x [(0,1)]  = "clé tuple"  # ajoute une nouvelle valeur dont la clé est (0,1)
y = { }         # crée un dictionnaire vide
z = dict ()     # crée aussi un dictionnaire vide
```

La table 14.4 regroupe les opérations qu'un dictionnaire supporte. La table 14.5 regroupe les méthodes d'un dictionnaire.

| | |
|---|---|
| `x in s` | vrai si `x` est un des éléments de `l` |
| `x not in s` | réciproque de la ligne précédente |
| `l + t` | concaténation de `l` et `t` |
| `l * n` | concatène `n` copies de `l` les unes à la suite des autres |
| `l[i]` | retourne le $i^{ème}$ élément de `l`, à la différence des T-uples, l'instruction `l[i]` = "3" est valide, elle remplace l'élément `i` par 3. |
| `l[i : j]` | retourne une liste contenant les éléments de `l` d'indices $i$ à $j$ exclu. Il est possible de remplacer cette sous-liste par une autre en utilisant l'affectation `l[i : j]` = `l2` où `l2` est une autre liste (ou un T-uple) de dimension différente ou égale. |
| `l[i : j : k]` | retourne une liste contenant les éléments de `l` dont les indices sont compris entre $i$ et $j$ exclu, ces indices sont espacés de $k : i, i+k, i+2k, i+3k, ...$ Ici encore, il est possible d'écrire l'affectation suivante : `l[i : j : k]` = `l2` mais `l2` doit être une liste (ou un T-uple) de même dimension que `l[i : j : k]`. |
| `len(l)` | nombre d'éléments de `l` |
| `min(l)` | plus petit élément de `l`, résultat difficile à prévoir lorsque les types des éléments sont différents |
| `max(l)` | plus grand élément de `l`, résultat difficile à prévoir lorsque les types des éléments sont différents |
| `sum(l)` | retourne la somme de tous les éléments |
| `del l[i : j]` | supprime les éléments d'indices entre `i` et `j` exclu. Cette instruction est équivalente à `l[i : j]` = []. |
| `list(x)` | convertit `x` en une liste quand cela est possible |

**Table 14.2 :** *Opérations disponibles sur les listes, identiques à celles des T-uples, on suppose que* `l` *et* `t` *sont des listes,* `i` *et* `j` *sont des entiers.* `x` *est quant à lui quelconque (voir également page 39).*

#### 14.2.2.3 Tableaux

Ce type ne fait pas partie du langage *Python* standard mais il est couramment utilisé.

```
import Numpy
a = Numpy.array ( [0,1] )
```

Il permet de convertir des listes en une structure plus appropriée au calcul. En contrepartie, il n'est pas aussi rapide d'ajouter ou supprimer des éléments.

## 14.3 Tests et boucles

### 14.3.1 Tests

Les tests permettent d'exécuter telle ou telle instruction selon la valeur d'une condition. Le test est suivi de : et les instructions dépendant de ce test sont indentées (décalées vers la droite).

```
if x < 5 :
    print x
    ...
```

Il peut y avoir une contrepartie :

| | |
|---|---|
| `l.count(x)` | Retourne le nombre d'occurrences de l'élément `x`. Cette notation suit la syntaxe des classes développées au chapitre 5. `count` est une méthode de la classe `list`. |
| `l.index(x)` | Retourne l'indice de la première occurrence de l'élément `x` dans la liste `l`. Si celle-ci n'existe, une exception est déclenchée (voir le paragraphe 6). |
| `l.append(x)` | Ajoute l'élément `x` à la fin de la liste `l`. Si `x` est une liste, cette fonction ajoute la liste `x` en tant qu'élément, au final, la liste `l` ne contiendra qu'un élément de plus. |
| `l.extend(k)` | Ajoute tous les éléments de la liste `k` à la liste `l`. La liste `l` aura autant d'éléments supplémentaires qu'il y en a dans la liste `k`. |
| `l.insert(i,x)` | Insère l'élément `x` à la position `i` dans la liste `l`. |
| `l.remove(x)` | Supprime la première occurence de l'élément `x` dans la liste `l`. S'il n'y a aucune occurrence de `x`, cette méthode déclenche une exception. |
| `l.pop([i])` | Retourne l'élément `l[i]` et le supprime de la liste. Le paramètre `i` est facultatif, s'il n'est pas précisé, c'est le dernier élément dont la valeur est d'abord retournée puis il est supprimé de la liste. |
| `l.reverse(x)` | Retourne la liste, le premier et dernier élément échangent leurs places, le second et l'avant dernier, et ainsi de suite. |
| `l.sort([f])` | Cette fonction trie la liste par ordre croissant. Le paramètre `f` est facultatif, il permet de préciser la fonction de comparaison qui doit être utilisée lors du tri. Cette fonction prend comme paramètre deux éléments `x` et `y` de la liste et retourne les valeurs -1,0,1 selon que `x` < `y`, `x` == `y` ou `x` > `y` (voir paragraphe 4.4). |

**Table 14.3 :** *Opérations permettant de modifier une liste on suppose que* `l` *est une liste,* `x` *est quant à lui quelconque (voir également page 40).*

```
if x < 5 :
    print x
    ...
else :
    print 5-x
    ...
```

S'il n'y a qu'une seule instruction, elle peut s'écrire en bout de ligne :

```
if x < 5 :  print x
else :    print 5-x
```

Il peut y avoir plusieurs conditions qui s'enchaînent :

```
if x < 5 :  print "x < 5"
elif x > 5 : print "x > 5"
else :  print "x == 5"
```

### 14.3.2 Boucles

Il y a deux types de boucles, la boucle `for` parcourt un ensemble, la boucle `while` continue tant qu'une condition est vraie. Comme pour les tests, une boucle est suivie du syumbol :, les lignes incluses dans cette boucle sont indentées à moins qu'il n'y en ait qu'une seule, auquel cas elle peut être écrite après le symbole : sur la même ligne.

| | |
|---|---|
| `x in d` | vrai si `x` est une des clés de `d` |
| `x not in d` | réciproque de la ligne précédente |
| `l[i]` | retourne l'élément associé à la clé `i` |
| `len(d)` | nombre d'éléments de `d` |
| `min(d)` | plus petite clé |
| `max(d)` | plus grande clé |
| `del l[i]` | supprime l'élément associé à la clé `i` |
| `list(d)` | retourne une liste contenant toutes les clés du dictionnaire `d`. |
| `dict(x)` | convertit `x` en un dictionnaire si cela est possible, en particulier, `d` est égal à `dict(d.items())` |

**Table 14.4 :** *Opérations disponibles sur les dictionnaires,* `d` *est un dictionnaire,* `x` *est quant à lui quelconque (voir également page 45).*

| | |
|---|---|
| `d.copy()` | Retourne une copie de `d`. |
| `d.has_key(x)` | Retourne `True` si `x` est une clé de `d`. |
| `d.items()` | Retourne une liste contenant tous les couples (clé, valeur) inclus dans le dictionnaire. |
| `d.keys()` | Retourne une liste contenant toutes les clés du dictionnaire `d`. |
| `d.values()` | Retourne une liste contenant toutes les valeurs du dictionnaire `d`. |
| `d.get(k[,x])` | Retourne `d[k]`, si la clé `k` est manquante, alors la valeur `None` est retournée à moins que le paramètre optionnel `x` soit renseigné, auquel cas, ce sera ce paramètre qui sera retourné. |
| `d.clear()` | Supprime tous les éléments du dictionnaire. |
| `d.update(d2)` | Pour chaque clé de `d1`, `d[k] = d1[k]` |
| `d.setdefault(k[,x])` | Retourne `d[k]` si la clé `k` existe, sinon, affecte `x` à `d[k]`. |
| `d.popitem()` | Retourne un élément et le supprime du dictionnaire. |

**Table 14.5 :** *Méthodes associées aux dictionnaires,* `d`, `d2` *sont des dictionnaires,* `x` *est quant à lui quelconque (voir également page 45).*

```
while condition :
    # lignes décalées
    # contenu de la boucle
```

Quelques exemples de boucles `for` :

```
for i in range(0,n) :             # parcourt tous les entiers de 0 à n-1 inclus
for i in xrange(0,n) :            # même chose mais en plus rapide
for i in xrange(n,0,-1) :         # parcourt tous les entiers de n à 1 inclus
                                  #                  dans le sens décroissant
for i in xrange(2,1000,3) :       # parcourt tous les entiers de 2 à 1000 de 3 en 3
                                  #                                    (2,5,8,...)
for e in li :                     # parcourt tous les éléments de la liste li
for cle,valeur in di.items () :   # parcourt tous les éléments du dictionnaire di
```

Pour toutes les boucles, l'instruction `break` permet de sortir de la boucle, l'instruction `continue` passe directement à l'itération suivante sans exécuter les instructions qui suivent l'instruction `continue`.

## 14.4 Fonctions

Les fonctions ou sous-programmes permettent de faire la même chose sans avoir à recopier le code informatique plusieurs fois dans le programme. Elle accepte aucun ou plusieurs paramètres, elle peut retourner aucun ou plusieurs résultats. Leur déclaration suit le schéma suivant :

```
def exemple_fonction (p1, p2, p3) :
    # code de la fonction
    return r1, r2

a,b = exemple_fonction (1,2,3)   # exemple d'appel de la fonction
```

L'instruction `return` n'est pas obligatoire mais si elle est présente à un ou plusieurs endroits, aucune autre instruction de la fonction ne sera exécutée après l'exécution de la première instruction `return` rencontrée lors de l'exécution de la fonction. Les fonctions peuvent être récursives et inclure des paramètres par défaut : ces paramètres reçoivent une valeur même si celle-ci n'est pas précisée lors de l'appel.

```
def exemple_fonction (p1, p2 = 4, p3 = 7) :
    # code de la fonction
    return r1, r2

a,b = exemple_fonction (1)   # = exemple_fonction (1,4,7)
a,b = exemple_fonction (1,2,3)   # = exemple_fonction (1,2,3)
a,b = exemple_fonction (1,2)   # = exemple_fonction (1,2,7)
a,b = exemple_fonction (1,p3 = 2)   # = exemple_fonction (1,4,2)
```

Les paramètres par défaut doivent tous être mis en fin de déclaration, l'exemple suivant n'est pas correct :

```
def exemple_fonction (p1, p2 = 4, p3) :
    # code de la fonction
    return r1, r2
```

En ce qui concerne les paramètres, les paramètres de type non modifiable sont passés par valeur (une modification à l'intérieur de la fonction n'a pas de répercution à l'extérieur).

```
def exemple_fonction (p1) :
    p1 = 3
a = 1
exemple_fonction (a)
print a  # affiche 1
```

Les paramètres de type modifiable sont passés par référence (une modification à l'intérieur de la fonction a des répercutions à l'extérieur).

```
def exemple_fonction (p1) :
    p1[0] = 3
a = [1]
%
exemple_fonction (a)
print a  # affiche [3]
```

## 14.5 Classes

Les classes sont un moyen de définir de nouveaux types modifiables de variables. Peu de programmes ne les utilisent pas. Une classe est un ensemble d'attributs (ou variables) et de méthodes (ou fonctions).

Un programme utilisant les classes est orienté objet. Il est possible de faire les mêmes choses avec ou sans classes mais leur utilisation rend d'ordinaire les grands programmes plus facile à comprendre et à construire.

### 14.5.1 Déclaration d'une classe

Pour déclarer une classe, on procède comme suit :

```
class ma_classe :
    def __init__ (self, att1, att2, att3) :
        self.att1 = att1
        self.att2 = att2
        self.att3 = att3
        self.att4 = att1 * att2 * att3

a = ma_classe (-1,1,2) # déclare une variable de type ma_classe
print a.att1  # affiche -1
print a.att2  # affiche 3
print a.att3  # affiche 4
print a.att4  # affiche -12
```

Lors de la déclaration de la variable `a`, le langage *Python* exécute la méthode `__init__` aussi appelée constructeur. Elle permet de définir les attributs de la classe directement à partir des paramètres ou comme le résultat d'un calcul ou d'une fonction. Le constructeur comme toutes les autres méthodes possède comme premier paramètre `self` qui permet d'accéder aux attributs et aux méthodes de la classe. Le programme suivant est équivalent au premier.

```
class ma_classe :
    def __init__ (self, att1, att2, att3) :
        self.att1 = att1
        self.att2 = att2
        self.att3 = att3
        self.att4 = self.calcule4 ()

    def calcule4 (self) :
        return self.att1 * self.att2 * self.att3

a = ma_classe (-1,1,2) # déclare une variable de type ma_classe
print a.att1           # affiche -1
print a.att2           # affiche 3
print a.att3           # affiche 4
print a.att4           # affiche -12
```

### 14.5.2 Attributs et méthodes

Les attributs sont déclarés le plus souvent à l'intérieur du constructeur, plus généralement à l'intérieur de toute méthode, voire à l'extérieure de la classe. Pour y faire référence à l'intérieur d'une méthode on fait précéder le nom de l'attribut de `self`., à l'extérieur de la classe, c'est le nom de l'instance suivi d'un point . qui précède le nom de l'attribut comme le montre le précédent exemple.

Une méthode est déclarée à l'intérieur de la classe. Elle accepte invariablement au moins un paramètre qui est `self` comme dans le précédent exemple. Les règles d'accès sont les mêmes que pour les attributs. Elles acceptent également la récursivité et les paramètres par défaut à l'exception du premier. Chaque instance de classe est également munie d'un dictionnaire `__dict__` qui recense tous les attributs.

```
class ma_classe :
    def __init__ (self, att1, att2, att3) :
```

```
        self.att1 = att1
        self.att2 = att2
        self.att3 = att3
        self.att4 = att1 * att2 * att3

a = ma_classe (1,2,3)
print a.att1                          # affiche 1
print a.__dict__ ["att1"]    # affiche aussi 1, ligne équivalente à la précédente
```

### 14.5.3 Opérateurs

Les opérateurs sont des méthodes qui permettent une manipulation plus simple des objets. Leur nom est fixé par convention par le langage *Python*, ils commencent et terminent par `__`.

```
class ma_classe :
    def __init__ (self, att1, att2, att3) :
        self.att1 = att1
        self.att2 = att2
        self.att3 = att3
        self.att4 = att1 * att2 * att3

    def __add__ (self, a) :
         return ma_classe (self.att1 + a.att1, self.att2 + a.att2, \
                       self.att3 + a.att3, self.att4 + a.att4)

a = ma_classe (1,2,3)
b = ma_classe (4,5,6)
c = a + b              # n'a de sens que si l'opérateur __add__ a été redéfini
```

Il existe un opérateur spécifique pour chaque opération, cet opérateur permet de donner un sens à une addition, une soustraction, ..., de deux instances d'une classe. L'opérateur `__str__` retourne une chaîne de caractères et est appelé par l'instruction `print`. L'opérateur `__cmp__` retourne un entier permettant à des instances de la classe d'être comparées et triées par une liste.

### 14.5.4 Copie d'instances

Les instances de classes sont des objets modifiables, comme pour les listes, une simple affectation ne signifie pas une copie mais un second nom pour désigner le même objet.

```
class ma_classe :
    def __init__ (self, att1, att2, att3) :
        self.att1 = att1
        self.att2 = att2
        self.att3 = att3
        self.att4 = att1 * att2 * att3

a = ma_classe (1,2,3)
b = a
b.att1 = -16
print a.att1  # affiche -16
print b.att1  # affiche -16
```

Il faut donc copier explicitement l'instance pour obtenir le résultat souhaité.

```
class ma_classe :
    def __init__ (self, att1, att2, att3) :
        self.att1 = att1
```

```
        self.att2 = att2
        self.att3 = att3
        self.att4 = att1 * att2 * att3

a = ma_classe (1,2,3)
import copy
b = copy.copy (a)
b.att1 = -16
print a.att1  # affiche 1
print b.att1  # affiche -16
```

Lorsque une classe inclut une variable de type classe, il faut utiliser la fonction `deepcopy` et non `copy`.

### 14.5.5 Héritage

L'héritage est l'intérêt majeur des classes et de la programmation orientée objet. Lorsqu'une classe hérite d'une autre, elle hérite de ses attributs et de ses méthodes. Le simple fait d'hériter crée donc une classe équivalente.

```
class ma_classe :
    def __init__ (self, att1, att2, att3) :
        self.att1 = att1
        self.att2 = att2
        self.att3 = att3
        self.att4 = att1 * att2 * att3

class ma_classe2 (ma_classe) :      # héritage simple
    pass                            # pour dire que la classe est vide
```

Mais hériter permet de faire deux choses :

1. ajouter des attributs et ajouter des méthodes
2. modifier le comportement d'une méthode existante

```
class ma_classe :
    def __init__ (self, att1) :
        self.att1 = att1
        self.att2 = self.calcul ()

    def calcul (self) :
        return self.att1 ** 2

class ma_classe2 (ma_classe) :
    def calcul (self) :
        # dans cette méthode, on change le comportement
        # de la méthode calcul tout en se servant de celui
        # de la classe mère
        return ma_classe.calcul (self) * self.att1

a = ma_classe (2)
b = ma_classe2 (2)
print a.att2   # affiche 4  = 2 * 2
print b.att2   # affiche 8   = (2*2) * 2
```

## 14.6 Fichiers

L'écriture et la lecture dans un fichier s'effectuent toujours de la même manière. On ouvre le fichier en mode écriture ou lecture, on écrit ou on lit, puis on ferme le fichier, le laissant disponible pour une utilisation

ultérieure. Ce paragraphe ne présente pas l'écriture ou la lecture dans un format binaire car celle-ci est peu utilisée dans ce langage.

### 14.6.1 Ecriture dans un fichier texte

L'écriture dans un fichier texte s'effectue toujours selon le même schéma :

```
f = open ("nom-fichier", "w") # ouverture en mode écriture "w" ou écriture ajout "a"

f.write (  s )                # écriture de la chaîne de caractères  s
f.write (  s2 )               # écriture de la chaîne de caractères  s2
...

f.close ()  # fermeture
```

Certains codes sont fort utiles lors de l'écriture de fichiers texte :

- `\n` passage à la ligne
- `\t` insertion d'une tabulation, indique un passage à la colonne suivante dans le logiciel Excel

### 14.6.2 Lecture dans un fichier texte

La lecture est le symétrique de l'écriture. En voici un exemple, la seule chose qui change d'un programme à l'autre est ce qu'on fait des lignes extraites.

```
f = open ("essai.txt", "r")   # ouverture du fichier en mode lecture
l = f.readlines ()            # lecture de toutes les lignes,
                              #   elles sont placées dans une liste
f.close ()                    # fermeture du fichier

for s in l : print s          # on affiche les lignes à l'écran
```

**Remarque 14.1 : code de fin de ligne**
Lors le programme précédent lit une ligne dans un fichier, le résultat lu inclut le ou les caractères (`\n` `\r` qui marquent la fin d'une ligne. C'est pour cela que la lecture est parfois suivie d'une étape de nettoyage.

```
f = open ("essai.txt", "r")     # ouverture du fichier en mode lecture
l = f.readlines ()              # lecture de toutes les lignes,
                                #   elles sont placées dans une liste placées dans une liste
f.close ()                      # fermeture du fichier

l_net = []                      # contiendra la liste nettoyée des lignes du fichier
for s in l :
    s2 = s.replace ("\n", "")   # on supprime le code de fin de ligne \n
    s2 = s2.replace ("\r", "")  # on supprime le code de fin de ligne \r
                                #                     (Windows uniquement)
    l_net.append (s2)           # on ajoute le résultat à la liste nettoyée
```

**Remarque 14.2 : fichier formaté**
Les fichiers textes ont de nombreux formats, on peut citer HTML ou XML qui sont des formats à balises. Leur lecture utilise des modules comme `HTMLParser` ou `xml.sax` dont la description sort du cadre de ce document. Un autre format est souvent utilisé avec le logiciel *Excel* ou tout autre tableur. Lorsqu'on enregistre une feuille de calcul sous format texte, le fichier obtenu est organisé en colonnes : sur une même ligne, les informations sont disposées en colonnes délimitées par un séparateur qui est souvent une tabulation (`\t`) ou un point virgule.

```
nom  ; prénom ; livre
Hugo  ; Victor  ; Les misérables
Kessel ; Joseph  ; Le lion
Woolf ; Virginia  ; Mrs Dalloway
Calvino ; Italo  ; Le baron perché
```

Pour lire ce fichier, il est nécessaire de scinder chaque ligne en une liste de chaînes de caractères. On utilise pour cela la méthode `split` des chaînes de caractères.

```
f = open ("essai.txt", "r")     # ouverture du fichier en mode lecture
l = f.readlines ()              # lecture de toutes les lignes, placées dans une liste
f.close ()                      # fermeture du fichier

for s in l :
    s2 = s.replace ("\n", "")   # on supprime le code de fin de ligne \n
    s2 = s2.replace ("\r", "")  # on supprime le code de fin de ligne \r (Windows uniquement)
    case = s2.split (";")
    if len (case) >= 3 :
        print case [1], " ", case [0], " a écrit ", case [2]
```

## 14.7 Modules

Le concept de module permet de répartir différentes parties d'un programme sur plusieurs fichiers. Il existe deux types de modules : ceux disponibles sur Internet (programmés par d'autres) et ceux que l'on programme soi-même. Les premiers sont souvent fournis avec un programme d'installation automatique ou dans le cas où ils sont manquants, des instructions permettant de l'installer. Les seconds sont écrits dans le même répertoire que le fichier principal. On enregistre le module suivant sous le nom `geometrie.py`.

```
# définition du module geometrie.py

def carre (x) :
    return x ** 2

class point :
    def __init__ (self,x,y) :
        self.x, self.y = x,y

    def norme (self) :
        return (self.x ** 2 + self.y ** 2) ** 0.5
```

Pour utiliser une fonction ou une classe du module `geometrie.py`, on utilise une des syntaxes suivantes :

1. Première syntaxe :

   ```
   import geometrie
   print geometrie.carre (1.5)
   p = geometrie.point (1,2)
   ```

2. Deuxième syntaxe :

   ```
   import geometrie as GEO  # on donne un pseudonyme au module geometrie
   print GEO.carre (1.5)
   p = GEO.point (1,2)
   ```

3. Troisième syntaxe : le module est utilisé très souvent, même un pseudonyme est trop long, il faut néanmoins s'assurer que les modules importés de cette même manière n'incluent pas des fonctions ou classes portant des noms identiques. Dans ce cas, c'est toujours le dernier qui gagne.

```
from  geometrie import *
print carre (1.5)
p = point (1,2)
```

Dans le cas des modules installés, les trois syntaxes d'utilisation sont aussi valables. On voit aussi souvent apparaître dans un module la condition :

```
if __name__ == "__main__" :
    # quelques instructions ici
```

Ces instructions ne sont exécutées que si le module est utilisé en tant que programme principal. Lorsque ce fichier est importé, elles ne sont jamais exécutées. Cela permet d'écrire des instructions qui permettent de vérifier si le module ne contient pas d'erreurs. Une fois cette étape effectuée, il ne sert à rien de la répéter à chaque fois que le module est importé. C'est pourquoi elles ne sont exécutées que si la condition `if __name__ == "__main__" :` est vérifiée, c'est-à-dire si le module est le programme principal et non un module.

## 14.8 Exceptions

Le petit programme suivant déclenche une erreur parce qu'il effectue une division par zéro.

```
def inverse (x):
    y = 1.0 / x
    return y
b = inverse (0)
print b
```

Il déclenche une erreur ou ce qu'on appelle une *exception*.

```
Traceback (most recent call last):
  File "cours.py", line 2, in ?
    y = 1.0 / x
ZeroDivisionError: float division
```

Le mécanisme des exceptions permet au programme de "rattraper" les erreurs, de détecter qu'une erreur s'est produite et d'agir en conséquence afin que le programme ne s'arrête pas :

```
def inverse (x):
    y = 1.0 / x
    return y
try :
    b = inverse (0)  # déclenche une exception
    print b
except :
    print "le programme a déclenché une erreur"
```

On protège la partie du code à l'aide des mots-clés `try` et `except`. Entre ces deux instructions, s'il se produit une erreur, le programme passe immédiatement à ce qui suit l'instruction `except`. On peut même récupérer le message d'erreur correspondant :

```
def inverse (x):
    y = 1.0 / x
    return y
```

```
try :
    print inverse (2)
    print inverse (0)
except Exception, exc:
    print "exception de type ", exc.__class__
         # affiche exception de type  exceptions.ZeroDivisionError
    print "message ", exc
         # affiche le message associé à l'exception
```

On peut aussi décider que le programme agira différemment selon l'erreur produite. Dans l'exemple suivant, le programme teste d'abord si l'erreur est de type `ZeroDivisionError` auquel cas il affiche le message *division par zéro*. Pour un autre type d'erreur, il regarde s'il y a d'autres instructions `except` qui s'y rapportent. S'il y en a une, il exécute les lignes qui la suivent, sinon, le programme s'arrête et déclenche une erreur.

```
def inverse (x):
    y = 1.0 / x
    return y
try :
    print (-2.1) ** 3.1
    print inverse (2)
    print inverse (0)
except ZeroDivisionError:
    print "division par zéro"
except Exception, exc:
    print "erreur insoupçonnée : ", exc.__class__
    print "message ", exc
```

Les instructions `try` et `except` peuvent apparaître dans le programme principal, dans une boucle, un test, une fonction, s'imbriquer les unes dans les autres. Il est possible de déclencher soi-même une exception avec l'instruction `raise` et ou de définir ses propres exceptions en créant une classe héritant d'une classe d'exception. L'exemple suivant regroupe tous ces cas.

```
class AucunChiffre (Exception) :
    """chaîne de caractères contenant
    aussi autre chose que des chiffres"""

    def __init__(self, s, f = "") :
        Exception.__init__(self, s)
        self.s = s
        self.f = f

    def __str__(self) :
        return """exception AucunChiffre, lancée depuis la fonction """ + self.f + \
        " avec le paramètre " + self.s

def conversion (s) :
    """conversion d'une chaîne de caractères en entier"""
    if not s.isdigit () :
        raise AucunChiffre, (s, "conversion")
    return int (s)

try :
    s = "123a"
    i = conversion (s)
    print s, " = ", i
except AucunChiffre, exc :
    print AucunChiffre.__doc__, " : ", exc
    print "fonction : ", exc.f
```

## 14.9 Erreurs, confusions fréquentes

### 14.9.1 Variables

#### 14.9.1.1 Chaîne de caractères = tableau de caractères

Une chaîne de caractères est un tableau de caractères : pour accéder à un caractère, on procède comme pour une liste.

```
s = "abcdefghijklmnopqrstuvwxyz"
print s [4]    # affiche "e"
print s [4:6]  # affiche "ef"
```

#### 14.9.1.2 Guillemets ou pas

Doit-on mettre des guillemets ou non ?

```
l = [ un, deux, trois, quatre ]
up = []
for i in range (0, len (l)) :
    up.append ( l [i].upper () )
```

Le code précédent ne fonctionne pas car il n'y a pas de guillemets autour de `un`, `deux`, `trois`, `quatre`. Le langage considère alors ces quatre mots comme des variables : un identificateur qui désigne une information. Mais comme ces variables n'existent pas, ces identifiants ne sont reliés à aucun contenu et l'interpréteur *Python* ne comprend pas.

Un mot entouré de guillemets définit un contenu. Sans guillemet, il définit une variable qui permet de manipuler un contenu tout simplement en donnant la possibilité au programmeur de le nommer. Autrement dit, pour manipuler une chaîne de caractères, il faut affecter ce contenu à une variable. Les guillemets n'apparaissent plus par la suite car on doit utiliser la variable pour la manipuler.

### 14.9.2 Boucles

#### 14.9.2.1 `range` ou pas `range`

Les deux programmes suivant sont équivalents. La seule différence réside dans l'écriture dans la boucle `for` qui utilise dans le premier cas la fonction `range` et dans l'autre non.

```
l = [ "un", "deux", "trois", "quatre" ]
up = []
for i in range (0, len (l)) :
    up.append ( l [i].upper () )
```

Lorsqu'on utilise la fonction `range`, on dispose lors de la boucle de deux informations, l'indice `i` et l'élément `l[i]`. Si l'indice n'est pas utile, il est possible de simplifier la boucle comme suit.

```
l = [ "un", "deux", "trois", "quatre" ]
up = []
for m in l :
    up.append ( m.upper () )
```

En général, on se sert de la boucle qui utilise la fonction `range` dans deux cas :

1. On souhaite faire des opérations sur les éléments qui précèdent ou suivent l'élément en question, ce qui nécessite de connaître l'indice.
2. On parcourt deux listes de même taille à la fois : l'indice désigne la position de deux éléments, un dans chaque liste.

#### 14.9.2.2 Initialisation

Une boucle est souvent utilisée pour faire une somme, calculer un maximum : garder un unique résultat en parcourant une liste. Une boucle de ce type est toujours précédée d'une étape d'initialisation qui consiste à donner une valeur au résultat : celle qu'il aurait si la liste était vide.

```
l = [ "un", "deux", "trois", "quatre" ]
s = ""
for m in l :
    s += m # concaténation des mots en une seule chaîne de caractères
```

### 14.9.3 Fonctions

#### 14.9.3.1 Différence entre `print` et `return`

A la fin d'un calcul, afin de voir son résultat, on utilise souvent l'instruction `print`. On peut se demander alors si à la fin de chaque fonction, il ne faudrait pas utiliser l'instruction `print`. A quoi servirait alors l'instruction `return` ? On suppose qu'un calcul est en fait le résultat de trois calculs à la suite :

```
a = calcul1 (3)
b = calcul2 (a)
c = calcul3 (b) # c résultat souhaité et affiché
```

Chaque terme `calculx` cache une fonction or seul le résultat de la dernière nous intéresse et doit être affiché. Pour les deux premières, la seule chose importante est que leur résultat soit transmis à la fonction suivante et ceci ne peut se faire que grâce à l'instruction `return`. L'instruction `print` insérée dans le code de la fonction `calcul1` ou `calcul2` permettra d'afficher le résultat mais ne le transmettra pas : l'instruction `return` est donc indispensable, `print` facultative.

En revanche, dans la dernière fonction `calcul3`, il est possible de se passer de `return` et de se contenter uniquement d'un `print`. Cependant, il est conseillé d'utiliser quand même `return` au cas où le résultat de la fonction `calcul3` serait utilisé par une autre fonction, `calcul4` par exemple.

# Chapitre 15

# Outils d'aide au développement

---

Ce chapitre présente différents outils qui accompagnent la création d'une application dans quelque langage de programmation que ce soit. Les premiers outils présentés sont des modules du langage *Python* qui permettent de fiabiliser un programme malgré de nombreuses modifications ou de mesurer le temps passé dans chaque fonction.

Les autres outils sont des applications facilitant le développement etle partage d'information. Le premier, *Tortoise SVN*, est un logiciel de suivi de version : il permet de conserver l'historique de vos modifications. Le second, *HTML Help Workshop*, permet de construire des fichiers d'aide au format *chm*. Le troisième, *InnoSetup*, permet de construire un installateur, un programme qui installera votre application sur une autre machine. Le dernier outil, *MoinMoin* est ce qu'on appelle un *Wiki*, il permet facilement de partager l'information au sein d'une entreprise en offrant à chacun la possibilité de modifier des pages internet et d'y effectuer des recherches.

**Remarque 15.1 : instructions d'installation et de configuration**
Les instructions qui permettent d'installer et de configurer les applications *Apache*, *SVN*, *MoinMoin*, ... sont extraites des sites internet de chaque applications. Elles sont valides pour les numéros de version cités dans ce document mais il est possible qu'elles évoluent dans le futur. Pour des versions plus récentes, il est probable que ces installations se modifient [1] .

## 15.1 Modules d'aide au développement

L'intérêt de la programmation réside rarement dans le programme construit mais plutôt dans ce qu'il permet de faire. Les outils qui suivent permettent de gagner du temps pas forcément lors de la conception mais plutôt lors de la finalisation d'une application [2]

### 15.1.1 Temps d'exécution

Le coût d'un algorithme définit sa complexité en fonction de la taille des données qu'il manipule mais cette information est rarement disponible. Lorsqu'un programme est trop long, avant de pouvoir le modifier, il faut savoir quels sont les éléments consommateurs de temps, c'est ce que permettent de faire les modules `profile` et `pstats`. Le code suivant a été ajouté au programme `genchm.py` définit au paragraphe 15.3.2, page 346.

```
def run () :
    files = [".\\genchm.py", ".\\genhelp.py", "os", "sys"]
    res   = genhelp.genhelp (files)
    print res  # ['genchm.html', 'genhelp.html', 'os.html', 'sys.html']
    genchm (res, "GenHelp")

if __name__ == "__main__" :
    # écrit des informations relatives à l'exécution de la fonction
```

1. Les versions testées lors de l'écriture de ce document sont *Apache 2.2.8*, *SVN 1.4.6*, *MoinMoin 1.6.3*, *InnoSetup 5.2.3*, *Tortoise SVN 1.4.8*, *HTML Help Workshop 4.74*, *MySQL 5.0.51*.
2. Certains site comme celui-ci http://sebsauvage.net/python/snyppets/index.html sont plus utiles lors de la conception, ils recensent de nombreux usages du langages toujours illustrés d'un exemple.

```
    # run () dans le fichier profile.txt
    import profile
    profile.run ('run()', "profile.txt")

    # récupère les informations stockées dans profile.txt
    # et construit le nombre de passages dans chaque
    # fonction et les temps d'exécution moyen
    import pstats
    p = pstats.Stats('profile.txt')
    p.strip_dirs().sort_stats(-1).print_stats()
```

Le résultat est le suivant :

```
Mon May 12 22:38:02 2008    profile.txt

         77 function calls in 5.557 CPU seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        5    0.000    0.000    0.000    0.000 :0(append)
        4    0.000    0.000    0.000    0.000 :0(close)
        2    0.000    0.000    0.000    0.000 :0(len)
        4    0.003    0.001    0.003    0.001 :0(open)
       16    0.000    0.000    0.000    0.000 :0(replace)
        1    0.001    0.001    0.001    0.001 :0(setprofile)
        2    0.000    0.000    0.000    0.000 :0(split)
        5    5.551    1.110    5.551    1.110 :0(system)
        4    0.000    0.000    0.000    0.000 :0(write)
        1    0.000    0.000    5.556    5.556 <string>:1(<module>)
        1    0.000    0.000    2.562    2.562 genchm.py:102(genchm)
        1    0.000    0.000    0.001    0.001 genchm.py:11(genhhp)
        1    0.000    0.000    5.556    5.556 genchm.py:113(run)
        1    0.000    0.000    0.001    0.001 genchm.py:42(gen_premierepage)
       13    0.000    0.000    0.000    0.000 genchm.py:54(genhh_input)
        1    0.000    0.000    0.001    0.001 genchm.py:61(genhhc)
        8    0.000    0.000    0.000    0.000 genchm.py:7(g)
        1    0.000    0.000    0.001    0.001 genchm.py:83(genhhk)
        4    0.000    0.000    2.993    0.748 genhelp.py:17(generate_help_file)
        1    0.000    0.000    2.994    2.994 genhelp.py:26(genhelp)
        0    0.000             0.000          profile:0(profiler)
        1    0.000    0.000    5.557    5.557 profile:0(run())
```

Ces informations nous révèle que la fonction la plus souvent appelée `replace` ne consomme presque pas de temps d'exécution. La fonction `generate_help_file` dure près de trois quarts de seconde à chaque appel. La fonction `run` est la plus longue car elle cumule le temps passé dans toutes les autres.

### 15.1.2 Tests systématiques avec `doctest`

Lorsqu'il s'agit d'un programme court, il n'est pas nécessaire de bien réfléchir à la structure d'un programme ni à vérifier de temps en temps que ce qu'on a écrit auparavant continue de fonctionner malgré toutes les modifications apportées au programme. Car il existe certains pièges récurrents en informatique inhérents à toutes les applications conséquentes. Le schéma suivant retrace l'historique de la création de trois fonctions.

| | | |
|---|---|---|
| `fonction A` | écrite en janvier | |
| `fonction B` | écrite en janvier | utilise `fonction A` |
| `fonction C` | écrite en juin | utilise `fonction A` |

En juillet, on s'aperçoit qu'il faut modifier `fonction C`, cette modification implique la modification de `fonction A`. Quel est l'impact sur `fonction B` qui est une vieille fonction mais dont le comportement ne doit

absolument pas changer ? Ce cas de figure se produit fréquemment lors de la conception d'une application et il n'est pas toujours facile de prévoir les conséquences d'une modifications d'une fonction lorsque celle-ci est appelée d'un peu partout dans le code.

Une des solutions proposées par le langage *Python*est le module `doctest`. Il permet de simplifier le travail de vérification pour des fonctions qui retourne un résultat. Plus précisément, il est possible de spécifier une liste d'appels à une fonction et de comparer le résultat avec un résultat attendu ou souhaité.

L'exemple suivant utilise une fonction d'addition de deux listes. Elle retourne une liste de même taille contenant la somme membre à membre des deux listes. On souhaite vérifier que cette fonction retourne les bonnes valeurs pour certaines valeurs d'entrées connues. Dans ce cas, on souhaite s'assurer que la fonction retourne bien :

| l1 | l2 | résultat |
|---|---|---|
| [] | [] | [] |
| $[1, 2]$ | $[3, -1]$ | $[4, 1]$ |

La première étape est d'ajouter ces résultats dans le commentaire associé à la fonction. On fait précéder l'appel de >>> et on écrit le résultat sur la ligne suivante.

```
>>> addition ( [1,2], [3,-1])
[4, 1]
```

Il suffit ensuite d'ajouter une fonction appelant ces tests pour le module en question :

```
def _test():
    import doctest
    print doctest.testmod()
```

Si le test est validé, il ne se passe rien lorsqu'on appelle la fonction `_test`, sinon le message suivant apparaît :

```
File "testdoc1.py", line 6, in __main__.addition
Failed example:
    addition ( [1,2], [3,-1])
Expected:
    [4,  1]
Got:
    [4, 1]
**********************************************************************
1 items had failures:
   1 of   2 in __main__.addition
***Test Failed*** 1 failures.
```

Dans ce cas précis, une erreur se déclenche car un espace de trop a été ajouté au résultat attendu. Il suffit d'appeler régulièrement ou à chaque modification du programme la fonction `test` de tous les modules d'un programme pour s'assurer que les résultats sont toujours ceux attendus. On peut mettre plusieurs tests bout à bout, le programme complet donne :

```
# -*- coding: cp1252 -*-
def addition (l1, l2):
    """cette fonction additionne deux listes
    >>> addition ( [], [])
    []
    >>> addition ( [1,2], [3,-1])
    [4, 1]
    """
```

```
    res = []
    for i in range (0, len (l1)) :
        res.append ( l1 [i] + l2 [i] )
    return res

def _test():
    import doctest
    doctest.testmod()

if __name__ == "__main__":
    _test()
```

Il reste à gérer le cas où la fonction retourne une exception. Il suffit d'inclure le texte suivant dans le commentaires [3]. Les deux premières lignes sont identiques à chaque fois, la dernière est l'exception lancée par la fonction.

```
    Traceback (most recent call last):
        ...
    Exception: listes de tailles différentes
```

Le programme complet suit :

```
# -*- coding: cp1252 -*-
def addition (l1, l2):
    """cette fonction additionne deux listes
    >>> addition ( [], [])
    []
    >>> addition ( [1,2], [3,-1])
    [4, 1]
    >>> addition ( [1], [3,-1])
    Traceback (most recent call last):
        ...
    Exception: listes de tailles différentes
    """
    if len (l1) != len (l2) :
        raise Exception ("listes de tailles différentes")
    res = []
    for i in range (0, len (l1)) :
        res.append ( l1 [i] + l2 [i] )
    return res

def _test():
    import doctest
    doctest.testmod()

if __name__ == "__main__":
    _test()
```

Le module `doctest` reste cependant très limité puisqu'il ne permet que de tester le résultat d'une fonction et non un paramètre modifié comme une liste. C'est une manière de faire simple et vite pour des fonctions courtes et retournant des résultats principalement numériques. Ce module permet néanmoins d'introduire le concept de tests automatiques d'une application ou tests unitaires développés par le paragraphe suivant.

### 15.1.3 Tests unitaires avec `unittest`

Les tests unitaires sont avant tout un concept, ils permettent de fiabiliser une application en répétant les mêmes tests à chaque modification du programme et chaque fois qu'une application doit être transmise aux

3. Il faut faire attention aux espaces résiduels en fin de ligne, ceux-ci doivent être supprimés dans le commentaires

utilisateurs finaux [4] Le module `unittest` facilite la mise en place de ces tests mais ils pourraient tout-à-fait être implémentés sans l'utiliser.

L'objectif reste identique à celui visé par le module `doctest`, il est simplement plus ambitieux. Cette fois-ci, le code correspondant aux tests sera séparé du code de l'application elle-même. Les tests inclus dont de deux types :

1. vérifier que la fonction fait bien ce pour quoi elle a été créé y compris dans les cas limites, on vérifiera par exemple qu'une fonction de tri retourne bien un tableau trié, qu'une fonction retourne bien le bon code d'erreur dans un cas précis
2. vérifier que la correction apportée suite à la découverte d'un bug continue bien de corriger ce bug

Ce paragraphe n'abordera pas en détail toutes les possibilités du module `unittest`, ils ne sont pas indispensables au travail d'un ingénieur à moins que son travail n'évolue vers la création d'une application qui dépasse son propre usage ou d'une application destinée à durer. Le programme dont on souhaite vérifier la fiabilité est celui du paragraphe précédent :

```
# -*- coding: cp1252 -*-
def addition (l1, l2):
    """cette fonction additionne deux listes"""
    if len (l1) != len (l2) :
        raise Exception ("listes de tailles différentes")
    res = []
    for i in range (0, len (l1)) :
        res.append ( l1 [i] + l2 [i] )
    return res
```

On écrit maintenant un autre fichier contenant les trois tests implémentés avec le module `doctest`. Deux concernaient des résultats retournés par la fonction `addition` et le dernier un cas générant une exception. Pour définir un test, le principe est le suivant :

1. créer une classe héritant de `unittest.TestCase`
2. ajouter une méthode dont le nom commence par `test`
3. dans le corps de cette méthode, écrire le code du test, ce dernier est validé si une série de conditions est vérifiée, pour chaque condition, on ajoutera l'instruction `assert condition`

Le test est passé avec succès si les conditions précédées du mot-clé `assert` sont vraies. Si l'une d'elle est fausse, le test a échoué et signifie que le programme est buggé. Pour lancer les tests, il faut appeler la fonction `unittest.main()`. Cela donne le programme suivant :

```
# -*- coding: cp1252 -*-
import unittest
from testunit1 import *

class TestCase_for_addition (unittest.TestCase):

    def test_addition_vide (self) :
        """test_addition_vide : on vérifie que l'addtion de deux listes retourne une liste vide"""
        assert [] == addition ( [], [] )

    def test_addition (self) :
        """test_addition : test de [1,2] + [3,-1] != [4,1]"""
        l1 = [1,2]
        l2 = [3,-1]
        l  = addition (l1, l2)
        assert l [0] == 4 and l [1] == 1

    def test_exception (self) :
```

4. C'est ce qu'on appelle une mise en production.

```
        """test_exception : on vérifie que l'addition
        de deux listes de tailles différentes génère une exception"""
        l1 = [1]
        l2 = [3,-1]
        try :
            l  = addition (l1, l2)  # la fonction doit lancer une exception
            assert False            # si elle ne le fait pas, alors le test a achoué
        except Exception, e :
            # on vérifie que l'exception générée n'est pas due à l'instruction assert False
            assert str (e.__class__ .__name__) != "AssertionError"
            # on vérifie ici que le message de l'exception est celui attendu
            assert str (e) == "listes de tailles différentes"

if __name__ == "__main__" :
    # on lance les tests
    unittest.main ()
```

Après avoir modifiée la fonction `addition` de façon à faire échouer le second test `test_addition`, le message retourné par *Python* est :

```
======================================================================
ERROR: test_addition : test de [1,2] + [3,-1] != [4,1]
----------------------------------------------------------------------
Traceback (most recent call last):
  File "testunit2.py", line 16, in test_addition
    assert l [0] == 4 and l [1] == 1
IndexError: list index out of range
```

Les premières lignes reprennent le commentaires associées à la méthode de test. Les lignes suivantes permettent de comprendre où l'erreur est apparue et quelle condition a fait échouer le test. La fonction `addition` a égalément été intentionnellement modifiée pour ne pas générer d'exception lorsque les tailles de listes sont différentes. Le message suivant apparaît :

```
======================================================================
FAIL: on vérifie que l'addition de deux listes de tailles différentes génère une exception
----------------------------------------------------------------------
Traceback (most recent call last):
  File "testunit2.py", line 27, in test_exception
    assert str (e.__class__ .__name__) != "AssertionError"
AssertionError
```

Le module `unittest` indique également le temps passé à effectuer ces tests et le nombre de tests lancés. Lorsqu'il existe une multitude de tests, il est sans doute préférable de les répartir sur plusieurs fichiers à la fois. Il faut pouvoir les lancer depuis un seul fichier qui passera tous les tests unitaires en séries. C'est l'objectif du programme suivant qui récupère tous les fichiers `test`$_*$`.py` du répertoire, les importe puis crée des suites de tests unitaires pour les méthodes `test`... des classes intitulées `Test`....

```
# -*- coding: cp1252 -*-
import unittest
import os

def get_test_file () :
    """retourne la liste de tous les fichiers *.py commençant par test_"""
    li = os.listdir (".")
    li = [ l for l in li if "test_" in l and ".py" in l and \
                            ".pyc" not in l and ".pyd" not in l]
    return li

def import_files (li) :
```

```
    """pour un fichier test_*.py, cherche les classes Test...
    et crée une suite de test pour ses méthodes commençant
    par test..., retourne la suite de test"""
    allsuite = []
    for l in li :
        fi = l.replace (".py", "")
        mo = __import__ (fi)
        cl = dir (mo)
        for c in cl :
            if len (c) < 5 or c [:4] != "Test" : continue
            # classe de test c
            testsuite = unittest.TestSuite ()
            exec compile ("di = dir (mo." + c + ")", "", "exec")
            for d in di :
                if len (d) < 5 or d [:4] != "test" : continue
                # method d.c
                exec compile ("t = mo." + c + "(\"" + d + "\")", "", "exec")
                testsuite.addTest (t)
        allsuite.append ((testsuite, l))

    return allsuite

def main () :
    """crée puis lance les suites de textes définis dans
    des programmes test_*.py"""
    li      = get_test_file ()
    suite   = import_files (li)
    runner  = unittest.TextTestRunner()
    for s in suite :
        print "running test for ", s [1]
        runner.run (s [0])

if __name__ == "__main__" :
    main ()
```

### 15.1.4 Ecrire des fichiers de log

module `logging`

### 15.1.5 Lancer un programme *Python* en ligne de commande

Certains développements nécessitent de pouvoir exécuter un programme *Python* en ligne de commande sans intervention extérieure. Ce peut être un programme de synchronisation de deux répertoires ou la récupération du contenu de pages HTML, des tâches souvent exécutées la nuit à une heure où personne n'est présent pour le faire. Prenons par exemple le cas d'un programme très simple de synchronisation qui recopie tous les fichiers d'un répertoire vers un autre. La fonctionne `copie_repertoire` effectue cette tâche.

```
# -*- coding: cp1252 -*-
import glob
import shutil
def copie_repertoire (rep1, rep2) :
    """copie tous les fichiers d'un répertoire rep1 vers un autre rep2"""
    li = glob.glob (rep1 + "/*.*")
    for l in li :
        to = l.replace (rep1, rep2)  # nom du fichier copié (on remplace rep1 par rep2)
        shutil.copy (l, to)
copie_repertoire ("c:/original", "c:/backup")
```

Cette tâche est en plus exécuté sur deux répertoires et on ne voudrait pas avoir deux programmes différents alors que la tâche est la même. On souhaite pouvoir lancer le programme *Python* et lui spécifier les deux répertoires, c'est-à-dire être capable de lancer le programme comme suit (voir également figure 15.1) :

```
c:\python25\python.exe synchro.py c:/original c:/backup
```

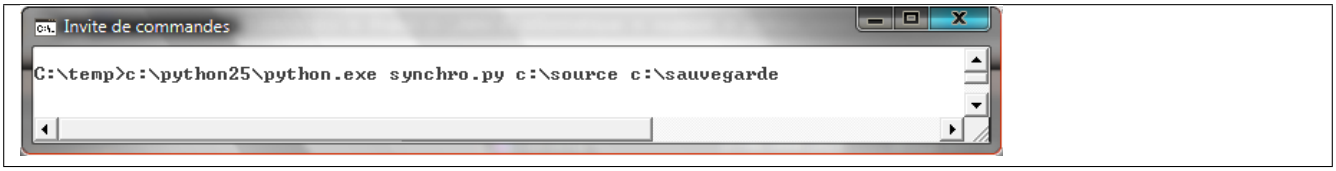



**Figure 15.1 :** *Un programme Python lancé en ligne de commande.*

Pour lancer un programme *Python* en ligne de commande, il faut écrire dans une fenêtre de commande ou un programme d'extension *bat* les instructions suivantes séparées par des espaces :

```
interpréteur_python programme_python argument1 argument2 ...
```

Il faut maintenant récupérer ces paramètres au sein du programme *Python* grâce au module `sys` qui contient une variable `argv` contenant la liste des paramètres de la ligne de commande. S'il y a deux paramètres, cette liste `argv` contiendra trois éléments :

1. nom du programme
2. paramètre 1
3. paramètre 2

Le programme suivante utilise ce mécanisme avec le programme `synchro` contenant la fonction `copie_repertoire` :

```
# -*- coding: cp1252 -*-
import glob
import shutil
def copie_repertoire (rep1, rep2) :
    """copie tous les fichiers d'un répertoire rep1 vers un autre rep2"""
    li = glob.glob (rep1 + "/*.*")
    for l in li :
        to = l.replace (rep1, rep2)  # nom du fichier copié
                                     # (on remplace rep1 par rep2)
        shutil.copy (l, to)

import sys
                      # sys.argv [0] --> nom du programme (ici, synchro.py)
rep1 = sys.argv [1]   # récupération du premier paramètre
rep2 = sys.argv [2]   # récupération du second paramètre
copie_repertoire (rep1, rep2)
```

**Remarque 15.2 : paramètres contenant des espaces**
Lorsque les paramètres écrits sur la ligne de commande contiennent eux-mêmes des espaces, il faut les entourer avec des guillemets.

```
c:\python25\python.exe c:\batch\synchro.py "c:\Program Files\source" "c:\Program Files\backup"
```

Pour des besoins plus complexes, le module `getopt` propose des fonctions plus élaborées pour traiter les paramètres de la ligne de commande.

### 15.1.6 Lancer un programme *Python* en ligne de commande depuis un programme *Python*

Il est parfois utile de lancer un programme exécutable en ligne de commande depuis un programme *Python*. C'est assez pratique pour ouvrir automatiquement un éditeur de texte ou lancer un navigateur affichant directement une page internet précise. Il existe plusieurs façons de faire cela. La première est l'utilisation de la fonction `system` du module `os`. L'exemple suivant ouvre automatiquement l'éditeur de texte *SciTe* sur le fichier `fichier.txt`.

```
os.system ("scite -open:fichier.txt")
```

Un autre moyen est l'utilisation du module `subprocess`. Il permet d'attendre ou non la fin de l'exécution de la ligne de commande.

```
import subprocess
args = ["scite.exe", "-open:fichier.txt"]
proc = subprocess.Popen(args)
retcode = proc.wait()            # on attend la fin de l'exécution
if retcode!=0:
    # ici, traiter les erreurs
```

Certains programmes en ligne de commande écrivent des informations dans la fenêtre d'exécution. C'est le cas par exemple d'une instruction `print` dans un programme *Python*. lorsqu'on exécute un programme en ligne de commande, ces informations sont perdues à moins de les récupérées. La première solution consiste à ajouter >> `output.txt` en fin de ligne de commande de sorte que ces informations seront stockées dans le fichier `output.txt`.

```
import subprocess
args = ["scite.exe", "-open:fichier.txt", ">>output.txt"]
proc = subprocess.Popen(args)
```

La seconde option consiste à rediriger la sortie vers le programme appelant toujours avec la fonction `Popen`.

(à venir)

## 15.2 Tortoise SVN : suivi de version

Le premier des outils est sans doute à installer avant de commencer à programmer. *Tortoise SVN*[5] [6] permet de conserver l'historique des modifications d'un fichier, d'une arborescence de fichiers. Le logiciel peut gérer tout type de fichiers, toutefois, il est particulier indiqué pour les fichiers textes. C'est un outil également qui facilite grandement le travail à plusieurs sur les mêmes fichiers.

Le principe est d'avoir un répertoire où sont stockées l'ensemble des fichiers et toutes les modifications qui y ont été apportées. Ce répertoire principal porte souvent le nom de *repository* (voir figure 15.2). Cette copie principale n'est pas modifiable directement. Il faut en faire autant de copies qu'il y a d'intervenants sur le projet.

### 15.2.1 Utilisation personnelle

Cet outil est également utile pour ses propres fichiers, pour conserver l'historique des modifications apportées à un ensemble de fichiers. La première étape est l'installation.

5. http://tortoisesvn.tigris.org/

6. *Tortoise SVN* comprend en fait deux composants, le premier *SVN* est le logiciel proprement dit, il existe sur tout type de plateforme. *Tortoise* est l'interface graphique qui permet d'utiliser *SVN* plus simplement mais elle est spécifique à *Windows*.

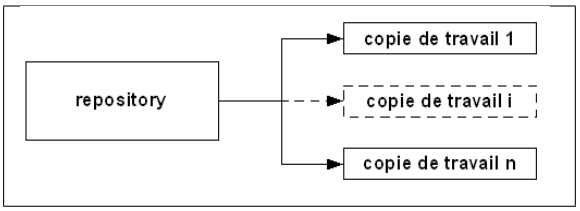



**Figure 15.2 :** *SVN conserve une copie principale que personne ne peut modifier directement. On peut en fait autant de copies qu'il y a de programmeurs. Chacun d'entre eux peut modifier sa copie sans interférer sur celle des autres.*

#### 15.2.1.1 Installation et création d'un repository

installation

L'installation commence par l'exécution d'un installateur *Windows*[7]. Celle-ci ne pose pas de problème particulier. Une fois celle-ci terminée, de nouveaux menus apparaissent dans l'explorateur *Windows* lorsqu'on clique sur le bouton droit de la souris sur un répertoire (voir figure 15.3). La création d'un repository consiste tout d'abord à créer un répertoire que SVN utilisera comme répertoire principal. C'est un répertoire que seul *SVN* modifiera, le modifier soi-même n'est pas conseillé même si certains fichiers de configuration peuvent l'être mais cela sort du cadre de ce livre.repository

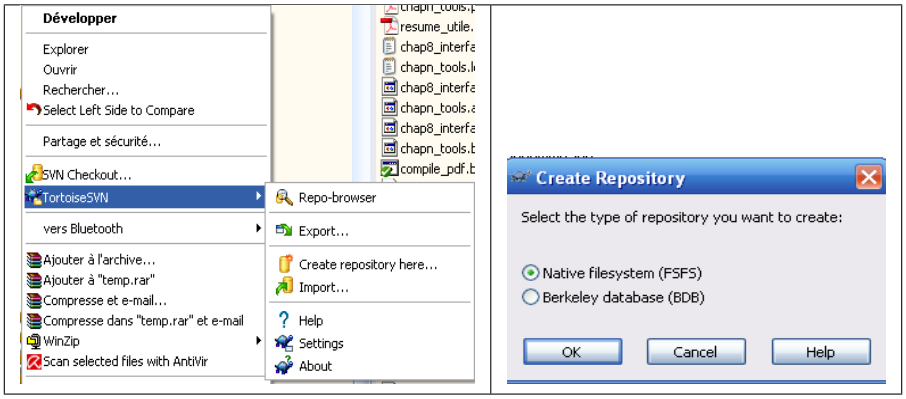



**Figure 15.3 :** *Après l'installation de SVN, de nouveaux menus apparaissent lors d'un clic droit de la souris au-dessus d'un répertoire. La seconde image intervient lorsqu'on cherche à construire un repository, SVN demande quel type d'archivage il doit choisir. Il est préférable de choisir le premier.*

L'étape suivante consiste à créer un répertoire dans le repository. Il faut encore cliquer sur le clic droit de la souris à l'endroit où a été créé le repository et choisir *RepoBrowser*. Ensuite, un nouveau clic droit de la souris dans la fenêtre qui s'ouvre permet de créer un nouveau répertoire (choisir *Add Folder*). Ce premier répertoire porte souvent le nom de *racine* ou *root* ou encore *trunk*.création d'un répertoire

#### 15.2.1.2 Utilisation

Le principe de *SVN* est de faire correspondre un répertoire du disque dur au premier répertoire du repository. Ce sera le répertoire de travail. Le nom de l'opération est *CheckOut* (voir figure 15.5).CheckOut *SVN* crée un répertoire si celui-ci n'existe pas et y insère également un répertoire caché *.svn*..svn Il contient des informations interne à *SVN*. Il ne faut ni le supprimer ni le modifier.

Ajouter un fichier au répertoire *racine* du disque dur n'a pas d'effet sur le repository. Pour l'y inclure, il faut utiliser l'opération *Commit* (voir la figure 15.6).Commit Il faut cocher la case associée au fichier créé, ajouter un commentaire et cliquer sur *Ok*.

Un icône vert apparaît adossé au fichier créé (voir figure 15.7), il signifie que le fichier est identique dans le repository et dans le répertoire associé. Sa modification entraîne l'apparition d'un icône rouge qui signifie

7. L'installation à usage personnel est simple, l'utilisation pour un usage multiple permet une utilisation personnelle mais nécessite des étapes de configurations plus importantes et l'installation de trois logiciels (voir paragraphe 15.2.2.1).

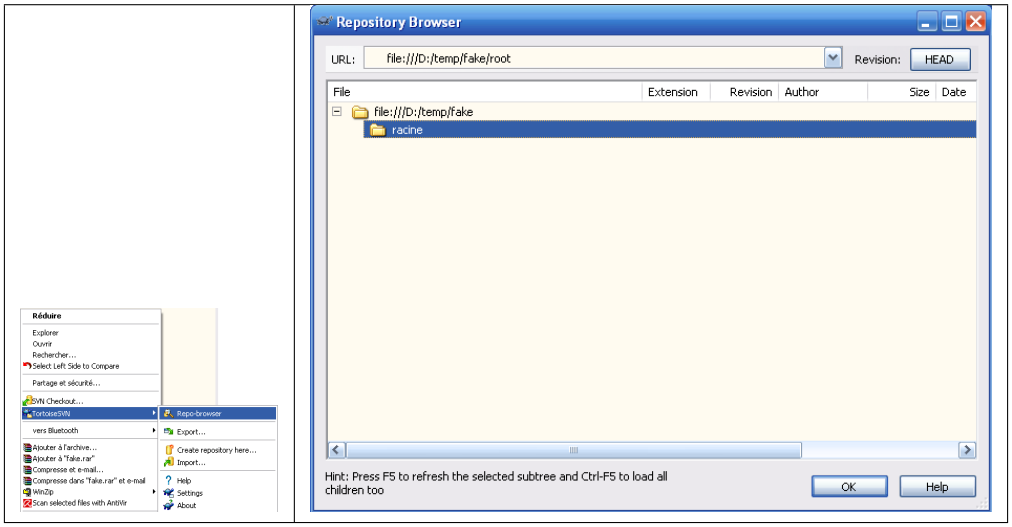



**Figure 15.4 :** *Création du premier répertoire du repository.*

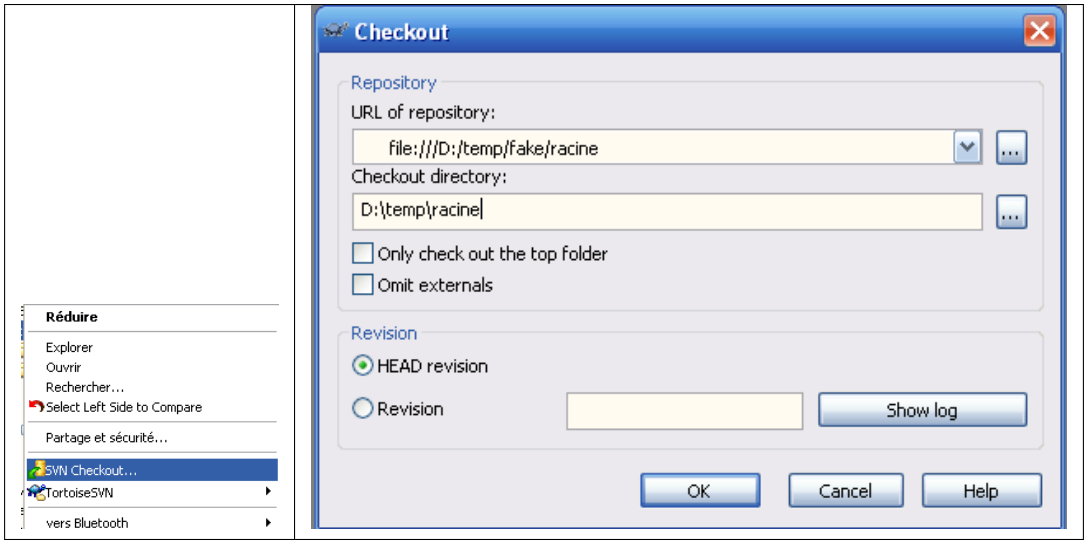



**Figure 15.5 :** *Création d'une correspondance entre un répertoire du disque dur et un répertoire du repository.*

que la copie du disque dur a été modifiée. La répétition de l'opération *Commit* permet de basculer les modifications dans le repository et de faire apparaître l'icône vert.

Un nouveau clic droit sur le fichier modifié et la sélection du sous-menu *Log* permet d'afficher l'ensemble des modifications intervenues sur ce fichier (voir figure 15.8).Log Un double-clic sur le fichier en question permet de comparer deux versions consécutives, de voir les lignes ajoutées, supprimées et modifiées. Cette fonctionnalité ne concerne que les fichiers textes.

Les autres menus ne seront pas détaillés, leur exploration est laissée au soin du lecteur. Toutefois, l'un d'entre deux permet de récupérer une vieille version d'un fichier, antérieure aux dernières modifications. De la même manière, si un fichier est supprimée par mégarde, celui-ci, à jamais présent dans le repository, peut être récupéré à tout moment.

#### 15.2.1.3 Configuration

Un clic droit sur le répertoire repository permet d'accéder à la partie *Settings*.Settings Le seul paramètre décrit dans ce paragraphe est celui intitulé *Global ignore patterns* de la section *General* (voir figure 15.9).

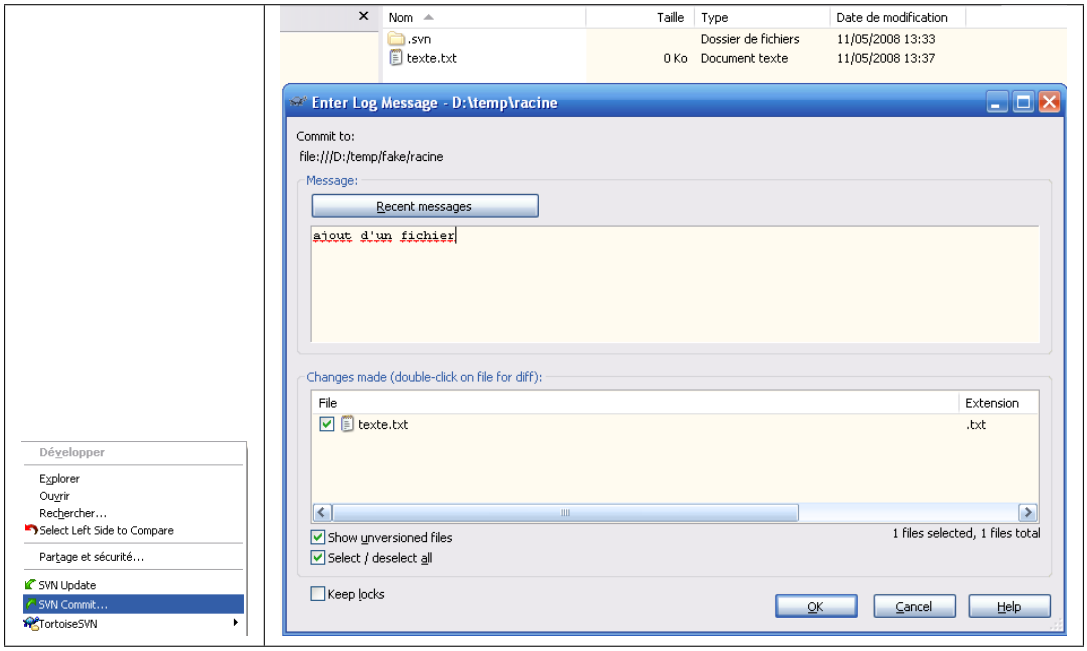



**Figure 15.6 :** *Ajout d'un fichier texte au repository.*




**Figure 15.7 :** *Icônes SVN.*

Il permet que certains fichiers ne soient pas pris en compte par *SVN*. Par exemple, lorsqu'on programme en *Python*, de nombreux fichiers d'extension *pyc* sont générés. Il ne sert à rien de les conserver dans le repository puisque ceux-ci seront de toutes façons recréés par *Python* si ceux-ci ont été supprimés ou si le fichier source à leur origine a été modifié : ce sont des fichiers temporaires. Pour éviter que *SVN* n'en tienne compte, il suffit d'ajouter l'expression **.pyc* dans le champ *Global ignore patterns*.

#### 15.2.1.4 Plusieurs copies du repository

Même si ce n'est pas souvent utile, il est possible de travailler avec plusieurs copies du repository. Chaque copie doit être construite avec la fonctionnalité *CheckOut*. Une fois ces deux copies créés, il faut pouvoir propager les modifcations d'une copie à l'autre. La fonctionnalité *Commit* appliquée à la copie modifiée permet de transmettre des modifications au repository qui contient l'unique version officielle. Ensuite, la fonctionnalité *Update*Update permet de mettre à jour en chargeant depuis le repository les dernières modifications. Ce principe est illustré par le schéma de la figure 15.10.

Certains problèmes peuvent survenir lorsqu'on travaille sur les mêmes fichiers depuis les deux copies. Il faut toujours rappatrier les dernières modifications enregistrées dans le repository : un *Commit* est toujours précédé d'un *Update*. *SVN* est capable dans la plupart des cas de gérer les conflits. Seuls restent les cas où les mêmes lignes du même fichier sont modifiées. Lors d'un *Update*, on peut récupérer une modification qui rentre en conflit sur la version en cours. Il faut alors résoudre ce conflit puis indiquer à *SVN* que le conflit est résolu (par l'intermédiaire des sous-menu accessibles avec la souris). *SVN* n'autorisera pas de *Commit* sur un fichier pour lequel il subsiste un conflit irrésolu.Resolve

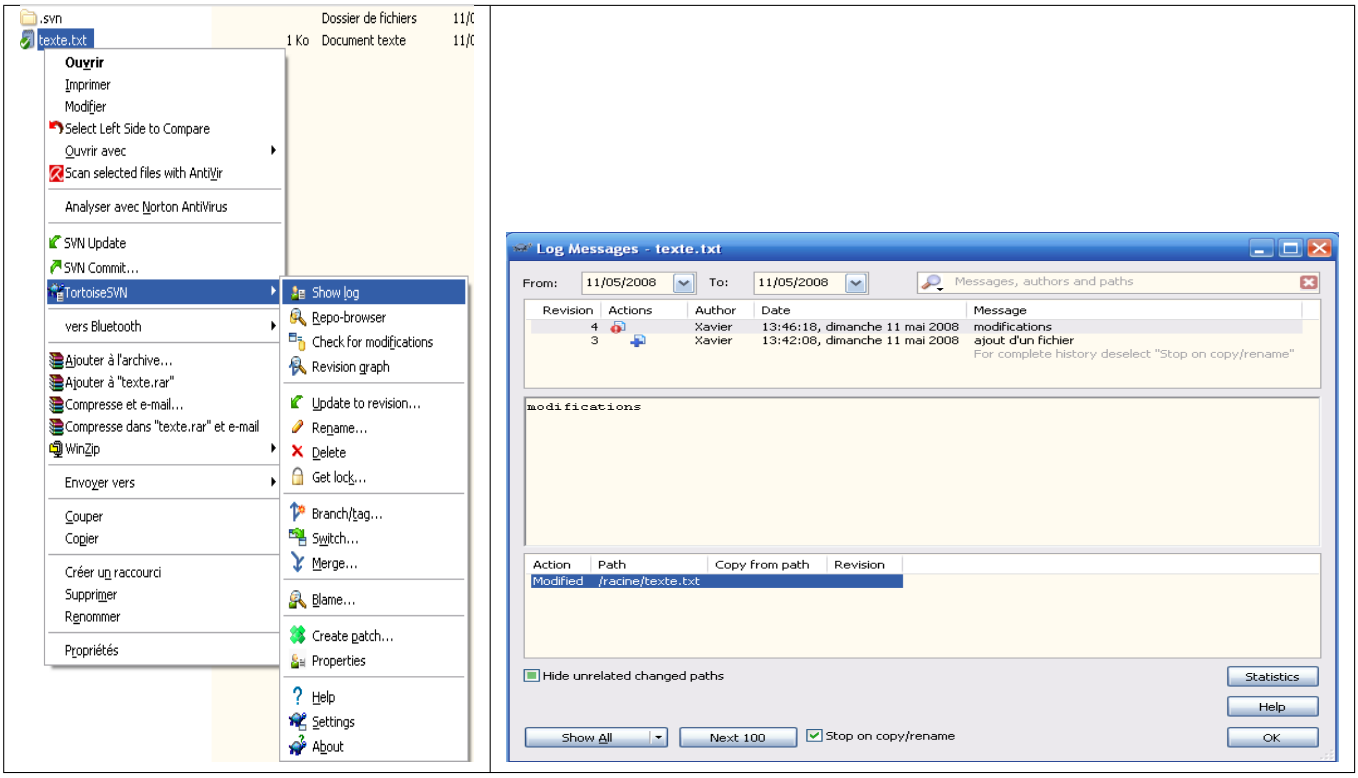



**Figure 15.8 :** *Historique des modifications d'un fichier.*

### 15.2.2 Utilisation à plusieurs

Une utilisation à plusieurs requiert une machine accessible par tous sur laquelle sera installé le repository. La communication entre cette machine et celles des autres utilisateurs se fera par l'intermédiaire d'un serveur. Le plus couramment utilisé est le serveur *Apache*. C'est une application qui va fonctionner en permanence sur l'ordinateur et qui permettra les échanges d'informations.

#### 15.2.2.1 Installation sur la machine contenant le repository

**Installation de *SVN*** installation

L'installation est différente car l'application *SVN* utilisée est légèrement différente et il faut faire attention que *SVN* et *Apache* ont des numéros de versions cohérents. On commence donc par l'installation de *SVN*. Il faut tout d'abord télécharger une version *SVN* pour une version *Apache* spécifique [8]

Ensuite, on va créer un repository. Pour cela, il faut ouvrir une fenêtre de commande *Windows* et écrire les instructions suivantes pour créer un repository à l'emplacement `c : \svnrepository` :

```
cd "répertoire d'installation de SVN\bin"
svnadmin create c:\svnrepository
```

**Installation de *Apache*** Voir paragraphe 15.9 page 383.

8. Au moment où est écrit ce livre, la version *SVN* est 1.4.6 pour *Apache 2.2*, *SVN* est téléchargeable à l'adresse http://subversion.tigris.org/project_packages.html et apparaît sous l'intitulé *Win32 packages built against Apache 2.2* pour *Windows*.

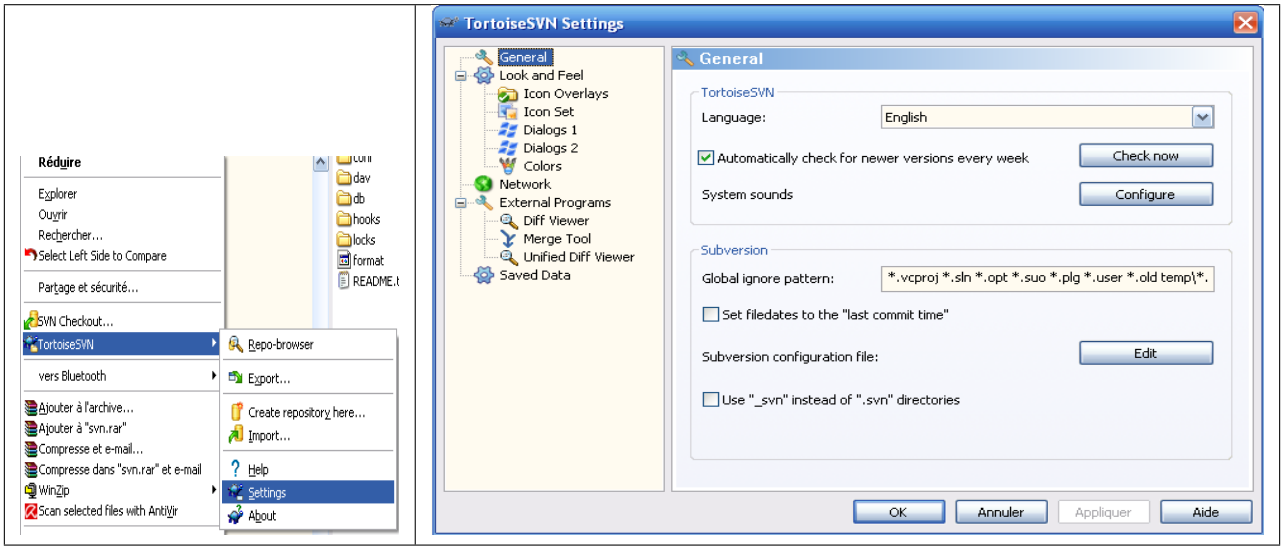



**Figure 15.9 :** *Configuration de SVN, il est possible de faire en sorte que SVN ne tienne pas compte de certains fichiers dans le champ Global ignore patterns. S'il y a plusieurs filtres, ils doivent être séparés par des espaces.*

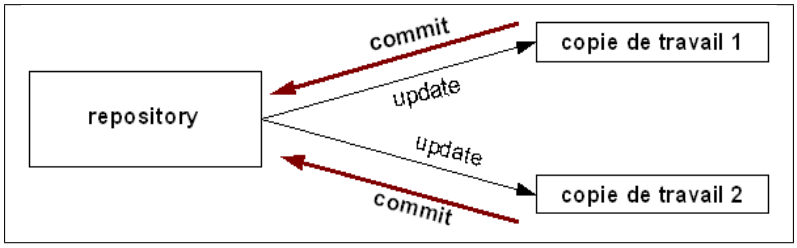



**Figure 15.10 :** *Lorsqu'on travaille avec plusieurs copies, la fonctionnalité Update fait les modifications depuis le repository jusqu'à une copie (celle sur laquelle on applique l'opération Update. La fonctionnalité Commit est le chemin inverse : on transmet les modifications depuis une copie vers le repository.*

#### 15.2.2.2 Installation sur les autres machines

L'installation est beaucoup plus simple puisqu'il suffit d'installer *Tortoire SVN* (voir paragraphe 15.2.1.1, page 337). Il suffira de faire une opération `checkout` depuis le repository http://localhost/svn/ http://127.0.0.1/svn/ où `127.0.0.1` est l'adresse IP de la machine où est installée le repository.

**Remarque 15.3 : connexion sur *Apache* impossible depuis une autre machine du réseau**
Il est possible que l'accès à la machine où est installé *Apache* soit impossible depuis une autre machine. Il y a deux causes possibles, la première est que les deux machines ne peuvent pas communiquer ensemble. Pour vérifier cela, il suffit d'ouvrir une fenêtre de commande *Windows* et d'écrire depuis la machine distante :

```
ping 127.0.0.1
```

On suppose que `127.0.0.1` est l'adresse IP de la machine où figure *Apache*. Si le résultat est négatif, il faut d'abord résoudre ce problème là. La seconde cause est le *firewall* (ou *pare-feu*) qui empêche les connexions entrantes. Il est possible que ce soit le *firewall* de *Windows* qui soit utilisé, il est possible de le désactiver depuis le panneau de contrôle ou le laisser activer mais autoriser une exception concernant le port qu'écoute le serveur *Apache*.

**Remarque 15.4 : *Apache* écoute un autre port que le port 80**
Au cas où *Apache* écoute un autre port que le port 80, il faut le préciser lorsqu'on se connecte depuis une machine distante. Si le port écouté est le port 81, il faudra écrire dans la barre d'adresse :

```
http://127.0.0.1:81/
```

#### 15.2.2.3 Backup

backup

Il peut être intéressant de sauvergarder régulièrement le repository au cas où la machine qui le contiendrait tombe en panne. Cette option ne sera pas détaillée ici mais elle est disponible avec `Apache` qui autorise l'exécution de tâches à intervalles réguliers.

#### 15.2.2.4 Via internet

*SVN* est plus souvent installé pour être utilisé à l'intérieur d'une entreprise et n'est accessible que depuis son réseau local. Néanmoins, il n'est pas exclu de travailler depuis des postes distants via Internet. Comme pour une utilisation interne, cela suppose qu'un ordinateur servira de repository, la différence viendra de l'URL permettant de se connecter au seveur *SVN* qui pointera vers l'extérieur. Toutefois, cette configuration soulève des problèmes de sécurité. Il faut que le serveur soit bien configuré pour ne laisser l'accès aux fichiers partagés qu'à ceux qui y sont autorisés.

#### 15.2.2.5 Limitations

Travailleur à plus d'une dizaine de personnes sur les mêmes fichiers peut parfois poser des problèmes. Comme un *Update* précède toujours un *Commit*, il arrive, en période de travail intense, qu'un autre programmeur réussisse à faire passer ses modifications entre la succession *Update-Commit* d'un autre programmeur.

La gestion des fichiers *SVN* est dite centralisée car il existe une seule version officielle de tous les fichiers. Cette configuration n'est pas toujours appropriée à la gestion du code source de projets open source auxquels participent un grand nombre de programmeurs. La configuration employée est dite décentralisée, ceci signifie qu'il n'existe plus de version officielle mais que chacun possède une version unique et identifiée. Il n'y a plus d'opération *Commit* ou *Update* mais simplement l'opération *Merge* qui consiste à fusionner les fichiers de deux versions. Avec ce schéma, le problème évoqué au paragraphe précédent n'existe plus.

#### 15.2.2.6 Module `pysvn`

Un module `pysvn`[9] permet de récupérer des informations sur un repository et de faire des modifications par l'intermédiaire du langage *Python*. Grâce à cela, on pourrait plus facilement piloter *SVN* et faciliter l'exécution de tâches récurrentes comme la mise à jour chaque nuit de tous les machines de chaque utilisateur. Il faut vérifier lors de l'installation que le module `pysvan` installé est cohérent à la fois avec la version de *Python* et la version de *SVN* installée.

## 15.3 HTML Help Workshop : construction d'un fichier d'aide

installation

La rédaction d'une documentation devrait accompagner toute application destinée à tout autre utilisateur que le concepteur lui-même. Un moyen simple est la rédaction de pages HTML. Ces pages peuvent ensuite compilées pour ne former qu'un seul fichier. A cet ensemble est adjoint une table des matières, un indexe, et un moteur de recherche simple. La première partie du travail est assurée par *Python* lui-même avec le module `pydoc`.. La seconde partie est assurée par le logiciel *HTML Help Workshop*[10].

9. téléchargeable depuis l'adresse http://pysvn.tigris.org/
10. http://msdn.microsoft.com/en-us/library/ms669985.aspx

### 15.3.1 Création automatique d'une documentation avec pydoc

Le module pydoc ne s'utilise pas directement à l'intérieur d'un programme *Python* mais en ligne de commande. L'exemple suivant génère l'aide associée au module random et celle du programme mon_programme.py dans le répertoire courant.

```
c:\python25\python c:\python25\lib\pydoc.py random
c:\python25\python c:\python25\lib\pydoc.py .\mon_programme.py
```

Pour l'inclure dans un programme *Python*, il faut reproduire un appel en ligne de commande grâce à la fonction system du module os :

```
import os
os.system (r"c:\python25\python c:\python25\lib\pydoc.py .\mon_programme.py")
```

Le nom d'un module standard doit apparaître sans extension, le nom d'un fichier peut apparaître sans extension s'il se trouve dans le répertoire depuis lequel est lancé l'appel à pydoc, sinon, il faut mettre son chemin complet avec au moins un symbol \. Le programme suivant crée automatique l'aide associée à une liste de fichiers ou de modules, le résultat est illustré par la figure 15.11.

```
# -*- coding: cp1252 -*-
"""genhelp.py : génération automatique de l'aide dans le répertoire d'exécution"""
import os
import sys
python_path = r"c:\python25\python"          # constante
pydoc_path  = r"c:\python25\lib\pydoc.py"    # constante

class ClassExemple :
    """classe vide, exemple d'aide"""
    def __init__ (self) :
        """constructeur"""
        pass
    def methode (self) :
        """unique méthode"""
        return 1

def generate_help_file (f, pyt = python_path, pyd = pydoc_path) :
    """génère l'aide associée à un fichier ou un module
    le nom de ce fichier peut apparaître sans son extension ou alors
    précédé de .\\ ou avec son chemin complet
    pyt est le répertoire de python
    pyd est l'emplacement de pydoc.py"""
    s = "call " + pyt + " " + pyd + " -w " + f
    os.system (s)

def replace_firstpage (file, page) :
    """la génération de l'aide chm s'arrête avant la fin si le lien <a href=".">index</a>
    est laissé dans les pages générées par pydoc, on le remplace par une page
    comme index.html"""
    f = open (file, "r")
    li = f.readlines ()
    f.close ()
    f = open (file, "w")
    for l in li :
        f.write (l.replace ("""<a href=".">index</a>""", page))
    f.close ()

def genhelp (files, pyt = python_path, pyd = pydoc_path, firstpage = "index.html") :
    """génère l'aide associée à des fichiers ou des modules,
    un fichier se distingue d'un module par son extension,
```

```
    retourne la liste des fichiers générés,
    pyt est le répertoire de python
    pyd est l'emplacement de pydoc.py
    firstpage voir fonction replace_firstpage"""
    res = []
    for f in files :
        print "génération de l'aide de ", f
        if ".py" in f : # fichier
            generate_help_file (f, pyt, pyd)
            g    = f.split ("\\")  # ne garde que le nom de fichier et non son chemin
            page = g [ len (g)-1].replace (".py", ".html")
        else :          # module
            generate_help_file (f, pyt, pyd)
            page = f + ".html"
        res.append (page)
        replace_firstpage (page, firstpage)
    return res

if __name__ == "__main__" :
    import sys
    import os

    files = [".\\genchm.py", ".\\genhelp.py", "os", "sys"]
    res   = genhelp (files)
    print res  # ['genchm.html', 'genhelp.html', 'os.html', 'sys.html']
```




**Figure 15.11 :** *Exemple généré avec* pydoc.

### 15.3.2 Compilation des pages HTML avec *HTML Help Workshop*

Les pages générées par l'exemple précédents vont être reprise par l'application *HTML Help Workshop* pour générer un fichier d'aide de type *chm*. Il est possible d'utiliser cette application de façon classique (voir figure 15.12) mais l'objectif de cette section sera de générer ce fichier de façon automatique grâce à un script *Python*.

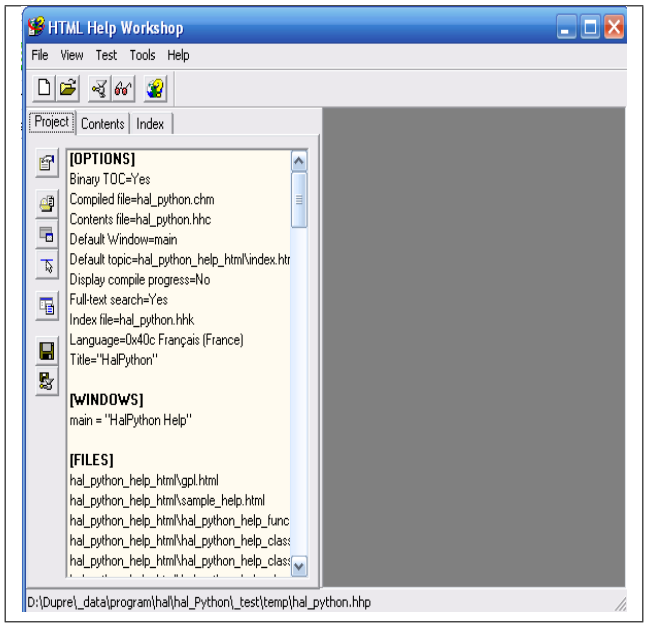



**Figure 15.12 :** *Fenêtre de l'application HTML Help Workshop.*

Pour cela, il faut construire trois fichiers :

1. help.hhp : le fichier de configuration de l'aide
2. help.hhc : le fichier contenant la table des matières (format HTML)
3. help.hhk : le fichier contenant l'indexe (format HTML)

L'utilisation de ce genre d'outils comme *HTML Help Workshop* est parfois laborieuse, apprendre à s'en servir n'a en soi pas beaucoup d'intérêt et une fois que la génération des fichiers *chm* est automatisée, il n'est pas rare qu'on oublie leur mode d'emploi. Les paragraphes qui suivent décrivent sommairement l'application et vous permettront de ne pas lui consacrer trop de temps. Le premier fichier d'extension *hhp* définit le projet, un exemple suit, seules les lignes commentées sont importantes.

```
[OPTIONS]
Compatibility=1.1
Full-text search=Yes
Contents file=help.hhc    # fichier table des matières
Default Window=main
Default topic=index.html  # page par défaut, première page à apparaître
Index file=help.hhk       # fichier index
Language=0x40C French
Binary TOC=YES
Create CHI file=No
Title="GenHelp"           # titre

[WINDOWS]                 # définition de la page principale, elle reprend des éléments cités ci-dessus
main="GenHelp", "help.hhc", "help.hhk" , "index.html", "index.html",,,,,0x23520,,0x387e,,,,,,,,0

[FILES]                   # ici commence la liste des fichiers à insérer dans le projet
genchm.html
genhelp.html
```

```
os.html
sys.html
index.html
```

Les fichiers *hhc* et *hhk* sont définis avec la même syntaxe HTML et leur structure est une arborescence. Ces deux fichiers commencent par l'en-tête suivant :

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN"><HTML><HEAD></HEAD><BODY>
```

Et se concluent par le pied de tête :

```
</BODY></HTML>
```

Chaque élément est composé d'un intitulé et d'une page HTML correspondante qui est facultative, ces deux informations vont être insérés dans le code HTML qui suit :

```
<LI><OBJECT type="text/sitemap">
<param name="Name" value="intitule">
<param name="Local" value="page HTML correspondante">
<param name="ImageNumber" value="11"></OBJECT>
```

Les niveaux et sous-niveaux sont définis par l'instruction < UL > qui permet d'ouvrir une section de niveau inférieure et < /UL > de la fermer. Appliqué à un exemple d'indexe, cela donne :

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN"><HTML><HEAD></HEAD><BODY>
<OBJECT type="text/site properties">param name="GenHelp" value="right">OBJECT>
<UL>
    <LI><OBJECT type="text/sitemap">
        <param name="Name" value="GenHelp">
        <param name="Local" value="index.html">
        <param name="ImageNumber" value="11"></OBJECT>
    <UL>
        <LI><OBJECT type="text/sitemap">
            <param name="Name" value="genchm">
            <param name="Local" value="genchm.html">
            <param name="ImageNumber" value="11"></OBJECT>
        <LI><OBJECT type="text/sitemap">
            <param name="Name" value="genhelp">
            <param name="Local" value="genhelp.html">
            <param name="ImageNumber" value="11"></OBJECT>
        <LI><OBJECT type="text/sitemap">
            <param name="Name" value="os">
            <param name="Local" value="os.html">
            <param name="ImageNumber" value="11"></OBJECT>
        <LI><OBJECT type="text/sitemap">
            <param name="Name" value="sys">
            <param name="Local" value="sys.html">
            <param name="ImageNumber" value="11"></OBJECT>
        <LI><OBJECT type="text/sitemap">
            <param name="Name" value="index">
            <param name="Local" value="index.html">
            <param name="ImageNumber" value="11"></OBJECT>
    </UL>
</UL>
</BODY></HTML>
```

La dernière étape est de construire un programme *Python* qui construit automatiquement les trois fichiers nécessaires à *HTML Help Workshop*. L'exemple suivant est assez simple, il pourrait être étoffé notamment en ajoutant à l'indexe le nom des classes et de leur méthodes ce qu'il ne fait pas.

```
# -*- coding: cp1252 -*-
"""genchm.py : génération automatique du fichier d'aide chm"""
import genhelp
import os
htmlworkshop_path = "\"c:\\Program Files\\HTML Help Workshop\\hhc.exe\""

def g (s) :
    """ajoute des guillements autour d'une chaîne de caractères"""
    return "\"" + s + "\""

def genhhp (files, premierepage, titre, \
            hhp = "help.hhp", hhc = "help.hhc", hhk = "help.hhk") :
    """génère le fichier hpp définissant le fichier d'aide chm,
    files est la liste des fichiers HTML
    premierepage est la page à afficher en premier
    titre est le titre de l'aide"""
    proj = """[OPTIONS]
Compatibility=1.1
Full-text search=Yes
Contents file=""" + hhc + """
Default Window=main
Default topic=""" + premierepage + """
Index file=""" + hhk + """
Language=0x40C French
Binary TOC=YES
Create CHI file=No
Title=""" + g (titre) + """

[WINDOWS]
main=""" + g (titre) + """, """ + g (hhc) + """, """ + g (hhk) + """ , """ + \
        g (premierepage) + """, """ + g (premierepage) + """,,,,,0x23520,,0x387e,,,,,,,,0

[FILES]
"""
    for f in files :
        proj += f + "\n"

    f = open (hhp, "w")
    f.write (proj)
    f.close ()

def gen_premierepage (files, titre, res = "index.html") :
    """génère la première page de l'aide au format HTML"""
    s = """<HTML><head><title>""" + titre + """</title></head><BODY>\n"""
    s += "<H1>" + titre + "</H1>\n"
    for f in files :
        s += "<A HREF=\"" + f + "\">" + f.replace (".html", "") + "</A><BR>\n"
    s += "</BODY></HTML>\n"

    f = open (res, "w")
    f.write (s)
    f.close ()

def genhh_input (entree, page, link = True) :
    """retourne la chaîne de caractères associée à une entrée de la table des matières"""
    res = """<LI><OBJECT type="text/sitemap"><param name="Name" value=\"""" + entree + """\">"""
    if link : res += """<param name="Local" value=\"""" + page + """\">"""
    res += """<param name="ImageNumber" value="11"></OBJECT>\n"""
    return res

def genhhc (files, premierepage, titre, hhc = "help.hhc") :
    """génère le fichier hhc, même paramètre que pour hhp"""
    res ="""<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN"><HTML><HEAD></HEAD><BODY>
        <OBJECT type="text/site properties">
```

```
        <param name=""" + g (titre) + """ value="right">
        </OBJECT>
        <UL>"""
    res += genhh_input (titre, premierepage, False)
    res += "<UL>\n"
    res += genhh_input ("modules", premierepage, False)
    res += "<UL>\n"
    for f in files :
        res += genhh_input (f.replace (".html", ""), f)
    res += "</UL>\n"
    res += "</UL>\n"
    res += "</UL>\n"
    res += "</BODY></HTML>\n"

    f = open (hhc, "w")
    f.write (res)
    f.close ()

def genhhk (files, premierepage, titre, hhk = "help.hhk") :
    """génère le fichier hhk, même paramètre que pour hhp"""
    res ="""<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN"><HTML><HEAD></HEAD><BODY>
        <OBJECT type="text/site properties">
        <param name=""" + g (titre) + """ value="right">
        </OBJECT>
        <UL>"""
    res += genhh_input (titre, premierepage)
    res += "<UL>\n"
    for f in files :
        res += genhh_input (f.replace (".html", ""), f)
    res += "</UL>\n"
    res += "</UL>\n"
    res += "</BODY></HTML>\n"

    f = open (hhk, "w")
    f.write (res)
    f.close ()

def genchm (files, titre, \
            hhp = "help.hhp", hhc = "help.hhc", hhk = "help.hhk") :
    """génère le fichier d'aide complet"""
    premierepage = "index.html"                     # génère la page de garde
    gen_premierepage (files, titre, premierepage)
    files.append (premierepage)
    genhhp (files, premierepage, titre)             # génère le fichier hhp
    genhhc (files, premierepage, titre)             # génère le fichier hhc
    genhhk (files, premierepage, titre)             # génère le fichier hhk
    os.system (htmlworkshop_path + " " + hhp)       # appelle HTML WorkShop en ligne de commande

if __name__ == "__main__" :
    files = [".\\genchm.py", ".\\genhelp.py", "os", "sys"]
    res   = genhelp.genhelp (files)
    print res  # ['genchm.html', 'genhelp.html', 'os.html', 'sys.html']
    genchm (res, "GenHelp")
```

**Remarque 15.5 : problèmes de compilation**

Il arrive que les messages d'erreurs de ce logiciel ne soient pas très explicites. Lors de la compilation du fichier *chm*, il s'intéresse aussi à toutes les pages dont l'hyperlien dans une des pages incluses dans le fichier *hpp*, c'est pourquoi le nombre de pages final peut être plus élevé que celui des pages recensées.

### 15.3.3 Prolongements

La documentation générée par le module `pydoc` n'est pas très agréable à lire. D'autres outils permettent d'améliorer la qualité des pages automatiquement générées et de générer d'autres formats que des pages HTML. Le premier est *epydoc*[11] qui fonctionne de façon similaire à `pydoc` mais déjà plus agréable. Le plus répandu est *Doxygen*[12] car il traite la plupart des langages informatique. Son utilisation fait appel à d'autres outils tels que *Graphviz*[13], *Latex*[14], ou encore le module `doxypy`[15].

## 15.4 InnoSetup : installateur

installation

Un installateur répond au problème : comment permettre à d'autres d'utiliser votre travail en peu de temps ? Une application a souvent besoin de plusieurs fichiers pour fonctionner et ceux-ci doivent respecter une certaine arborescence. *InnoSetup*[16] est une application qui répond simplement à ce problème. Il permet de concevoir un programme exécutable (d'extension *exe*) qui recopie aux bons endroits les bons fichiers. Il permet plus précisément de :

1. grouper plusieurs fichiers ensemble en un seul, ces fichiers peuvent provenir de différents répertoires depuis la machine du concepteur
2. recopier ces fichiers à des endroits spécifiques sur la machine de l'utilisateur
3. exécuter certaines tâches avant la première utilisation
4. construire un sous-menu dans le menu *Démarrer / Tous les programmes* de *Windows*

### 15.4.1 Sections *InnoSetup*

#### 15.4.1.1 section [Setup]

Toutes ces actions sont définies par un fichier texte d'extension *iss*. Celui-ci commence par un en-tête définissant les options générales.

```
[Setup]                                  ; nom de la section
AppName=HalPython                        ; nom de l'application à installer
AppVerName=HalPython 1.5.1162            ; numéro de version
AppPublisher=Xavier Dupré                ; auteur
AppPublisherURL=http://www.xavierdupre.fr/ ; site web
AppSupportURL=http://www.xavierdupre.fr/   ; site web, support
AppUpdatesURL=http://www.xavierdupre.fr/   ; site web, mise à jour
DefaultDirName={pf}/HalPython            ; emplacement par défaut sur la machine utilisateur
                                         ; le code {pf} signife le répertoire \textit{Program Files}
DefaultGroupName=HalPython               ; nom par défaut du sous menu dans
                                         ;                              Démarrer / Tous les programmes
OutputDir=c:                             ; répertoire où sera écrit l'installateur
OutputBaseFilename=setup_HalPython_py25  ; nom de l'installateur ou setup
Compression=lzma                         ; type de compression, ne pas changer
SolidCompression=yes                     ; type de compression, ne pas changer
VersionInfoVersion=1.0.0                 ; version, ne pas changer
```

11. http://epydoc.sourceforge.net/
12. http://www.stack.nl/~dimitri/doxygen/
13. http://www.graphviz.org/, cet outil sert à représenter des graphes.
14. *Latex* n'est pas un outil en soi, c'est un langage permettant de créer des documents à usage le plus souvent scientifique. *Miktex* fournit un compilateur sur *Windows* (http://miktex.org/).
15. http://code.foosel.org/doxypy
16. http://www.jrsoftware.org/isinfo.php

#### 15.4.1.2 section [Files]

Ensuite, il faut définir les quatre étapes définies précédemment. Les deux premières seront regroupées en une seule section [Files] qui indiquera en même temps l'emplacement d'un fichier chez le concepteur et sa destination chez l'utilisateur.

```
[Files]
Source: "chemin_concepteur"; DestDir: "chemin_utilisateur"  ;
Source: "..\hal_dll.dll"; DestDir: "{app}\hal_python"       ; autre exemple
```

L'installateur construit par *InnoSetup* demande à l'utilisateur de spécifier un chemin d'installation. Le code *app* fait référence à celui-ci lorsqu'il est utilisé dans la définition du chemin de destination.

#### 15.4.1.3 section [Run]

La partie exécution permet d'exécuter certaines opérations comme des opérations sur les fichiers en lignes de commande, la compilation de fichiers, l'installation d'un module *Python* ou encore l'installation d'autres applications. Par exemple, les lignes suivantes permettent de lancer l'installation du langage *Python* puis de le supprimer (suppression du fichier *python-2.5.2.msi*).

```
[Run]
Filename: "{win}\system32\msiexec.exe"; Parameters: "/i ""{app}\python-2.5.2.msi"" /qr ALLUSERS=1";
Filename: "{cmd}"; Parameters: "/c del python-2.5.2.msi"; WorkingDir: "{app}";
```

#### 15.4.1.4 section [Icons]

La section [Icons] permet d'ajouter un sous-menu dans le menu *Démarrer / Tous les programmes*. Le nom de ce sous-menu est précisé dans la section [Setup].

```
[Icons]
; liens vers diverses applications
Name: "{group}\PyScripter"; Filename: "{app}\PyScripter.exe"; WorkingDir: "{app}"
Name: "{group}\PyScripter Help"; Filename: "{app}\pyscripter.chm"; WorkingDir: "{app}"
Name: "{group}\Help"; Filename: "{app}\hal_python.chm"; WorkingDir: "{app}"
Name: "{group}\smallest sample with PyScripter"; Filename: "{app}\small_sample.bat"; WorkingDir: "{app}"

; hyperlien ou url vers un site
Name: "{group}\Website"; Filename: "{app}\hal_python.url"

; lien vers le désinstalleur créé automatiquement par InnoSetup
Name: "{group}\uninstall"; Filename: "{uninstallexe}"; WorkingDir: "{app}"
```

Le fichier hal_python.url contient le texte suivant :

```
[InternetShortcut]
URL=http://www.xavierdupre.fr/hal_python/hal_python_help_html/index.html
```

La section [Tasks] permet de créer un icône sur le bureau relié au sous-menu définit auparavant.

```
[Tasks]
Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}"; GroupDescription: "{cm:AdditionalIcons}";
    Flags: checkedonce
```

#### 15.4.1.5 section [Registry]

Cette section permet de modifier les clés de registres de *Windows*. Cela permet de mémoriser le numéro de version ou encore le chemin utilisé lors d'une installation précédente.

```
[Registry]
Root: HKLM; Subkey: "SOFTWARE\HalPython\InstallPath"; ValueType: string; ValueName: "";
      ValueData: "{app}"; Flags: uninsdeletekey
Root: HKLM; Subkey: "SOFTWARE\HalPython\Version"; ValueType: string;
     ValueName: ""; ValueData: "1.5.1162"; Flags: uninsdeletekey
```

#### 15.4.1.6 section [Code]

Enfin, si certaines opérations à effectuer sont trop complexes pour être décrites par ces sections, il est possible de coder des fonctions, d'ajouter des boîtes de dialogues permettant de choisir des options. Cette dernière possibilité ne sera pas décrite ici.

### 15.4.2 Examples

#### 15.4.2.1 Exemple 1 : installateur pour un module *Python*

L'exemple suivant définit un installateur pour un module *Python* construit avec *Boost Python*. Il inclut également l'installateur de *Python* 2.5 ainsi qu'un éditeur de texte. De ce fait, cet unique installateur inclut tout ce qu'il faut pour pouvoir se servir du module. Le résultat sera un sous-menu dans le menu *Démarrer / Tous les programmes* incluant différents liens dont un ouvrant l'éditeur de texte sur un exemple de programme *Python* utilisant le module installé. Certaines fonctions ont été ajoutés afin que le langage *Python* 2.5 ne soit pas installé si celui-ci l'a déjà été.

Tout texte après un point virgule est considéré comme un commentaire, les instructions des sections [Run], [Tasks], [Registry] sont écrites sur deux lignes pour des raisons de présentation mais doivent l'être sur une seule.

```
; section non commentée
[Setup]
AppName=HalPython
AppVerName=HalPython 1.5.1162
AppPublisher=Xavier Dupré
AppPublisherURL=http://www.xavierdupre.fr/hal_python/hal_python_help_html/index.html
AppSupportURL=http://www.xavierdupre.fr/hal_python/hal_python_help_html/index.html
AppUpdatesURL=http://www.xavierdupre.fr/hal_python/hal_python_help_html/index.html
DefaultDirName={pf}/HalPython
DefaultGroupName=HalPython
OutputDir=D:\Dupre\_data\site\hal_python\executable
OutputBaseFilename=setup_HalPython_py25
Compression=lzma
SolidCompression=yes
VersionInfoVersion=1.0.0

[Code]

; cette fonction retourne le chemin d'installation de Python 2.5
; le résultat est vide si celui-ci n'a pas été installé
; le chemin est celui stocké dans la clé de registre
;           HKLM\Software\Python\PythonCore\2.5\InstallPath
function GetPythonPath(Param: String): String;
begin
  Result := '';
  Result := ExpandConstant('{reg:HKLM\Software\Python\PythonCore\2.5\InstallPath,|}');
  if Result <> '' then
```

```
    Result := Result + '\pythonw.exe';
end;

; cette fonction retourne {win}\system32\msiexec.exe
; si Python 2.5 a été installé, vide sinon
function GetPythonPathExec(Param: String): String;
begin
  Result := GetPythonPath () ;
  if Result <> '' then
    Result := ExpandConstant('{win}\system32\msiexec.exe') ;
end;

; clés de registre
; mémorise le répertoire d'installation
; et le numéro de version
[Registry]
Root: HKLM; Subkey: "SOFTWARE\HalPython\InstallPath"; ValueType: string; ValueName: "";
        ValueData: "{app}"; Flags: uninsdeletekey
Root: HKLM; Subkey: "SOFTWARE\HalPython\Version"; ValueType: string; ValueName: "";
        ValueData: "1.5.1162"; Flags: uninsdeletekey

; icône sur le bureau
[Tasks]
Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}"; GroupDescription: "{cm:AdditionalIcons}";
     Flags: checkedonce

; fichiers
[Files]
; le module en question
Source: "..\hal_dll.dll"; DestDir: "{app}\hal_python";
Source: "..\hal_dll_ext.dll"; DestDir: "{app}\hal_python";
Source: "..\hal_dll_model.dll"; DestDir: "{app}\hal_python";
Source: "..\hal_python.dll"; DestDir: "{app}\hal_python";
Source: "..\wxwindows_old.dll"; DestDir: "{app}\hal_python";
Source: "..\boost_python.dll"; DestDir: "{app}\hal_python";
Source: "..\_graphviz_draw.exe"; DestDir: "{app}\hal_python";
Source: "..\_hal_script.exe"; DestDir: "{app}\hal_python";
Source: "..\hal_python.py"; DestDir: "{app}\hal_python";
Source: "..\install\__init__.py"; DestDir: "{app}\hal_python";
Source: "..\install\setup.py"; DestDir: "{app}";
Source: "..\_test\temp\hal_python.chm"; DestDir: "{app}";

; les fichiers annexes
Source: "hal_python.url"; DestDir: "{app}";    URL de l'application
Source: "pyscripter.chm"; DestDir: "{app}";    aide de l'éditeur PyScripter
Source: "PyScripter.exe"; DestDir: "{app}";    éditeur PyScripter
Source: "python-2.5.2.msi"; DestDir: "{app}";  installateur Python 2.5
Source: "sample.bat"; DestDir: "{app}";        fichier de commande
Source: "sample.py"; DestDir: "{app}";         exemple de programme

; création des icônes
[Icons]
Name: "{group}\PyScripter"; Filename: "{app}\PyScripter.exe"; WorkingDir: "{app}"
Name: "{group}\PyScripter Help"; Filename: "{app}\pyscripter.chm"; WorkingDir: "{app}"
Name: "{group}\Help"; Filename: "{app}\hal_python.chm"; WorkingDir: "{app}"
Name: "{group}\smallest sample with PyScripter"; Filename: "{app}\small_sample.bat"; WorkingDir: "{app}"
Name: "{group}\Website"; Filename: "{app}\hal_python.url"
Name: "{group}\uninstall"; Filename: "{uninstallexe}"; WorkingDir: "{app}"

[Run]
; installe Python 2.5 si GetPythonPathExec retourne un résultat non vide
; passe à l'instruction suivante sinon
Filename: "{code:GetPythonPathExec}"; Parameters: "/i ""{app}\python-2.5.2.msi"" /qr ALLUSERS=1";
          StatusMsg: "Installing Python 2.5..."; Flags: skipifdoesntexist
```

```
; installe le module  si GetPythonPath retourne un résultat non vide
; passe à l'instruction suivante sinon
; exécute en fait l'insttruction python setup.py install
Filename: "{code:GetPythonPath}"; Parameters:"setup.py install"; WorkingDir: "{app}";
          StatusMsg: "Installing HalPython for Python 2.5..."

; supprime l'installation de Python 2.5
Filename: "{cmd}"; Parameters: "/c del python-2.5.2.msi"; WorkingDir: "{app}";
```

Le fichier `sample.bat` contient la ligne suivante :

```
PyScripter.exe sample.py
```

#### 15.4.2.2 Exemple 2 : mise à jour du module

L'installateur suivant assure la mise à jour du module. Il évite ainsi d'inclure dans le setup de gros fichiers comme l'installateur de *Python*ou l'éditeur de texte. Ce fichier de description est presque identique au précédent, les parties ne concernant pas le module ont été supprimées et une fonction `InitalizeSetup` a été ajoutée. Cette dernière est appelée au démarrage de l'installateur, elle vérifie que tous les éléments sont nécessaires au bon fonctionnement du module. En particulier, si *Python* 2.5 n'est pas installé, l'installateur affiche un message puis abandonne.

```
; section non commentée
[Setup]
AppName=HalPython
AppVerName=HalPython 1.5.1162
AppPublisher=Xavier Dupré
AppPublisherURL=http://www.xavierdupre.fr/hal_python/hal_python_help_html/index.html
AppSupportURL=http://www.xavierdupre.fr/hal_python/hal_python_help_html/index.html
AppUpdatesURL=http://www.xavierdupre.fr/hal_python/hal_python_help_html/index.html
DefaultDirName={pf}/HalPython
DefaultGroupName=HalPython
OutputDir=D:\Dupre\_data\site\hal_python\executable
OutputBaseFilename=setup_HalPython_update_py25
Compression=lzma
SolidCompression=yes
VersionInfoVersion=1.0.0

[Code]
function GetPythonPath(Param: String): String;
begin
  Result := '';
  Result := ExpandConstant('{reg:HKLM\Software\Python\PythonCore\2.5\InstallPath,|}');
  if Result <> '' then
    Result := Result + '\pythonw.exe';
end;

function GetHalPythonPath(Param: String): String;
begin
  Result := '';
  Result := ExpandConstant ('{reg:HKLM\Software\HalPython\InstallPath,|}') ;
end;

function InitializeSetup(): Boolean;
begin
  Result := True ;
  MsgBox (GetPythonPath (''), mbConfirmation, MB_OK) ;
  MsgBox (GetHalPythonPath (''), mbConfirmation, MB_OK) ;
  if GetPythonPath ('') = '' then begin
    MsgBox('Python 2.5 has not been installed. You should download
```

```
            the complete Setup with Python 2.5 included instead of updating.', mbError, MB_OK) ;
    Result := False ;
  end else if GetHalPythonPath ('') = '' then begin
    MsgBox('HalPython for Python 2.5 has not been installed.
            You should download the complete Setup with Python 2.5 included instead of updating.',
            mbError, MB_OK) ;
    Result := False ;
  end
end;

[Tasks]
Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}"; GroupDescription: "{cm:AdditionalIcons}";
      Flags: checkedonce

[Files]
Source: "..\hal_dll.dll"; DestDir: "{app}\hal_python";
Source: "..\hal_dll_ext.dll"; DestDir: "{app}\hal_python";
Source: "..\hal_dll_model.dll"; DestDir: "{app}\hal_python";
Source: "..\hal_python.dll"; DestDir: "{app}\hal_python";
Source: "..\wxwindows_old.dll"; DestDir: "{app}\hal_python";
Source: "..\boost_python.dll"; DestDir: "{app}\hal_python";
Source: "..\_graphviz_draw.exe"; DestDir: "{app}\hal_python";
Source: "..\_hal_script.exe"; DestDir: "{app}\hal_python";
Source: "..\hal_python.py"; DestDir: "{app}\hal_python";
Source: "..\install\__init__.py"; DestDir: "{app}\hal_python";
Source: "..\install\setup.py"; DestDir: "{app}";
Source: "hal_python.url"; DestDir: "{app}";
Source: "small_sample.bat"; DestDir: "{app}";
Source: "sample.py"; DestDir: "{app}";
Source: "..\_test\temp\hal_python.chm"; DestDir: "{app}";

[Registry]
Root: HKLM; Subkey: "SOFTWARE\HalPython\InstallPath"; ValueType: string; ValueName: "";
      ValueData: "{app}"; Flags: uninsdeletekey
Root: HKLM; Subkey: "SOFTWARE\HalPython\Version"; ValueType: string; ValueName: "";
      ValueData: "1.5.1162"; Flags: uninsdeletekey

[Icons]
Name: "{group}\PyScripter"; Filename: "{app}\PyScripter.exe"; WorkingDir: "{app}"
Name: "{group}\PyScripter Help"; Filename: "{app}\pyscripter.chm"; WorkingDir: "{app}"
Name: "{group}\Help"; Filename: "{app}\hal_python.chm"; WorkingDir: "{app}"
Name: "{group}\smallest sample with PyScripter"; Filename: "{app}\small_sample.bat"; WorkingDir: "{app}"
Name: "{group}\Website"; Filename: "{app}\hal_python.url"
Name: "{group}\uninstall"; Filename: "{uninstallexe}"; WorkingDir: "{app}"

[Run]
Filename: "{code:GetPythonPath}"; Parameters:"setup.py install"; WorkingDir: "{app}";
     StatusMsg: "Installing HalPython for Python 2.5..."
```

### 15.4.3 Quelques trucs en plus

#### 15.4.3.1 Créer un raccourci

Le code suivant permet d'insérer un raccourci dans le sous-menu inséré dans *Démarrer / Tous les programmes*. La première partie appelle la fonction `GetLnkSciTE` qui crée un raccourci `scite.exe sample.y`. La seconde partie consiste à insérer une ligne permettant d'ajouter le raccourci au sous-menu.

```
[Code]
function GetLnkSciTE(Param:String): String;
begin
    Result := CreateShellLink(ExpandConstant('{app}\SciTE sample.lnk'),
                              'Opens SciTE with a sample',
                              ExpandConstant('{app}\wscite\scite.exe'),
```

```
                                ExpandConstant('"{app}\sample.py"'),
                                ExpandConstant('{app}'),
                                '',
                                0,
                                SW_SHOWNORMAL);
end;

[Icons]
Name: "{group}\smallest sample with SciTE"; Filename: "{code:GetLnkSciTE}"; WorkingDir: "{app}"
```

#### 15.4.3.2 Executer des tâches plus compliquées

Il n'est pas évident d'apprendre encore un nouveau langage tel que celui utilisé par *InnoSetup*. Pour des tâches plus compliquées rien n'empêche d'exécuter un programme *Python* récupérant les paramètres dont il a besoin sur sa ligne de commande. L'exemple suivant permet de remplacer dans un fichier installé une chaîne de caractères par une autre. Le programme prend trois paramètres sur sa ligne de commande : le fichier à modifier, la chaîne de caractères à remplacer et la chaînes qui la remplace.

```
def replace_string (file, s1, s2) :
    f = open (file, "r")
    li = f.readlines ()
    f.close ()
    f = open (file, "w")
    for l in li :
        s = l.replace (s1, s2)
        f.write (s)
    f.close ()

if __name__ == "__main__" :
    import sys
    file = sys.argv [1]  # fichier à modifier
    s1   = sys.argv [2]  # chaîne à remplacer
    s2   = sys.argv [3]  # nouvelle chaîne
    replace_string (file, s1, s2)
```

On cherche à remplacer dans le cas présent la chaîne " = `pythonw`" par " = `c : \python25\pythonw`".

```
[Code]
; retourne le chemin du programme pythonw.exe
function GetPythonPathW(Param: String): String;
begin
  Result := '';
  Result := ExpandConstant('{reg:HKLM\Software\Python\PythonCore\2.5\InstallPath,|}');
  if Result <> '' then
    Result := Result + '\pythonw';
end;

; construit les paramètres de la ligne de commande
; "c:\Program Files\HalPython\replace_conf.py" "=pythonw" "=c:\python25\pythonw"
function GetReplaceConf(Param:String):String;
begin
    Result := ExpandConstant ('"{app}\replace_conf.py"')
    Result := Result + ' ' + ExpandConstant ('"{app}\wscite\python.properties"')
    Result := Result + ' "=pythonw"'
    Result := Result + ' "=' + GetPythonPathW ('') + '"' ;
end;

[Run]
; exécution de la ligne de commande
Filename: "{code:GetPythonPath}"; Parameters:"{code:GetReplaceConf}"; WorkingDir: "{app}";
```

Au final, la ligne de commande exécutée est la suivante :

```
c:\python25\pythonw "c:\Program Files\HalPython\replace_conf.py" "=pythonw" "=c:\python25\pythonw"
```

**Remarque 15.6 : `pythonw.exe` et `python.exe`**
Dans ce cas, c'est le programme exécutable `pythonw.exe` et non `python.exe` qui a été utilisé. Ceci évite l'apparition la fenêtre de la ligne de commande.

## 15.5 MoinMoin : wiki

Un *Wiki* est un outil permettant d'échanger facilement de l'information au sein d'une entreprise ou via Internet. Le *Wiki* le plus connu est sans doute *Wikipedia*. Ce site a pour objectif de créer une encyclopédie que chacun peut modifier. L'intérêt d'un *Wiki* dans une entreprise est de pouvoir créer une sorte d'encyclopédie à usage interne. A cela s'ajoute la possibilité d'inclure des pièces jointes, de garder une trace des différentes modifications apportées, ou de gérer les droits de chaque utilisateur, tout le monde ne pourra pas lire ou écrire ou créer des pages ou certaines pages. L'outil proposé ici est *MoinMoin*, c'est un *Wiki* écrit en langage *Python*.

Un autre avantage d'un *Wiki* est la possibilité d'effectuer une recherche au sein des pages créées dans le *Wiki*. A cette recherche s'ajoute aussi des statistiques comme le nombres de pages créées, le compte de visites de chaque pages, la possibilité d'être informé d'un changement sur une page. Un *Wiki* a pour objectif de faciliter l'accès à l'information d'une entreprise. Pour donner un exemple d'utilisation de *MoinMoin*, je citerai le site *WikiPython*[17] qui propose de nombreux exemples en français sur de nombreux sujets.

Les différentes étapes d'installation et de configuration sont extraites de http://moinmo.in/HelpOnInstalling/ApacheOnWin32 (en anglais). Il est sans doute utile de préciser ce qu'est un script CGI[18]. Ces scripts sont en fait des programmes exécutés par un serveur, ils peuvent exécuter certaines tâches sur le serveur et construire des pages HTML à la volée. Typiquement, lorsque le texte d'une page HTML varie (elle affiche l'heure par exemple), cela signifie qu'elle inclut un script CGI qui insère dans la page l'heure actuelle, c'est-à-dire l'heure à laquelle la page a été demandée. Concrètement, lorsqu'un internaute se connecte à une page HTML :

1. Le serveur reçoit la requête.
2. Le serveur examine le code de la page HTML.
3. Il commence à retourner la page HTML à l'internaute.
4. Lors de ce processus, s'il détecte la présente d'un script, il l'exécute et envoie à l'internaute tout ce que le script produit en terme d'affichage.
5. Le serveur a terminé sa tâche lorsque tous les scripts sont exécutés.

Les scripts CGI peuvent être écrit en de nombreux langages, les plus courants sont *PHP* et *JavaScript*. *Python* fait partie également de la liste des langages utilisés. C'est le cas de *Ruby* aussi.

### 15.5.1 Installation

installation

La première étape consiste à installer un serveur comme le serveur *Apache*, cette installation est identique à celle du paragraphe 15.9 (page 383). L'étape de configuration d'*Apache* requise pour *MoinMoin* pourra s'ajouter à celle effectuée pour *SVN*.

17. http://wikipython.flibuste.net/
18. CGI : Common Gateway Interface

La seconde étape consiste à installer *MoinMoin*[19]. Une fois téléchargé puis décompressé le fichier `MoinMoin1.6.3 tar.gz`[20], il reste à l'installer comme un module *Python* en tapant les instructions suivantes en ligne de commande :

```
C:
cd c:\TEMP\moin-1.6.3
C:\Python25\python setup.py install --record=install.log
```

La dernière étape de l'installation consiste à créer un répertoire dans lequel seront placés toutes les informations relatives au *Wiki*. Cela signifie qu'il est possible de créer plusieurs *Wiki*. Les instructions qui suivent sont à exécuter en ligne de commande, elles supposent que le *Wiki* devra être créer dans le répertoire `c : \Moin` et que *Python* a été installé dans le répertoire `c : \Python25`. Dans le cas contraire, il suffit de les adapter.

```
cd C:\
md Moin
md Moin\mywiki
md Moin\mywiki\data
md Moin\mywiki\underlay
cd c:\Python25\share\moin
xcopy data       C:\Moin\mywiki\data /E
xcopy underlay   C:\Moin\mywiki\underlay /E
copy  config\*.* C:\Moin\mywiki\*.*
copy  server\*.* C:\Moin\mywiki\*.*
```

Il reste à configurer *Apache* et *MoinMoin* pour que les deux applications communiquent. Par la suite, on supposera que le *Wiki* est créé dans le répertoire `c : \Moin` et que *Python* a été installé dans le répertoire `c : \Python25`. Dans le cas contraire, il suffit d'adapter ces lignes.

### 15.5.2 Configuration d'*Apache*

installation Voir paragraphe 15.9.2 page 384.

### 15.5.3 Configuration de *MoinMoin*

Les indications suivantes sont tirées du site officiel de *MoinMoin*[21]. Il y a principalement deux fichiers à modifier, le premier gère l'exécution du code *Python* au sein de *Apache*, le second gère l'apparence de *MoinMoin*.

#### 15.5.3.1 Fichier `moin.cgi`

Ce fichier est disponible dans le répertoire `C : /Moin/mywiki`. Il faut remplacer la ligne :

```
#!/usr/env/bin python
```

par la ligne :

```
#!c:/Python25/python.exe
```

19. Il est disponible à l'adresse : http://moinmo.in/
20. Le site de *MoinMoin* déconseille l'utilisation de *WinZip* pour décompresser le fichier téléchargé mais plutôt *WinRar* ou *7-Zip*.
21. plus précisément de http://moinmo.in/HelpOnConfiguration

Cette ligne spécifie om l'interpréteur *Python* est disponible sur la machine du serveur.

**Remarque 15.7 : pour aller plus loin**
Il est possible, si vous êtes courageux, d'optimiser *MoinMoin* en utilisant d'autres types de CGI [22], notamment les *FastCGI* dont l'exécution est plus rapide qu'un CGI classique. Dans ce cas, il faudra penser à répéter cette manœuvre pour les fichiers du type `moin.*`.

#### 15.5.3.2 Fichier `wikiconfig.py`

Ce fichier définit l'apparence et les droits d'accès de *MoinMoin*. Après l'installation, ce fichier contient de nombreuses lignes commentées concernant les options principales. Tout d'abord, deux options qui permettent à *MoinMoin* de définir l'emplacement du *Wiki* :

```
data_dir = r'C:\Moin\mywiki\data'
data_underlay_dir = r'C:\Moin\mywiki\underlay'
```

Ensuite, il est préférable qu'une personne non identifiée ne puisse pas défaut ni attacher une pièce jointe, ni détruire ou renommer une page, pour cela il faut ajouter la ligne :

```
actions_excluded = ['DeletePage', 'RenamePage', ]
```

On définit ensuite un "superuser", le seul ayant tous les droits sur *MoinMoin*, le nom d'*admim* [23] lui est donné (ce n'est pas une obligation) :

```
superuser = [u"admin", ]
```

Les droits utilisateurs sont définis par défaut par la ligne suivante. Par défaut, le "superuser" peut tout faire :

```
acl_rights_before = u"admin:read,write,delete,revert,admin"
```

Il reste à définir le nom du site et le nom de la première page :

```
sitename = u'NomSite'
page_front_page = u"FirstPage"
```

**Remarque 15.8 : `wikiconfig.py`, paramètre `url_prefix_static`**
Il est préférable de ne pas toucher au paramètre `url_prefix_static`, celui-ci ne semble pas fonctionner complètement. Il est préférable d'utiliser les alias avec *Apache*.

**Remarque 15.9 : `wikiconfig.py`, langue par défaut**
Si les termes anglais vous rebutent, il est possible de faire passer une partie des intitulés en français en modifiant le paramètre suivant :

```
language_default = 'fr'
```

Le *Wiki* est maintenant en état de fonctionner et vous devriez avoir une page correspondant à celle de la figure 15.13.

22. Le rôle des CGI est brièvement expliqué au paragraphe 15.5.
23. pour *administrator*

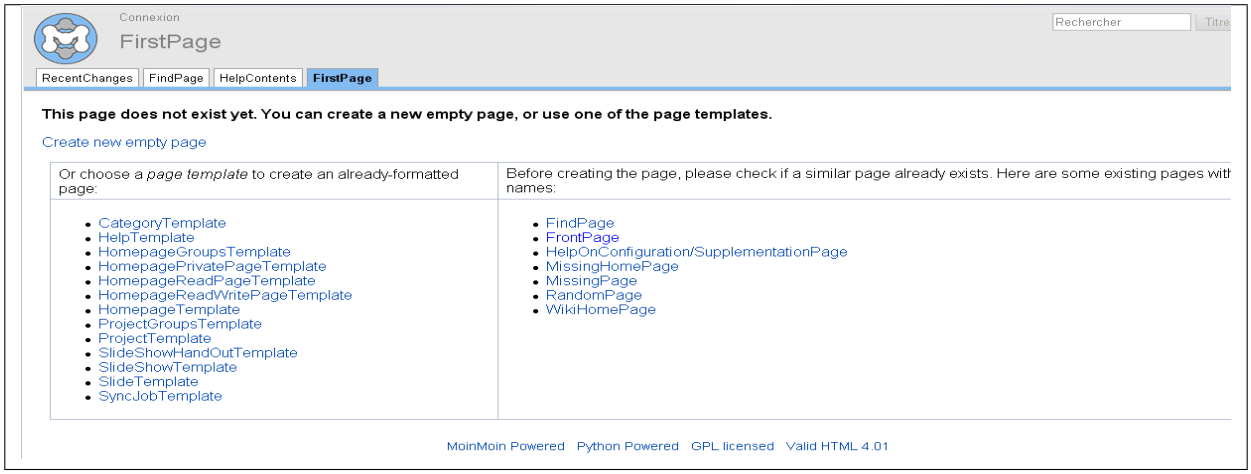



**Figure 15.13 :** *Première page du Wiki une fois configuré. Son contenu est vide car ce dernier n'a pas encore été précisé. C'est aussi ce qui s'affiche lorsqu'une page n'existe pas encore mais que le Wiki y fait référence.*

### 15.5.3.3 Sécurité, création d'utilisateurs

Il est utile de restreindre maintenant les droits de chaque utilisateurs. La première chose à faire est de créer un utilisateurs administrateurs. Pour cela il faut cliquer sur *Connexion* pour atteindre la page de la figure 15.14.

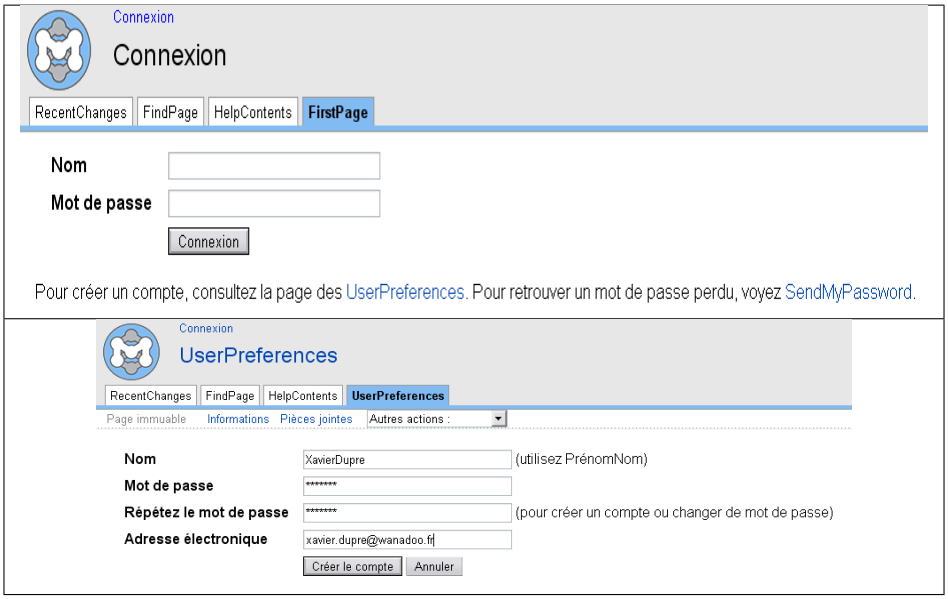



**Figure 15.14 :** *Page de connexion, il faut rentrer un login et un mot de passe ou définir un nouvel utilisateur en cliquant sur UserPreferences si vous n'avez pas encore de compte. Il est déconseillé d'utiliser des accents, ils peuvent provoquer certains comportements erratiques.*

Une fois qu'un utilisateur administrateur est créé, il faut insérer la ligne suivante dans le fichier `wikiconfig.py`. Elle permet de spécifier que seul l'administrateur (ici dupre) a tous les droits, tout autre utilisateur n'a que le droit de lire les pages sans pouvoir les modifier. Au cas om cet utilisateur essaierait, il tomberait sur la page de la figure 15.15.

```
acl_rights_default = u"dupre:read,write,delete,revert,admin All:read"
```

*MoinMoin* offre la possibilité de créer des groupes d'utilisateurs, de se connecter à un annuaire de type

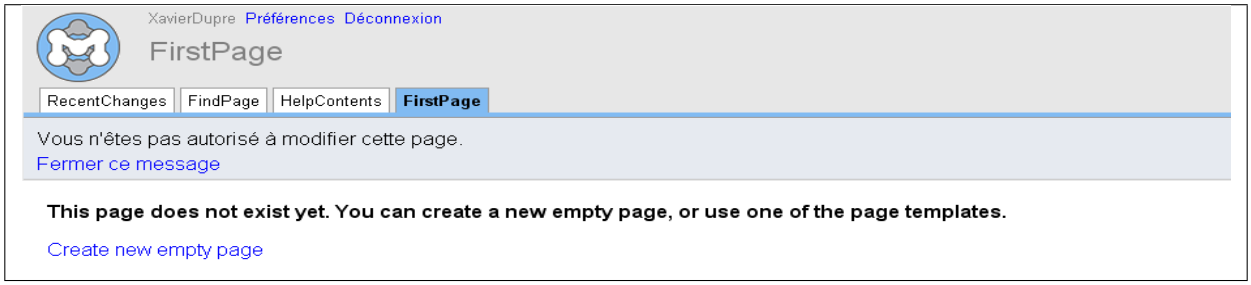



**Figure 15.15 :** *Page qui apparaît lorsqu'un utilisateur essaye de modifier une page mais qu'il n'en a pas le droit.*

*LDAP* ou *ActiveDirectory*. Le site de *MoinMoin* est assez complet et décrit tout ce qu'il est possible de faire avec ce *Wiki*[24] Les étapes de configuration présentées dans ce livre sont assez sommaires et conviennent pour une utilisation au sein d'une entreprise de quelques dizaines d'employés tout au plus et sur un réseau interne. Pour des réseaux de plus en grandes envergures, il est conseillé de consulter le site de *MoinMoin*, voire d'opter pour un `Wiki` différent.

#### 15.5.3.4 Apparence

Il est possible de modifier l'apparence du site *Wiki*, la variable `theme_default` permet de définir l'aspect général. Il existe trois thèmes prédéfinis `'classic'`, `'modern'` et `'rightsidebar'`.

```
theme_default = 'modern'
```

Il est aussi possible de développer son propre thème[25] mais cela requiert un peu de temps ou aller télécharger un thème existant[26] en prenant toutefois soin de vérifier que ce thème fonctionne avec la version de *MoinMoin* installée.

### 15.5.4 Utilisation sur un réseau d'entreprise

Aucune autre configuration n'est a priori nécessaire pour utiliser ce *Wiki* sur le réseau d'une entreprise. La donnée de l'adresse IP suffit à se connecter au *Wiki* installé sur une autre machine. Si cela ne fonctionnait pas, les deux remarques 15.3 et 15.4 page 15.4 devraient vous résoudre à résoudre ce problème.

### 15.5.5 Ecrire des pages

#### 15.5.5.1 Création d'une page

Le *Wiki* est maintenant configuré, il reste encore savoir comment écrire quelques pages à commencer par la première qui est à pour le moment l'allure de celle de la figure 15.13. Il suffit de cliquer sur *Create an empty page* pour afficher l'écran de la figure 15.16. Le *Wiki* ne manipule que du texte, toute image doit être traitée comme une pièce jointe. La syntaxe *Wiki*, simple, permet tout de même d'afficher des titres, des tableaux, des listes, une table des matières[27]. La figure 15.17 donne un court exemple de qu'il est possible de faire avec *MoinMoin*.

24. Tout est y est écrit en anglais exclusivement.
25. voir à l'adresse http://moinmo.in/MoinDev/ThemeDevelopment
26. voir à l'adresse http://moinmoin.wikiwikiweb.de/ThemeMarket
27. voir un résumé en français de la syntaxe à l'adresse http://moinmo.in/HelpOnMoinWikiSyntax.




**Figure 15.16 :** *Page qui apparaît lorsqu'un utilisateur essaye de modifie une page.*




**Figure 15.17 :** *D'un côté, la syntaxe MoinMoin, de l'autre le rendu dans un navigateur.*

### 15.5.5.2 Ajouter des liens

On peut considérer deux types de liens à l'intérieur d'un *Wiki* : les liens externes qui sont une simple adresse internet et les liens internes qui font référence à une page du *Wiki*. Pour ajouter un lien externe à une page, il suffit de l'incoroporer au texte de la page ou entre double crochets si on veut que ce lien apparaisse avec un autre texte.

```
http://www.xavierdupre.fr/mywiki/XavierDupre
[[http://www.xavierdupre.fr/mywiki/XavierDupre|wiki de Xavier Dupré]]
```

Les liens internes sont des références à des pages déjà écrites ou des pages à créer si le *Wiki* détecte que ce lien interne n'existe pas. Un lien interne est un mot contenant deux ou plusieurs majuscule comme `LienInterne` ou une phrase entre crochets comme [[`lien interne`]].

#### 15.5.5.3 Ajouter des pièces jointes

L'ajout d'une pièce jointe s'effectue en cliquant sur l'intitulé *Pièce jointe*. Il suffit ensuite d'ajouter le texte `attachment : nom_de_la_piece_jointe` pour y donner accès à tous les lecteurs de la page.

#### 15.5.5.4 Droits spécifiques à une page

Il est possible d'attribuer des droits spécifiques à une page. Le paramètre `acl_rights_before` de `wikiconfig.py` (voir paragraphe 15.5.3.2) définit les droits par défaut de chaque page du *Wiki*. Il est néanmoins possible de restreindre l'accès à certaines pages du *Wiki*. Par exemple, par défaut, tout le monde peut lire chaque page mais on voudrait que chaque utilisateurs est un espace réservé inaccessible à quiconque autre que lui. Dans ce cas, on ajoutera au début du texte de cette page le texte suivant :

```
#acl dupre:read,write
```

Dans ce cas, personne d'autre que l'utilisateur `dupre` ne pourra avoir accès ni en lecture ni en écriture à la page où est insérée cette ligne. Cette ligne écrase les droits par défaut. Le texte de la page peut ensuite être écrit comme n'importe quelle autre page.

### 15.5.6 Script CGI

#### 15.5.6.1 Script simple

configuration CGI *MoinMoin* fonctionne à base de script CGI : ce sont des pages HTML entièrement produites par des scripts écrits en *Python*. Exécuté par le ssrveur *Apache*, tout instruction `print` d'un script CGI sera automatiquement envoyé au navigateur. Les lignes qui suivent donne un exemple d'introduction à cette technique. Il commence par la configuration du serveur *Apache* auquel on indique un nouveau répertoire dans lequel seront placés des script CGI écrits en *Python*.

```
<Directory "C:/Program Files/Apache Software Foundation/Apache2.2/python_cgi">
    Options FollowSymLinks
    AllowOverride None
    Order deny,allow
    Allow from all
</Directory>

Alias /pycgih "C:/Program Files/Apache Software Foundation/Apache2.2/python_cgi"
ScriptAlias /pycgi "C:/Program Files/Apache Software Foundation/Apache2.2/python_cgi"
```

Dans ce répertoire, on placera des pages HTML et des scripts CGI. Pour éviter de longues adresses URL, on définit deux alias différents mais dont la cible est le même répertoire `python_cgi`. *Apache* n'accepte pas que deux directives `Alias` et `ScriptAlias` portent le même nom.

```
#!c:/Python25/python.exe
# ligne 1 : emplacement de python (obligatoire)
# ligne 2 : la page ne contiendra que du texte
print "Content-Type: text/plain\n\n"
# premier message au navigateur
print "premier message\n"
```

Il faut maintenant créer une page HTML qui pointe vers ce script :

```
<html><body>
<h1> Lien vers un script CGI écrit en Python </h1>
<a href="/pycgi/hello.cgi">hello.cgi</a>
</body></html>
```

Il ne reste plus qu'à écrire l'adresse de cette page dans le navigateur :

```
http://localhost/pycgih/index.html
```

La page affichée fait apparaître un titre suivi d'un unique lien vers le script CGI. Cliquer dessus affiche le message souhaité.

**Remarque 15.10 : erreur, première ligne** #!c : /Python25/python.exe
Il ne faut pas oublier cette première ligne qui indique à *Apache* où trouver l'interpréteur *Python* qui lui permettra de comprendre le script. Dans le cas contraire, la page aboutit à une erreur. Si cette erreur subsiste toujours même après l'ajout de cette ligne, cela signifie que l'interpréteur *Python* n'est pas installé à cet endroit où que le script a produit une erreur. Dans ce cas, il peut être utile de remplacer l'extension `cgi` par `py` puis d'exécuter le script CGI comme s'il était un programe *Python*.

#### 15.5.6.2 Script HTML

plain

Le paragraphe précédent a présenté un script CGI permettant d'afficher du texte générée par un script *Python*. Pour mettre en forme ce texte, il faut utiliser la syntaxe HTML et préciser depuis le script que le format de sortie du script est bien du HTML. Pour cela, il faut modifier une ligne du script CGI, `html` remplace `plain` :

```
print "Content-Type: text/html\n\n"
```

Par exemple :

```
#!c:/Python25/python.exe
# -*- coding: cp1252 -*-
print "Content-Type: text/html\n\n"
# premier message au navigateur,
# les passages à la ligne \n sont sans effet à l'affichage
print "<html><body>\n"
print "<b>premier message en gras</b>\n"
print "</body></html>"
```

**Remarque 15.11 : les accents dans un script CGI**
L'instruction # −∗− `coding` : `cp1252` −∗− est toujours nécessaire lorsque le script CGI contient des accents. Dans le cas contraire, l'interpréteur *Python* déclenche une exception et le serveur affiche un message d'erreur.

#### 15.5.6.3 Attraper les exceptions

Lorsqu'une erreur survient, le texte affiché correspond au traitement effectué par le script CGI jusqu'à ce que l'erreur se soit produite. Pour pouvoir remonter jusqu'à la cause de l'erreur, il est possible d'attraper l'exception engendré par le script et d'afficher les informations relatives à cette erreur. Le script suivant génère volontairement une erreur en exécutant une instruction inconnue `stop`. L'exception est alors attrapée puis on utilise le module `traceback` pour afficher la ligne et le type d'erreur généré.

```
#!c:/Python25/python.exe
# -*- coding: cp1252 -*-
print "Content-Type: text/html\n\n"
print "<html><body>\n"
try:
    print "<b>premier message en gras</b>\n"
```

```
    print "<br>déclenchement d'une erreur (instruction inconnue)<br>"
    stop
except:
    import sys
    import traceback
    print "<pre>"
    traceback.print_exc(file=sys.stdout)
    print "</pre>"
print "</body></html>"
```

Le navigateur affiche le message suivant :

```
Traceback (most recent call last):
  File "C:/Program Files/Apache Software Foundation/Apache2.2/python_cgi/hello4.cgi", line 8, in
    stop
NameError: name 'stop' is not defined
```

**Remarque 15.12 : module `traceback` et erreur de syntaxe**
Ce procédé ne protège pas le script contre les erreurs de syntaxe, un guillement manquant, une mauvaise indentation ou une parenthèse mal fermée.

#### 15.5.6.4 Résumé de la syntaxe HTML

Il peut être utile de décrire très succintement la syntaxe HTML. Ce langage est un langage à balise, c'est-à-dire que toute information est insérée au sein d'une syntaxe qui ressemble à :

```
<balise>  ....  </balise>
```

Les balises peuvent être écrites avec des minuscules ou des majuscules, le langage HTML ne fait pas la différence. Toute page HTML s'écrit de la forme :

```
<html>
<head>
.... entête de la page, n'est pas affiché, il est possible d'y définir
.... des feuilles de styles pour un affichage amélioré
.... ou la définition de nouvelles balises
.... la ligne
<meta content="text/html; charset=ISO-8859-1" http-equiv="content-type">
.... permet de définir un jeu de caractères incluant les accents français
</head>
<body>
.... contenu de la page
</body>
</html>
```

Le contenu de la page doit figurer entre les balises < `body` > et < /`body` >. La table 15.1 regroupe les balises de style les plus utilisées et la table 15.2 des balises courantes de présentations. Les balises citées dans ces tables permettent de présenter des résultats au format HTML. Pour des présentations plus sophistiquées, une recherche sur Internet donnera la plupart du temps une réponse [28]. Le langage étant en perpétuelle évolution, les possibilités offertes ne cessent de s'étendre.

#### 15.5.6.5 Appeler une page avec des paramètres

Les programmes *Python* sont capables d'accepter des paramètres lorsqu'ils sont exécutés en ligne de commande. Ce mécanisme existe également en ce qui concernent les pages HTML. Cela consiste à faire un URL d'un point d'interrogation puis d'une suite de couple `param` = `valeur` séparé par le symbole &.

28. voir également http://fr.wikipedia.org/wiki/HTML

| balises | effet |
|---|---|
| <! − − − − > | commentaires, ces informations ne sont pas affichées par le navigateur |
| < h1 >< /h1 > | titre de premier niveau |
| < h2 >< /h2 > | titre de second niveau |
| ... | ... |
| < br > | aller à la ligne |
| < hr > | tracer une ligne horizontale |
| < b >< /b > | gras |
| < i >< /i > | italique |
| < ins >< /ins > | souligné |
| < del >< /del > | barré |

**Table 15.1 :** *Balises HTML relatives à la mise en forme de caractères.*

| balises | effet |
|---|---|
| < ahref = "url" > texte < /a > | lien, url désigne l'hyperlien, le texte à afficher par le navigateur est situé entre les balises |
| < aname = "ancre" >< /a > | poser une ancre |
| < ahref = "#ancre" > texte < /a > | lien vers une ancre à l'intérieur d'une page |
| < imgsrc = "imagename" >< /img > | lien vers une image, les attributs width et height pour modifier la taille de l'image lorsqu'elle est affichée. |
| < p >< /p > | paragraphe |
| < table >< /table > | insertion d'un tableau, l'attribut border permet de préciser l'épaisseur des bords. < tableborder = "1" > |
| < tr >< /tr > | insertion d'une ligne dans un tableau |
| < td >< /td > | insertion d'une case dans un tableau |
| < td >< /td > | insertion d'une case dans un tableau |
| < ul >< /ul > | création d'une liste |
| < li > | insertion d'un intitulé au sein d'une liste |

**Table 15.2 :** *Balises HTML relatives à la mise en page. Le nom du fichier suit le mot* src *dans la syntaxe,* < imgsrc = "..." >< /img >, src *est appelé un attribut. les valeurs associées aux attributs sont toujours entourées de guillemets. Lorsqu'il n'y a rien entre deux balises, l'écriture peut être écourtée en* < imgsrc = "..."/ >.

```
http://URL?param1=valeur1&param2=valeur2&...
```

Le programme suivant récupère la valeur de ces paramètres.

```
#!c:/Python25/python.exe
# -*- coding: cp1252 -*-
print "Content-Type: text/html\n\n"
print "<html><body>\n"
import cgi
par = cgi.FieldStorage()
if len (par) == 0 :
    print "<b> aucun paramètre à cette page </b>"
else :
    print "<b>liste des paramètres </b><br>"
    print """<table border="1">"""
    for p in par :
        print "<tr>"
```

```
        name  = str (p)
        value = par.getvalue (name)
        print "<td>", name, "</td><td>", value, "</td>"
        print "</tr>"
    print "</table>"
print "</body></html>"
```

#### 15.5.6.6 Formulaires

Le langages HTML donne la possibilité de construire des formulaires puis de récupérer les données saisies par l'internaute. Bien souvent, la page qui affiche le formulaire est un script CGI capable de traiter les données qu'il s'envoie lui-même. L'utilisateur sélectionne l'adresse :

```
http://localhost/pycgi/pyform.cgi
```

Puis le formulaire, lorsque l'utilisateur appuie sur *Envoyer*, appelle l'URL suivant :

```
http://localhost/pycgi/pyform.cgi?nom=dupre&prenom=xavier
```

C'est ce que fait le script CGI suivant. Il affiche d'abord le formulaire de la figure 15.18.

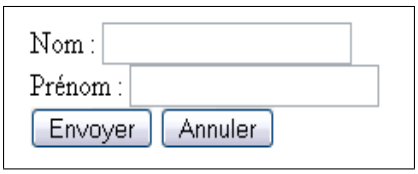



**Figure 15.18 :** *Exemple de formulaire au format HTML engendré par un script CGI écrit en Python.*

```
#!c:/Python25/python.exe
# -*- coding: cp1252 -*-
print "Content-Type: text/html\n\n"
print "<html><body>\n"
def print_form () :
    # création du formulaire avec deux champs nom et et prenom
    print """
    <FORM action="/pycgi/pyform.cgi" method=post>
    <P>
    <LABEL for="nom">Nom : </LABEL> <INPUT type="text" name="nom"><BR>
    <LABEL for="prenom">Prénom : </LABEL> <INPUT type="text" name="prenom"><BR>
    <INPUT type=submit value="Envoyer"> <INPUT type=reset value="Annuler">
    </P>
    </FORM>
    """

import cgi
par = cgi.FieldStorage ()
if len (par) == 0 :
    # s'il n'y a aucun paramètre, on affiche le formulaire
    print_form ()
else :
    # sinon, cela signifie que le formulaire a rappelé le script CGI
    # avec des paramètres et le traitement suit une direction différente
    # ici, cela consiste à afficher les informations reçues en gras
    nom = par.getvalue ("nom")
    prenom = par.getvalue ("prenom")
    print "Votre nom est <b>", nom, "</b> et votre prénom <b>", prenom, "</b>"
print "</html></body>\n"
```

## 15.6 SQL

Le langage SQL est devenu incontournable pour quiconque souhaite stocker, consulter de grandes quantité de données. Les paragraphes qui suivent ne feront que présenter sommairement le langage SQL, il s'agit ici de l'utiliser via un programme *Python* plutôt que de le maîtriser.

### 15.6.1 Les bases de données

Le langage SQL est lié à ce qu'on appelle les *bases de données*. C'est simplement un ensemble de tableaux ou plutôt *table*. Une table est définie par un nombre de colonnes, chacune porte un nom et ne contient qu'un type donné d'information. Par exemple, lorsqu'on décrit un élève dans le système informatique d'une école, on retiendra par exemple :

| information | nom | type |
|---|---|---|
| un numéro | num | nombre entier |
| son nom | nom | chaîne de caractères (30 maximum) |
| son prénom | prenom | chaîne de caractères (30 maximum) |
| sa date de naissance | date | date |
| son adresse | adresse | chaîne de caractères (100 maximum) |
| son genre | genre | booléen |
| sa nationalité | codepays | un nombre entier |
| sa classe | classe | nombre entier |

Cette table, qu'on appelle `ELEVE`, sera représentée sous la forme d'un tableau. Le numéro permet de désigner chaque élève plus facilement dans le système informatique que par son nom et son prénom, il permet aussi de distinguer portant même nom et prénom. Ce numéro porte souvent le nom d'*index*.

| num | nom | prenom | date | adresse | genre | codepays | classe |
|---|---|---|---|---|---|---|---|
| 1 | dupre | xavier | 11/08/1975 | ... paris | True | 33 | 19 |
| 2 | dupre | gilles | 24/12/1946 | ... charleville | True | 33 | 56 |

Le pays est entier, on aurait pu choisir une chaîne de caractères également. Toutefois, il est possible de définir une seconde table qu'on appelle `PAYS` avec la liste des pays et le numéro qui leur sont attribués.

| pays | codepays |
|---|---|
| France | 33 |
| Royaume-Uni | 44 |

Lorsqu'on cherche des élèves d'une certaine nationalité sans connaître son code, il suffit de chercher d'abord le code dans la table `PAYS` puis de chercher les élèves ayant ce code dans la table `ELEVE`. En langage SQL [29], la première requête s'ecrit :

```
SELECT codepays
FROM PAYS
WHERE pays = 'France'
```

La seconde requête utilise les résultats de la première, en lanage SQL, cela donne :

```
SELECT num,nom,prenom
FROM ELEVE
WHERE codepays IN (
                    SELECT codepays
                    FROM PAYS
                    WHERE pays = 'France'
                  )
```

A ces deux tables, on ajoute une table des matières enseignées :

29. Il n'est pas tenu compte des passages à la ligne, ils apportent plus de lisibilité.

| information | nom | type |
|---|---|---|
| numéro | num | nombre entier |
| matière | matiere | chaîne de caractères (30 maximum) |
| niveau | niveau | nombre entier |

Puis une table des notes attribuées :

| information | nom | type |
|---|---|---|
| numéro élève | nume | nombre entier |
| numéro matière | numm | nombre entier |
| note | note | nombre réel |

Si on relie les colonnes portant la même information, on aboutit au graphe de la figure 15.19. Ces liens illustrent le concept de *base de données relationnelle*. Il serait possible de représenter toutes ces données sous la forme d'un unique tableau mais il aurait beaucoup de colonnes et autant de lignes que de notes. Autrement dit, les mêmes informations sont présentes en de nombreux endroits. Pour éviter la redondance d'un tel système, il faut scinder ce tableau en autant de tables que nécessaire pour faire disparaître la la redondance.

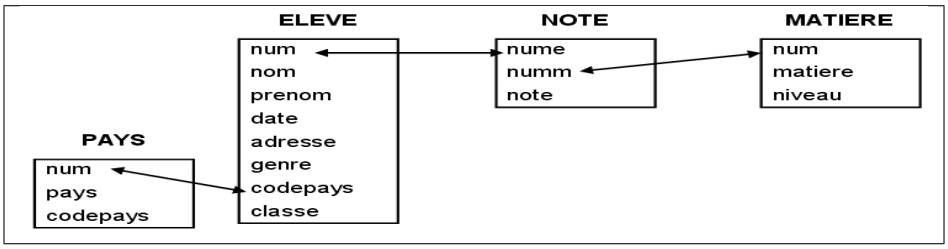



**Figure 15.19 :** *Graphe illustrant les bases de données relationnelles, les champs reliés entre eux ont le même sens. Un numéro d'élève désigne aussi bien un élève dans la table* `ELEVE` *que dans la table* `NOTE`*. Si on connaît le numéro d'un élève, on connaît également toutes les autres informations le concernant, c'est pourquoi, dans la table* `NOTE`*, on inscrit que son numéro. Toute autre information serait superflue.*

Le langage SQL permet de recherche des informations dans cet ensemble de tables. Par exemple, la requête suivante permet de construire la liste des élèves suivi de la moyenne de leurs note en français :

```
SELECT nom,prenom,AVG(note) FROM ELEVE,NOTE
WHERE num = nume and
      numm IN ( SELECT num FROM MATIERES WHERE matiere = 'français' )
GROUP BY nume
```

Le chapitre suivant montrera comment créer ces différentes tables, y insérer des lignes (ou enregistrements), lancer des requêtes.

### 15.6.2 Utiliser une base de données

La façon la plus simple d'utiliser une base de données est d'utiliser l'application *SQLite*[30]. C'est d'autant plus avantageux que ce module est fourni avec l'installation standard de *Python*. La base de données, qui est un ensemble de plusieurs tables, est en fait stockés dans un unique fichier. On lit et on écrit dans ce fichier grâce au module `sqlite3`. Et l'unique moyen d'écrire, de lire, de consulter est d'utiliser des requêtes SQL. Par exemple, la création de la table `ELEVE` se fait grâce à la requête suivante :

```
CREATE TABLE ELEVE (
     num integer primary key,
     nom varchar (30),
     prenom varchar (30),
```

30. http://www.sqlite.org/

```
     date date,
     adresse varchar (100),
     codepays integer,
     classe integer)
```

Le mot-clé `primary key` permet d'indiquer au langage SQL qu'il peut utiliser ce champ comme un index : il n'y aura jamais deux élèves avec des numéros identiques. C'est aussi une clé qui est attribuée automatiquement par le langage SQL. Si une table contient un tel champ, sa consultation est plus rapide. Il ne reste plus qu'à exécuter cette requête grâce au module `sqlite3` :

```
import sqlite3 as SQL
cx = SQL.connect("madatabase.db3")
cur = cx.cursor()

cur.execute ("""
CREATE TABLE ELEVE (
     num integer primary key,
     nom varchar (30),
     prenom varchar (30),
     date date,
     adresse varchar (100),
     codepays integer,
     classe integer)
""")
```

La première instruction permet d'ouvrir une connexion à la base de données, d'indiquer qu'on s'en sert. La seconde instruction permet de créer ce qu'on appelle un curseur. Cela permet d'avoir un accès à la base qui autorise des accès simultanés en nombre limité. La troisième instruction exécute la requête proprement dite.

```
cur.execute ("CREATE TABLE NOTE (nume integer, numm integer, note double)")
cur.execute ("CREATE TABLE MATIERES (num integer, matiere varchar (30))")
cur.execute ("CREATE TABLE PAYS (codepays integer, pays varchar (30))")
```

Il faut maintenant insérer des valeurs dans chacune des tables :

```
import sqlite3 as SQL
cx = SQL.connect("madatabase.db3")
cur = cx.cursor()

cur.execute ("""
CREATE TABLE ELEVE (
     num integer primary key,
     nom varchar (30),
     prenom varchar (30),
     date date,
     adresse varchar (100),
     codepays integer,
     classe integer)
""")
```

Il ne faut pas oublier l'instruction `cx.commit()` qui met à jour la base de données. Sans cette instruction, les tables ne sont pas modifiées et pendant l'exécution de cette instruction, toutes les autres requêtes sont momentanément interrompues le temps que la modifications de la base soit effective. Le programme suivant permet de vérifier que les modifications ont bien été prises en compte :

```
import sqlite3 as SQL
cx = SQL.connect("madatabase.db3")
```

```
cur = cx.cursor()
cur.execute("select * from ELEVE")
for row in cur.fetchall(): print row
```

Il affiche [31] :

```
(1, u'dupre', u'xavier', u'11/08/1975', u'---- paris', 33, 19)
(2, u'dupre', u'gilles', u'24/12/1946', u'---- charleville', 33, 56)
```

Il reste à insérer d'autres valeurs dans ces tables [32] :

```
cur.execute ("INSERT INTO PAYS (codepays, pays) VALUES (33, 'France')")
cur.execute ("INSERT INTO PAYS (codepays, pays) VALUES (44, 'Royaume-Uni')")
cur.execute ("INSERT INTO MATIERES (matiere, num) VALUES ('français', 1)")
cur.execute ("INSERT INTO MATIERES (matiere, num) VALUES ('mathématiques', 2)")
cx.commit ()
```

Il ne reste plus qu'à ajouter des notes :

```
note = [(1, 1, 12), (1, 1, 14), (1, 2, 16), (1, 2, 8), (1, 2, 12), \
        (2, 1, 8),  (2, 1, 9),  (2, 2, 11), (2, 2, 8), (2, 2, 12)]
for n in note :
    req = "INSERT INTO NOTE (nume,numm,note) VALUES " + str (n)
          cur.execute (req)
cx.commit ()
```

Le programme suivant permet de déterminer la liste des prénoms des élèves suivi de la moyenne de leurs notes :

```
req = """
SELECT nom,prenom,AVG(note) FROM ELEVE,NOTE
WHERE num = nume and
      numm IN ( SELECT num FROM MATIERES WHERE matiere = 'français' )
GROUP BY nume
"""
cur.execute (req)
for row in cur.fetchall(): print row
```

Le résultat de la requête donne pour notre exemple :

```
(u'dupre', u'xavier', 13.0)
(u'dupre', u'gilles', 8.5)
```

En ajoutant une condition avec le mot-clé `HAVING`, il est possible de ne retourner que la liste des élèves ayant la moyenne :

```
req = """
SELECT nom,prenom,AVG(note) FROM ELEVE,NOTE
WHERE num = nume and
      numm IN ( SELECT num FROM MATIERES WHERE matiere = 'français' )
GROUP BY nume
HAVING AVG(note) >= 10
"""
cur.execute (req)
for row in cur.fetchall(): print row
```

31. Le symbol u devant les chaînes de caractères indique qu'elles sont codées au format unicode et que les accents sont pris en compte.

32. En langage SQL, le mot-clé `UPDATE` sert à modifier un enregistrement.

### 15.6.3 Utiliser un serveur de base de données

Le module `sqlite3` propose en quelque sorte de stocker des informations de façon plus intelligente qu'un simple fichier texte puisqu'il met à disposition des outils permettant de consulter rapidement ces informations sans avoir à lire intégralement les données pour les mettre dans la mémoire de l'ordinateur au travers de listes par exemple. Toutefois, les informations sont localisées dans un unique fichier et elles ne sont pas faciles à partagées à travers un réseau.

Un serveur de bases de données offre les mêmes services au niveau du stockage et de la consultation des données mais il permet de partager ces informations avec quiconque a accès à ce serveur. Le serveur de base de données organise la mise en commun de grandes volumes d'informations. Un serveur de bases de données est un serveur au même titre que l'application *Apache* : c'est une application qui tourne en permanence, de préférence sur un ordinateur entièrement dédié à ce service, et qui attend les requêtes des utilisateurs provenant de tous les utilisateurs ayant la possiblité de se connecter à ce serveur.

#### 15.6.3.1 Installer un serveur de base de données sous *Windows*

installation

Les paragraphes qui suivent décrivent l'installation de serveur *MySQL*. Il est parfois plus simple d'utiliser l'application *EasyPHP*[33] qui en simplifie l'installation lorsqu'elle est couplé avec le serveur *Apache* (voir également le paragraphe 15.9).

Il existe de nombreux serveurs de base de données. Les plus connus sont *Microsoft SQL Server*, *Oracle*, *Sybase*. Parmi les serveurs gratuits, on trouve *MySQL*, ou encore *PostGreSQL*. Ces applications sont souvent regroupées sous le terme de *SGBD* ou encore *Système de gestions de base de données*.

L'installation de *MySQL*. n'est pas très compliquée. Il faut dans un premier temps téléchercher *MySQL*[34] puis suivre les instructions de la figure 15.20. Il est tout de même possible de changer les paramètres de la configuration sans avoir à refaire l'installation, il suffit d'aller dans le sous-menu ajouter par *MySQL* dans le menu *Démarrer / Tous les programmes*.

33. http://www.easyphp.org/
34. http://dev.mysql.com/downloads/mysql/5.0.html#downloads

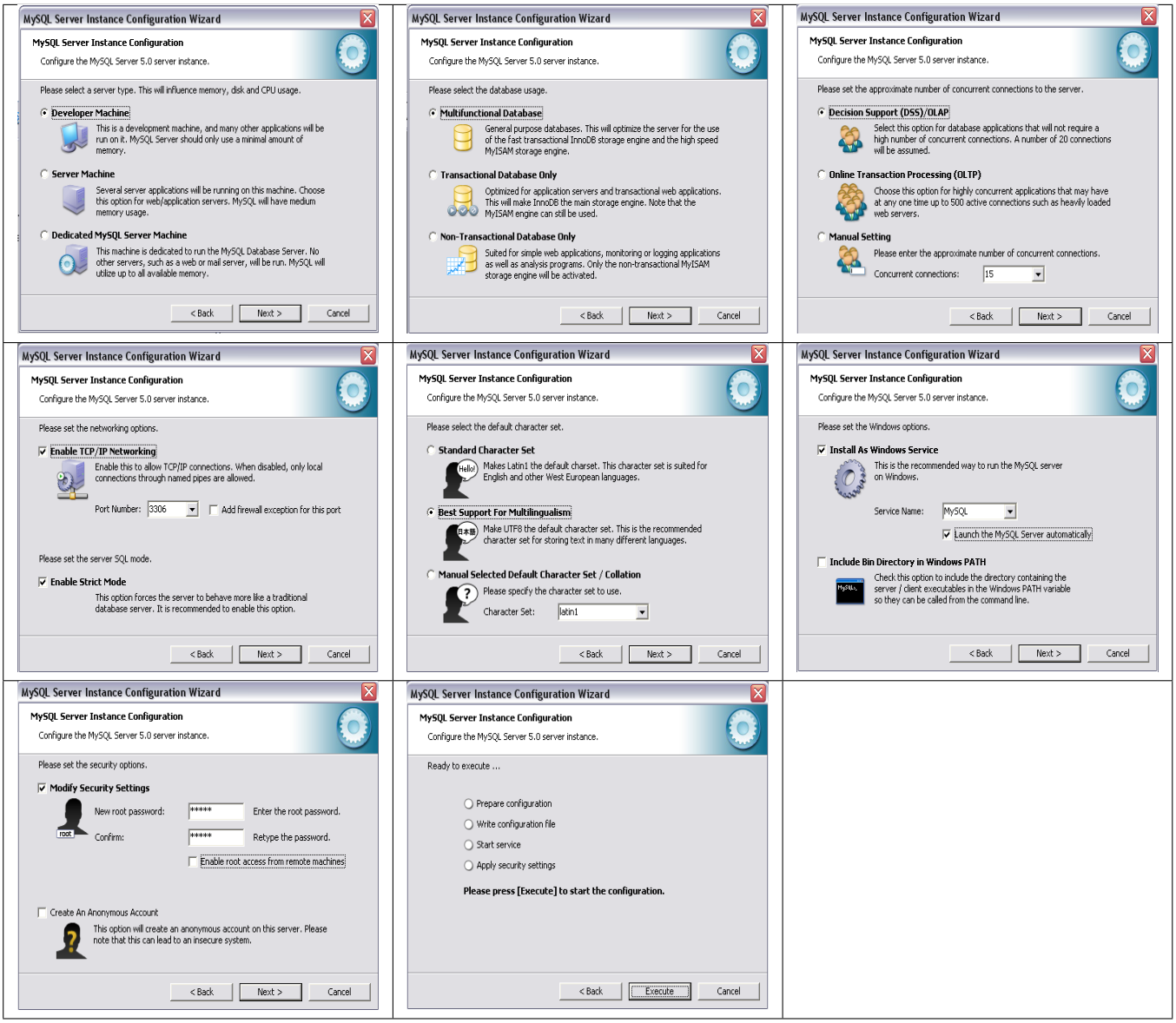



**Figure 15.20 :** *L'installation de MySQL fait intervenir une petite dizaine d'écrans qui se succèdent. Chacun d'eux propose quelques options d'installation que l'utilisateur doit choisir. Voici les choix pour une utilisation sur une machine de bureau qui n'est pas exclusivement réservé à MySQL et qui autorise une dizaine de connexion simultanée. Cette base n'est donc pas configurée pour un usage intense. Les derniers écrans font référence à la sécurité. Il est conseillé de définir un mot de passe root ou administrateur. C'est le seul utilisateur ayant tout pouvoir sur la gestion du système.*

Il faut maintenant installer un second morceau qui est le pilote *ODBC* pour *MySQL*[35]. Il n'y a cette fois-ci rien à configurer et tout se passe presque sans intervention. Cette seconde étape est facultative mais elle rend l'écriture de programmes plus facile et surtout indépendante du serveur de base de données choisie.

Avant d'utiliser *Python* pour lancer des requêtes, le serveur nouvellement installé est entièrement vide et il faut d'abord créer ce qu'on appelle une *database*. C'est principalement un ensemble de tables. Pour cela, il faut cliquer sur l'intitulé *MySQL Command Line client* du menu *Démarrer / Tous les programmes / MySQL / MySQL Server 5.0*. Il vous est d'abord demandé un mot de passe, celui renseigné lors de l'installation. Ensuite il faut taper la ligne suivante :

35. Il est téléchargeable à l'adresse http://dev.mysql.com/doc/refman/5.0/fr/myodbc-windows-binary-installation.html.

```
CREATE DATABASE datatest ;
```

L'instruction ne devrait pas retourner d'erreur et à partir de ce moment, la database `datatest` est créée. Il sera alors possible d'y connecter un pilote ODBC pour s'en servir dans un programme *Python*.

Il reste une dernière installation à effectuer, celle du module `pywin32` [36] qui permet d'utiliser une connexion ODBC. Son installation utilise un programme exécutable et ne requiert aucune intervention.

#### 15.6.3.2 Configurer une connexion ODBC sous *Windows*

Il existe de nombreux modules *Python* qui permet de se connecter directement à un serveur de base de données particulier comme le module `MySQLdb` [37] qui accède au serveur *MySQL*. Même si cet accès est le plus rapide, l'inconvénient de ce système est qu'il faut utiliser un module différent pour chaque serveur de bases de données [38].

Il existe un moyen d'homogénéiser les accès aux serveurs en ajoutant une couche intermédiaire appelée *ODBC* ou *Object Database Connectivity*. Grâce à ce système, le code nécessaire à l'utilisation d'un serveur ne dépend plus du système de base de données utilisés. Celui-ci peut être *MySQL* ou *Microsoft SQL Server* ou encore un fichier texte, *Excel*, une base de données *Access*. Il suffit pour cela de déclarer ce pilote ODBC au système d'exploitation *Windows*. Il faut pour cela aller dans le panneau de configuration, icône *Outils d'administration* puis choisir *Database ODBC*. On tombe alors sur une fenêtre du type de celle présentée à la figure 15.21. La configuration du pilot ODBC suit les instructions de la figure 15.22.

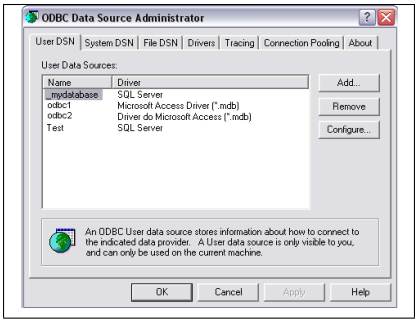



**Figure 15.21 :** *Fenêtre gérant les accès ODBC sous Windows. La fenêtre montre quatre connexions ODBC disponibles de types différents. Il suffit de cliquer sur le bouton Add pour ajouter une connexion ODBC et ouvrir un accès à une nouvelle source, que ce soit un fichier texte, un fichier Excel, un serveur, ...*

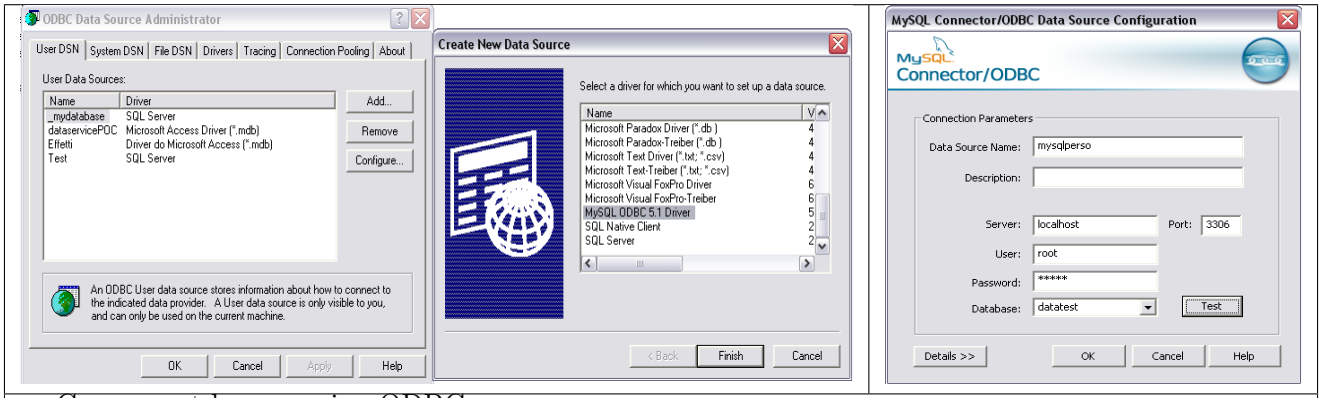
#### 15.6.3.3 Utilisation d'un serveur de base de données

En utilisant une connexion ODBC, les lignes écrites au paragraphe 15.6.2 ne diffèrent excepté celle qui ouvre la connexion ODBC qui s'effectue avec un module différent. Le module à utliser devient `odbc`. La création d'une table fait appel au script suivant :

```
import odbc
cx = odbc.odbc("mysqlperso")
cur = cx.cursor()

cur.execute ("""
CREATE TABLE ELEVE (
     num integer primary key,
```

36. https://sourceforge.net/projects/pywin32/
37. http://sourceforge.net/projects/mysql-python
38. On trouve à l'adresse http://www.pygresql.org/ le module pour le serveur *PostGreSQL*.




Concernant la connexion ODBC :

- Son nom est *mysqlperso*, il est possible de le changer.
- Le server est *localhost*, c'est la machine locale, si c'est une machine distante mais connectée au réseau interne, il faut ici préciser son adresse IP.
- Le port est par défaut 3306, dans un cas classique, ce paramètre est déjà renseigné.
- Le nom d'utilisateur et le mot de passe sont un de ceux définis sur le serveur de base de données, si aucun utilisateur n'a été ajouté, alors, il faut choisir l'utilisateur *root* et le mot de passe renseigné lors de l'installation (dans ce cas *admin*).
- Enfin, il faut indiquer un nom de database qui existe sur le serveur de base de données. (Pour la créer, voir le paragraphe 15.6.3.1 (page 372).

**Figure 15.22 :** *La configuration d'une connexion ODBC se passe en deux étapes, la première consiste à ajouter une connexion en cliquant sur le bouton Add de la première fenêtre. La seconde fenêtre permet de choisir le type de connexion, dans le cas de MySQL, il faut choisir MySQL ODBC 5.1 Driver. La troisième fenêtre définit les paramètres de la connexion : Le bouton Test permet de s'assurer que ces paramètres sont bien valides.*

```
    nom varchar (30),
    prenom varchar (30),
    date date,
    adresse varchar (100),
    codepays integer,
    classe integer)
""")
```

L'ajout de données diffère un peu de `sqlite3` car le serveur SQL requiert une valeur par défaut pour la colonne `num` alors qu'elle était renseigné automatiquement par `sqlite3`. Le langage SQL est en principe le même d'un serveur SQL à l'autre mais il arrive parfois que certains logiciels proposent des spécificités non compréhensibles par un autre SGBD. Par exemple, le format des dates par défaut n'est pas le même sur *MySQL* :

```
import odbc
cx = odbc.odbc("mysqlperso")
cur = cx.cursor()

cur.execute ("""INSERT INTO ELEVE (num, nom, prenom, date, adresse, codepays, classe)
                VALUES (1, 'dupre', 'xavier', '1975-08-11', '---- paris', 33, 19) ;""")
cur.execute ("""INSERT INTO ELEVE (num, nom, prenom, date, adresse, codepays, classe)
                VALUES (2, 'dupre', 'gilles', '1946-12-24', '---- charleville', 33, 56) ;""")
cx.commit ()
```

Pour lancer une requête, le code reste le même :

```
import odbc
cx = odbc.odbc("mysqlperso")
```

```
cur = cx.cursor()
cur.execute ("SELECT * from ELEVE")
for row in cur.fetchall () :
    print row
```

**Remarque 15.13 : SGBD et les dates**
La gestion des dates peut être différente d'un SGBD à l'autre, avec *MySQL*, elles apparaissent sous forme d'objets. Pour pouvoir l'afficher, il faut remplacer la dernière du programme précédent par :

```
    print [ str (r) for r in row ]
```

Mise à part la connexion, les autres requêtes et le code *Python* les exécutant extraits des exemples développés au paragraphe 15.6.2 seront identiques avec *MySQL*.

#### 15.6.3.4 ODBC et script CGI

**Problème de mot de passe**
Lorsqu'on cherche à se connecter à une connexion ODBC, celle-ci requiert parfois un login et un mot de passe. Par exemple, les deux lignes qui suivent permettent d'ouvrir une connexion ODBC vers un serveur de type *SQL Server* de *Microsoft*.

```
import odbc
cx = odbc.odbc("odbc_sql_server")
```

L'exécution du programme retourne le message suivant annonçant l'impossibilité de l'ouverture de la connexion :

```
dbi.operation-error: [Microsoft][ODBC SQL Server Driver][SQL Server]Login failed for user ''.
The user is not associated with a trusted SQL Server connection. in LOGIN
```

Dans le cas, il faut répéter le login et le mot de passe utilisé lors de la configuration de la connexion ODBC comme ceci :

```
import odbc
cx = odbc.odbc("odbc_sql_server/login/password")
```

Une autre syntaxe comme celle qui suit est également correcte :

```
cx = odbc.odbc("DSN=odbc_sql_server;UID=login;PWD=password")
```

**Service *Apache*** service installation
Il arrive aussi que la connexion ODBC ne fonctionne pas car la table ou le serveur SQL n'est pas trouvé. Lorsqu'on exécute le script CGI comme un programme *Python*, celui-ci fonctionne parfaitement. Ceci s'explique par le fait que le serveur *Apache* fonctionne avec un nom d'utilisateur qui ne lui permet pas de créer une connexion ODBC. Il existe deux façons de résoudre ce problème. La première consiste à configurer le service *Apache* et à le relancer avec le bon login et le bon mot de passe. La seconde option consiste à ne plus lancer le serveur *Apache* comme un service *Windows* mais comme un programme depuis une fenêtre de commande. Il faut donc exécuter les trois lignes qui suivent :

```
cd c:\Program Files\Apache Software Foundation\Apache2.2\bin
httpd -k stop
httpd
```

La première ligne permet de se placer dans le répertoire d'*Apache*. La seconde arrête l'exécution du serveur *Apache* en tant que service, la dernière ligne redémarre le serveur *Apache* en tant que programme indépendant. En contrepartie, il ne faut pas fermer la fenêtre de commande *Windows* à moins de vouloir arrêter le serveur *Apache*. Cette solution fonctionne localement, il est possible que le serveur ne soit plus accessible depuis une autre machine connectée sur le même réseau interne.

Une autre option consiste à donner au service *Apache* l'accès au serveur de base de données. Pour cela, il faut associer à au serveur *Apache* un utilisateur et donner à cet utilisateur les droits nécessaires à une bonne utilisateur de *Apache*. La première consiste à créer cet utilisateur (voir figure 15.23) puis à lui conférer les bons droits.

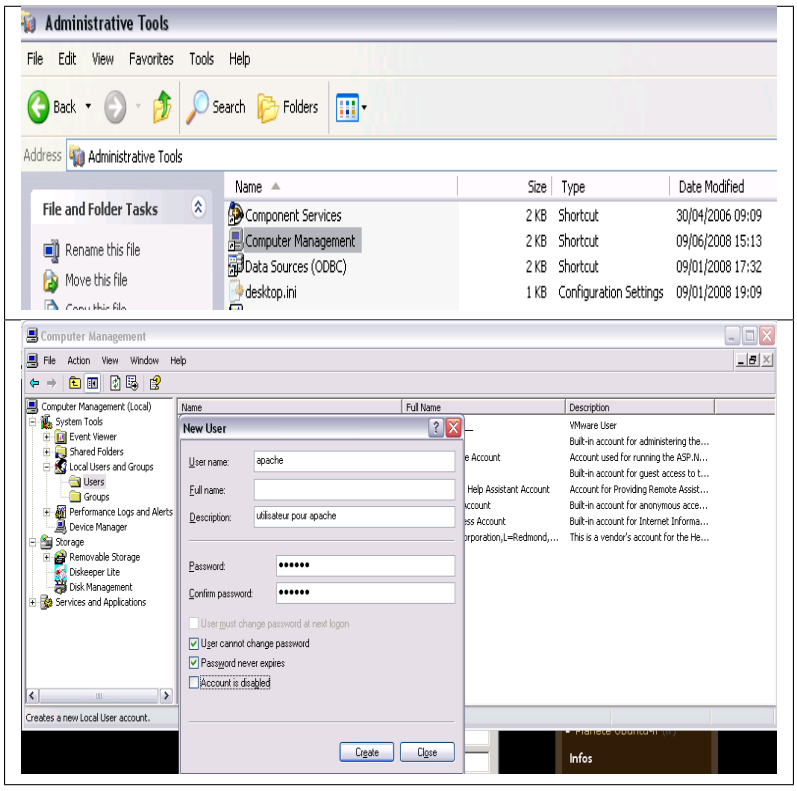



**Figure 15.23 :** *Pour ajouter un utilisateur, il faut aller dans le Panneau de Contrôle, choisir Outils d'administration puis choisir Computer Management. La seconde image indique comment ajouter un utilisateur en cliquant avec le bouton droit sur le répertoire Users ou Utilisateurs.*

L'étape suivante consiste à donner à cet utilisateur nouvellement créé les droits nécessaires à la modification des répertoires d'*Apache* (voir la figure 15.24). L'étape suivante est l'attribution de droits administrateur à l'utiliseur `apache` (figure 15.25). La dernière étape est la configuration du service *Apache* ce qu'illustre la figure 15.26.

#### 15.6.3.5 Sans ODBC

Il arrive que sur certains systèmes, la configuration d'*Apache* en tant que service soit difficile pour un non expert bien que cela soit rendu nécessaire par l'utilisation de pilotes ODBC dans de scripts CGI écrits en *Python*. Il existe une alternative qui est de se connecter directement au serveur *MySQL*. Il faut pour cela utiliser le module `MySQLdb`[39]. Le code permettant de lancer une requête diffère peu de celui utiliser pour une connexion ODBC comme le montre les deux exemples suivant.

39. http://sourceforge.net/projects/mysql-python

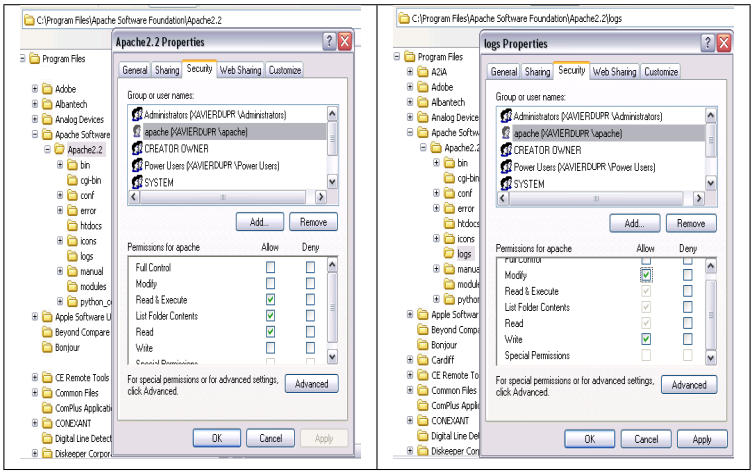



**Figure 15.24 :** *L'utilisateur* `apache` *créé par les étapes illustrées la figure 15.23 doit avoir les droits d'accès en lecture et exécution pour le répertoire d'installation d'Apache. Il doit aussi avoir les droits en écriture pour le répertoire logs. Si Apache utilise des extensions comme SVN ou des répertoires de pages ou de scripts situés ailleurs, il faudra également ajouter l'utilisateur* `apache` *à la liste des utilisateurs autorisés à lire ou écrire ces emplacements.*

```
import odbc
cx = odbc.odbc("mysqlperso")

cur = cx.cursor()
cur.execute ("SELECT * from ELEVE")
for row in cur.fetchall () :
    print row
cur.close ()
```

```
import MySQLdb
cx  = MySQLdb.connect('localhost','root', \
        'admin', 'datatest')
cur = cx.cursor()
cur.execute ("select * from ELEVE" )
for row in cur.fetchall() :
    print row
cur.close()
cx.close()
```

Il faut toutefois penser à fermer la connexion vers le serveur *MySQL*, tâche que le pilote ODBC gère seul. Le serveur *MySQL* ne peut supporter trop de connexions ouvertes au même moment [40].

Si le serveur de base de données est différent de *MySQL*, il suffit de taper *python* + *<nom du serveur>* dans un moteur de recherche pour trouver le module qui permet de communiquer avec ce serveur. Par exemple, pour *Oracle*, on trouve le module `cx – oracle` [41]. La syntaxe permettant de lancer une requête est sensiblement la même. Un serveur déroge à cette règle, il s'agit de *MSSQL Server* de *Microsoft* qui utilise le module `win32com` [42] dont voici un exemple d'utilisation.

```
import win32com.client
connexion = win32com.client.gencache.EnsureDispatch('ADODB.Connection')
connexion.Open("Provider='SQLOLEDB';Data Source='localhost';Initial Catalog='datatest';User ID='root';Password='admin
recordset = connexion.Execute('select * from client')[0]
while not recordset.EOF:
    for i in recordset.Fields : print i,
    print "<br>\n"
    recordset.MoveNext()
connexion.Close()
```

ODBC fournit une interface de connexion qui ne change pas quelque soit le serveur SQL utilisé, c'est tout l'avantage d'un tel système. Accéder directement au serveur est plus rapide mais plus contraignant si la base de données est transférée sur un serveur différent. C'est pourquoi il est recommandé de regrouper le code qui a trait aux bases de données. Ceci peut être fait manuellement ou en utilisant un module qui propose une unique interface quelque soit le serveur de base de données utilisé.

40. Ceci dépend de la configuration du serveur et de la puissance de l'ordinateur sur lequel il est installé.
41. http://cx-oracle.sourceforge.net/
42. http://python.net/crew/mhammond/win32/Downloads.html

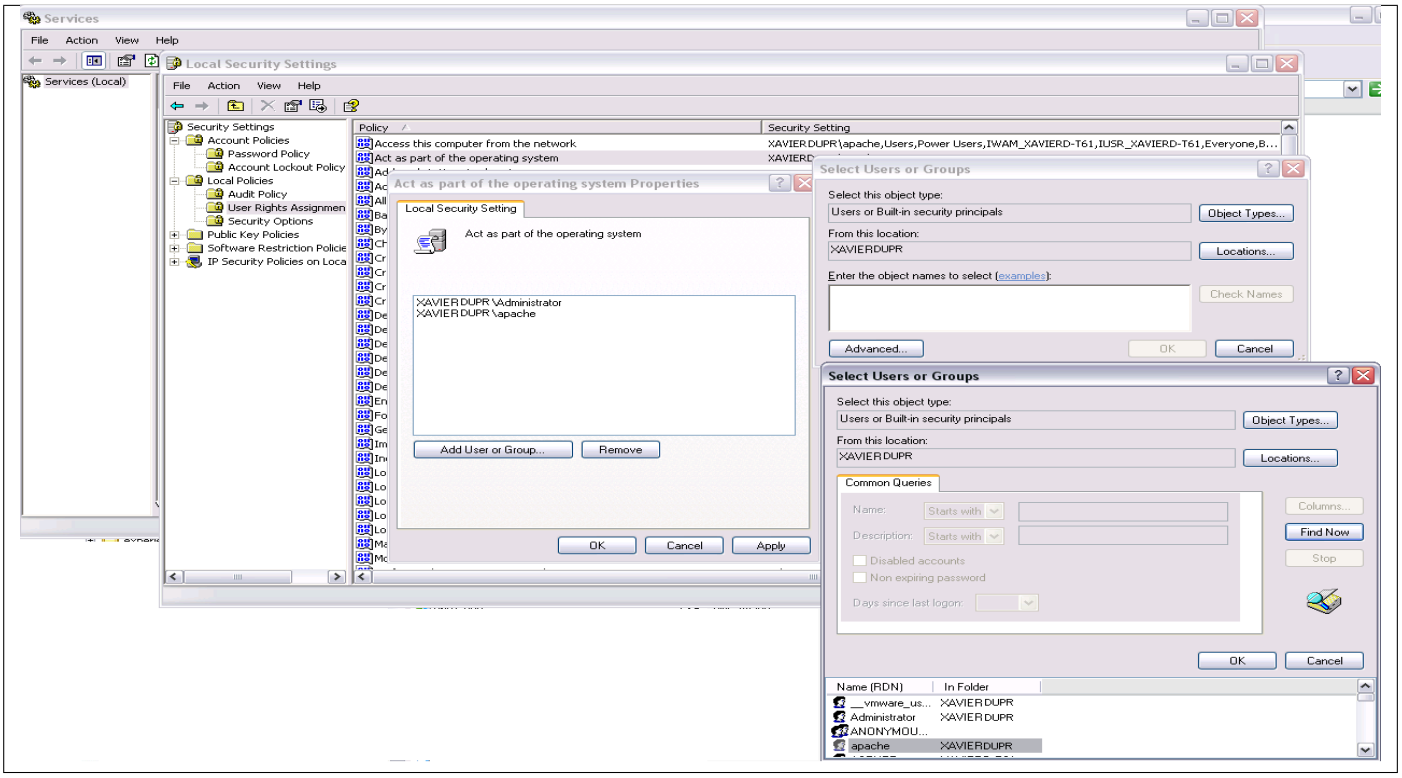



**Figure 15.25 :** *Il faut donner à l'utilisateur* `apache` *des droits d'administrateurs. Il faut pour cela aller dans le Panneau de configuration, sur Outils d'Administration puis sur Local Secruity Policy. On termine par l'ajout de l'utilisateur* `apache` *au droit Act a part of the operating system.*

`SQLObject` [43] s'appuie sur les classes pour représenter les tables d'une base de données. Une table est considérée comme une liste d'instances d'un même objet dont les attributs sont stockés dans la table. Cette conception objet remplace en partie le langage SQL et est bien adapté aux bases de données relationnelles qui permettent de lier des attributs de plusieurs tables.

## 15.7 Modules divers

### 15.7.1 PDF

De même que *Python* propose des outils pour écrire et lire des fichiers *zip*, il existe des modules permettant de constuire des fichiers *pdf* et d'extraire des informations depuis des documents de ce format. Le module `reportLab` [44] permet de construire automatique des fichiers au format *pdf*. Ceci évite par exemple l'utilisation d'application intermédiaire comme *Latex* par exemple.

Un autre module permet d'extraire des informations textuelles depuis un fichier *pdf* à condition que celui-ci ne soit pas crypté : `pdftools` [45]. Le programme suivant permet d'extraire tout le texte d'un document *pdf*.

```
from pdftools.pdffile import PDFDocument
from pdftools.pdftext import Text

def contents_to_text (contents):
    for item in contents:
```

5

43. http://www.sqlobject.org/
44. http://www.reportlab.org/rl_toolkit.html
45. http://www.boddie.org.uk/david/Projects/Python/pdftools/

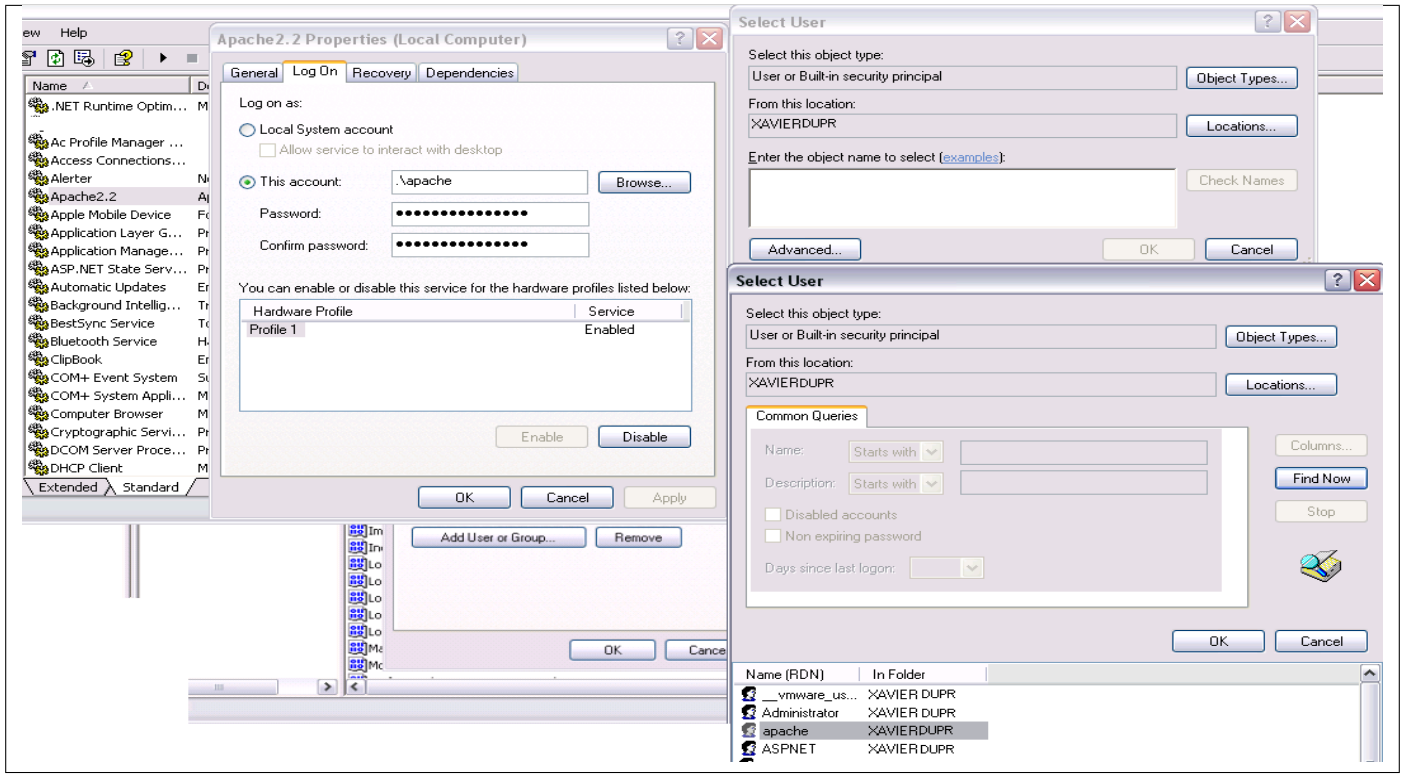



**Figure 15.26 :** *Il faut maintenant préciser au service Apache qu'il doit utiliser les droits de l'utilisateur* `apache`. *Il faudra redémarrer pour prendre en compte les modifications.*

```
        if isinstance (item, type ([])):
            for i in contents_to_text (item):
                yield i
        elif isinstance (item, Text):
            yield item.text

doc = PDFDocument ("declaration.pdf")
n_pages = doc.count_pages ()
text = []

for n_page in range (1, (n_pages+1)):
    print "Page", n_page
    page = doc.read_page (n_page)
    contents = page.read_contents ().contents
    text.extend (contents_to_text (contents))

print "".join (text)

f = open ("ok.txt", "w")
f.write ("".join (text))
f.close ()
```

Le module `pyPDF` [46] propose des fonctionnalités équivalentes. Le module le plus récemment mis à jour est souvent le meilleur choix.

46. http://pybrary.net/pyPdf/

### 15.7.2 Tesseract

*Tesseract* est un outil gratuit de reconnaissance de l'écriture imprimée (OCR) qui est capable de transcrire le texte depuis une page imprimée puis scannée. Il s'utilise depuis le module `pytesser`[47] Il fonctionne assez simplement et il suffit d'exécuter le programme `pytesser.py` pour le faire fonctionner.

## 15.8 Graphisme

### 15.8.1 GNUPlot

*GNUPlot*[48] est un logiciel permettant de tracer des graphes. Il a son propre langage de programmation et dispose d'une large panoplie de types de graphismes. Il existe à la fois sous *Windows* et *Linux*. Le programme suivant construit un script GNUPlot et l'exécute de façon à construire automatiquement l'image du graphique.

```
# -*- coding: cp1252 -*-

import sys
import os

default_path     = r"C:\Program Files\gp423win32\gnuplot\bin\pgnuplot.exe"
default_temp_dir = "tempgnuplot"
imagenumber      = 0

def execute_script_gnuplot (scr) :
    global default_temp_dir
    global imagenumber
    global default_path

    if not os.path.exists (default_temp_dir) : os.mkdir (default_temp_dir)

    # avant
    scr          = "set term png\n" + scr
    image        = default_temp_dir + ("/image_%05d.png" % imagenumber)
    imagenumber += 1
    scr          = "set out \"" + image + "\"\n" + scr

    # après
    scr += "show out\n"
    scr += "exit\n"

    name = default_temp_dir + "/gnuscript.txt"
    f    = open (name, "w")
    f.write (scr)
    f.close ()

    line = default_path + " " + name
    os.system (line)

    return image

def build_script_gnuplot (series, seriesname, title = None, \
                      xlabel = None, ylabel = None, histo = False) :
    global default_temp_dir
    global default_path

    if not os.path.exists (default_temp_dir) : os.mkdir (default_temp_dir)
    scr = ""
```

47. http://code.google.com/p/pytesser/
48. http://www.gnuplot.info/

```
    if xlabel != None : scr += "set xlabel \"" + xlabel + "\"\n"
    if ylabel != None : scr += "set ylabel \"" + ylabel + "\"\n"
    if title  != None : scr += "set title \"" + title + "\"\n"

    scr += "set grid\n"
    if histo : scr += "set style data histograms\n"
    else : scr += "set style data lines\n"
    scr += "plot "

    id = 0
    for s,lab in zip (series, seriesname) :
        name = default_temp_dir + "/series%d.txt" % (id)
        id  += 1
        f    = open (name, "w")
        for l in s :
            if histo : f.write ("%f\n" % (l [1]))
            else : f.write ("%f\t%f\n" % (l [0], l [1]))
        f.close ()
        scr += "\"" + name + "\" title \"" + lab + "\", "
    scr = scr [:len (scr)-2]
    scr += "\n"

    return execute_script_gnuplot (scr)


if __name__ == "__main__" :
    print "chemin pour gnuplot ", default_path

    series = [ [], [] ]
    for i in range (0, 100) :
        x = float (i) / 100
        y = x ** 0.5
        z = 1.0 - y
        series [0].append ( (x,y) )
        series [1].append ( (x,z) )

    image = build_script_gnuplot (series, ["serie 1", "serie 2"], \
                        xlabel="abscisses", ylabel="ordonnées", histo = False)
    print "image ", image
    image = build_script_gnuplot (series, ["serie 1", "serie 2"], \
                        xlabel="abscisses", ylabel="ordonnées", histo = True)
    print "image ", image
```

### 15.8.2 MatPlotLib

Ce module imite la syntaxe de *Matlab* en ce qui concerne la création de graphe. Il propose les mêmes fonctionnalités. Le programme suivant montre comment tracer une série financière.

```
def GraphOHLC (title, dates, ohlc, file, size = (800,400), t1 = 0, t2 = -1) :
    """dessine une série financière avec matplotlib
    - title est une titre
    - dates est une liste de date
    - ohlc est une liste de dictionnaire avec les clés
                      ("Open", "High", "Low", "Close", "Volume")
    - file est un nom de fichier, le graphe y sera sauvegardé
    - size est la taille désirée pour le graphique
    - t1 est l'indice de la première date du graphique
    - t2 est l'indice de la dernière date du graphique
    """

    # on coupe la série si besoin
    if t2 == -1 : t2 = len (dates)
```

```
dates = dates [t1:t2]
ohlc = ohlc [t1:t2]

import pylab

# on crée les labels pour les abscisses
# sans afficher toutes les dates car elles deviennent illisibles sinon
# on en affiche que 10
ind  = pylab.arange (len (dates))
lab  = ["" for i in ind]
for i in range (0,len (dates),len(dates)/10) : lab [i] = dates [i]

# on crée la figure
fig  = pylab.figure (figsize=(size [0]/100, size [1]/100))
ax   = fig.add_subplot(111)

# la première série à tracer est celle des volumes sous forme d'histogramme
for i in range (0, len (ohlc)) :
    oh = ohlc [i]

    # l'histogramme est vert si c'est un jour de hausse
    # rouge si c'est un jour de baisse
    if i == 0 : co = (0.5,0.5,0.5)
    elif ohlc [i]["Close"] > ohlc [i-1]["Close"] : co = (0.0,1.0,0.0)
    else : co = (1.0,0.0,0.0)
    if len (dates) > 300 :
        pylab.plot ( [i, i], [0, oh ["Volume"]], "k", color = co)
    elif len (dates) > 100 :
        pylab.plot ( [i, i], [0, oh ["Volume"]], "k", \
                                    linewidth = 2.0, color = co)
    else :
        pylab.plot ( [i, i], [0, oh ["Volume"]], "k", \
                                     linewidth = 4.0, color = co)

# on écrit la légende du volume
v = [0.0 for i in ind]
pylab.plot (ind, v, "c.")
pylab.ylabel ("volume")

# on construit un second axe au graphique pour la série des prix
ymin, ymax = pylab.ylim()
pylab.ylim (ymin, ymax*3)
pylab.xticks (ind, lab, fontsize = "small", rotation = 13  )
ax2 = pylab.twinx()

# puis un dessine les prix sur le graphique selon le même schéma,
# la même série de points
# (i-0.5, Open) (i,Open) (i,Low) (i,High) (i,Close) (i+0.5,Close)
for i in range (0, len (dates)) :
    oh = ohlc [i]
    co = (0.0,0.0,1.0)
    pylab.plot ([i-0.5, i], [oh["Open"],  oh["Open"]],  "k", \
                                linewidth = 1.0, color = co)
    pylab.plot ([i, i],     [oh["Low"],   oh["High"]],  "k", \
                                linewidth = 1.0, color = co)
    pylab.plot ([i, i+0.5], [oh["Close"], oh["Close"]], "k", \
                                linewidth = 1.0, color = co)

# on termine par le titres, la légende du second axe et celles des abscisses
pylab.title   (title)
pylab.ylabel  ("euros")
pylab.xticks  (ind, lab, fontsize = "small", rotation = 13)
pylab.xlabel  ("dates")
# il ne reste plus qu'à sauver la figure
pylab.savefig (file, orientation='paysage')
```

## 15.9 Apache : serveur

installation

Les paragraphes qui suivent décrivent l'installation et la configuration du serveur *Apache*. Celui est utilisé par des applications comme*SVN*, *MoinMoin* ou tout simplement, il offre la possibilité d'écrire des applications Internet comme des script CGI en langage *Python*. Il est parfois plus simple d'utiliser l'application *EasyPHP*[49] qui en simplifie l'installation surtout s'il faut installer ensuite le serveur de base de données *MySQL*. La suite vient avec l'installation d'*Apache* version 2.2[50]

### 15.9.1 Installation

L'installation commence par un premier écran qu'il faut remplir en suivant les instructions de la figure 15.27. Pour vérifier que l'installation a fonctionné, il suffit d'ouvrir un navigateur comme *Firefox*. Après avoir rentré http://localhost/ dans la barre d'adresse, le message *It works !* apparaît si *Apache* fonctionne.

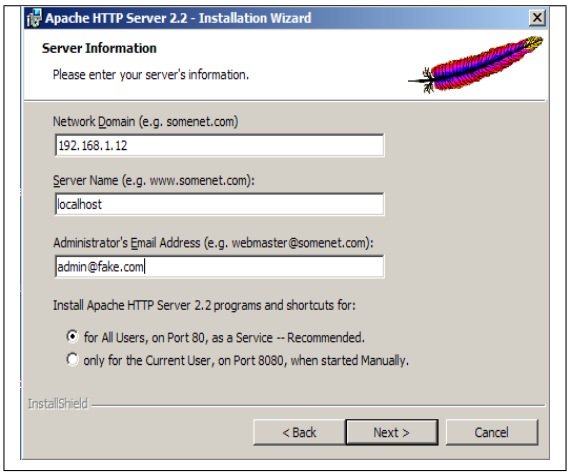



**Figure 15.27 :** *Installation de Apache,* `DomainName` *est un nom de domaine, dans un réseau local, cela correspond à l'adresse IP de l'ordinateur. (écrire* `ipconfig /all` *en ligne de commande pour l'obtenir.* `ServerName` *est le nom du serveur, le plus souvent* `localhost` *qui désigne l'ordinateur local. L'adresse email peut être n'importe quel email.*

**Remarque 15.14 : changement de port pour *Apache***
Il est possible de le port par défaut 80 soit déjà utilisé par d'autres services. Le port se résume à un numéro, un emplacement, où le serveur *Apache* va écouter et envoyer des informations. Toute application qui veut envoyer des informations à *Apache* doit spécifier le nom de la machine sur laquelle est installée *Apache* ainsi que le port sur lequel il écoute. Si deux serveurs écoutent le port 80, lequel va recevoir un message, il ne sera pas non plus possible de les distinguer puisqu'ils écoutent à la même entrée. Pour le changer, il faut aller dans le fichier `httpd.conf` et changer la ligne `Listen 80` par : port

```
Listen 81
```

Ensuite, dans l'exporateur internet, il faut préciser ce port qui n'est plus le port par défaut 80 et entrer l'adresse suivante : http://localhost:81/ ou ajouter `http://nom_de_machine:81` si le serveur *Apache* est appelée depuis une autre machine du réseau.

49. http://www.easyphp.org/
50. Lors de l'écriture du livre, la version était 2.2.8 et téléchargeable à l'adresse http://httpd.apache.org/.

### 15.9.2 Configuration de *Apache* pour *MoinMoin*

Il reste à configurer *Apache* pour évoluer de paire avec *SVN*. Il y a quatre étapes :

1. Copier les fichiers `mod_dav_svn.so` et `mod_authz_svn.so` depuis le répertoire d'installation de *SVN* (sous-répertoire *bin*) dans le répertoire d'installation de `Apache` (sous-répertoire *module*).
2. Ouvrir le fichier de configuration de *Apache* depuis le menu *Démarrer* de *Windows* et insérer les lignes suivantes :

```
LoadModule dav_module modules/mod_dav.so
LoadModule dav_svn_module modules/mod_dav_svn.so
LoadModule authz_svn_module modules/mod_authz_svn.so

<Location /svn/>
    DAV svn
    SVNListParentPath on
    SVNParentPath "c:/svnrepository/"
    SVNPathAuthz on
    #AuthzSVNAccessFile "C:/Program Files/Apache Software Foundation/Apache2.2/bin/apachesvnauth
    AuthName "Subversion Repositories"
    AuthType Basic
    AuthBasicProvider file
    AuthUserFile "C:/Program Files/Apache Software Foundation/Apache2.2/bin/apachesvnpasswd"
    require valid-user
</Location>
```

   Il faut remplacer `c : /svnrepository/` par le répertoire du repository créé et `C : /Program Files/Apache SoftwareFou` par le répertoire d'installation de *Apache*.
3. Il reste à insérer des utilisateurs authorisés à utiliser `SVN`. Pour cela il faut ouvrir une fenêtre de commande *Windows* :

```
cd C:\Program Files\Apache Software Foundation\Apache2.2\bin
htpasswd.exe -c apachesvnpasswd utilisateur
```

   Il vous sera demandé de rentrer un mot de passe pour chaque utilisateur.
4. Il reste à redémarrer le serveur `Apache` afin qu'il prenne en compte ces modifications. Ce peut être fait depuis le sous-menu *Apache* du menu *Démarrer / Tous les programmes*.

Pour vérifier que cela fonctionne, il faut ouvrir un navigateur et écrire http://localhost/svn/. L'explorateur vous demandera de vous identifier, il suffira de rentrer un des utilisateurs et son mot de passe définis en étape 3.

**Remarque 15.15 : problème de chargement du module `mod_dav_svn.so`**
Il arrive parfois que le serveur n'arrive pas à redémarrer après avoir modifié le fichier `httpd.conf` car le module `mod_dav_svn.so` ne peut être chargé [51]. Cette erreur peut survenir si la version de *Windows* n'est pas récente où sur laquelle n'a pas été installée la dernière ou les dernières mises à jour de *Windows*.

D'autres précisions concernant la configuration d'*Apache* sont décrites aux paragraphes 15.6.3.4 et 15.6.3.4. installation

La configuration d'*Apache* fait intervenir deux éléments. Le premier est la définition d'alias.alias Il s'agit de dire à *Apache* que tous les adresses URL contenant la même racine (ici http://localhost/mywiki) font partie du *Wiki* et contiennent des scripts *Python*. Il faut ajouter les lignes suivantes dans le fichier `httpd.conf`.

```
ScriptAlias /mywiki "C:/Moin/mywiki/moin.cgi"
Alias /moin_static163 "C:/Python25/share/moin/htdocs"
```

La seconde étape est l'attribution de droits au répertoire contenant plusieurs répertoires essentiels au `Wiki` mais qui sont situés en dehors du chemin `DocumentRoot`. Toujours dans le fichier `httpd.conf`, il faut ajouter les lignes en fin de fichier :

51. Cette information est présente dans le fichier `logs/error.log` dans le répertoire d'installation d'*Apache*.

```
<Directory "C:/Python25/share/moin/htdocs">
    Order deny,allow
    Allow from all
</Directory>
<Directory "C:/Moin/mywiki">
    Options FollowSymLinks
    AllowOverride None
    Order deny,allow
    Allow from all
</Directory>
```

Il ne faut pas oublier de redémarrer le serveur pour que ces modifications soient bien prises en compte. Avec cette configuration, il suffit que chaque script *Python* commence par une ligne indiquant l'emplacement de l'interpréteur *Python*.

```
#!c:/python25/python.exe
```

Ou sous Linux :

```
#!/usr/bin/python
```

Cette première ligne est appelée par *Apache* un *shebang*. Si un script CGI doit être exécuté sur des serverus *Apache* installés sur deux systèmes d'exploitation différent, cette ligne empêche le passage de l'un à l'autre. *Apache* peut être configurer de façon à ce qu'il sache automatiquement quoi faire lors de l'exécution d'un script CGI d'extension `.py`.

```
ScriptInterpreterSource registry
<Directory "C:/Python25">            # il faut également une section
    AllowOverride None               # comme celle-ci définie
    Options ExecCGI                  # pour le répertoire contenant
    Order allow,deny                 # le script CGI
    Allow from all                   #
</Directory>
AddType application/x-httpd-py .py
AddHandler cgi-script .py
Action application/x-httpd-py "C:/python25/python.exe"
```

**Remarque 15.16 : `Alias`, *Apache* et *MoinMoin***
Les alias sous *Apache* permettent de remplacer une chemin donné par un autre dans tous les hyperliens. Ici `/moin_static163/` sera remplacé par `C : /Python25/share/moin/htdocs/`. Un dernier détail, pour une version ultérieure de *MoinMoin*, il est possible que l'alias défini pour `moin_static163` diffère, 163 est en effet la liste des chiffres du numéro de version. Pour en être sûr, il suffit d'afficher le *source* d'une page du `Wiki` depuis l'navigateur. Toutefois, il est possible de modifier ce chemin depuis *MoinMoin*.

**Remarque 15.17 : *Apache*, comprendre une erreur**
Dans le répertoire d'installation de *Apache*, on trouve le sous-répertoire `logs` et le fichier `error.log` dans lequel figurent des informations concernant l'exécution du serveur. Si une page HTML ou un script ne s'exécute pas comme il le devrait, un message d'erreur est inséré dans ce fichier si l'erreur se situe au niveau de *Apache*. Les exceptions *Python* sont également présentes dans ce fichier lorsqu'un script CGI [52] provoque une erreur d'exécution.

D'autres remarques concernant des problèmes éventuels survenant lors de l'installation d'*Apache* sont décrites au paragraphe 15.6.3.4. installation

52. Le rôle des CGI est brièvement expliqué au paragraphe 15.5.

### 15.9.3 Configuration pour des applications CGI

La configuration du serveur *Apache* pour l'exécution de script CGI en langage *Python* passe toujours par la précision des bons droits pour le répertoire contenant le langage *Python* et celui contenant le script CGI.

```
<Directory "C:/Python25">
    AllowOverride None
    Options ExecCGI
    Order allow,deny
    Allow from all
</Directory>
<Directory "C:/_home/query-intent/projects/common_tools/cgi/">
    AllowOverride None
    Options FollowSymLinks ExecCGI
    Order allow,deny
    Allow from all
</Directory>
```

Il faut préciser ensuite que l'extension *.py* est celle d'un script CGI et qu'il doit s'exécuter avec l'interpréteur *Python* :

```
<IfModule mime_module>
    TypesConfig conf/mime.types
    AddType application/x-httpd-py .py
    AddHandler cgi-script .py
</IfModule>
Action application/x-httpd-py "C:/python25/python.exe"
```

Il ne reste plus qu'à donner un alias au répertoire contenant le script CGI :

```
ScriptAlias /py/ "C:/script_cgi_python/"
```

Par exemple, le fichier suivant nommé `script.py` est placé dans le répertoire précédent.

```
header = """Content-type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>"""

footer = """</body></html>"""

print header
print "<b>Script CGI</b><br>\n"
print footer
```

Pour l'exécuter via un navigateur, il suffit alors de taper l'adresse :

```
http://localhost:80/py/script.py
```

# Chapitre 16

# Algorithmes usuels

---

## 16.1 Calcul d'une intégrale

### 16.1.1 Description de l'objectif

On cherche à calculer une intégrale en utilisant la méthode des rectangles (voir figure 16.1). L'intervalle

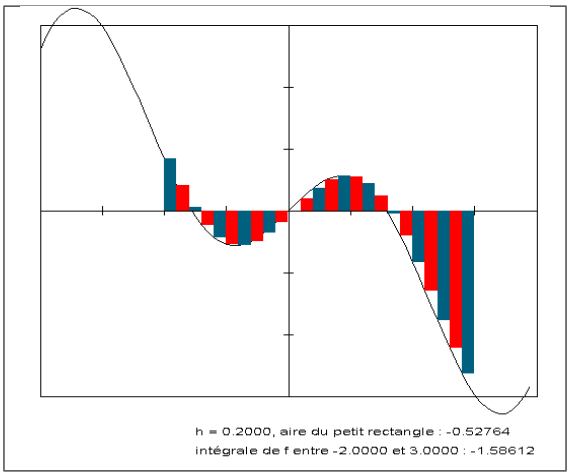



**Figure 16.1 :** *Illustration du calcul approché d'une intégrale à l'aide de la méthode des rectangles.*

de l'intégrale est noté $[a, b]$ et la fonction à intégrer $f$. On divise cet intervalle en $n$ petits segments et on fait la somme des aires des petits rectangles délimités par l'axe des abscisses et la courbe de la fonction $f$.

$$\int_a^b f(x)dx \approx \frac{b-a}{n} \sum_{i=1}^{n} f\left(a + i\frac{b-a}{n}\right) \tag{16.1}$$

On pourra prendre par exemple :

$$\begin{array}{ll} a & = -2 \\ b & = 3 \\ f(x) & = xcos(x) \\ n & = 20 \end{array} \tag{16.2}$$

### 16.1.2 Différents programmes

Le calcul approché d'une intégrale est un calcul mathématique simple. Il ne repose que sur une boucle et celle-ci obéit à la règle que toute boucle doit être précédée d'une étape d'initialisation (ici, `sum = 0`). C'est une étape qui n'apparaît pas souvent dans la phase de rédaction d'un algorithme, il n'y a pourtant pas d'execption à cette règle.

```
# -*- coding: cp1252 -*-
import math

n = 2000
a = -2.0
b = 3.0
h = (b-a)/n

sum = 0
for i in xrange (1,n):
    x = a + h * i
    sum += h * x * math.cos (x)

print "intégrale : ", sum

import scipy.integrate as integral            # on importe le module d'intégration
def fff(x): return x * math.cos (x)           # on définit la fonction à intégrer
sum = integral.quad (fff, a,b, args=()) [0]   # on appelle la fonction d'intégration
print "intégrale avec scipy : ", sum          # on affiche le résultat
```

Le module `SciPy` [1] propose de nombreux algorithmes mathématiques tels que le calcul d'intégrales. Le programme précédent se résume alors à quelques lignes.

```
import scipy.integrate as integral            # on importe le module d'intégration
def fff(x): return x * math.cos (x)           # on définit la fonction à intégrer
sum = integral.quad (fff, a,b, args=()) [0]   # on appelle la fonction d'intégration
print "intégrale avec scipy : ", sum          # on affiche le résultat
```

Le programme affiche les lignes suivantes :

```
intégrale :  -1.96640795695
intégrale avec scipy :  -1.96908048953
```

## 16.2 Tri simple et tri fusion

### 16.2.1 Trier une liste avec *Python*

Pour trier une liste par ordre croissant, il suffit d'utiliser la méthode `sort` associée aux listes.

```
x = [1,3,2,5,8,4,9,0,7,6]
x.sort ()
print x
```

Cependant, le tri est très courant en informatique et il est parfois utile de s'inspirer de telles méthodes pour résoudre des problèmes similaires. On distingue principalement deux familles d'algorithmes qui permettent de trier un ensemble. La première, la plus simple, inclus notemment le *tri par sélection*. Les algorithmes de cette classe ont un coût [2] en $O(n^2)$ où $n$ est le nombre d'éléments à trier. La seconde classe d'algorithmes inclut le tri rapide ou *quicksort* qui est détaillé dans le TD 4. Elle inclut également le *tri par fusion* décrit au paragraphe 16.2.3 ou le tri par insertion dichotomique décrit au paragraphe 16.2.4. Le point commun de ces algorithmes est leur coût qui est en $O(n \ln n)$.

1. Le module SciPy propose de nombreux algorithmes, algorithmes de classification, intégration, optimisation, génération de nombres aléatoires, traitement du signal, interpolation, algorithmes génétiques, algèbre linéaire, transformée de Fourier, dessin de courbes 2D, 3D, adresse : http://www.scipy.org/.

2. Le coût d'un algorithme est une estimation ou l'exact nombre d'opérations nécessaires à l'algorithme pour traiter les informations qu'il reçoit. Le coût sera en général exprimé à l'aide du symbol $O(...)$ qui désigne un multiple d'une expression. $O(n)$ désigne par exemple un multiple de $n$.

### 16.2.2 Tri par sélection

Bien que moins performant que d'autres algorithmes de tri, le tri par sélection est pourtant souvent utilisé sur de petits ensembles parce qu'il est très simple et très court à programmer.

**Algorithme 16.1 : tri par sélection**
On suppose qu'on possède une suite finie quelconque de $n$ éléments $(u_1, \ldots, u_n)$. On pose $l \longleftarrow n$.
**Etape A : recherche du maximum**
On détermine l'indice $i^*$ de l'élément le plus grand inclus dans la suite $(u_1, \ldots, u_l)$ : $i^* = \arg\max \{u_i \mid i \in \{1, \ldots, l\}\}$.
**Etape B : échange**
On échange les positions des éléments $u_l$ et $u_{i^*}$. Puis on décrémente $l \longleftarrow l - 1$. Si $l > 1$, on retourne à l'étape A, dans le cas contraire, le tri est terminé.

Ce tri s'appelle *tri par sélection* car, à chaque étape B, le plus grand élément remonte vers le haut de la liste, vers la surface. On peut se demander si, après l'application de cet algorithme, la liste est effectivement triée et comment cet algorithme serait implémenté en langage *Python*. Le coût de cet algorithme est en $O(n^2)$.

### 16.2.3 Tri par fusion

Le tri par fusion applique le principe *diviser pour régner*. Etant données deux suites d'éléments triés, de longueurs respectives $l_1$ et $l_2$, il est très facile d'obtenir une troisième suite d'éléments triés de longueur $l_1 + l_2$, par interclassement ou fusion des deux précédentes suites. On dispose de deux indices pour chacunes des listes. On suppose que tous les éléments ayant un indice inférieur ont déjà été insérés dans la liste final. La fusion consiste à comparer les deux éléments indicés de chaque liste puis à choisir le plus petit des deux. Il reste à incrémenter l'indice de la liste à laquelle appartient le dernier élément inséré.

**Algorithme 16.2 : fusion de deux listes triées**
On suppose que $(a_1, \ldots, a_{l_1})$ et $(b_1, \ldots, b_{l_2})$ sont deux listes triées dans l'ordre croissant. L'objectif de l'algorithme est de construire la liste $(c_1, \ldots, c_{l_1+l_2})$ égale la fusion des deux listes précédentes de sorte que cette liste soit triée dans l'ordre croissant.

$i \longleftarrow 1$
$j \longleftarrow 1$
$k \longleftarrow 1$
**tant que** $(k \leqslant l_1 + l_2)$ **faire**
    **si** $j > l_2$ **ou** $(i \leqslant l_1$ **et** $a_i \leqslant b_j)$ **alors**
        $c_k \longleftarrow a_i$
        $i \longleftarrow i + 1$
        $k \longleftarrow k + 1$
    **sinon**
        $c_k \longleftarrow b_j$
        $j \longleftarrow j + 1$
        $k \longleftarrow k + 1$
    **fin si**
**fin tant que**

Pour trier un ensemble $E$ en utilisant la méthode du tri par fusion, on trie d'abord toutes les paires d'éléments de $E$. On fusionne ensemble ces paires d'éléments deux par deux pour obtenir des ensembles de quatre éléments. On fusionne à nouveau des groupes de quatre éléments deux par deux pour obtenir des groupes de huit éléments. On réitère le processus jusqu'à ce que l'ensemble $E$ soit trié, ce qui mène à

l'algorithme suivant.

**Algorithme 16.3 : tri par fusion**
On suppose que $(a_1, \ldots, a_l)$ est une liste qu'il faut trier par ordre croissant.
On suppose que $l > 1$.
$n \longleftarrow 1$
**tant que** $(n < l)$ **faire**
  $i \longleftarrow 1$
  **tant que** $(i \leqslant l)$ **faire**
    $m \longleftarrow \min(l, i+n)$
    $b \longleftarrow \min(l, m+n)$
    **si** $m < b$ **alors**
      On utilise l'algorithme 16.2 de fusion de listes sur les deux sous-listes $(a_i, \ldots, a_m)$ et $(a_{m+1}, \ldots, a_b)$. La liste fusionnée est la liste $(t_i, \ldots, t_b)$.
      **pour** $j = i$ **à** $b$ **faire**
        $a_j \longleftarrow t_j$
      **fin pour**
    **fin si**
    $i \longleftarrow b + 1$
  **fin tant que**
  $n \longleftarrow n * 2$
**fin tant que**

Le coût de cet algorithme est en $O(n \ln n)$. Cependant, l'algorithme 16.2 implique de conserver en mémoire un tableau d'indices le temps de la fusion et ce tableau temporaire est de même taille que la liste fusionnée. Cette contrainte n'existe pas avec le tri par sélection ou le tri rapide mais le coût de ces algorithmes est supérieur à celui du tri par fusion.

### 16.2.4 Tri par insertion

Le tri par insertion nécessite lui aussi l'utilisation d'un tableau intermédiaire. On prend un par un les éléments de la liste de départ pour les insérer dans une autre liste, initialement vide, de sorte que celle-ci reste constamment triée. On vide la première liste pour emplir la seconde. A chaque insertion d'un nouvel élément dans la liste triée, il faut chercher la position adéquate pour qu'elle reste triée. Cette étape utilise

une recherche dichotomique.

**Algorithme 16.4 : recherche dichotomique**
Soit $l = (l_1, \ldots, l_n)$ une liste supposée triée par ordre croissant. On cherche à déterminer la position d'insertion d'un nouvel élément $x$ de sorte que la liste $l$ reste triée.

$a \longleftarrow 1$
$b \longleftarrow n$
**tant que** $(a \leqslant b)$ **faire**
  $m = \left\lfloor \frac{a+b}{2} \right\rfloor$
  **si** $x = l_m$ **alors**
    $a \longleftarrow m$
    $b \longleftarrow m$
  **sinon**
    **si** $x < l_m$ **alors**
      $b \longleftarrow m - 1$
    **sinon**
      $a \longleftarrow m + 1$
    **fin si**
  **fin si**
**fin tant que**
$x$ doit être inséré entre les éléments $l_{a-1}$ et $l_a$. La position cherchée est $a$.

On utilise l'algorithme suivant pour trier une liste par insertion dichotomique.

**Algorithme 16.5 : tri par insertion dichotomique**
On suppose que $(l_1, \ldots, l_n)$ est une liste qu'il faut trier par ordre croissant. On suppose que $n > 1$. Le résultat cherché est la liste $c$.
$c \longleftarrow (l_1)$
**pour** $i = 2$ **à** $n$ **faire**
  $m$ est la position déterminée par l'algorithme 16.4 de recherche dichotomique appliqué à l'élément $l_i$ et la liste $c$. On insère l'élément $l_m$ dans la liste $c$ entre les éléments d'indice $m - 1$ et $m$.
**fin pour**

Le coût d'une recherche dichotomique est en $O(\ln n)$ où $n$ est le cardinal de l'ensemble de recherche. Le coût d'un algorithme par insertion dichotomique est en $O(n \ln n)$. Ce coût peut varier en fonction de la structure utilisée pour stocker les données. Si c'est sous forme de graphe, le tri par insertion est en fait un tri rapide ou *quicksort*. En revanche, si les données sont insérées dans un tableau, même si la recherche de la position d'un élément à insérer est rapide, insérer cette information dans le tableau revient à décaler les données pour créer une place vide. Dans ce cas, tout dépend du coût de ce décalage.

### 16.2.5 Démonstration de l'algorithme de tri par fusion

La démonstration que l'algorithme 16.1 trie la suite $(u_1, \ldots, u_n)$ s'effectue par récurrence. Cet algorithme permet bien de classer un ensemble composé d'un seul élément puisque tout ensemble d'un élément est trié. On suppose maintenant que l'algorithme peut trier une suite de $n-1$ éléments et on cherche à démontrer que l'algorithme peut trier une suite de $n$ éléments.

Le premier passage par les étapes A et B permet de décomposer la suite $(u_1, \ldots, u_n)$ en deux sous-suites. La première suite $(v_k)_{1 \leqslant k \leqslant n-1}$ est composée de $n-1$ éléments, la seconde suite $(w_k)_{k=1}$ contient un seul élément, supérieur à tous les autres, $w_1 = \arg\max \{u_n \mid n \in \{1, \ldots, n\}\}$.

Les $n-1$ passages qui suivent (étapes A et B) correspondent à l'application de l'algorithme 16.1 sur une suite de $n-1$ éléments, ici $(v_k)_{1 \leqslant k \leqslant n-1}$. D'après l'hypothèse de récurrence, l'algorithme trie correctement cette suite $(v_k)_{1 \leqslant k \leqslant n-1}$ pour donner la suite ordonnée $(v_k^*)_{1 \leqslant k \leqslant n-1}$ vérifiant $v_1^* \leqslant v_2^* \leqslant \ldots \leqslant v_{n-1}^*$. Etant

donné que $\forall k,\ w_1 \geqslant v_k^*$, la suite $\left(v_1^*, v_2^*, ..., v_{n-1}^*, w_1\right)$ est aussi une suite ordonnée dont les éléments sont ceux de la suite $(u_1, \ldots, u_n)$.

### 16.2.6 A propos de coût d'algorithme

Le tri par fusion est en $O(n \ln n)$. Il consiste à diviser un tableau en deux parties égales, à trier chacune des parties puis à fusionner les deux listes triées. Le coût de cette dernière opération est entre $O(n)$ s'il y a $n$ éléments à trier. Si on note le coût de l'algorithme $f(n)$ alors la fonction $f$ vérifie la récurrence :

$$f(2n) = 2f(n) + 2n \tag{16.3}$$

Si on prolonge la récurrence, cela donne :

$$f(n) = 2f\left(\frac{n}{2}\right) + n \tag{16.4}$$
$$f\left(\frac{n}{2}\right) = 2f\left(\frac{n}{4}\right) + \frac{n}{2} \tag{16.5}$$
$$f\left(\frac{n}{4}\right) = 2f\left(\frac{n}{8}\right) + \frac{n}{4} \tag{16.6}$$
$$... \tag{16.7}$$

Et :

$$f(n) = 2f\left(\frac{n}{2}\right) + n \tag{16.8}$$
$$= 4f\left(\frac{n}{4}\right) + 2n \tag{16.9}$$
$$= 8f\left(\frac{n}{8}\right) + 3n \tag{16.10}$$
$$... \tag{16.11}$$

La récurrence s'arrête lorsque $\frac{n}{2^k} < 1$ auquel cas le coût est nul. Le coût de l'algorithme par fusion est donc $n\frac{\ln n}{\ln 2}$ soit $O(n \ln n)$. On peut montrer que le coût du tri par fusion est $O(n \ln n)$ simplement à partir de la récurrence. On va tout d'abord déterminer le lien entre deux fonctions de classe $C^1$ vérifiant la récurrence (16.3). On suppose pour cela qu'il existe deux fonctions $f$ et $g$ qui vérifient (16.3). On pose $h = f - g$. $h$ vérifie :

$$h(2n) = f(2n) - g(2n) = 2f(n) - 2g(n) = 2h(n) \tag{16.12}$$

Comme $f$ et $g$ sont de classe $C^1$ sur $[0, \infty[$, $h$ l'est aussi. On en déduit que :

$$\forall k \in \mathbb{N},\ h(x) = 2^k h\left(2^{-k}x\right) \tag{16.13}$$
$$\forall k \in \mathbb{N},\ h'(x) = h'\left(2^{-k}x\right) \tag{16.14}$$

La dérivée $h'$ est continue, on démontre donc que $\forall x \in \mathbb{R}, h'(x) = h'(0) = \alpha$. Par extension, $h(x) = \alpha x + \beta$. Afin que la fonction $h$ vérifie la condition 16.12, il faut que $\beta = 0$. L'ensemble des solutions de l'équation 16.12 est donc $\{h(x) = \alpha x \mid \alpha \in \mathbb{R}\}$.

On pose $f(n) = n\frac{\ln n}{\ln 2}$. Si $g$ vérifie également $g(2n) = 2g(n) + 2n$ alors il existe $\alpha \in \mathbb{R}$ tel qu'on puisse écrire $g$ de la manière suivante :

$$g(n) = f(n) + \alpha n \tag{16.15}$$

Comme $O(n)$ est négligeable devant $O(n \ln n)$, ceci démontre d'une autre façon que le coût d'un tri par fusion est en $O(n \ln n)$.

### 16.2.7 Coût minimal d'un tri

Tous les tris présentés ici sont des tris par comparaison successives (excepté le tri d'entiers présenté au paragraphe 16.2.8). En effet, ils n'utilisent que le résultat de la comparaison entre deux éléments. Pour tous les tris de ce type là, il est possible de démontrer que leur coût ne peut être plus rapide que $O(n \ln n)$.

Le tri consiste à ordonner des éléments $(x_1, \ldots, x_n)$. On considère que cette suite est aussi une permutation $\sigma$ de l'ensemble ordonné $\big(x_1 = y_{\sigma(1)}, \ldots, x_n = y_{\sigma(n)}\big)$. Le tri revient à trouver la permutation $\sigma$ dans l'ensemble des permutations qui en contient $n!$. Il faut préciser aussi que le tri doit permettre de retrouver cette permutation quelle qu'elle soit et ne pas s'appliquer uniquement sur un ensemble réduit de permutations. Pour cela, on peut seulement comparer deux éléments et le résultat de cette comparaison est binaire.

Au mieux, chaque comparaison permet d'éliminer la moitié des permutations. Admettons que chaque comparaison permette de diviser l'ensemble des permutations possibles par 2, on définit $k$ tel que $2^k \geqslant n!$ ; il faudra donc au moins $k$ comparaisons afin de déterminer la permutation $\sigma$. Il ne reste plus qu'à trouver un équivalent de $k$, la formule de Stirling va nous aider :

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \tag{16.16}$$

On cherche donc $k$ tel que :

$$\begin{aligned} 2^k &\sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \\ k &\sim \ln\sqrt{2\pi n} + n\ln\frac{n}{e} && (16.17) \\ k &\sim n\ln n && (16.18) \end{aligned}$$

Le coût d'un tri par comparaisons successives est au mieux de $O(n \ln n)$. Comme c'est le coût du tri par fusion, ce raisonnement permet d'affirmer qu'un tri de coût $O(n \ln n)$ existe et qu'il est impossible de faire mieux.

### 16.2.8 Tri d'entiers

Dans certains cas, il est possible d'utiliser une méthode de tri plus rapide encore à condition que l'ensemble à trier le soit dans un ensemble discret tel que les entiers. Cette information supplémentaire sur les éléments à trier exclu ce tri des tris à comparaisons successives. La démonstration du paragraphe 16.2.7 ne s'applique

pas dans ce cas.

**Algorithme 16.6 : tri d'entiers**
On suppose que $(l_1, \ldots, l_n)$ est une liste d'entiers qu'il faut trier par ordre croissant. On détermine tout d'abord $m = \min(l_1, \ldots, l_n)$ et $M = \max(l_1, \ldots, l_n)$. On définit la suite $(p_m, \ldots, p_M)$. La première étape va compter le nombre d'occurrences de chaque valeur.

**Etape A : nombre d'occurrences**
**pour** $k = m$ **à** $M$ **faire**
$p_k = 0$

**fin pour**
**pour** $i = 1$ **à** $n$ **faire**
$p_{l_i} \longleftarrow p_{l_i} + 1$

**fin pour**

On détermine ensuite le nombre de valeurs inférieures à un certain niveau, $(P_m, \ldots, P_M)$. Elle correspond à la fonction de répartition du vecteur $(l_1, \ldots, l_n)$ tandis que le vecteur $(p_m, \ldots, p_M)$ correspond à sa fonction de densité.

**Etape B : fonction de répartition**
$P_m \longleftarrow p_m$ **pour** $k = m + 1$ **à** $M$ **faire**
$P_k \longleftarrow P_{k-1} + p_k$

**fin pour**

On termine par l'obtention de la suite triée $(s_1, \ldots, s_n)$.

**Etape C : tri**
$k \longleftarrow 0$
**pour** $i = 2$ **à** $n$ **faire**
**tant que (** $P_k < i$ **) faire**
$k \longleftarrow k + 1$

**fin tant que**
$s_{i-1} \longleftarrow k + m$
**fin pour**

Chaque boucle de cet algorithme excepté est une boucle qui dépend soit de la taille de l'échantillon à trier soit de l'écart entre le maximum et le minimum. La dernière étape est de façon évidente la plus coûteuse ou au moins aussi coûteuse que toutes les autres. Le coût de l'algorithme est en $O(\max(n, M - m))$. Cet algorithme est lié à un théorème statistique. Si $X$ est une variable aléatoire et $P$ sa fonction de répartition alors $P^{-1}(X)$ suit une loi uniforme sur l'intervalle $[0, 1]$. Il suffit d'appliquer ce théorème à une loi discrète, $P^{-1}(x)$ correspond à la position de $x$ dans la liste triée. Ce tri ne s'applique qu'à des entiers, il serait néanmoins possible de discrétiser un tableau de réels à condition de savoir l'écart minimum entre deux valeurs du tableaux, information qu'il est aisé de déterminer une fois le tableau trié.

### 16.2.9 Programme du tri par sélection

```
# -*- coding: cp1252 -*-
import random

def construit_suite(n):
    """construit une liste de n nombres entiers compris entre 0 et 99"""
    l = []
    for i in range(0,n):
```

```
        l.append (random.randint(0,100))
    return l

def tri_selection(l):
    """ tri une liste l, tri par sélection"""
    # première boucle, répétition des étapes A et B
    for i in range(0,len(l)):

        # recherche du maximum, on suppose pour commencer
        # qu'il est à la position 0
        pos = 0
        # boucle de l'étape A
        for j in range(1,len(l)-i):
            if l [pos] < l [j]: pos = j

        # échange de l'étape B
        # la position du maximum est conservé dans la variable pos
        # on fait l'échange avec l'élément à la position len(l)-i-1
        k       = len(l)-i-1
        ech     = l [k]
        l [k]   = l [pos]
        l [pos] = ech

l = construit_suite(8)          # création d'une suite aléatoirement
print "liste non triée :\t",l   # affichage
tri_selection (l)               # tri
print "liste triée     :\t",l   # affichage
```

### 16.2.10 Programme du tri par fusion

```
# -*- coding: cp1252 -*-
import random

def construit_suite(n):
    """construit une liste de n nombres entiers compris entre 0 et 99"""
    l = []
    for i in range(0,n):
        l.append (random.randint(0,100))
    return l

def fusion_liste (l1,l2):
    """fusionne deux listes l1 et l2 triées par ordre croissant
    de sorte que la liste résultante soit également triée"""
    i,j,k = 0,0,0
    fin   = len (l1) + len (l2)
    l = []
    while k < fin :
        if j >= len (l2) or (i < len (l1) and l1 [i] < l2 [j]) :
            l.append (l1 [i])
            k += 1
            i += 1
        else :
            l.append (l2 [j])
            k += 1
            j += 1
    return l

def tri_fusion(l):
    """ tri une liste l par fusion"""
    if len (l) <= 1 : return None # on élimine le cas simple
    n = 1
    while n < len (l) :
        for i in xrange (0, len (l), 2*n):
```

```
            a  = i
            m  = min (len (l), a + n)
            b  = min (len (l), a + 2 * n)
            if m < b:
                t        = fusion_liste (l [a:m], l [m:b])
                l [a:b] = t
        n *= 2

l = construit_suite(13)          # création d'une suite aléatoirement
print l [0:3]
print "liste non triée :\t",l    # affichage
tri_fusion (l)                   # tri
print "liste triée     :\t",l    # affichage
```

### 16.2.11 Programme du tri par insertion dichotomique

```
# -*- coding: cp1252 -*-
import random

def construit_suite(n):
    """construit une liste de n nombres entiers compris entre 0 et 99"""
    l = []
    for i in range(0,n):
        l.append (random.randint(0,100))
    return l

def recherche_dichotomique (l,x):
    """cherche l'élément x dans la liste l, la liste l est
    supposée être triée par ordre croissant"""
    a = 0
    b = len(l)-1
    while a <= b :
        m = (a+b) // 2
        r = cmp (x, l[m])
        if r == 0 : return m
        elif r == -1 : b = m-1
        else : a = m + 1
    return a

def tri_insertion(l):
    """ tri une liste l par insertion dichotomique, on suppose que l est non vide"""
    lt = []
    for x in l :
        if len (lt) == 0 :
            lt.append (x)
            continue
        m = recherche_dichotomique (lt, x)
        lt.insert (m, x)
    l [0:len (l)] = lt


l = construit_suite(13)         # création d'une suite aléatoirement
print l [0:3]
print "liste non triée :\t",l   # affichage
tri_insertion (l)               # tri
print "liste triée     :\t",l   # affichage
```

### 16.2.12 Programme du tri d'entiers

```
# -*- coding: cp1252 -*-
import random
```

```
def construit_suite(n):
    """construit une liste de n nombres entiers compris entre 0 et 99"""
    l = []
    for i in range(0,n):
        l.append (random.randint(0,10))
    return l

def tri_entiers(l):
    """ tri une liste l, les éléments à trier sont entiers"""
    # première boucle, minimum, maximum
    m = l [0]
    M = l [0]
    for k in range(1,len(l)):
        if l [k] < m : m = l [k]
        if l [k] > M : M = l [k]

    # calcul du nombre d'occurrences
    p = [0 for i in range (m,M+1) ]
    for i in range (0, len (l)) :
        p [ l [i] - m ] += 1

    # fonction de répartition
    P = [0 for i in range (m,M+1) ]
    P [0] = p [0]
    for k in range (1, len (p)) :
        P [k] = P [k-1] + p [k]

    # tri
    pos = 0
    for i in range (1, len (l)) :
        while P [pos] < i : pos += 1
        l [i-1] = pos + m
    l [len (l)-1] = M


l = construit_suite(8)          # création d'une suite aléatoirement
print "liste non triée :\t",l   # affichage
tri_entiers (l)                 # tri
print "liste triée     :\t",l   # affichage
```

Quatrième partie

# VISUAL BASIC FOR APPLICATIONS (VBA)

# Chapitre 17

# Introduction

## 17.1 Visual Basic et macros

Excel est un logiciel très répandu dès qu'on manipule des chiffres. Certaines de ces manipulations reviennent très souvent et on éprouve le besoin de les automatiser. Les logiciels ont fait apparaître les macros : une succession d'opérations que l'utilisateur pouvait enregistrer puis reproduire. Dans Microsoft Office, à chaque macro correspond un programme informatique qui définit les opérations effectuées.

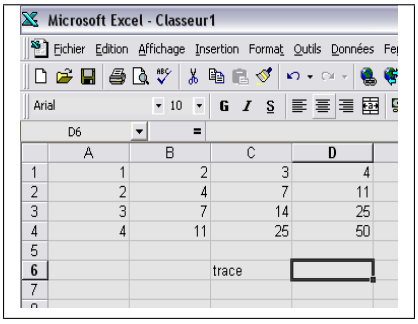



**Figure 17.1 :** *Une matrice sous Excel 4x4, on souhaite en calculer la trace.*

La figure 17.1 montre une matrice. On veut enregistrer une macro qui va mémoriser les opérations permettant d'entrer la formule de la trace de la matrice. La figure 17.2 montre les trois étapes qui permettent de créer cette macro.




**Figure 17.2 :** *On lance l'enregistrement d'une macro dans la première image, on lui donne un nom dans la seconde, on saisit la formule et on arrête l'enregistrement dans la troisième image.*

Une fois que la macro est enregistrée, Excel crée un code écrit en Visual Basic qui correspond à ces opérations. Pour le voir, on ouvre l'éditeur Visual Basic.

## 17.2 Première procédure

On cherche ensuite à écrire une macro qui modifie la matrice de façon à ce que la somme des coefficients sur chaque ligne soit égale à 1. Il est difficile d'enregistrer la succession d'opérations comme dans le paragraphe

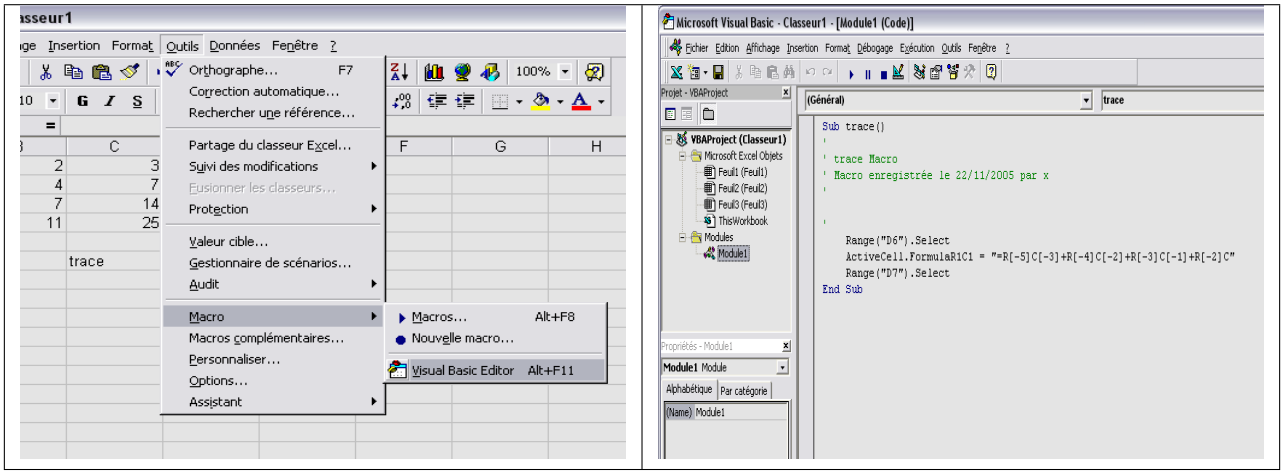



**Figure 17.3 :** *On ouvre l'éditeur Visual Basic dans Excel (cette opération est la même dans Word, Access ou toute autre application de la gamme Microsoft Office). Il faut ensuite cliquer sur Module1 pour voir apparaître le code de la macro. Les lignes commençant par une apostrophe sont des commentaires et ne sont pas pris en compte lors de l'exécution de la macro. La première de la macro signifie que la case qui doit recevoir le résultat est sélectionnée. La seconde ligne est la saisie de la formule. La troisième ligne est un clic dans une case voisine.*

précédent. En effet, la normalisation de la matrice nécessite de calculer d'abord la somme des coefficients sur une ligne et ensuite de diviser chaque coefficient par la somme obtenue pour sa ligne. Il est impossible de réaliser ce calcul directement sur la feuille Excel sans recopier la matrice à moins de programmer cette macro. On retourne donc dans l'éditeur Visual Basic pour y ajouter les lignes suivantes :

```
Sub normalisation()
    ' cette fonction renormalise une matrice 4x4
    Dim s As Double
    Dim cel As String
    Dim i As Long
    Dim j As Long

    For i = 1 To 4
        s = 0
        For j = 1 To 4
            s = s + Worksheets("Feuil1").Cells(i, j)
        Next
        For j = 1 To 4
            Worksheets("Feuil1").Cells(i, j) = Worksheets("Feuil1").Cells(i, j) / s
        Next
    Next
End Sub
```

Une fois cette étape terminée, on revient dans la fenêtre *Excel*. On souhaite ajouter un bouton à la barre des tâches et appeler la macro `normalisation` lorsqu'on clique sur ce bouton. Pour cela, il faut suivre les instructions de la figure 17.4.

## 17.3 Macro liée à une feuille Excel

Il est possible, au milieu d'une feuille Excel, d'insérer un bouton lui-même relié à une macro. Lorsque vous changez la valeur d'une cellule, Excel calcule à nouveau toutes les formules qui en dépendent. Parfois, ces calculs sont compliqués, et lorsqu'il faut changer dix valeurs, ils sont effectués dix fois. La manipulation d'Excel s'en trouve ralentie. Il est alors possible d'insérer un bouton à l'intérieur d'une feuille Excel de

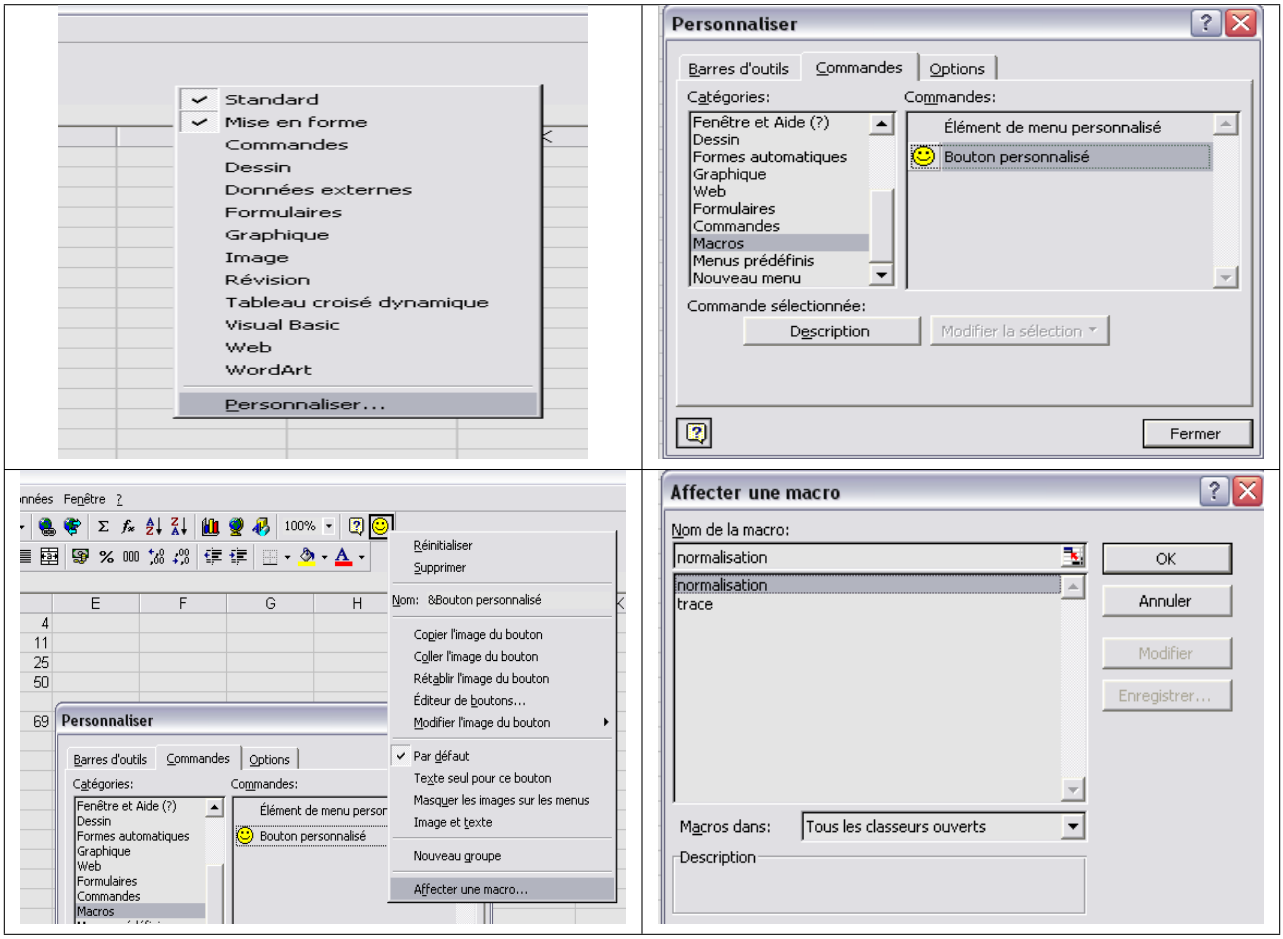



**Figure 17.4 :** *On clique d'abord sur le clic droit de la souris au niveau de la barre des tâches puis sur Personnaliser (première image). Une fenêtre apparaît, il faut cliquer sur l'onglet Commande puis sur Macro (deuxième image). La troisième étape consiste à déplacer (on clique dessus et on relâche à l'endroit désiré) le bouton personnalisé dans une barre des tâches. Il s'insère alors dedans. A nouveau un clic droit de la souris sur ce nouveau bouton, puis on clique sur Affecter une macro (troisième image). Une dernière fenêtre apparaît (dernière image), il suffit de sélectionner* `normalisation` *pour associer cette procédure au nouveau bouton. De retour dans Excel, on clique sur le nouveau bouton. La procédure s'exécute et la matrice est aussitôt modifiée.*

sorte que, lorsque ce bouton est pressé, c'est une macro qui est exécutée et qui effectue ces coûteux calculs lorsque vous l'avez décidé.

La figure 17.5 montre les étapes à suivre pour insérer un bouton dans une feuille de calcul. La figure 17.6 montre comment modifier le nom du bouton et le rattacher à une macro.

## 17.4 Erreurs

Il arrive parfois qu'une procédure ne fonctionne pas ou plante. Dans le cas présent, pour faire planter la procédure `renormalisation`, il suffit de modifier la première ligne de la matrice pour n'y mettre que des zéros. Un nouveau clic sur le bouton associé à cette macro et une erreur est provoquée comme le montre la figure 17.7. Le logiciel Excel est assez souple puisqu'il autorise la programmation et les erreurs ne provoque pas l'arrêt d'Excel.

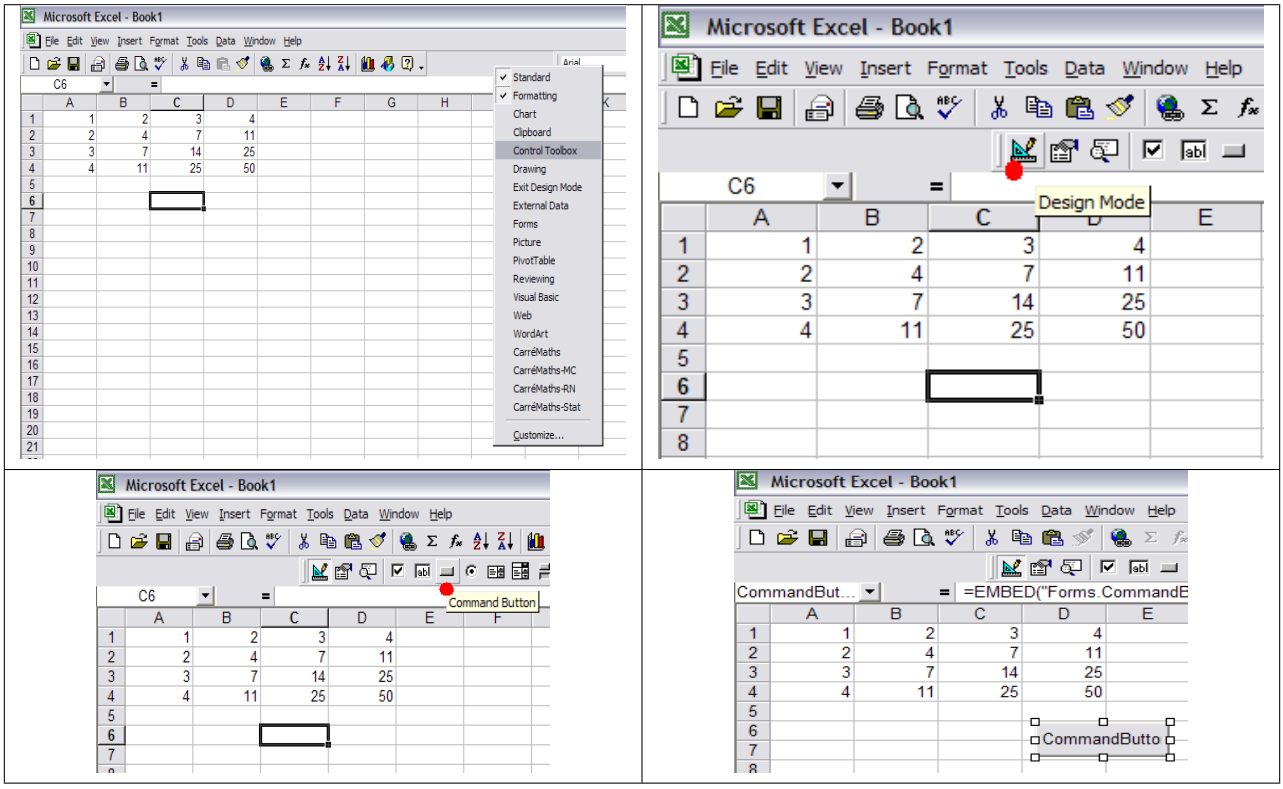



**Figure 17.5 :** *La première étape consiste à dessiner le bouton dans la feuille de calcul. Il faut faire d'abord apparaître une barre d'outils nommé Control ToolBox en cliquant sur le bouton droit de la souris au niveau des barres d'outils (première image). Il faut ensuite préciser à Excel qu'on désire insérer un bouton (ou plus généralement un contrôle). On clique pour cela comme indiqué sur la seconde image pour être en Design Mode. On clique alors sur un icône ressemblant à un bouton (troisième image). Il ne reste plus qu'à dessiner le bouton sur la feuille (quatrième image). On clique à un endroit puis on maintient appuyé jusqu'à ce que le bouton soit assez grand. Il est possible par la suite de le déplacer (uniquement en Design Mode. (suite figure 17.6, page 403)*

## 17.5 Autres possibillités

Visual Basic permet la réalisation de petits programmes. Même si les calculs sont possibles, ils sont plus lents qu'avec un langage comme le C++. Néanmoins, l'avantage est qu'Excel dispose d'une interface dans laquelle il est facile d'afficher les résultats d'un calcul ou de construire un graphique. Il est même possible d'introduire des boîtes de dialogue.

Excel n'est pas la seule application, Word, Access, Outlook sont toutes des applications qui peuvent être pilotées avec Visual Basic. Toutefois Excel et Access (base de données) sont les deux applications principales avec lesquels on écrit des programmes en VBA.

Pour Excel, je citerai le livre [Riva2005]. Il illustre l'utilisation de VBA et Excel au travers de la réalisation d'un pricer de produits financiers. Il s'étend peu sur le langage VBA mais le livre commence par un résumé sur le langage et les objets manipulés dans les programmes qui suivent en exemple.

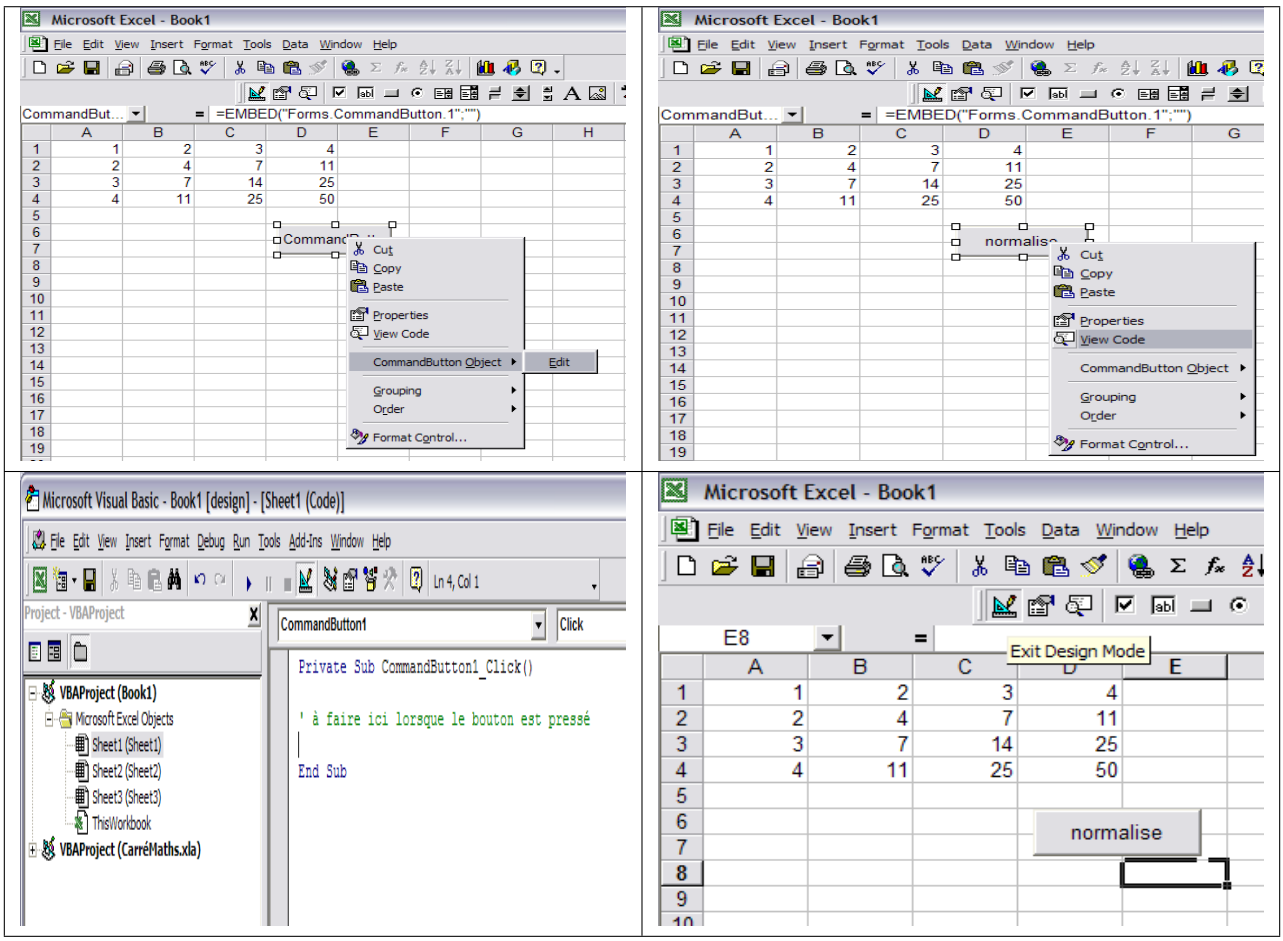



**Figure 17.6 :** *Après avoir fait apparaître un menu local en cliquant sur le bouton droit de la souris au niveau du bouton, la première image de cette série montre où faut il aller pour modifier le nom du bouton. Une fois que cela est fait, on fait apparaître de nouveau ce même menu local (seconde image) et on clique sur View Code pour faire apparaître le code de la macro associée au bouton. La macro apparaît dans l'éditeur Visual Basic (troisième image), il ne reste plus qu'à la définir. A chaque fois que le bouton sera pressé, c'est cette macro qui sera exécutée. La dernière image est là pour rappeler qu'une fois toutes ces étapes terminées, il ne faut pas oublier de revenir dans Excel et de sortir du mode Design Mode pour utiliser normalement Excel et surtout pouvoir cliquer sur le bouton.*

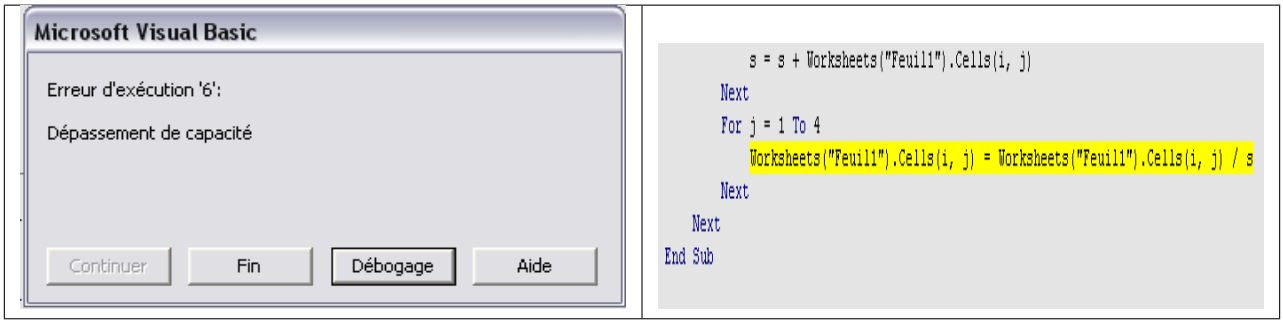



**Figure 17.7 :** *Une erreur survient car la somme des coefficients de la première ligne est nulle ce qui implique une division par zéro. C'est opération est impossible et provoque l'ouverture de la fenêtre de la première image. Après avoir cliqué sur Débogage, on se retrouve dans l'éditeur Visual Basic avec une ligne en jaune : c'est la ligne qui a provoqué l'erreur. Le menu Affichage (ou View en anglais) permet de faire apparaître une fenêtre Watch. Cette fenêtre facilite la mise au point des programmes car lorsque le programme s'arrête en cours d'exécution, il est possible d'afficher le contenu des variables en cours d'utilisation.*

# Chapitre 18

# Le langage VBA

---

Le VBA est semblable à tout autre langage de programmation, il contient des boucles, tests, fonctions ou encore procédures. Il diffère un peu par les types de variables qu'il propose, les objets qu'il manipule, liés aux applications Microsoft Office. Ce chapitre résume la syntaxe de ce langage.

Tout programme écrit en VBA peut être réparti sur plusieurs modules bien que ceux-ci soient enregistrés dans un seul fichier (une feuille Excel, un document Word, ...).

Il ne faut pas oublier l'aide fournie par Microsoft. Lorsque vous avez un doute sur une fonction ou un mot-clé, il vous suffit de l'écrire et d'appuyer sur la touche F1. Vous aurez alors automatiquement l'aide associé à ce mot, s'il existe. Une autre manière d'obtenir des informations sur ce langage consiste à enregistrer une macro en rapport avec la tâche à programmer et d'extraire du code VBA produit par Microsoft Office les informations désirées.

## 18.1 Préliminaires

Les commentaires commencent par une apostrophe et se termine par la fin de la ligne. Il n'y a pas de séparateur d'instructions en VBA, le langage impose une instruction par ligne. Lorsque l'une d'elle est trop longue et qu'il est préférable pour des raisons de lisibilité d'étendre une instruction sur plusieurs lignes, chacune d'elles excepté la dernière doit se terminer par _.

```
MsgBox "première partie" & _
       "seconde partie\n" & _
       "troisième partie"
```

## 18.2 Variables et opérateurs

### 18.2.1 Types prédéfinis

La déclaration d'une variable se fait toujours selon le même schéma :

```
Dim nom_variable as nom_type
```

Les principaux types de variables sont regroupés dans la table 18.1. Voici quelques exemples de déclarations :

```
    Dim a As Integer
    Dim s As String
    Dim v As Variant
```

Le langage VBA nécessite la déclaration des variables avant leur utilisation. Toutefois, lorsqu'elle n'est pas présente, Excel ou tout autre application de Microsoft Office ne remarque pas toujours l'erreur car il est possible que le programme ait un sens même dans ce cas. Il est donc conseillé d'ajouter au début de

tout module l'instruction `Option Explicit`. De cette manière, l'oubli d'une déclaration sera détecté avant même l'exécution du programme.

VBA autorise également les tableaux. Ils possèdent une ou plusieurs dimensions. Le premier indice est 0 et le dernier `dim1`. La déclaration se fait comme suit :

`Dim nom_variable (dim1, dim2, ...) as nom_type`

VBA permet aussi de changer les indices d'un tableau de `debut` à `fin` inclus :

`Dim nom_variable (debut To fin, ...) as nom_type`

Pour accéder à un élément du tableau, il faut utiliser les parenthèses. Quelques exemples :

```
' mois est un tableau de chaînes de caractères
' dont les indices vont de 0 à 12 inclus
Dim Mois(12) As String

' Matrice est un tableau à deux dimensions d'entiers
Dim Matrice(3, 4) As Integer

' Matrice2 est un tableau de réels à deux dimensions pour lesquels
' les indices désirés sont explicitement spécifiés
Dim Matrice2(1 To 5,  4 To 9,  3 To 5) As Double

' accès à un paramètre
Dim t(3 To 6) As Integer
t(3) = 4
```

### 18.2.2 Opérateur

Les opérations sont les opérations standards +,-,/,*, ^ (puissance), `Mod` (congruence). On peut concaténer les chaînes de caractères grâce à l'opérateur &.

```
3 + 4          ' vaut 7
31 Mod 5       ' vaut 1
2^4            ' vaut 16
"un" & "deux"  ' vaut undeux
```

Les opérateurs de comparaisons sont <,>,<=,>=,=[1]. Les opérations logiques `And` et `Or` permettent de combiner les comparaisons.

1. L'opérateur de comparaison ne contient qu'un seul signe =.

| type | définition |
|---|---|
| `Byte` | octet, entier de 0 à 255 |
| `Boolean` | booléen, `True` ou `False` |
| `Integer` | entier de -32768 à 32768 |
| `Long` | entier de -2.147.483.648 et 2.147.483.647 |
| `Double` | réel, 14 chiffres de précision, exposant allant jusqu'à 308 |
| `Date` | date |
| `String` | chaîne de caractères |
| `Variant` | équivalent à tout type, en particulier les tableaux |

**Table 18.1 :** *Principaux types de variables en VBA.*

```
((3 < 4) And (5 < 6)) Or (2 > 1)
```

**Remarque 18.1 : chaîne de caractères, caractères spéciaux**
Lorsqu'on veut insérer un passage à la ligne dans une chaîne de caractères, il faut concaténer le symbol `vbCrLf`.

```
s = "première ligne" & vbCrLf & "seconde ligne"
```

### 18.2.3 Conversion

Les conversions sont parfois implicitement effectuées. Lorsqu'elles ne marchent pas, il faudra faire appel à des fonctions de conversions explicites. Pour faire simple, il en existe deux qui sont souvent utilisées :

1. La fonction `Str` convertit n'importe quoi en chaîne de caractères.
2. La fonction `Val` convertit n'importe quoi en une valeur numérique.

Exemple :

```
Dim s As String
Dim a As Double
s = Str (3.14)
a = Val ("3.14")
```

### 18.2.4 Types définis par l'utilisation

Il est possible de créer des types en combinant plusieurs types déjà existant, comme les structures en langage C. On utilise le mot-clé `Type` et la syntaxe obéit au format suivant :

```
Type Contacts
    Nom As String
    Prenom As String
    Age As Integer
End Type

Sub procedure ()
    Dim ct As Contacts
    ct.Nom = "Microsoft"
    ct.Prenom = "VBA"
    ct.Age = 10     ' environ
End Sub
```

Cet exemple fait apparaître le mot-clé `Sub` qui désigne le début d'une procédure [2]. La définition d'un type ne peut être faite à l'intérieur d'un sous-programme (fonction ou procédure).

Lorsque le nom de la variable est trop long [3] ou simplement qu'on désire éviter sa répétition, on utilise le mot-clé `With` et la syntaxe équivalente suivante :

```
Type Contacts
    Nom As String
    Prenom As String
    Age As Integer
End Type
```

2. ou fonction ne retournant aucun résultat
3. cas fréquent lorsqu'on manipule les objets de Microsoft

```
Sub essai()
    Dim ct As Contacts
    With ct
        .Nom = "Microsoft"
        .Prenom = "VBA"
        .Age = 10
    End With
End Sub
```

Il n'est pas possible avec le mot-clé `Type` de définir une classe mais le paragraphe suivant aborde ce point.

### 18.2.5 Classes

La déclaration d'une classe s'effectue dans un module à part : un module de classe. La figure 18.1 montre comment créer un module de classe.

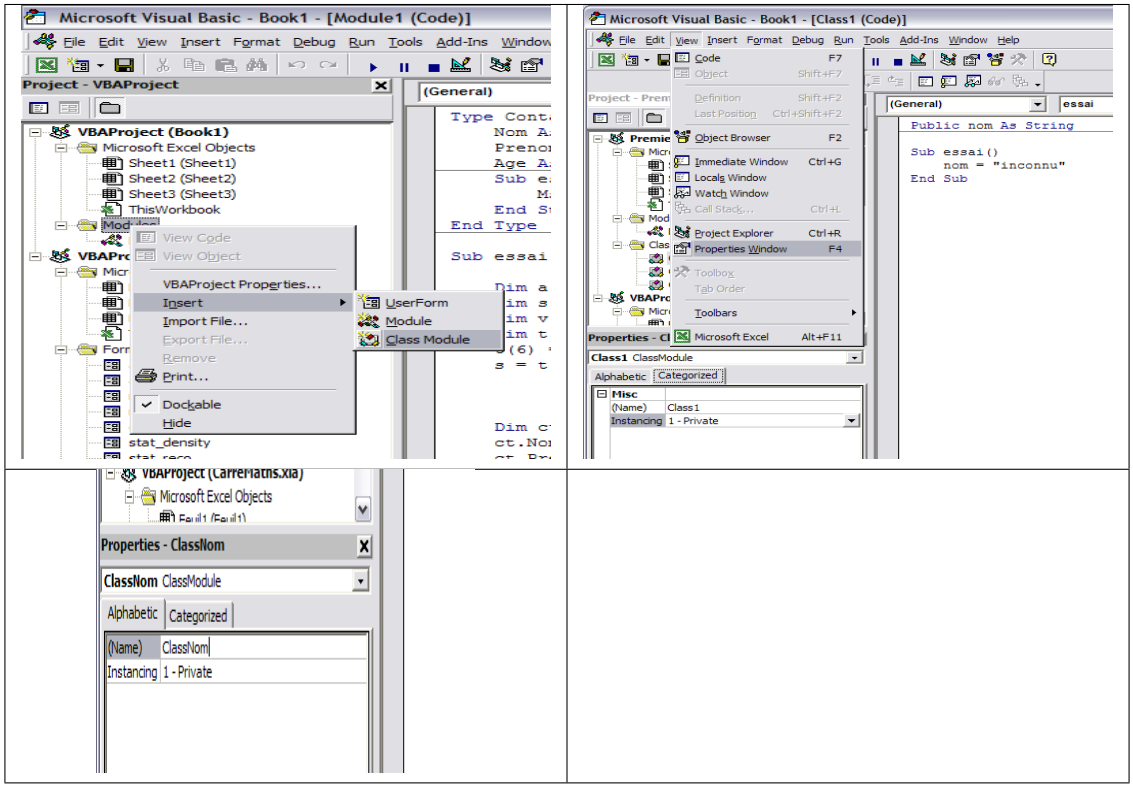



**Figure 18.1 :** *Création d'un module de classe en cliquant sur le clic droit de la souris dans la fenêtre* `VBAProject`*. La première image montre comment créer le module de classe. La seconde et la troisième montrent comment donner un nom à la classe créée. Le nom choisi est ici* `ClassNom`*.*

A l'intérieur de ce module de classe, les attributs de la classe sont déclarés comme une variable globale via le mot-clé `Public`. Voici un exemple de module de classe incluant un attribut et une méthode. Son nom est `ClassNom` :

```
Public nom As String

Sub essai()
    nom = "inconnu"
End Sub
```

La déclaration des variables de types classes diffère aussi de celle des autres types, il faut ajouter le mot-clé `New`. New

```
    Dim c As New ClassNom
    c.nom = "eeee"
    c.essai
```

VBA offre la possibilité d'ajouter un constructeur et un destructeur. Ils possèdent des noms particuliers :

```
Private Sub Class_initialize()
    ' code du constructeur
End Sub

Private Sub Class_Terminate()
    ' code du destructeur
End Sub
```

## 18.3 Affichage

L'affichage du résultat d'un calcul dépend de l'application dans laquelle on exécute le programme VBA. Ce sera dans la case d'une feuille Excel, dans un paragraphe d'un document Word, dans un mail d'Outlook, dans une table d'Access. Avant d'aborder ces applications, il existe un moyen simple de faire apparaître à l'écran un message à l'intérieur d'une boîte de dialogue. La figure 18.2 montre ce qui surgit à l'écran.

```
MsgBox "message"
```

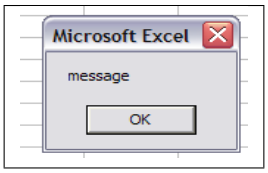



**Figure 18.2 :** *Boîte de dialogue dans laquelle apparaît* "`message`"*. Le programme s'arrête puis reprend lorsque le bouton* `OK` *est pressé.*

Lorsque le message est long, il est nécessaire d'insérer des sauts de lignes, il faut utiliser le symbole `vbCrLf` (voir remarque 18.1). La boîte de dialogue peut inclure plusieurs boutons en ajoutant un second paramètre lors de l'appel de la fonction `MsgBox`.

```
Reponse = MsgBox ("Voulez-vous continuer ?", vbYesNo)
```

Les différentes valeurs possibles sont `vbAbortRetryIgnore`, `vbOKCancel`, `vbRetryCancel`, `vbYesNo`, `vbYesNoCancel`. La valeur retournée dépend du bouton pressé, cette valeur peut être `vbOK`, `vbCancel`, `vbAbort`, `vbRetry`, `vbIgnore`, `vbYes`, `vbNo`.

Il est aussi possible de demander un renseignement à l'utilisateur avec la fonction `InputBox`. Le résultat est une chaîne de caractères. C'est le moyen le plus simple mais VBA offre la possibilité de définir des boîtes de dialogue plus complexe appelées *formulaire* qui permettent de saisir plusieurs informations dans la même fenêtre.

```
Dim Message As String
Message = InputBox("intitulé de la question", "nom de la boîte de dialogue", _
                                   "réponse par défaut")
```

## 18.4 Tests

### 18.4.1 Tests avec `If`

Les tests sont écrits grâce aux mots-clé `If` et `Then` :

```
If condition Then
    ' faire..
End If
```

S'il n'y qu'une instruction à exécuter lorsque le test est vérifié, il est possible de l'écrire sur une ligne et la dernière ligne `End If` disparaît :

```
If condition Then   ' faire
```

Il peut y avoir une contre partie :

```
If condition Then
    ' faire..
Else
    ' sinon faire...
End If
```

On peut aussi enchaîner les tests :

```
If condition1 Then
    ' faire..
ElseIf condition2 Then
    ' faire...
Else
    ' sinon faire...
End If
```

### 18.4.2 Tests avec `Select Case`

Il existe une autre manière d'écrire des tests lorsque les instructions à exécuter dépendent de la valeur d'une variable discrète.

```
Dim NiveauEau As Integer
Dim Mention As String
NiveauEau = 10
Select Case NiveauEau
    Case 0
        Mention = "sec"
    Case 1 To 5
        Mention = "presque sec"
    Case 6 To 10
        Mention = "normal"
    Case 11 To 15
        Mention = "trop d'eau"
    Case 16 To 19
        Mention = "inondations"
    Case Else
        Mention = "on déménage"
End Select
```

## 18.5 Boucles

### 18.5.1 Boucle `For`

```
Dim s as Integer
Dim i as Integer ' déclarer la variable de la boucle
s = 0
For i = 1 To 10
    s = s + 1
Next i           ' passer au i suivant
```

Il est possible de faire en sorte que la variable de boucle soit incrémentée d'une valeur positive ou négative à chaque itération.

```
Dim s as Integer
Dim i as Integer ' déclarer la variable de la boucle
s = 0
For i = 1 To 10 Step 2  ' i = 1 3 5 ... 9
    s = s + 1
Next i           ' passer au i suivant
```

Pour sortir de la boucle `For` avant sa fin prévue, il faut utiliser l'instruction `Exit For`.

### 18.5.2 Boucle `While`

```
Dim s as Integer
Dim i as Integer ' déclarer la variable de la boucle
s = 0
i = 1
While i <= 10
    s = s + 1
    i = i + 1
Wend
```

Il n'est pas possible de sortir d'une boucle `While`.

### 18.5.3 Boucle `Loop`

```
Dim s as Integer
Dim i as Integer ' déclarer la variable de la boucle
s = 0
i = 1
Do While i <= 10
    s = s + 1
    i = i + 1
Loop
```

Pour sortir de la boucle `Do While` avant sa fin prévue, il faut utiliser l'instruction `Exit Do`.
Même chose mais la condition est placée et évaluée à la fin de la boucle :

```
Dim s as Integer
Dim i as Integer ' déclarer la variable de la boucle
s = 0
i = 1
Do
```

```
    s = s + 1
    i = i + 1
Loop While i <= 10
```

## 18.6 Fonctions et procédures

### 18.6.1 Procédures

C'est une fonction qui ne retourne pas de résultat.

```
Sub exemple_procedure ()
   ' code de la procédure
End Sub
```

Il est possible de sortie prématurément de la procédure en écrivant `Exit Sub`. Contrairement à d'autres langages, il ne faut pas mettre de parenthèses autour des paramètres d'une procédure lorsqu'on l'appelle.

```
Sub exemple_procedure (ByVal param1 As Long)
   ' code de la procédure
End Sub
Sub main ()
    ' appel de la procédure
    exemple_procedure 3
End Sub
```

### 18.6.2 Fonctions

```
Function exemple_fonction () As Integer
   ' code de la procédure
   ...
   ' retourner le résultat
   exemple_fonction =
End Function
```

Il est possible de sortir prématurément de la procédure en écrivant `Exit Function`. Toutefois, avant d'exécuter une telle instruction (qui est facultative), il ne faut pas oublier de préciser quel est son résultat en écrivant le nom de la fonction suivant du signe = et du résultat à retourner.

### 18.6.3 Variables statiques

Les variables statiques sont créées une seule fois au cours de l'exécution du programme et conservent la dernière valeur qui leur a été affectée lorsque le programme quitte la procédure ou la fonction qui la contient. Au prochain appel de cette procédure ou fonction, la variable statique a retenu sa précédente valeur.

```
Sub proc ()
   Static i As Integer
End Sub
```

### 18.6.4 Passages des paramètres

Comme dans tous les langages, il existe manières de transmettre des paramètres à une fonction :

1. Le passage par valeur : les paramètres sont recopiés et peuvent être modifiés dans la fonction ou la procédure sans conséquence à l'extérieur du sous-programme.
2. Le passage par adresse : les paramètres ne sont pas recopiés. Lorsqu'ils sont modifiés dans la fonction ou la procédure, ces modifications demeurent après que le sous-programme est terminé.

Dans l'exemple qui suit, le premier paramètre est passé par valeur, le second par référence.

```
Function exemple_fonction (ByVal s As String, ByRef v As Variant) As Long
    ' ...
End Function
```

En règle générale, les petits paramètres (Long, Double, String) sont passés par valeur car leur copie ne coûte pas grand-chose. Les paramètres plus gros comme les tableaux ou n'importe quel objet Microsoft Office, sont passés par référence car leur copie est coûteuse en temps d'exécution. Il arrive même que certains objet n'accepte pas les copies et donc les passages par valeur.

Les paramètres passés par valeur acceptent des valeurs par défaut.

### 18.6.5 Surnom : `Set`

Certaines variables ont parfois des noms longs qu'il est ennuyeux de recopier à chaque utilisation. Il est possible de leur donner un surnom grâce au mot-clé `Set`.

```
Dim nom_tres_tres_long As String
Dim s As String
Set s = nom_tres_tres_long
' par la suite, s et nom_tres_tres_long désignent la même variable
```

### 18.6.6 Fonctions couramment utilisées

Comme dans tous les langages, VBA possède des fonctions qui permettent de manipuler les chaînes de caractères, des conversions de type, de mesurer la longueur d'un tableau. Il ne s'agit pas ici de dresser l'inventaire de ces fonctions mais plutôt de donner une liste écourtée des fonctions le plus couramment utilisées, ce que fait la table 18.2. A ces fonctions, il faut aussi ajouter les fonctions numériques que *Excel* propose comme `sin`, `cos`, `abs`, `exp`, `log`, ...

## 18.7 Erreurs

Les exceptions n'existent pas en VBA mais un autre mécanisme le remplace.

```
Sub proc ()
        On Error Goto erreur
          '
          ' code susceptible de produire une erreur
          '
        Exit Sub
erreur:
        '
        ' en cas d'erreur
        '
End Sub
```

| fonction | objectif |
|---|---|
| `UBound(t)` | retourne le plus grand indice d'un tableau |
| `LBound(t)` | retourne le plus petit indice d'un tableau, il arrive souvent qu'on écrit la boucle suivante : `For i = LBound(t) To UBound(t)` |
| `Len(s)` | retourne la longueur d'une chaîne de caractères |
| `Split(s, sep)` | découpe une chaîne de caractères en un tableau, le second paramètre est le séparateur de morceaux |
| `Str(x)` | convertit `x` en une chaîne de caractères |
| `Val(s)` | convertit `s` en un réel |
| `Rnd()` | retourne un nombre aléatoire compris en 0 et 1 |
| `Round(x, d)` | arrondit un nombre `x` avec `d` chiffres après la virgule |
| `Now()` | retourne la date de maintenant |
| `Dir(s)` | Retourne le premier fichier dans le répertoire `s` ou désigné par `s`, un second appel à cette fonction retournera le fichier suivant ou sera vide si celui n'existe pas. |
| `Shell(s)` | Exécute une ligne de commande DOS. |
| `Printfile, s` | Ecrit une chaîne de caractères `s` dans un fichier `file` (`file` est un entier). |
| `Line Inputfile, s` | Lit une chaîne de caractères `s` depuis un fichier `file` (`file` est un entier). |

**Table 18.2 :** *Fonctions couramment utilisées en VBA.*

## 18.8 Fichiers

Les fonctions de lecture et d'écriture sont souvent peu utilisées en VBA parce que ces opérations s'effectuent la plupart du temps avec *Excel* lui-même. Lire et écrire dans un fichier texte est une tâche simple dans la plupart des langages, elle l'est également en VBA.

### 18.8.1 Ecriture

A la différence d'autres langages où un fichier est identifié par un type spécial, en VBA, un fichier est relié à un entier qu'il faut choisir entre 1 et 500. Chaque fichier ouvert doit être assicié à un numéro différent. Comme il est peu fréquent d'ouvrir une mulitude de fichiers simultanément, ce numéro est souvent 1. Toutefois, si ce cas se produisait, la fonction `FreeFile` retourne un numéro non utilisé.

```
Dim intFile As Integer
Dim filename as String
Dim s as String
intFile = 1                             ' ou intFile = FreeFile
filename = "c:\file.txt"                ' nom du fichier
s = "contenu du fichier"                ' chaîne à enregistrer dans le fichier
Open filename For OutPut As intFile     ' ouverture du fichier
Print intFile, s                        ' écriture dans le fichier
Close intFile                           ' fermeture du fichier
```

Comme dans les autres langages, il est possible d'ouvrir le fichier en mode écriture ou en mode `Append` auquel cas le fichier est créé s'il n'existe pas ou agrandi s'il existe : tout ce qui est écrit est ajouté à la fin.

### 18.8.2 Lecture

La lecture est un processus symétrique. Le code est inchangé à ceci près que l'ouverture du fichier est en mode lecture et qu'une boucle permet de lire toutes les lignes depuis la première jusqu'à la dernière.

```
Dim sLine as String
Dim intFile As Integer
```

```
intFile = 1                             ' ou intFile = FreeFile
Open sNomeFile For Input As intFile     ' ouverture en mode lecture

Do While Not EOF(1)                     ' tant qu'on n'a pas atteint la fin du fichier
    Line Input intFile, sLine           ' on lit une ligne et on place le contenu dans sLine
Loop
Close intFile                           ' on ferme le fichier
```

# Chapitre 19

# Exercices autour de l'automatisation de tâches quotidiennes

---

## 19.1 Création de PDF puis envoi automatique de mails

### 19.1.1 Enoncé

En préambule, cet exercice utilise un logiciel gratuit *Ghostscript* composé de deux éléments *GSview*[1] et *Ghostscript*[2] qui doit être installé en premier. La seconde requête est la présence d'une imprimante de type *Postscript* ou alors l'installation du pilote d'imprimante *HP LaserJet 5P/5MP PostScript* qui ne requiert pas la présence physique de l'imprimante ni de disque d'installation ou de téléchargement.

Un professeur de maths a instauré une coutume lors de la dernière séance de la semaine. Le vendredi est consacré à la correction d'un examen ou d'exercices. Ce professeur a décidé de regrouper les élèves par deux : le premier a obtenu une bonne à l'examen précédent tandis que le second a obtenu une mauvaise note. Il a expliqué aux élèves qu'il leur enverrait par email tous les mercredi la composition des groupes de la séance du vendredi suivant afin qu'ils ne perdent pas trop de temps à s'installer dans la salle.

Au bout de quelques semaines, cette tâche est devenue assez contraignante, elle nécessitait vingt minutes à chaque fois, vingt-cinq minutes s'il fallait rallumer l'ordinateur et se reconnecter à internet parce qu'il avait oublié. Ce professeur décida donc d'optimiser la tâche à partir d'une feuille Excel regroupant les notes des élèves et leur adresse email. Il a isolé les grandes étapes de son modeste programme :

1. tri des notes par ordre croissant
2. composition des groupes, le premier avec le dernier, le second avec l'avant-dernier et ainsi de suite
3. création d'un fichier Word contenant le résultat
4. conversion de ce fichier en un fichier Acrobat (PDF) que les élèves peuvent imprimer mais pas modifier
5. envoi de ce fichier à tous les élèves

Malgré l'ampleur de la tâche, ce professeur s'est dit qu'en perdant quelques heures pour concevoir le programme, il gagnerait vingt minutes chaque semaine, soit une quinzaine d'heures sur l'année, sans compter les heures qu'il gagnerait l'année prochaine si l'expérience s'avère fructueuse. Presque surpris, c'est plein d'entrain qu'il s'est mis à programmer.

**1)** La première étape consiste à construire une feuille Excel avec quelques élèves, leurs notes et leur email. On pourra s'inspirer de la figure 19.1. Il est conseillé de choisir un nombre pair d'élèves. Si vous utilisez de vrais emails, il est probable que vous ayez à nettoyer un peu votre boîte de message après cette séance de TD, voire beaucoup selon l'humour.

1. http://pages.cs.wisc.edu/~ghost/gsview/
2. http://pages.cs.wisc.edu/~ghost/doc/AFPL/index.htm

| | A | B | C |
|---|---|---|---|
| 1 | **Prénom** | **Note** | **Mail** |
| 2 | Archibald | 12 | Archibald@math.fr |
| 3 | Aladdin | 4 | Aladdin@math.fr |
| 4 | Albert | 18 | Albert@math.fr |
| 5 | Alphonse | 11 | Alphonse@math.fr |
| 6 | Candy | 7 | Candy@math.fr |
| 7 | Cendrillon | 1 | Cendrillon@math.fr |
| 8 | Constance | 2 | Constance@math.fr |
| 9 | Cunégonde | 6 | Constance@math.fr |
| 10 | Gertrude | 7 | Gertrude@math.fr |
| 11 | Gontrand | 6 | Gontrand@math.fr |
| 12 | Hubert | 8 | Hubert@math.fr |
| 13 | Téophile | 15 | Téophile@math.fr |
| 14 | | | |

**Figure 19.1 :** *Début pour la question 1.*

**2)** La première tâche consiste à trier les données ou de préfèrence une copie des données au cas où le programme ne serait pas bon du premier coup. On découpe cette épreuve en quatre :

1. sélection des informations à copier
2. copie
3. sélection de la copie
4. tri

Plutôt que de vous donner la syntaxe de chacune des commandes, l'objectif est d'enregistrer une macro faisant exactement ces quatre tâches et de se servir du code produit par *Excel* comme début de programme.

**3)** On s'occupe de la seconde tâche : composer les groupes. Cette partie ne nécessite pas de connaissance particulière d'Excel à part lire ou écrire le contenu d'une case. Il suffit d'utiliser pour cela la fonction `Cells` dont l'aide de Microsoft vous fournira le mode d'emploi.

**4)** Il faut maintenant construire un fichier Word. C'est la partie la plus longue dont voici les détails :

1. On sélectionne le tableau des couples. On s'inspire de la question 2.
2. On le copie. On s'inspire aussi de la question 2.
3. On le colle dans un nouveau document Word.

```
' on crée un document Word
Dim word As Object
Set word = CreateObject("Word.Application")
word.Documents.Add

' on met en forme
word.Selection.Font.Size = 16
word.Selection.Font.Bold = True

' écriture d'un petit texte dans ce nouveau document et on va deux fois à la ligne
word.Selection.TypeText "Organisation de la classe vendredi prochain" _
            & vbCrLf & vbCrLf

' vbCrLf insère un paragraphe dans le document Word
' le symbole & permet de concaténer plusieurs chaînes de caractères

' on ajoute le tableau dans le document word
```

```
word.Selection.Paste

' on change la mise en forme
word.Selection.Font.Size = 12
word.Selection.Font.Bold = False

' on écrit un petit message de fin
word.Selection.TypeText vbCrLf & vbCrLf & "Bon courage et à vendredi" _
   & vbCrLf & vbCrLf & "Votre professeur de mathématiques"
```

4. On imprime de document dans un fichier. Il faut vérifier que l'imprimante *HP LaserJet 5P/5MP PostScript* est disponible sur votre ordinateur (aller dans le menu imprimer). Sinon, il faut choisir une imprimante *PostScript* comme *Garamond**Klee* à l'ENSAE.

```
' on imprime le document dans un fichier posscript
' l'imprimante doit être installée sur votre ordinateur
word.ActivePrinter = "HP LaserJet 5P/5MP PostScript"
word.PrintOut Filename:="c:\vendredi.doc", PrintToFile:=True, _
           OutputFileName:="c:\vendredi.ps"

' Fermeture de ce document :
word.ActiveDocument.Close
Set word = Nothing
```

**5)** Il faut convertir le fichier `vendredi.ps` en `vendredi.pdf` parce que Microsoft ne sait pas produire directement un fichier Acrobat [3]. Deux options sont possibles, la première utilise la version Windows de latex :

```
Shell ("C:\texmf\miktex\bin\epstopdf.exe c:\vendredi.ps")
```

La seconde solution, parfaite pour l'ENSAE, utilise le logiciel gratuit GhostScript :

```
Dim s As String
s = Chr(34)
Shell (s & "C:\Program Files\Ghostgum\gs8.60\bin\gswin32c" & s & _
    " -q -dNOPAUSE -dBATCH -sDEVICE=pdfwrite -dPDFSETTINGS=/prepress -sOutputFile=" & _
    "c:\vendredi.pdf d:\vendredi.ps")
```

**6)** Il ne reste plus qu'à envoyer le mail à tous les élèves. Quelques indices...

```
Dim look As Object
Set look = CreateObject("Outlook.Application")
Dim email As Object
Set email = look.CreateItem(olMailItem)
email.to = ...
email.Body = "organisation de la séance de vendredi, voir pièce jointe"
email.Subject = "cours de maths"
email.Attachments.Add "c:\vendredi.pdf"
email.Send
```

Il ne reste plus qu'à déterminer l'adresse à laquelle sera envoyée le mail, c'est-à-dire la concaténation de toutes les adresses des élèves séparées par un point virgule.

**7)** Il est possible comme en Python de prendre en compte le fait que le programme qu'on vient d'écrire ne marche pas. Par exemple, il marche sur un ordinateur mais pas sur un autre parce que les logiciels nécessaires ne sont pas tous installés. Dans ce cas, VBA propose la syntaxe suivante :

3. Open Office, l'équivalent gratuit de Microsoft Office, le fait très bien. Il est probable que les prochaines versions de Microsoft Office le fassent également.

```
Sub procedure_protege_contre_les_erreurs ()

    On Error Goto Marqueur1

       ' s'il se produit une erreur dans ces lignes,
       ' le programme va directement à la ligne qui
       ' comment par Marqueur1

    On Error Goto Marqueur2

       ' s'il se produit une erreur dans ces lignes,
       ' le programme va directement à la ligne qui
       ' comment par Marqueur2

       ' la procédure s'arrête ici
       Exit Sub

Marqueur1 :

    ' on affiche un message d'erreur
    MsgBox "Une erreur s'est produite dans la première partie."
    Exit Sub

Marqueur2 :

    ' on affiche un message d'erreur
    MsgBox "Une erreur s'est produite dans la seconde partie."
    Exit Sub

End Sub
```

Il est fort probable que le programme écrit à l'ENSAE ne marche pas ailleurs parce que l'imprimante utilisée n'est pas disponible, que le programme de conversion au format Acrobat n'existe pas ou n'est pas au même endroit. Il est préférable d'afficher un message explicite dans ces cas-là plutôt que le programme ne s'arrête et laisse apparaître l'éditeur VBA.

**8)** Pour terminer, on se propose d'ajouter un bouton dans la feuille Excel qui permet d'exécuter le programme sans passer par l'éditeur VBA. Suivez les instructions des figure 19.2 et 19.3.

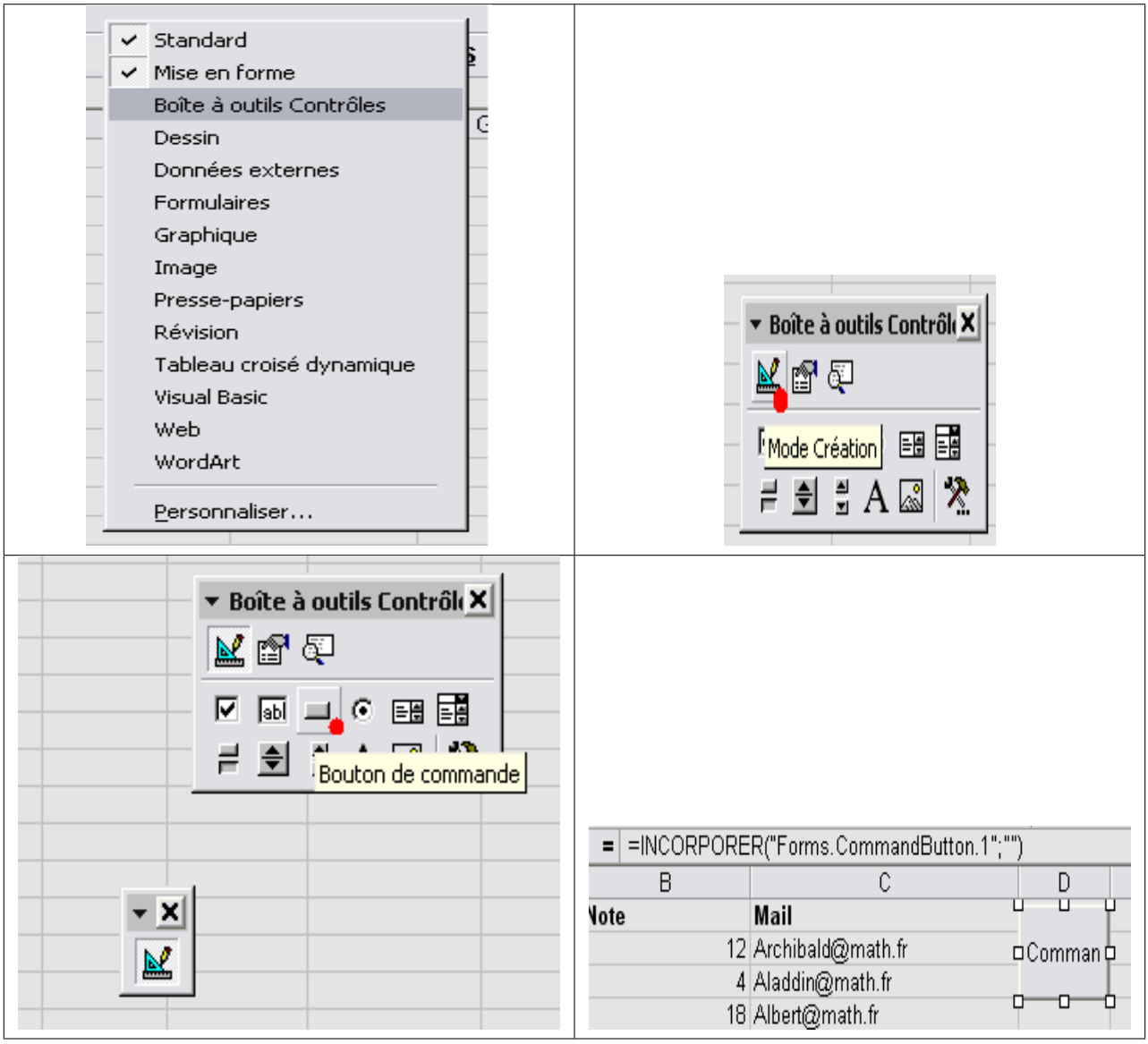



**Figure 19.2 :** *Bande dessinée : clic du bouton droit de la souris sur la barre d'outils, sélectionner Boîte à Outils. Activer le mode Création. Clic sur le bouton et l'insérer dans la feuille Excel.*

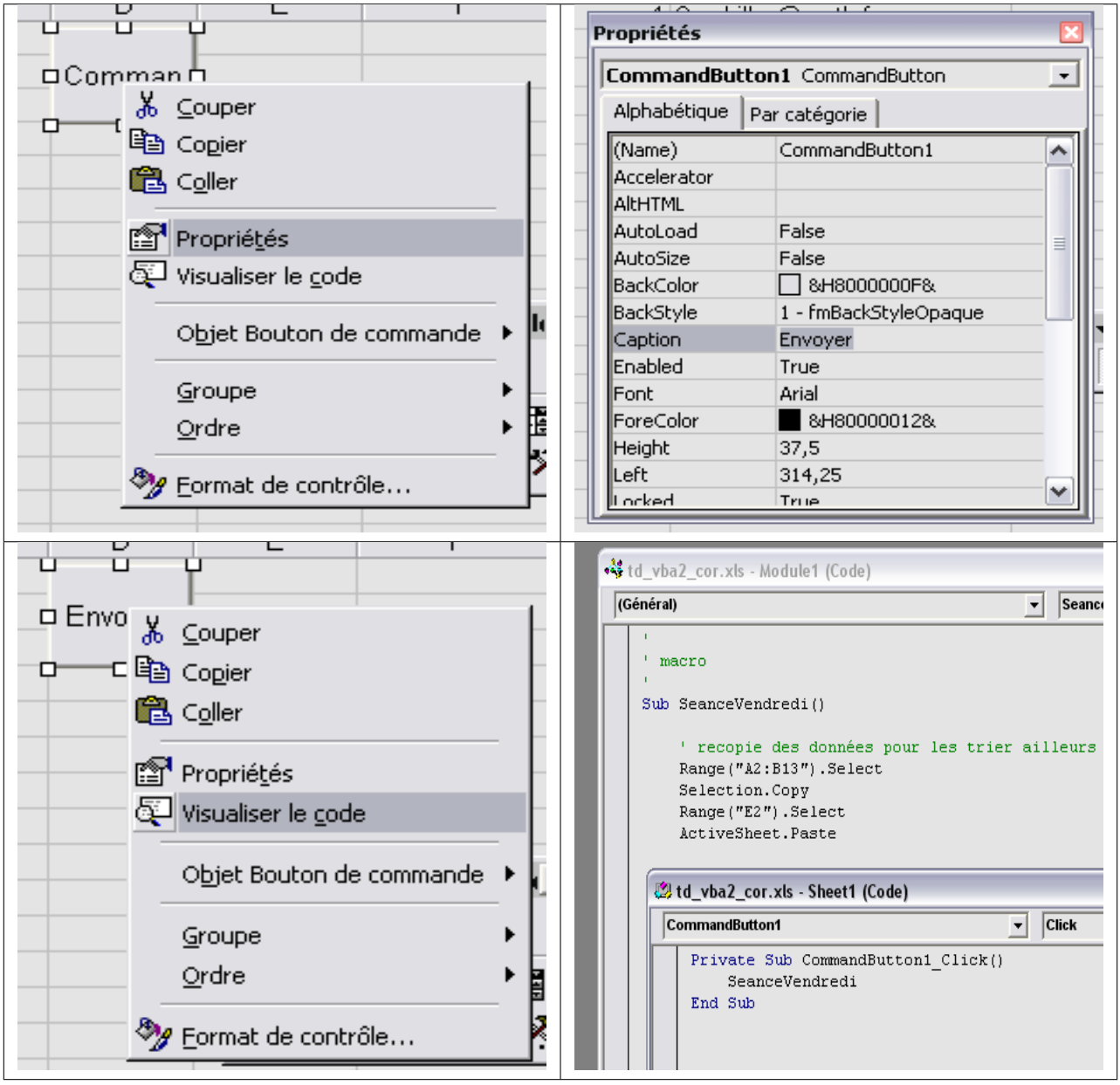



**Figure 19.3 :** *Bande dessinée : clic du bouton droit sur le bouton inséré, sélectionner Propriété. Changer l'intitulé Caption. Fermer la fenêtre des propriétés, choisir Visualiser le code. Une fenêtre s'ouvre dans l'éditeur VBA, elle contient le sous-programme qui sera exécuté à chaque pression du bouton inséré dans la feuille de calcul. C'est ici qu'il faut appeler la macro* `SeanceVendredi`*. Il suffit ensuite de basculer vers la feuille de calcul, de quitter le mode Création puis de cliquer sur le bouton de la feuille pour vérifier que tout marche bien.*

**9)** Pour ceux qui ont encore du courage, il faudrait modifier le code de la macro pour qu'elle puisse prendre en compte un nombre pair variable d'élèves.

**10)** Et si le nombre d'élèves est impair, il faut former un groupe de trois.

### 19.1.2 Correction

L'exécution de la macro `SeanceVendredi` nécessite l'utilisation du logiciel *GhostScript* (voir paragraphe 19.1.1), il écrit également des fichiers dans le répertoire `c : /temp` qui doit être présent. il faut également vérifier

qu'une imprimante *HP LaserJet 5P/5MP PostScript* est présente. Dans le cas contraire, il faut soit l'installer soit changer son nom dans le programme qui suit. Il est également possible d'utiliser *Miktex*[4] mais son installation est beaucoup plus longue.

```
'
'  convertit un fichier file1.ps en un fichier file2.pdf
'
Function conversionPDF(ByVal file1 As String, ByVal file2 As String) As String

    ' les guillemets
    Dim s As String
    s = Chr(34)

    ' version Ghostcript 8.53
    On Error GoTo solution2:
    Shell (s & "C:\Program Files\Ghostgum\gs8.53\bin\gswin32c" & s & _
               " -q -dNOPAUSE -dBATCH -sDEVICE=pdfwrite -dPDFSETTINGS=/prepress -sOutputFile=" & _
               file2 & " " & file1)
    conversionPDF = file2
    Exit Function

solution2:

    ' version Ghostcript 8.60
    On Error GoTo solution3:
    Shell (s & "C:\Program Files\gs\gs8.60\bin\gswin32c.exe" & s & _
               " -q -dNOPAUSE -dBATCH -sDEVICE=pdfwrite -dPDFSETTINGS=/prepress -sOutputFile=" & _
               file2 & " " & file1)
    conversionPDF = file2
    Exit Function

solution3:

    ' version Miktex, il faut que Miktex soit installé
    ' http://miktex.org/
    MsgBox "miktex"
    On Error GoTo pasdesolution
    Shell (s & "C:\texmf\miktex\bin\epstopdf.exe" & s & " " & file1)
    conversionPDF = file2
    Exit Function

pasdesolution:
    conversionPDF = ""

End Function

'
' macro
'
Sub SeanceVendredi()

    ' recopie des données pour les trier ailleurs
    Range("A2:B13").Select
    Selection.Copy
    Range("E2").Select
    ActiveSheet.Paste

    ' on sélectionne la zone à trier et on trie
    Range("E2:F13").Select
    Selection.Sort Key1:=Range("F2"), Order1:=xlAscending, Header:=xlNo, _
                        OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom

    ' on forme les couples
```

4. http://miktex.org/

```
' le plus fort avec le moins fort
Dim i As Long  ' position du premier
Dim j As Long  ' position du dernier
Dim k As Long  ' position du couple
Dim m1 As String
Dim m2 As String

i = 2
j = 13
k = 15

While i < j
    Cells(k, 2) = Cells(i, 5)
    Cells(k, 3) = Cells(j, 5)

    ' on fait aussi la somme des notes du groupe
    Cells(k, 4) = Cells(i, 6).Value + Cells(j, 6).Value

    k = k + 1
    i = i + 1
    j = j - 1
Wend

'
' on crée un document Word à envoyer aux élèves
'

' on copie la sélection
Range("B15:D20").Select
Selection.Copy

' on crée un document Word
Dim word As Object
Set word = CreateObject("Word.Application")
word.Documents.Add

' on met en forme
word.Selection.Font.Size = 16
word.Selection.Font.Bold = True

' écriture d'un petit texte dans ce nouveau document et on va deux fois à la ligne
word.Selection.TypeText "Organisation de la classe vendredi prochain" & vbCrLf & vbCrLf

' on ajoute le tableau dans le document word
word.Selection.Paste

' on change la mise en forme
word.Selection.Font.Size = 12
word.Selection.Font.Bold = False

' on écrit un petit message de fin
word.Selection.TypeText vbCrLf & vbCrLf & "Bon courage et à vendredi" & vbCrLf & vbCrLf & _
                                          "Votre professeur de mathématiques"

' sauvegarde de ce document ainsi créé :
word.ActiveDocument.SaveAs "c:\temp\vendredi.doc"

' pour voir le résultat et éviter que Word ne se ferme tout de suite
' word.Visible = True
' MsgBox "Cliquer pour continuer"
' word.Visible = False

' si l'impression se passe mal, on va à PasImprimante
On Error GoTo PasImprimante
```

```
    ' on imprime le document dans un fichier posscript
    ' l'imprimante doit être installée sur votre ordinateur
    word.ActivePrinter = "HP LaserJet 5P/5MP PostScript"
    word.PrintOut Filename:="c:\temp\vendredi.doc", PrintToFile:=True, _
                  OutputFileName:="c:\temp\vendredi.ps"

    ' Fermeture de ce document :
    word.ActiveDocument.Close
    Set word = Nothing

    ' conversion de ps vers pdf
    Dim ist As String
    ist = conversionPDF("c:\temp\vendredi.ps", "c:\temp\vendredi.pdf")
    If ist = "" Then GoTo PasPDF


    ' envoi d'un mail
    ' voici la version sans document attaché mais elle marche sans outlook
    'ActiveWorkbook.FollowHyperlink _
            Address:="mailto:xavier.dupre@wanadoo.fr?subject=cours de maths&body=groupes pour vendredi", _
            NewWindow:=True

    ' on récupère les adresses dans une chaîne de caractères
    Dim tous As String
    For i = 2 To 13
        tous = tous & Cells(i, 3)
        If i < 13 Then tous = tous & ";"
    Next i

    On Error GoTo PasOutlook

    ' envoi d'un mail avec Outlook
    Dim look As Object
    Set look = CreateObject("Outlook.Application")
    Dim email As Object
    Set email = look.CreateItem(olMailItem)
    email.to = tous
    email.Body = "organisation de la séance de vendredi, voir pièce jointe"
    email.Subject = "cours de maths"
    email.Attachments.Add "c:\temp\vendredi.pdf"
    email.Send

    Set email = Nothing

    Exit Sub

PasPDF:
    MsgBox "Il est impossible de convertir le fichier ps en PDF."
    Exit Sub

PasImprimante:
    ' Fermeture de ce document :
    word.ActiveDocument.Close
    Set word = Nothing
    MsgBox "Il est impossible d'imprimer."
    Exit Sub

PasOutlook:
    MsgBox "Outlook n'est pas disponible ou la conversion en PDF s'est mal passée."
    Exit Sub

End Sub
```

**fin correction TD 19.1.1** □

## 19.2 Création d'une présentation à partir de photos de vacances

### 19.2.1 Enoncé

Après un long séjour en vacances et une grande moisson de photos numériques, un baroudeur souhaite présenter quelques extraits de ses nombreuses péripéthies. Etant tout aussi baroudeur sur un ordinateur, il opte pour une aventureuse présentation sur PowerPoint à base de programme VBA. Une heure ou deux et le tour sera joué... enfin presque.

**1)** La première étape consiste à récupérer la liste des images d'un répertoire contenant des images. L'aide de Visual Basic sur la fonction `Dir` vous donnera un exemple à recopier qu'il faudra adapter pour chercher des images dont les extensions sont `jpg`, `jpeg`, `tif`, `png`, `bmp`.

**2)** A partir de la fonction précédente, il faut maintenant créer un bouton qui exécute une procédure :

1. qui récupère le nom d'un répertoire dans la première case de la feuille de calcul,
2. qui obtient la liste de toutes les images de ce répertoire,
3. qui écrit le nom de toutes ces images dans des cases de la feuille de calcul.

Petit rappel, pour accéder ou modifier le contenu d'une case, on utilise l'instruction :

```
Cells (4,3).Value = 3  ' on modifie le contenu de la ligne 4 colonne 3
```

Il sera peut-être utile de revenir sur les séances de TD précédentes : il faut d'abord créer un bouton en mode *Design* et lui associer une procédure ou macro. A l'intérieur de cette macro, on exécute les trois tâches décrites précédemment. L'indice d'un élément d'un tableau doit apparaître en parenthèses.

**3)** Il faut maintenant créer une présentation PowerPoint. On ouvre l'application PowerPoint :

```
    Dim power As Object
    Set power = CreateObject("PowerPoint.Application")
```

On crée une présentation :

```
    Dim pres As Object
    Set pres = power.Presentations.Add(WithWindow:=msoTrue)
```

On rend l'application visible :

```
    power.Visible = True
```

**4)** Ensuite, pour chaque image, il faut :

1. créer un slide, numéro `nb`

   ```
       Set slide = pres.Slides.Add(Index:=nb, Layout:=12)
       power.ActiveWindow.ViewType = 1
   ```

2. effectuer une petite magouille PowerPoint pour sélectionner le slide qu'on vient de créer

   ```
       power.ActiveWindow.Selection.Unselect
       power.ActiveWindow.View.GotoSlide Index:=nb
       power.ActiveWindow.Selection.Unselect
   ```

3. insérer l'image dont le nombre est `imnom` dans le slide

```
power.ActiveWindow.Selection.SlideRange.Shapes.AddPicture _
        (Filename:=imnom, LinkToFile:=False, SaveWithDocument:=True, _
        Left:=0, Top:=0).Select
```

4. dire qu'on a terminé de manipuler le slide par l'instruction :

```
Set slide = Nothing
```

**5)** Il suffit maintenant de répéter ces opérations pour chaque image du répertoire à l'aide d'une boucle. Quelques indices :

1. La boucle `For ... Next`.
2. Les plus petit et plus grand indices d'un tableau sont retournés par les fonctions `LBound` et `UBound`.
3. Si `res` est un tableau, `res(nb)` est l'élément d'indice `nb` du tableau. Dans notre cas, le tableau est celui retourné par la fonction de la question 1.

**6)** Les images ne sont pas toutes de tailles identiques, il faut les agrandir ou les faire rapetisser, les placer au centre. Tout d'abord une petite magouille VBA pour sélectionner l'image dont il faut modifier les propriétés :

```
power.ActiveWindow.Selection.Unselect
power.ActiveWindow.View.GotoSlide Index:=nb
power.ActiveWindow.Selection.Unselect
power.ActiveWindow.Selection.SlideRange.Shapes(1).Select
```

Grâce à ces lignes, on peut accéder ou modifier les informations suivantes :

```
With power.ActiveWindow.Selection.ShapeRange
    .Left   ' coordonnées du coin supérieur gauche
    .Top    '

    .Width  ' largeur et
    .Height ' hauteur de l'image
End With
```

La dernière information nécessaire sont les dimensions du slide. Il semblait possible de les obtenir grâce à `power.Width` et `power.Height` mais les images ne sont manifestement pas centrées. Le plus simple est, à l'intérieur d'une présentation PowerPoint, d'enregistrer une macro pendant qu'on trace une droite terminant dans le coin inférieur droit. Le code VBA de cette macro donnera les coordonnées les dimensions du slide.

Réduire, agrandir, centrer les images, tout est possible.

**7)** On termine la présentation en modifiant le fond (noir) et par la mise en place de transition automatique entre les slides. Le plus simple est d'enregistrer une macro qui change le fond blanc des slide en noir et qui modifie les transitions entre slides puis de recopier ce code dans le programme VBA.

**8)** C'est presque fini mais on voudrait que la création de la présentation soit effectuée par l'intermédiaire de la boîte de dialogue figure 19.4. Dans un premier temps, on va créer la boîte de dialogue. Il faut aller dans l'éditeur VBA et créer ce qu'on appelle un *UserForm* (figure 19.5).

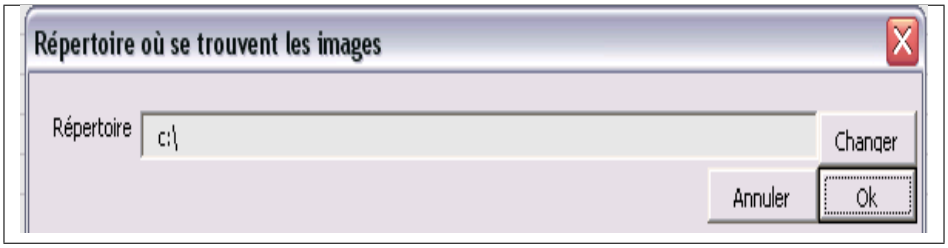



**Figure 19.4 :** *La boîte de dialogue qui permet de saisir le répertoire des images.*

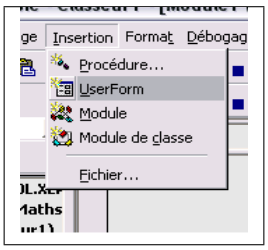



**Figure 19.5 :** *Comment insérer une boîte de dialogue ou UserForm.*

De nombreuses fenêtres apparaissent à l'écran (figure 19.6). Tout d'abord, la boîte de dialogue elle-même, vide. A côté, une première boîte à outils qui permet d'insérer ce qu'on appelle des *contrôles*. Le terme contrôle regroupe tout ce qui permet d'interagir avec l'utilisateur. Dans notre cas, ils se résumeront à une zone de saisie permettant de récupérer un chemin, une zone de texte indiquant à quoi sert la zone de texte et trois boutons comme dans la figure 19.4. La dernière fenêtre est celle des propriétés, entre autres, il y a celle intitulée *Caption* qui est le titre de la fenêtre ou du contrôle sélectionné [5], c'est-à-dire ce qui apparaît à l'écran. Il y a aussi la propriété *Name* est le nom de variable désignant le contrôle, c'est-à-dire son nom qui le désigne dans le programme (différent de qui apparaît à l'écran).

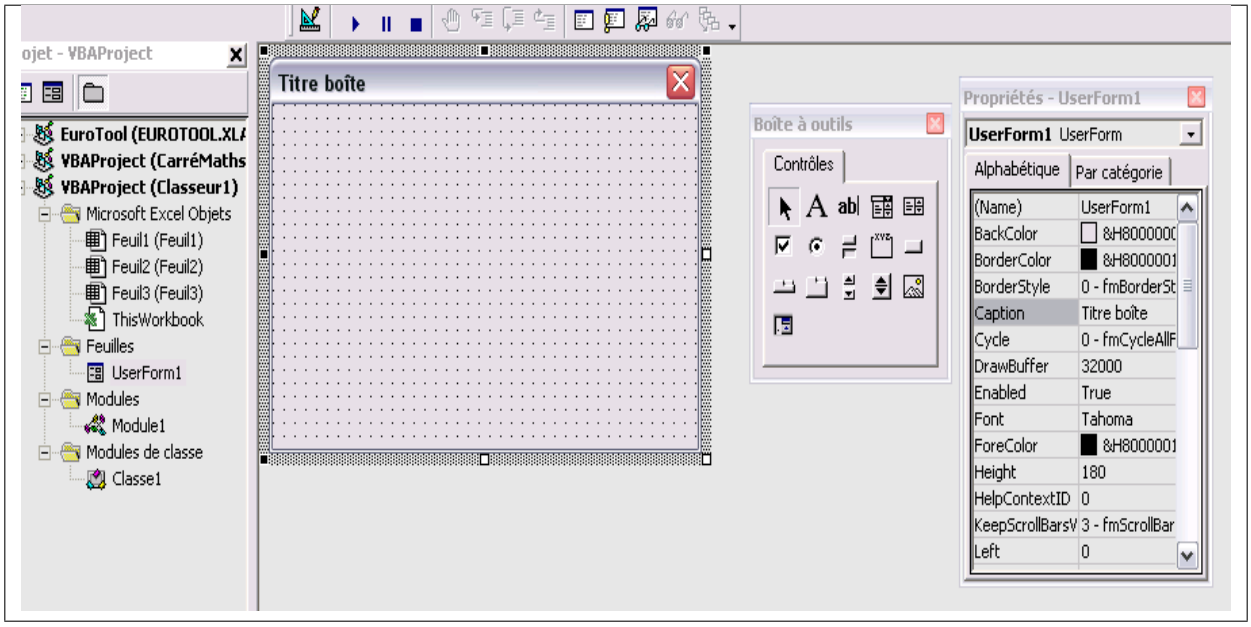



**Figure 19.6 :** *Trois fenêtres : celles de la boîte dialogue, celles des contrôles, celles des propriétés.*

Votre première mission, si vous l'acceptez, est de réussir à imiter la boîte de dialogue de la figure 19.4. En particulier, la boîte de dialogue doit avoir comme *nom* `image_dialog` et le *nom* de la zone de saisie doit être `repertoire`.

**9)** Il reste à préciser ce que doit faire Excel lorsqu'on presse chacun des boutons à commencer par le premier, celui de la feuille de calcul et qui pour l'instant crée la présentation. Et au lieu de la créer, il va maintenant afficher la boîte de dialogue. On suppose que son nom (Name) est `image_dialog` (celui défini dans ses propriétés). Elle va apparaître à l'aide des trois lignes suivantes :

```
Load image_dialog    ' création de la boîte de dialogue
image_dialog.repertoire.Text = Cells(1, 1).Value
   ' on stipule que dans la zone de saisie (dont le nom est repertoire),
   ' il doit y avoir lorsque la fenêtre apparaît
```

5. L'object sélectionné, la boîte de dialogue, un contrôle qu'elle contient, est entourée de huit petits carrés.

```
      ' le contenu de la première cellule de la feulle de calcul
    image_dialog.Show  ' Excel reprend le contrôle et attend vos instructions
                       ' après avoir affiché la boîte de dialogue
```

Pour l'instant, si vous cliquer sur le bouton de la feuille de calcul, la boîte de dialogue apparaît totalement inerte et sans aucune réaction face aux clics de souris excepté sur la petite croix en haut à droite.

**10)** On revient donc à la boîte de dialogue. Lorsqu'on clique sur le bouton *Annuler*, celle-ci doit disparaître sans rien faire. Dans l'éditeur VBA, il faut cliquer deux fois sur le bouton pour faire apparaître la procédure qui est exécutée lorsqu'on clique dessus. A cet endroit, ajoutez la ligne suivante qui détruit la boîte de dialogue :

```
' VBA ajoute automatiquement la première et la dernière de ces trois lignes
' variable_Annuler n'est pas la légende (Caption) du bouton Annuler
' mais son nom (Name)
Private Sub variable_Annuler_Click()
   Unload image_dialog
End Sub
```

**11)** On s'intéresse maintenant au bouton *Ok*. Il doit d'abord récupérer le répertoire de la zone de saisie et lancer la création de la présentation.

```
    Unload image_dialog                                                    ' on détruit la boîte de dialogue
    Cells(1, 1).Value = repertoire.Text ' on récupère le répertoire saisi
                                        ' et on le met dans la première cellule de la feuille
    ' et ici : lancer la création du diaporama
```

**12)** Enfin, le dernier bouton qui consiste à ouvrir une autre boîte de dialogue permettant de sélectionner le répertoire :

```
Private Sub Changer_Click()
    Dim s As String
    s = BrowseForFolderShell("Choisissez un répertoire", repertoire.Text)
    repertoire.Text = s
End Sub
```

Il reste à préciser ce qu'est la fonction `BrowseForFolderShell`. Compliquée sur Excel 2000, elle est plus simple sur les versions utlérieures.

**Version Excel 2000 :**

```
Function BrowseForFolderShell(title As String, repertoire As String) As String
    Dim objShell As Object
    Dim objFolder As Object
    Dim strFolderFullPath As String

    Set objShell = CreateObject("Shell.Application")
    Set objFolder = objShell.BrowseForFolder(0, titre, 0, repertoire)

    If (Not objFolder Is Nothing) Then
        On Error Resume Next
        If IsError(objFolder.Items.Item.path) Then
           strFolderFullPath = CStr(objFolder): GoTo Here
        End If
        On Error GoTo 0
        If Len(objFolder.Items.Item.path) > 3 Then
            strFolderFullPath = objFolder.Items.Item.path & Application.PathSeparator
        Else
            strFolderFullPath = objFolder.Items.Item.path
```

```
        End If
    Else
        BrowseForFolderShell = repertoire
        Exit Function
    End If

Here:
    BrowseForFolderShell = strFolderFullPath

    Set objFolder = Nothing
    Set objShell = Nothing

End Function
```

**Et la version Excel XP, plus simple et plus jolie (à l'écran) :**

```
Function BrowseForFolderShell(title As String, repertoire As String) As String
    Dim fd As FileDialog
    Set fd = Application.FileDialog(msoFileDialogFolderPicker)
    fd.title = title
    fd.AllowMultiSelect = False
    fd.InitialFileName = repertoire & "\"

    If fd.Show = -1 Then
        Dim v As Variant
        For Each v In fd.SelectedItems
            BrowseForFolderShell = v
        Next
    Else
        BrowseForFolderShell = repertoire
    End If

    Set fd = Nothing
End Function
```

*FIN.*

Le baroudeur, quant à lui, a abandonné au bout d'une heure : un flamant rose est passé par là suivi par le poisson d'Arizona Dream.

En ce qui concerne la question 1, une recherche rapide sur Internet a permis de trouver l'autre moyen qui suit :

```
Function ListFileInFolder(chemin As String) As Variant
    Dim i As Long
    Dim res() As String

    ' mise en place de la liste de fichier
    With Application.FileSearch
         .NewSearch
         .FileType = msoFileTypeAllFiles
         .Filename = "*.jpg;*.jpeg;*.tif;*.png;*.bmp"
         .SearchSubFolders = False
         .LookIn = chemin
         If .Execute() > 0 Then
            ReDim res(.FoundFiles.Count)
            For i = 1 To .FoundFiles.Count
                res(i) = .FoundFiles(i)
            Next i
         End If
    End With
```

```
    ListFileInFolder = res
End Function
```

### 19.2.2 Correction

La correction est répartie sur plusieurs blocs : deux blocs de fonctions et un bloc relié au bouton d'un fenêtre permettant de sélectionner un chemin via une autre fenêtre. Le premier bloc contient une fonctions qui retourne les listes des fichiers dans un répertoire et le code de la macro principale.

```
'
' retourne la liste des images d'un répertoire
'
Function ListFileInFolder(chemin As String) As Variant
    Dim i As Long
    Dim res() As String

    ' mise en place de la liste de fichier
    With Application.FileSearch
         .NewSearch
         .FileType = msoFileTypeAllFiles
         .Filename = "*.jpg;*.jpeg;*.tif;*.png;*.bmp"
         .SearchSubFolders = False
         .LookIn = chemin
         If .Execute() > 0 Then
            ReDim res(.FoundFiles.Count)
            For i = 1 To .FoundFiles.Count
                res(i) = .FoundFiles(i)
            Next i
         End If
    End With
    ListFileInFolder = res
End Function

'
' création d'un power point avec une liste d'images
'
Sub macro_film()

    Dim chemin As String
    Dim image As Variant
    Dim ligne As Long
    Dim colnne As Long

    ' on récupère le chemin sélectionné
    chemin = Selection.Value

    ' et sa position
    ligne = Selection.Row
    colonne = Selection.Column

    ' on récupère la liste des images
    image = ListFileInFolder(chemin)

    ' on crée un document PowerPoint
    Dim power As Object
    Set power = CreateObject("PowerPoint.Application")

    ' création d'une présentation
    Dim pres As Object
    Set pres = power.Presentations.Add(WithWindow:=msoTrue)
```

```
' on crée autant de pages qu'il y a de graphiques
Dim page As Object

' pour voir le résultat et éviter que Word ne se ferme tout de suite
power.Visible = True

' on insère les images
Dim nb As Long
For nb = 1 To UBound(image)

    ' on écrit le nom de l'image dans la cellule
    Cells(ligne + nb + 1, colonne).Value = image(nb)

    ' on ajoute un slide vide
    Set slide = pres.Slides.Add(Index:=nb, Layout:=12)
    power.ActiveWindow.ViewType = 1

    ' on sélectionne l'image pour changer sa taille
    ' magouille PowerPoint
    power.ActiveWindow.Selection.Unselect
    power.ActiveWindow.View.GotoSlide Index:=nb
    power.ActiveWindow.Selection.Unselect

    ' on insère une image
    power.ActiveWindow.Selection.SlideRange.Shapes.AddPicture _
            (Filename:=image(nb), LinkToFile:=False, SaveWithDocument:=True, _
            Left:=0, Top:=0).Select

    ' on sélectionne l'image pour changer sa taille
    ' magouille PowerPoint
    power.ActiveWindow.Selection.Unselect
    power.ActiveWindow.View.GotoSlide Index:=nb
    power.ActiveWindow.Selection.Unselect
    power.ActiveWindow.Selection.SlideRange.Shapes(1).Select

    ' on la position en haut à gauche
    With power.ActiveWindow.Selection.ShapeRange
        .Left = 0#
        .Top = 0#
    End With

    ' on sélectionne l'image pour changer sa taille
    ' magouille PowerPoint
    power.ActiveWindow.Selection.Unselect
    power.ActiveWindow.View.GotoSlide Index:=nb
    power.ActiveWindow.Selection.Unselect
    power.ActiveWindow.Selection.SlideRange.Shapes(1).Select

    ' on modifie sa taille
    With power.ActiveWindow.Selection.ShapeRange
        ' si l'image est trop petite
        If .Height < .Width And .Width < 400 Then .Width = 400
        If .Width < .Height And .Height < 400 Then .Height = 400
        ' si l'image est trop grande
        If .Width > power.Width Then .Width = power.Width
        If .Height > power.Height Then .Height = power.Height
        ' on centre l'image
        .Left = (power.Width - .Width) / 2
        .Top = (power.Height - .Height) / 2

        ' on écrit les dimensions de l'image dans les cellules suivantes
        Cells(ligne + nb + 1, colonne + 1).Value = .Left
        Cells(ligne + nb + 1, colonne + 2).Value = .Top
        Cells(ligne + nb + 1, colonne + 3).Value = .Width
```

```
            Cells(ligne + nb + 1, colonne + 4).Value = .Height
        End With

        Set slide = Nothing
    Next nb

    ' on met un fond noir
    With power.ActivePresentation.SlideMaster.Background
        .Fill.Visible = msoTrue
        .Fill.ForeColor.SchemeColor = 2
        .Fill.Transparency = 0#
        .Fill.Solid
    End With
    With power.ActivePresentation.Slides.Range
        .FollowMasterBackground = msoTrue
        .DisplayMasterShapes = msoTrue
    End With

    ' on règle les transitions entre transparents
    With power.ActivePresentation.Slides.Range.SlideShowTransition
        .EntryEffect = 2819
        .Speed = 3
        .AdvanceOnClick = msoTrue
        .AdvanceOnTime = msoTrue
        .AdvanceTime = 1
        .SoundEffect.Type = 0
    End With
End Sub
```

Le second bloc ou module contient une fonction permettant de sélectionner un répertoire.

```
Function BrowseForFolderShell(title As String, repertoire As String) As String
    Dim objShell As Object
    Dim objFolder As Object
    Dim strFolderFullPath As String

    Set objShell = CreateObject("Shell.Application")
    Set objFolder = objShell.BrowseForFolder(0, titre, 0, repertoire)

    If (Not objFolder Is Nothing) Then
        On Error Resume Next
        If IsError(objFolder.Items.Item.path) Then strFolderFullPath = CStr(objFolder): GoTo Here
        On Error GoTo 0
        '// Is it the Root Dir?...if so change
        If Len(objFolder.Items.Item.path) > 3 Then
            strFolderFullPath = objFolder.Items.Item.path & Application.PathSeparator
        Else
            strFolderFullPath = objFolder.Items.Item.path
        End If
    Else
        BrowseForFolderShell = repertoire
        Exit Function
    End If

Here:
    BrowseForFolderShell = strFolderFullPath
    Set objFolder = Nothing
    Set objShell = Nothing
End Function
```

Le dernier bloc définit les fonctions attachés aux boutons de la fenêtre de la figure 19.4 page 425.

```
Private Sub Changer_Click()
    Dim s As String
```

```
    s = BrowseForFolderShell("Choisissez un répertoire", repertoire.Text)
    repertoire.Text = s
End Sub

Private Sub Ok_Click()
    ' on supprime la boîte de dialogue
    Unload image_dialog
    Cells(1, 1).Value = repertoire.Text
End Sub

Private Sub Annuler_Click()
    ' on supprime la boîte de dialogue
    Unload image_dialog
End Sub
```

**fin correction TD 19.2.1** □

## 19.3 Résolution d'un sudoku

### 19.3.1 Enoncé

Réjoui(e) ou agacé(e) à l'idée de résoudre un Sudoku ? Et d'utiliser un ordinateur pour le résoudre à votre place ? C'est pourtant ce qui va vous arriver durant les deux heures qui suivent. La figure 19.7 montre ce qu'est un Sudoku. Ce carré incomplet qu'il s'agit de remplir obéit à certaines règles :

1. Chaque case contient un chiffre entre 1 et 9.
2. Dans chaque ligne, il ne peut y avoir deux chiffres égaux.
3. Dans chaque petit carré, il ne peut y deux chiffres égaux.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | 0 | 0 | 7 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 1 | 9 | 5 | 0 | 0 | 0 |
| 0 | 9 | 8 | 0 | 0 | 0 | 0 | 6 | 0 |
| 8 | 0 | 0 | 0 | 6 | 0 | 0 | 0 | 3 |
| 4 | 0 | 0 | 8 | 0 | 3 | 0 | 0 | 1 |
| 7 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 6 |
| 0 | 6 | 0 | 0 | 0 | 0 | 2 | 8 | 0 |
| 0 | 0 | 0 | 4 | 1 | 9 | 0 | 0 | 5 |
| 0 | 0 | 0 | 0 | 8 | 0 | 0 | 7 | 9 |

**Figure 19.7 :** *Exemple de Sudoku.*

A l'aide de ces informations, on va essayer de construire un programme sous *Excel* qui résoud les Sudoku. La figure 19.8 donne un exemple du résultat qu'on cherche à obtenir. On sélectionne la plage contenant le Sudoku, on sélectionne la plage devant recevoir le résultat, on clique sur le bouton et le tour est joué ou presque.

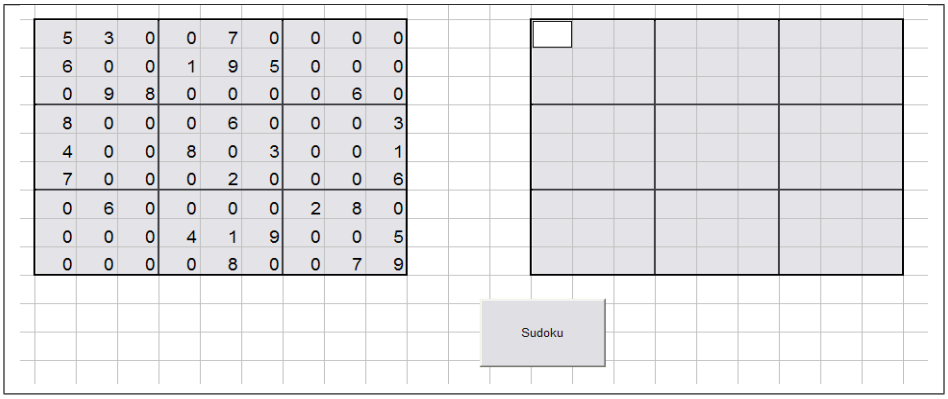



**Figure 19.8 :** *Mode d'emploi : le Sudoku est défini par un chiffre en 1 et 9 si celui-ci est connu, 0 s'il est inconnu. On sélectionne à la souris la première plage, on appuie ensuite sur la touche* Ctrl *et on sélection encore à la souris la second plage. Il ne reste plus qu'à cliquer sur le bouton toujours à la souris.*

Comment écrire ce programme ? On pourrait choisir une case vide, y poser un chiffre au hasard mais différent de ceux déjà présents sur la même ligne et dans le petit carré auquel appartient la case vide. Une fois cette case remplie, on obtient un nouveau Sudoku qu'on cherche à nouveau à résoudre. Cette constatation nous dirige vers une fonction récursive qu'on appellera `resolution`. On peut résumer cette fonction en trois étapes :

1. on choisit une case vide
2. on place dans cette case un chiffre qui vérifie les règles d'un Sudoku
3. si toutes les cases sont pleines, c'est fini, sinon on rappelle à nouveau la fonction `resolution` sur ce nouveau Sudoku qui contient une case vide de moins.

Ce découpage suggère trois tâches bien définies. Choisir une case vide peut-être fait aléatoirement dans un premier temps. Nous verrons plus tard comment faire mieux. Pour l'instant, nous allons nous contenter d'un programme simple, qui marche même s'il est lent. La seconde étape maintenant : placer un chiffre dans une case vide, à choisir entre 1 et 9 ou mieux encore dans l'ensemble des chiffres non déjà présents sur la même ligne ou dans le petit carré. Nous allons donc écrire une fonction qui retourne l'ensemble des chiffres possibles pour une case vide. Cette fonction se présente sous cette forme :

```
Function nombre_possible_pour_case(ByRef su As Variant, _
                                                ByVal i As Long, ByVal j As Long) As Variant
    ' ....
End Function
```

Cette fonction prend comme paramètres :

1. `su` : un tableau à deux dimensions, `su(k,l)` désigne la case ligne `k`, colonne `l`
2. `i`, `j` : deux coordonnées désignant la case pour laquelle on veut déterminer l'ensemble des chiffres possibles

Le type `Variant` désigne un type inconnu, en particulier un tableau à deux dimensions. Il existe un nom précis pour tous les types de variables simples `Long`, `String`, `Double`. Les autres comme les tableaux sont souvent désignés par le type `Variant`.

**1)** La première chose à faire est de vérifier que la case `i, j` est vide. La fonction `nombre_possible_pour_case` doit donc commencer par ces lignes :

```
Function nombre_possible_pour_case(ByRef su As Variant, _
                                                ByVal i As Long, ByVal j As Long) As Variant
    Dim res() As Long     ' on crée le résultat sans connaître son contenu
    If .............
```

```
        ReDim res(0)                  ' on crée un tableau vide
        nombre_possible_pour_case = res ' c'est le résultat de la fonction
        Exit Function                   ' on quitte la fonction
    End If
End Function
```

A vous de trouver la condition à écrire sur la troisième ligne. Si cette case n'est pas vide, il n'y a pas de chiffres possibles pour cette case et le résultat de la fonction est un tableau vide. C'est ce que font les trois lignes suivantes.

**2)** Si la case est vide, on doit déterminer l'ensemble des chiffres possibles pour cette case. Ou plutôt, ne serait-il pas plus facile de recenser l'ensemble des chiffres qui ne sont pas possibles, c'est-à-dire l'ensemble des chiffres déjà placés soit sur la même ligne, soit dans le même petit carré ? Il suffit pour cela de parcourir la ligne et le petit carré qui contiennent la case `i, j` et de noter tous les chiffres qu'on y rencontre. On crée donc un tableau de neuf cases qui contient `0` si le chiffre est possible et `1` si le chiffre n'est pas possible.

```
    Dim paspossible(9) As Long
    Dim k As Long
    For k = 1 To 9
        paspossible(k) = 0  ' au départ, tous les chiffres sont possibles
    Next k
```

Il faut maintenant éliminer tous les chiffres déjà présents sur la ligne `i`. On rappelle que les cases `su(k, l)` sont soit égales à `0` si elles sont vides, soit leur valeur est comprise entre `1` et `9`. C'est à vous d'écrire et de modifier le tableau `paspossible`. Quelques indices : il y a une boucle et un test.

**3)** On doit maintenant éliminer les chiffres présents dans le petit carré. On calcule pour cela deux entiers `ii`, `jj` grâce aux trois lignes qui suivent :

```
    Dim ii, jj As Long
    ii = i - ((i - 1) Mod 3)
    jj = j - ((j - 1) Mod 3)
```

Que vaux `ii` lorsque `i` varie entre `1` et `9` ? Et `jj` ? Ne pourrait-on pas en déduire quelques lignes permettant d'éliminer les chiffres déjà présents dans le petit carré qui contient la case `i, j` ?

**4)** Maintenant qu'on a éliminé tous les chiffres pas possibles, on peut construire l'ensemble des chiffres possibles. On va d'abord les compter :

```
    Dim n As Long
    n = 0
    For k = 1 To 9
        If ............... Then n = n + 1
    Next k
```

Le résultat attendu est stocké dans la variable `n`. Il suffit juste d'écrire la bonne condition.

**5)** Il reste plus qu'à construire le résultat `res` de la fonction en complétant le bout de programme suivant :

```
    ReDim res(n)
    n = 0
    For k = 1 To 9
        If ................  Then
            ............
            ............
        End If
    Next k

    ' fin
    nombre_possible_pour_case = res
```

Petite remarque : si on n'avait pas d'abord compté le nombre de chiffres pas possibles, on n'aurait pas pu créer le bon nombre de cases dans le tableau `res` (première ligne de l'extrait qui précède ce paragraphe).

**6)** La fonction `nombre_possible_pour_case` retourne donc l'ensemble des chiffres possibles pour une case vide d'un Sudoku. On va d'abord considérer que la case vide qu'on choisit de remplir est la première case vide trouvée. Il ne reste plus qu'à écrire la fonction récursive `resolution` : c'est la fonction qui résoudra le Sudoku. Elle prend comme entrée un tableau Sudoku et retourne un tableau Sudoku pour la solution. On choisit comme convention que ce tableau sera vide si la solution n'existe pas.

```
Function resolution(ByRef su As Variant) As Variant
    ' .....
End Function
```

Il y a trois étapes :

1. On trouve la première case vide.
2. On calcule l'ensemble des chiffres possibles.
3. Pour tous les chiffres possibles, on rappelle la fonction `resolution` avec une case vide en moins.

Deux questions : quand peut-on savoir que le Sudoku est résolu ? Est-il possible d'avoir un ensemble de chiffres possibles vides ? A quoi cela correspond-il et que fait-on dans ce cas-là ?

**7)** Ecrire le code qui permet de trouver la première case vide et écrivez ce que la fonction doit faire si cette case n'existe pas. N'hésitez pas à lire les deux questions suivantes.

**8)** Appelez la fonction `nombre_possible_pour_case` pour connaître l'ensemble des chiffres possibles. Que fait-on si l'ensemble est vide ? La fonction `UBound` permet de savoir si un tableau est vide ou non, regardez l'aide pour comprendre comment s'en servir.

**9)** La fonction `resolution` doit maintenant ressembler à quelque chose comme :

```
Function resolution(ByRef su As Variant) As Variant
    ' étape 1
    Dim i,j,vi,vj As Long
    vi = -1
    For i = 1 To 9
        For j = 1 To 9
            If su (i,j) = 0 Then
               ...................
            End If
        Next j
     Next i

     If vi = ..... Then
         resolution = ..........
         Exit Function
     End If

     ' étape 2
     Dim ens As Variant
     ens = nombre_possible_pour_case (.......)
     If UBound (ens) ...... Then
         Dim res(0) As Long
         resolution = res
         Exit Function
     End If

     ' étape 3
     Dim k As Long
     Dim copie,solution As Variant
     copie = su
     For k = 1 To .....
         copie ( ...... ) = ens (k)
```

```
        solution = resolution (copie)
        If ..... Then
            .....
            Exit Function
        End If
    Next k

    ' au fait, a-t-on réussi ou non si on arrive ici,
    ' lors de l'exécution du programme ?
    resolution = .....
End Function
```

Surtout, enregistrer votre programme avant chaque exécution : si votre programme est entré dans une boucle infini, il faudra fermer Excel de manière autoritaire (en utilisant le gestionnaire des tâches). C'est pourquoi, on propose d'ajouter une variable qui va compter le nombre d'itérations ou d'appels à la fonction `resolution`.

```
Dim nbiter As Long
Function resolution(ByRef su As Variant) As Variant
    nbiter = nbiter + 1
End Function
```

**10)** Est-il possible de choisir une meilleure case que la première vide ? Et si parmi toutes les cases vides, il y en avait une pour laquelle l'ensemble des chiffres possibles est vide ou ne contient qu'un seul élément ? Comment modifier la fonction `resolution` pour tenir compte de cette information et accélérer la résolution ?

**11)** Il ne reste plus qu'à extraire le Sudoky depuis une feuille de calcul pour l'envoyer à la fonction `resolution`, ce qui n'est pas forcément une mince affaire. Comme d'habitude, on écrit une macro qu'on pourra relier plus tard à un bouton comme lors des séances précédentes. On vérifie d'abord que l'utilisateur a bien sélectionné 81*2 cases. En VBA, l'ensemble des cases sélectionnées est `Selection`.

```
Sub macro_sudoku()
    Dim i As Long
    i = 0
    For Each ch In Selection
        i = i + 1  ' pointeur d'arrêt ici
    Next ch
    If i <> 81 * 2 Then
        MsgBox "Vous n'avez pas sélectionné 81 * 2 cases, on sélectionne la plage B2:J10 + N2:V10"
        Range("B2:J10,N2:V10").Select
        Range("N2").Activate
    End If
End Sub
```

Après avoir recopié ce petit bout de programme, placez un pointeur d'arrêt sur la cinquième ligne et regardez le contenu de `ch`. Déduisez-en le moyen d'accéder au contenu d'une case et la manière dont VBA parcourt l'ensemble des cases sélectionnées (en ligne ou en colonne ?).

Lorsqu'on essaye le bouton alors que plage de cellules a été sélectionnée, il arrive que cette sélection disparaisse. Dans ce cas, il faut modifier la propriété `TakeFocusOnClick` et la définir sur `False` (voir figure 19.9).

**12)** Il faut maintenant récupérer le contenu des 81 premières cases de la sélection pour construire le tableau du Sudoku. Il y a plusieurs manières de le faire, voici l'esquisse d'une.

```
    Dim sudoku(9, 9) as Long
    i = 1
    j = 1
    For Each ch In Selection
        sudoku(i, j) = ........
```

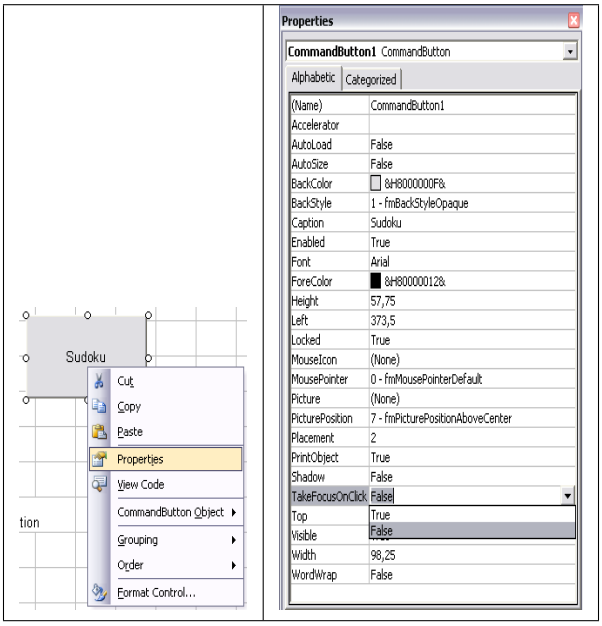



**Figure 19.9 :** *Modifier les propriétés d'un bouton, ici la propriété* `TakeFocusOnClick` *qui permet d'éviter que le bouton ne modifie la sélection lorsque celui-ci est actionné.*

```
        If j = 9 Then
            ......
        Else
            j = j + 1
        End If
    Next ch
```

**13)** On lance la résolutation.

```
    Dim r As Variant
    nbiter = 0
    r = resolution(sudoku)
```

**14)** Il n'y a plus qu'à remplir les 81 cases suivantes de la sélection avec la solution.

**15)** Deux choix s'offrent à vous dorénavant. Le premier consiste à modifier le programme pour obtenir tous les Sudoku qui vérifient la configuration de départ. Le second choix dépend de votre amour inconditionnel des Sudoku, un amour qui vous enjoint de saboter délibérément votre programme pour que jamais il n'ose encore une fois vous répondre en dix fois moins de temps qu'il ne vous en faudrait pour terminer votre Sudoku. Cela dit, si vous comptiez vous rabattre sur le démineur, il est sans doute heureux qu'il n'y ait pas de séance de VBA supplémentaire.

### 19.3.2 Correction

```
Option Explicit

Dim nbiter As Long

'
'  retourne le nombre de cases non vides
' on compte toutes celles qui ne contiennent pas 0
```

```
'
Function sudoku_cases_non_vide(ByRef su As Variant) As Long
    Dim n As Long
    Dim i As Long
    Dim j As Long

    n = 0
    For i = 1 To 9
        For j = 1 To 9
            If su(i, j) > 0 Then n = n + 1
        Next j
    Next i

    sudoku_cases_non_vide = n
End Function

'
' retourne l'ensemble des nombres possibles pour une case
' en tenant compte des contraintes
'
Function nombre_possible_pour_case(ByRef su As Variant, _
                                                ByVal i As Long, ByVal j As Long) As Variant

    Dim res() As Long

    ' on regarde d'abord si la case est vide
    If su(i, j) > 0 Then
        ReDim res(0)
        nombre_possible_pour_case = res
        Exit Function
    End If

    ' on crée un tableau,
    ' si paspossible (i) : alors le chiffre i est déjà
    ' pris ailleurs dans la ligne, dans la colonne ou dans le petit carré
    ' qui contiennent la case i,j

    Dim paspossible(9) As Long
    Dim k As Long
    For k = 1 To 9
        paspossible(k) = 0  ' au départ, tous sont possibles
    Next k

    ' vérification des contraintes en ligne et en colonne
    For k = 1 To 9
        If su(i, k) > 0 Then
            paspossible(su(i, k)) = 1
        End If

        If su(k, j) > 0 Then
            paspossible(su(k, j)) = 1
        End If
    Next k

    ' vérification des contraintes dans le petit carré de la case i,j
    Dim ii, jj, iii, jjj As Long
    ii = i - ((i - 1) Mod 3)
    jj = j - ((j - 1) Mod 3)

    For iii = ii To ii + 2
        For jjj = jj To jj + 2
            If su(iii, jjj) > 0 Then
                paspossible(su(iii, jjj)) = 1
            End If
        Next jjj
```

```
    Next iii

    ' nombre de possibles = tous ceux qui ne sont pas dans pospossible
    ' on les compte d'abord
    Dim n As Long
    n = 0
    For k = 1 To 9
        If paspossible(k) = 0 Then n = n + 1
    Next k

    ' puis on les met dans res
    ReDim res(n)
    n = 0
    For k = 1 To 9
        If paspossible(k) = 0 Then
            n = n + 1
            res(n) = k
        End If
    Next k

    ' fini
    nombre_possible_pour_case = res

End Function

'
' retourne l'ensemble des nombres possibles pour une case
' en tenant compte des contraintes
'
Function get_best_solution(ByRef su As Variant) As Variant
    Dim i, j, mi, mj As Long
    Dim pos As Variant

    ' on regarde d'abord si toutes les cases sont encore viables
    For i = 1 To 9
        For j = 1 To 9
            If su(i, j) = 0 Then
                pos = nombre_possible_pour_case(su, i, j)
                If UBound(pos) = 0 Then
                    Dim r(0) As Long
                    get_best_solution = r
                    Exit Function
                End If
            End If
        Next j
    Next i

    ' on teste la case qui offre le moins de chiffres possibles vérifiant
    ' les contraintes
    Dim l As Long
    l = 0
    For i = 1 To 9
        For j = 1 To 9
            If su(i, j) = 0 Then
                pos = nombre_possible_pour_case(su, i, j)
                If UBound(pos) = 1 Then
                    Dim rrr(2) As Long
                    rrr(1) = i
                    rrr(2) = j
                    get_best_solution = rrr
                    Exit Function
                ElseIf l = 0 Or UBound(pos) < l Then
                    l = UBound(pos)
                    mi = i
                    mj = j
```

```
                End If
            End If
        Next j
    Next i

    If l > 0 Then
        ' s'il y a une solution
        Dim rr(2) As Long
        rr(1) = mi
        rr(2) = mj
        get_best_solution = rr
    Else
        ' s'il n'y en a pas
        ' excusez le nom de la variable (rrrr),
        ' la portée d'une variable en VBA est la procédure
        ' même si sa déclaration est à l'intérieur d'un bloc
        Dim rrrr(0) As Long
        get_best_solution = rrrr
    End If
End Function

'
' résolution du sudoku, su est le sudoku à résoudre
'
Function resolution(ByRef su As Variant) As Variant
    ' premier cas, le sudoku est déjà résolu,
    ' auquel cas, c'est fini
    ' la variable nbiter compte le nombre d'itération pour la résolution
    ' il vaut mieux vérifier que ce nombre ne devient pas trop grand,
    ' sinon, il est possible que le programme entre dans une boucle infinie
    ' ce qui oblige l'utilisateur à relancer Excel après l'avoir détruit l'application
    ' dans le gestionnaire des tâches
    If sudoku_cases_non_vide(su) = 81 Or nbiter > 2000 Then
        resolution = su
        Exit Function
    End If

    nbiter = nbiter + 1

    Dim copie As Variant
    copie = su

    ' retourne la case la plus sympathique
    Dim b As Variant
    b = get_best_solution(copie)

    ' s'il existe une case impossible
    If UBound(b) = 0 Then
        Dim r(0) As Variant
        resolution = r
        Exit Function
    End If

    Dim i, j As Long
    i = b(1)
    j = b(2)

    Dim nb As Variant
    Dim sol As Variant
    nb = nombre_possible_pour_case(copie, i, j)

    ' sinon on teste toutes les solutions possibles pour une case
    Dim k As Long
    For k = 1 To UBound(nb)
        copie(i, j) = nb(k)
```

```
        sol = resolution(copie)
        If UBound(sol) > 0 Then
            resolution = sol
            Exit Function
        End If
    Next k

    ' pas de solution
    Dim re(0) As Long
    resolution = re
End Function

'
' macro appelée lorsque le bouton est enclenché
'
Sub macro_sudoku()
    Dim sudoku() As Variant
    Dim i, j As Long
    Dim nb As Long
    Dim ch

    ' vérification
    i = 0
    For Each ch In Selection
        i = i + 1
    Next ch
    If i <> 81 * 2 Then
        MsgBox "Vous n'avez pas sélectionné 81 * 2 cases, on sélectionne la plage B2:J10 + N2:V10"
        Range("B2:J10,N2:V10").Select
        Range("N2").Activate
    End If

    ' on remplit le sudoku avec les 81 premières cases
    ReDim sudoku(9, 9)
    i = 1
    j = 1
    For Each ch In Selection
        sudoku(i, j) = ch.Value
        If j = 9 Then
            j = 1
            i = i + 1
            If i = 10 Then Exit For
        Else
            j = j + 1
        End If
    Next ch

    ' on résoud le sudoku
    Dim r As Variant
    nbiter = 0
    r = resolution(sudoku)

    If UBound(r) > 0 Then
        ' s'il y a une solution, on remplit les cases
        i = 1
        j = 1
        For Each ch In Selection
            If i >= 10 Then
                ch.Value = r(i - 9, j)
            End If
            If j = 9 Then
                j = 1
                i = i + 1
            Else
                j = j + 1
```

```
            End If
        Next ch
    Else
        ' s'il n'y a pas de solution, on remplit les cases de zéros
        i = 1
        j = 1
        For Each ch In Selection
            If i >= 10 Then
                ch.Value = 0
            End If
            If i = 9 Then
                i = 1
                j = j + 1
            Else
                i = i + 1
            End If
        Next ch
    End If
End Sub
```

**fin correction TD 19.3.1** □

# Chapitre 20

# VBA et *Python*

---

Il est souvent possible de trouver des extensions d'*Excel* pour effectuer un traitement compliqué comme déterminer les valeurs propres et les vecteurs propres d'une matrice. Par exemple, une recherche sur *Google* avec les mots-clés *excel + valeur propre*[1] ne retourne pas de résultats probants. La plupart des extensions *Excel* ne sont pas gratuites comme le logiciel *XLSTAT*[2] qui propose des outils d'analyse des données sous *Excel*.

Une recherche équivalente avec les mots-clés *python + valeur propre* ou *python + eigenvalue* retourne plusieurs réponses qui font apparaître entre autre le module `scipy` qui un module gratuit utilisable avec *Python*.

Il n'est pas non plus toujours évident de construire des graphiques complexes sous *Excel* et des logiciels comme *R*[3] sont plus adaptés à cette tâche.

## 20.1 *Excel* et *Python*

### 20.1.1 Version simple

L'objectif est ici de connecter un script *Python* à un bouton de la barre de boutons de *Excel*, ce script devra calculer le quantile $\alpha$ d'une série de valeurs prises depuis une feuille *Excel* comme illustré sur la figure 20.1. La fonction calculant le quantile ne pose pas de problème.

```
def quantile (valeurs, alpha) :
    """calcul le quantile d'ordre alpha"""
    if len (valeurs) == 0 : raise Exception ("liste vide")
    valeurs.sort ()
    i = int (alpha * len (valeurs))
    return valeurs [i]
```

La communication entre *Python* et *Excel* s'effectue grâce au module `win32com.client`. Ce module n'a pas besoin de savoir si *Excel* est déjà ouvert ou non. Il peut tout à fait accéder à une feuille de calcul en train d'être modifié ou ouvrir un fichier. Dans notre cas, pour utiliser *Python* avec *Excel*, on va supposer que *Excel est déjà ouvert* est qu'on cherche à effectuer un calcul sur la feuille active. Le script suivant permet de calculer un quantile d'une liste de valeurs placées dans la plage `A2 : A25`, le quantile cherché est défini dans la cellule `C2` et le résultat sera enregistré dans la cellule `C3`.

```
numToAlpha = 'abcdefghijklmnopqrstuvwxyz'.upper ()
def celluple (s) :
    """convertit une cellule en couple ligne,colonne  C2 --> 2,3,  AA1 --> 1,27"""
    if len (s) <= 1 : return 1,1
    if "0" <= s [1] <= "9" : return int (s [1:]), numToAlpha.find (s [0])+1
    else : return int (s [1:]), (numToAlpha.find (s [0])) * 26 + numToAlpha.find (s [1]) + 1
```

1. ou eigen values en anglais
2. http://www.xlstat.com/fr/home/
3. L'URL suivant http://addictedtor.free.fr/graphiques/allgraph.php donne un aperçu de ce qu'il est possible de dessiner avec textitR. Les codes sources sont également accessibles et permettent de reproduire les exemples présentées.

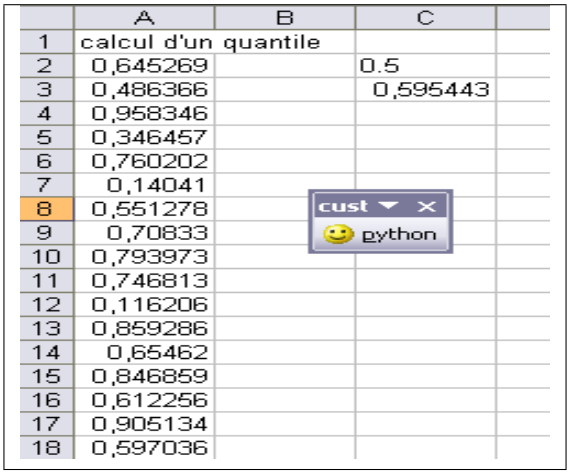



**Figure 20.1 :** *Construire une macro associée à un bouton qui calcule un quantile d'une série de valeurs.*

```
import win32com.client                                  # module
excel = win32com.client.Dispatch ('Excel.Application')  # connexion à Excel

tab = 'A2:A25'              # plage de valeurs
par = celluple ('C2')       # position du quantile demandé
res = celluple ('C3')       # position du résultat

feuille = excel.ActiveSheet                                 # on récupère la feuille active
plage   = [ v [0].Value for v in feuille.Range (tab) ]      # on convertit une plage en une liste
param   = float (feuille.Cells (par [0], par [1]).Value)    # on convertit une valeur en un réel

try :
    # on calcule le quantile
    import quantile
    q = quantile.quantile (plage, param)
    feuille.Cells (res [0], res [1]).Value = q          # on place le résultat dans la plage C3
except Exception, e :
    # au cas où une erreur se produit
    feuille.Cells (res [0], res [1]).Value = str (e)    # on place l'erreur dans la plage C3
```

Le programme précédent est enregistré sous le nom `excel.py` à côté de la feuille de calcul. Si le langage *Python* a été installé dans le répertoire `c : /python25`, le code VBA de la macro qui va lancer le script *Python* devient :

```
Public Sub UpdatePythonCode()
    Shell ("c:\python25\pythonw excel.py")
End Sub
```

**Remarque 20.1 : `python` ou `pythonw`**
L'utilisation du programme `pythonw.exe` au lieu de `python.exe` évite l'apparition d'une fenêtre de commande intempestive.

Le paragraphe 17.2 page 399 indique comment associer un bouton à cette macro.

### 20.1.2 Version améliorée

Cette configuration est assez rigide puisqu'il faut modifier le script *Python* à chaque fois qu'on souhaite modifier le nombre de valeurs à prendre en compte. Une solution consiste à modifier le script exécuté depuis *Excel*, on procède pour cela en trois étapes :

1. Lecture d'un programme *Python*, ce programme est identique au programme précédent à ceci près qu'on aura remplacé la plage et les cellules (lignes 11 à 13) par :

```
tab = __PLAGE__               # ligne 11
par = celluple (__PAR__)      # ligne 12
res = celluple (__RES__)      # ligne 13
```

2. On remplace dans ce script certains paramètres qui dépendent du contenu de la feuille.
3. On écrit ce nouveau script dans un fichier temporaire.
4. On exécute ce script *Python* grâce à la fonction `Shell`.

Les trois premières étapes ne sont pas indispensables, elles rendent simplement plus facilement adaptable le script *Python* qu'on souhaite exécuter ; de cette manière, dès que le nombre de valeurs évolue, on modifie la macro VBA et non le script *Python*. Le code VBA devient :

```
Public Function LoadString(ByVal file As String)
    ' ouvre un fichier et construit une unique chaîne de caractères à partir de son contenu
    Dim res As String
    Dim sLine As String
    Open file For Input As #1
    Do While Not EOF(1)
        Line Input #1, sLine
        res = res & sLine & vbCrLf  ' on ajoute les sauts de lignes qui sont oubliés lors de la lecture
    Loop
    Close #1
    LoadString = res
End Function

Public Function WriteString(ByVal s As String, ByVal file As String)
    ' on écrit une unique chaîne de caractères dans un fichier
    Open file For Output As #1
        Print #1, s
    Close #1
End Function

Public Sub UpdatePythonCode()
    ' code la macro
    Dim range1 As String
    Dim par As String
    Dim res As String
    python = LoadString("excel_sample.py")
    python = Replace(python, "__PLAGE__", "'A2:A25'")
    python = Replace(python, "__PAR__", "'C2'")
    python = Replace(python, "__RES__", "'C3'")
    WriteString python, "temp.py"
    Shell ("c:\python25\pythonw temp.py")
End Sub
```

Pour éviter de répéter les lignes de la fonction `UpdatePythonCode` pour chaque script *Python*, on peut construire une fonction qui lance un script *Python*en une seule ligne. Tout d'abord, il faut remplacer dans le script *Python* les plages d'entrées et de sorties par :

```
tab = __PAR0__                # ligne 11
par = celluple (__PAR1__)     # ligne 12
res = celluple (__RES0__)     # ligne 13
```

Le code VBA associé est le suivant :

```
Public Sub RunPythonScript(ByVal script As String, ByVal par As String, ByVal res As String)
    Dim p As Variant
```

```
    Dim r As Variant
    Dim python As String
    Dim i As Long
    Dim temp As String

    p = Split(par)  ' on découpe la liste des plages d'entrées séparées par des espaces
    r = Split(res)  ' on découpe la liste des plages de sorties séparées par des espaces

    python = LoadString(script)   ' on charge le script Python

    ' on remplace les paramètres d'entrée par les bonnes plages
    For i = 0 To UBound(p)
        temp = "__PAR" & Val(i) & "__"
        python = Replace(python, temp, "'" & p(i) & "'")
    Next i

    ' on remplace les paramètres de sortie par les bonnes plages
    For i = 0 To UBound(r)
        temp = "__RES" & Val(i) & "__"
        python = Replace(python, temp, "'" & r(i) & "'")
    Next i

    ' la fin est identique, on écrit le fichier temp.py puis on l'exécute
    WriteString python, "temp.py"
    Shell ("c:\python25\pythonw temp.py")
End Sub

Public Sub UpdatePythonCode()
    RunPythonScript "excel_sample2.py", "A2:A25 C2", "C3"
End Sub
```

## 20.2 *Python*, valeurs propres

### 20.2.1 Sans *Excel*

Le calcul des valeurs propres fait intervenir le module `scipy`[4] qui propose des fonctions autour de l'algèbre linéaire mais aussi concernant la transformée de Fourier, les ondelettes... L'installation est simple sous *Windows* puisqu'il suffit d'exécuter un programme bien qu'il faille d'abord installer le module `numpy`[5].

Le programme suivant construit une matrice `3x3` symétrique puis calcule les valeurs propres (`l`) et les vecteurs propres (`v`) qui sont retournées par la fonction `scipy.linalg.eig`. Il ne reste plus qu'à utiliser ce programme sous *Excel*.

```
import scipy.linalg
import numpy
import random
m = [ [ random.gauss (0,1) for i in range (0,3) ] for i in range (0,3) ]
mat = numpy.array (m)
mat = mat + mat.transpose ()
l,v = scipy.linalg.eig (mat)
print l
print v
```

### 20.2.2 Avec *Excel*

Le code *Python* a été scindé en deux. La première partie contient des fonctions qui permettent de récupérer des informations depuis *Excel* puis d'en insérer dans une feuille. Ce fichier s'appelle `excel_function.py`.

4. http://www.scipy.org/
5. http://numpy.scipy.org/

Le programme affiche une première ligne avec les valeurs propres qui peuvent être complexes si la matrice n'est pas diagonalisable dans $\mathbb{R}$ puis la matrice des vecteurs propres.

```
numToAlpha = 'abcdefghijklmnopqrstuvwxyz'.upper ()
def celluple (s) :
    """convertit une cellule en couple ligne,colonne"""
    if len (s) <= 1 : return 1,1
    if "0" <= s [1] <= "9" : return int (s [1:]), numToAlpha.find (s [0])+1
    else : return int (s [1:]), (numToAlpha.find (s [0])) * 26 + numToAlpha.find (s [1]) + 1

def nombre_colonne (plage) :
    """retourne le nombre de colonnes d'une plage"""
    p = plage.split (":")
    return celluple (p [1]) [1] - celluple (p [0]) [1] + 1

def lit_matrice (plage, feuille) :
    """transcrit une plage de la feuille Excel en matrice"""
    temp = [ v [0].Value for v in feuille.Range (plage) ]
    nc   = nombre_colonne (plage)
    nl   = len (temp) / nc
    res  = [ [ temp [i*nc+j] for j in range (0,nc) ] for i in range (0,nl) ]
    return res

def ecrit_matrice (valeur, vecteur, plage, feuille) :
    """transcrit une matrice vers une plage d'une feuille Excel"""
    # on calcule la premiere cellule de la plage
    if ":" in plage : i,j = celluple (plage.split (":") [0])
    else :            i,j = celluple (plage)
    # les valeurs propres sont complexes
    for l in range (0, len (valeur)) :
        if valeur [l].imag != 0 :  feuille.Cells (i, j + l).Value = str (valeur [l])
        else : feuille.Cells (i, j + l).Value = valeur [l].real
    # les vecteurs propres sont reels
    for k in range (0, len (vecteur)) :
        for l in range (0, len (vecteur [k])) :
            feuille.Cells (i + k + 2, j + l).Value = str (vecteur [k][l])
```

Ce premier fichier contient des fonctions qui reviennent fréquemment lorsqu'on fait communiquer *Python* et *Excel*. Le second fichier `excel_eig.py` contient le code qui lance le calcul des valeurs propres et c'est le seul qui doit être adapté à chaque nouveau traitement.

```
from excel_function import *
import win32com.client
excel = win32com.client.Dispatch ('Excel.Application')

mat = __PAR0__ # = 'A1:C3' ajout pour debugger
res = __RES0__ # = 'A5'    ajout pour debugger

feuille = excel.ActiveSheet

try :
    import scipy.linalg
    import numpy
    mat = numpy.array (lit_matrice (__PAR0__, feuille))
    valeur,vecteur = scipy.linalg.eig (mat)
    ecrit_matrice (valeur, vecteur, __RES0__, feuille)
except Exception, e :
    if ":" in res : i,j = celluple (res.split (":") [0])
    else :          i,j = celluple (res)
    feuille.Cells (i, j).Value = str (e)
```

Il ne reste plus qu'à écrire le code VBA en utilisant les fonctions présentées au paragraphe 20.1.2 :

```
Public Sub UpdatePythonCode()
    RunPythonScript "excel_eig.py", "A1:C3", "A5"
End Sub
```

**Remarque 20.2 : attendre l'exécution d'un script sous *Excel***
La fonction `Shell` a un inconvénient : elle n'attend pas la fin de l'exécution du script *Python*. Cela signifie que si l'exécution du script est longue, les cases mises à jour ne le seront que plus tard et que la pression du bouton qui lance le script pourra être perçue comme sans effet. Cela signifie aussi qu'il est possible de lancer l'exécution simultanée de deux scripts. Pour remédier à cela, il faut remplacer la fonction `Shell` par celle-ci `ShellWait` :

```
Option Explicit  ' rend explicite les déclarations de variables

Private Const PROCESS_QUERY_INFORMATION As Long = &H400
Private Const STILL_ACTIVE As Long = &H103
Public gszErrMsg As String

' utilisation d'une fonction OpenProcess remplaçant Shell, elle ne fait pas partie du langage VBA
' mais elle fait partie d'une DLL système de Windows
' la ligne qui suit déclare la fonction OpenProcess et l'ajoute à la liste de celles disponibles
Private Declare Function OpenProcess Lib "kernel32" (ByVal dwDesiredAccess As Long, _
                                                     ByVal bInheritHandle As Long, _
                                                     ByVal dwProcessId As Long) As Long

' permet d'obtenir un code d'erreur après l'exécution de OpenProcess
Private Declare Function GetExitCodeProcess Lib "kernel32" (ByVal hProcess As Long, _
                                                            lpExitCode As Long) As Long

' remplacement de la fonction Shell
' retourne True si l'exécution s'est bien passée, False sinon
' les paramètres d'entrée sont les mêmes que ceux de la fonction Shell
Private Function ShellWait(ByVal szCommandLine As String, _
                           Optional ByVal iWindowState As Integer = vbHide) As Boolean

    Dim lTaskID As Long
    Dim lProcess As Long
    Dim lExitCode As Long
    Dim lResult As Long

    On Error GoTo ErrorHandler  ' si une erreur se produit, aller au label ErrorHandler

    ' exécute la ligne de commande szCommandLine
    lTaskID = Shell(szCommandLine, iWindowState)

    ' vérifie qu'il n'y a pas d'erreur
    If lTaskID = 0 Then Err.Raise 9999, , "Shell function error."

    ' attrape le processus lancé par Shell et qui est en cours d'exécution
    lProcess = OpenProcess(PROCESS_QUERY_INFORMATION, 0&, lTaskID)

    ' regarde s'il y a des erreurs
    If lProcess = 0 Then Err.Raise 9999, , "Unable to open Shell process handle."

    ' attend la fin de l'exécution du script
    Do
        ' lExitCode est égale à STILL_ACTIVE tant que le script continue d'être exécuté
        lResult = GetExitCodeProcess(lProcess, lExitCode)
        DoEvents
    Loop While lExitCode = STILL_ACTIVE

    ShellWait = True
    Exit Function
```

```
ErrorHandler:
    ' gestion des erreurs (gszErrMsg conserve le message d'erreur)
    gszErrMsg = Err.Description
    ShellWait = False
End Function
```

## 20.3 Des graphes complexes avec *R* sous *Excel*

*R*[6] est un outil gratuit permettant de faire des statistiques poussées. C'est aussi comme *Python* un langage de programmation mais dont la spécialité est la manipulation et l'analyse de données. Toutefois, même si *R* est un outil plus puissant que *Excel*, tout le monde est familier avec *Excel*. En passant par *Python* et le module `PyR`[7], il devient possible d'étendre *Excel* notamment par des graphes plus évolués[8].

L'installation de *R* s'effectue grâce à un exécutable. L'installation du module `RPy` nécessite d'abord celle du module `numpy`[9] mais elle s'effectue aussi grâce à un exécutable. Il faut néanmoins s'assurer que les modules installés le sont pour la bonne version de *Python* et la bonne version de *R*.

### 20.3.1 Graphes avec *R* seul

On cherche à construire une représentation graphique d'un cours financier. Ce programme utilise le package `tseries` qu'il faut installer en suivant les instructions de la figure 20.2.

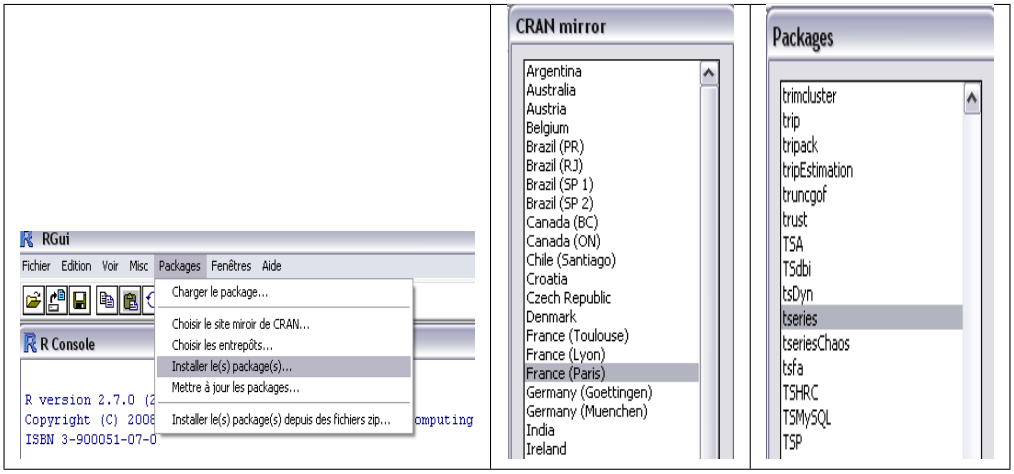



**Figure 20.2 :** *Pour installer le package* `tseries`*, il suffit de cliquer sur le menu Package / Installer le package. Si cela n'a pas été encore fait, R demande un entrepôt, les packages R sont copiés à plusieurs endroits du même type dans le monde, il suffit de choisir un site (image 2). La dernière image indique comment choisir le package souhaité.*

Le langage *R*[10] propose des fonctions pour récupérer des cours financier sur Internet[11] La première étape consiste à dire qu'on utilise le module `tseries` :

```
library (tseries)
```

6. http://www.r-project.org/
7. http://rpy.sourceforge.net/index.html
8. voir l'adresse http://addictedtor.free.fr/graphiques/allgraph.php pour des exemples de graphes
9. http://numpy.scipy.org/
10. Le site de *R* fournit quelques tutoriels comme celui-ci http://cran.r-project.org/doc/manuals/R-intro.pdf qui est écrit en anglais. Il existe une version française moins complète appelé *R pour les débutants* : http://www.lsp.ups-tlse.fr/Besse/pub/R/rdebuts_fr.pdf.
11. La fonction utilisée dans l'exemple qui suit utilise le site *Yahoo Finance*.

Cette instruction peut n'être utilisée qu'une seule fois même si l'appeler à chaque fois ne changera pas le comportement du programme qui suit. Ce dernier contient trois lignes, la prmeière récupère les données depuis Internet, la seconde ne conserve que les 100 dernières valeurs et la troisième affiche le graphique de la figure 20.3.

```
x=get.hist.quote("AXA", quote = c("Open", "High", "Low", "Close"), retclass = "ts")
x100=window(x,start=end(x)[1]-100)
plotOHLC(x100,ylab="prix",main="AXA")
```

Le même programme peut être adapté pour sauvegarder le graphique sous forme d'image et non plus l'afficher.

```
x=get.hist.quote("AXA", quote = c("Open", "High", "Low", "Close"), retclass = "ts")
x100=window(x,start=end(x)[1]-100)
png ("c:\\temp\\essai2.png", width=700, height=500)  # enregistrement au format png
plotOHLC(x100,ylab="prix",main="AXA")
dev.off ()                                           # fin de l'enregsitrement
```

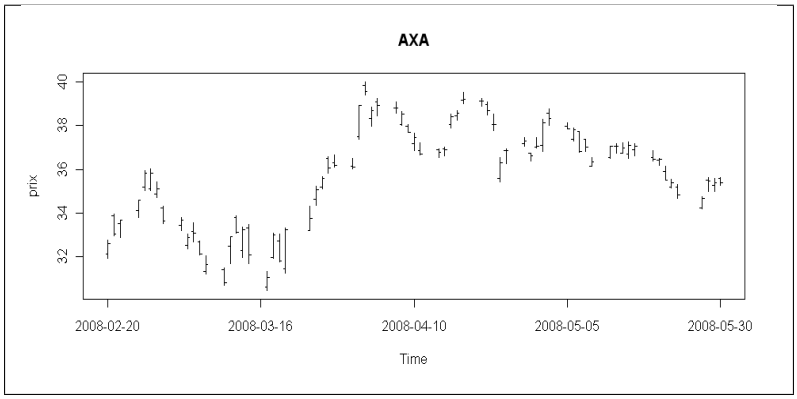



**Figure 20.3 :** *Les 100 dernières valeurs du cours AXA dans un graphique OHLC (Open High Low Close).*

Il est possible de charger les données depuis un fichier texte du type suivant :

```
      Date        Open        High         Low       Close       Volume
20/03/08       56.01       56.75       55.26       55.84      2434000
19/03/08       57.53       57.95       56.40       56.77      2714400
18/03/08       58.85       58.99       57.13       57.61      3265900
...
```

Il utiliser dans le cas la fonction `read.table` qui charge un fichier contenant le nom des colonnes en première ligne et le nom de ses lignes en première colonnes.

```
r <- read.table ("edf.txt", header = TRUE, row.names = 1)
```

Différentes fonctions permettent d'accéder aux données stockés dans l'objet chargé qui est de type `data.frame` :

```
r [[1]]            # accède à la colonne 1 en tant que vecteur
r [["Open"]]       # accède à la colonne Open en tant que vecteur
row.names (r)      # retourne le nom des lignes
length(r)          # retourne le nombre de colonnes
length (r [[1]])   # retourne le nombre de lignes
ts(r)              # convertit r en une série temporelles
```

### 20.3.2 *R* avec *Python*

Le module `rpy` permet de piloter le logiciel *R* depuis un programme *Python*. La première façon est d'utiliser l'interface de `rpy` qui permet d'appeler les fonctions de *R*. Cette façon n'est pas la plus intéressante, il est préférable tout d'abord de concevoir un script avec le programme *R* puis de l'exécuter via un programme *Python* avec la syntaxe suivante :

```
from rpy import *    # importer le module rpy

# définition du programme R
code = """library (tseries)
x=get.hist.quote("AXA", quote = c("Open", "High", "Low", "Close"), retclass = "ts")
x100=window(x,start=end(x)[1]-100)
png ("essai2.png", width=700, height=500)
plotOHLC(x100,ylab="prix",main="AXA")
dev.off ()"""

r(code)   # exécution du code
```

Cette façon de faire permet de maîtriser un logiciel *R* plutôt que son interface sous *Python*. A terme, ce choix facilitera la possible utilisation du langage *R* au sein d'un site internet. Bien que *Python* soit également un langage de script CGI, le langage *R* propose également ce genre d'extension avec le projet *Rweb* [12] ou encore la possibilité de configurer *Apache* [13] pour l'utilisation de scripts CGI écrits en *R*.

Par conséquent, l'utilisation dans ce document du module `pyr` se limitera aux conversions de données depuis *Python* vers *R* et réciproquement. L'essentiel des traitements sera écrits sous la forme de programme écrits en langage *R*. L'avantage de cette méthode est aussi de profiter des nombreux scripts *R* disponibles sur Internet [14]. L'inconvénient est la nécessité d'apprendre les premiers rudiments du langage [15].

Il existe un autre inconvénient à l'utilisation du module `pyr` lors de l'affichage des graphes. Le programme suivant identique au précédent a pour objectif d'afficher le cours d'une série financière dans un graphique qui disparaît aussitôt l'exécution du programme *Python* terminée.

```
from rpy import *
code = """library (tseries)
x=get.hist.quote("AXA", quote = c("Open", "High", "Low", "Close"), retclass = "ts")
x100=window(x,start=end(x)[1]-100)
plotOHLC(x100,ylab="prix",main="AXA")
r(code)
```

Si ce programme est exécuté avec le logiciel *Scite* [16], la fenêtre disparaît. S'il est exécuté avec *PyScritper* [17] ou avec l'éditeur de programmes fournit avec *Python*, la fenêtre graphique reste active jusqu'à l'exécution du même programme ou d'un autre.

Pour contourner ce problème, le plus simple est sans doute de sauvergarder le graphique sous forme d'image puis d'écrire une page au format HTML qui peut être visualiser depuis n'importe quel navigateur. C'est aussi ce passage temporaire par une image qui sera utilisé pour transmettre le graphique à *Excel*. L'exemple suivant montre comment créer cette page HTML.

12. http://www.math.montana.edu/Rweb/
13. Le site suivant http://biostat.mc.vanderbilt.edu/twiki/bin/view/Main/RApacheProject décrit commence configurer le serveur *Apache* dans ce sens.
14. Le site http://addictedtor.free.fr/graphiques/ fournit des exemples de code *R* permettant de reproduire une grande variété de graphiques.
15. Le site officiel http://www.r-project.org/ fournit de nombreux manuels, quelques recherches permettent d'en trouver des traductions en français.
16. voir paragraphe 2.4.1, page 17
17. voir paragraphe 2.4.2, page 19

```
# partie R
from rpy import *
code = """library (tseries)
x=get.hist.quote("AXA", quote = c("Open", "High", "Low", "Close"), retclass = "ts")
x100=window(x,start=end(x)[1]-100)
png ("essai2.png", width=700, height=500)
plotOHLC(x100,ylab="prix",main="AXA")
dev.off ()"""
r(code)

# partie HTML
html = """<html><body><h1>depuis R</h1><img src="NAME"/></body></html>"""
html = html.replace ("NAME", "essai2.png")
f = open ("essai2.html", "w")
f.write (html)
f.close ()
```

On retrouve les instructions `png` et `dev.off` qui indiquent au langage *R* le début et la fin de l'enregistrement dans une image les instructions graphiques incluses entre ces deux instructions. Les dernières lignes du programme servent à créer la page HTML qui permettra de visualiser le résultat dans un navigateur.

### 20.3.3 *R* avec *Python* et *Excel*

L'objectif de cette partie est d'insérer le code précédent dans une feuille *Excel*. Le programme *Python* doit récupérer les cours d'une action, tracer un graphe puis insérer le tout sur la feuille de calcul. Le premier obstacle est l'insertion d'une image dans une feuille de calcul [18] :

```
ActiveSheet.Pictures.Insert("essai2.png").Select
```

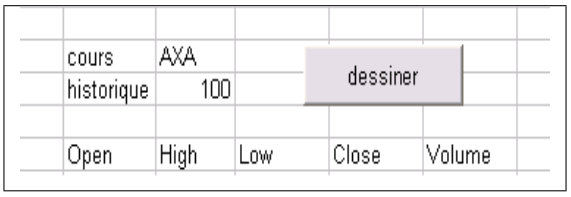



**Figure 20.4 :** *R + Excel*

### 20.3.4 *R* et *Excel*

Il est possible de se passer de *Python*, le langage *R* fournit les outils qui permettent de lire et d'écrire des feuilles de calcul au format *Excel*. Toutefois, *R* ne peut pas piloter l'application *Excel* alors que *Python* le peut. Cette fonctionnalité ne justifie pas l'emploi de *Python* comme intermédiaire entre *R* et *Excel*, cela dépend de l'objectif à atteindre et du temps nécessaire pour se sentir à l'aise en *R* et en *Python*.

Les lignes qui suivent expliquent comment faire cohabiter les trois langages. Leur assemblage utilise la fonction `ShellWait` décrite par la remarque 20.2 [19] qui permet d'attendre la fin de l'exécution d'un script. Le principe est simple :

1. Un script *Python* est généré, celui contient les instructions *R* à exécuter qui correspondent au téléchargement des $n$ dernières valeurs du cours d'une action et à leur représentation graphique.

18. Pour trouver en VBA la fonction désirée, il suffit le plus souvent d'enregistrer une macro, d'exécuter la tâche souhaitée grâce aux menus d'*Excel* puis de regarder le code créé.
19. page 448

2. La macro *VBA* exécute le script *Python* puis attend la fin du processus.
3. Le script *Python* écrit les valeurs de la série dans la feuille *Excel*.
4. La macro *VBA* récupère le graphe généré par les instructions *R*.

Le script *Python* utilisé par ce système est celui qui suit. Il contient les mots-clés `__PAR0__`, `__PAR1__` et `__RES0__` qui seront remplacés par des cellules dans le code *VBA*.

```
def finance (action, last, image = "temp.png") :
    # dessin de la série
    from rpy import *
    code = """library (tseries)
    x=get.hist.quote(""" + "\"" + action + "\"" + """,
               quote = c("Open", "High", "Low", "Close"), retclass = "ts")
    x100=window(x,start=end(x)[1]-""" + str (last) + """)
    png ("c:\\\\essai2.png", width=1000, height=500)
    plotOHLC(x100,ylab="prix",main=""" + "\"" + action + "\"" + """)
    dev.off ()"""
    r(code)

    # extraction avec volume pour export vers Excel
    code = """x=get.hist.quote(""" + "\"" + action + "\"" + """,
                    quote = c("Open", "High", "Low", "Close", "Volume"), retclass = "ts")
    x100=window(x,start=end(x)[1]-""" + str (last) + """)"""
    r (code)

    # conversion en python
    series = r ("x100")
    return series

def copie_tableau (series, i,j) :
    """recopie un tableau dans un feuille"""
    for xi in range (0, len (series)) :
        for xj in range (0, len (series [xi])) :
            feuille.Cells (i + xi, j + xj).Value = series [xi][xj]

import win32com.client
from celluple import *
excel = win32com.client.Dispatch ('Excel.Application')

act = celluple (__PAR0__)
par = celluple (__PAR1__)
res = celluple (__RES0__)

feuille = excel.ActiveSheet
action  = str (feuille.Cells (act [0], act [1]).Value)
param   = float (feuille.Cells (par [0], par [1]).Value)

try :
    series = finance (action, param)
    copie_tableau (series, res [0], res [1])
except Exception, e :
    feuille.Cells (res [0], res [1]).Value = str (e)
```

Il ne reste plus qu'à écrire le code *VBA* qui permet de créer le script *Python*, de l'exécuter puis de charger l'image du graphe généré par les instructions *R*.

```
' charge un fichier texte
Public Function LoadString(ByVal file As String)
    Dim res As String
    Dim sLine As String
    Open file For Input As #1
    Do While Not EOF(1)
        Line Input #1, sLine
```

```
        res = res & sLine & vbCrLf
    Loop
    Close #1
    LoadString = res
End Function

' écrit un fichier texte
Public Function WriteString(ByVal s As String, ByVal file As String)
    Open file For Output As #1
        Print #1, s
    Close #1
End Function

' exécute un script Python
Public Sub RunPythonScript(ByVal script As String, ByVal par As String, ByVal res As String)
    Dim p As Variant
    Dim r As Variant
    Dim python As String
    Dim i As Long
    Dim temp As String

    p = Split(par)
    r = Split(res)

    python = LoadString(script)

    For i = 0 To UBound(p)
        temp = "__PAR" & Val(i) & "__"
        python = Replace(python, temp, "'" & p(i) & "'")
    Next i
    For i = 0 To UBound(r)
        temp = "__RES" & Val(i) & "__"
        python = Replace(python, temp, "'" & r(i) & "'")
    Next i

    WriteString python, "temp.py"
    ShellWait ("c:\python25\pythonw temp.py")
End Sub

' lance l'exécution d'un script Python puis charge une image
Public Sub UpdatePythonCode()
    RunPythonScript "rplot2.py", "C3 C4", "B7"
    ActiveSheet.Pictures.Insert("essai2.png").Select
End Sub

' cette dernière fonction est celle attachée à la pression du bouton "dessiner"
Private Sub dessiner_Click()
    UpdatePythonCode
End Sub
```

# Cinquième partie

# EXEMPLES VARIÉS

La compétence d'un ingénieur ne repose pas sur ses capacités à programmer toutefois une certaine aisance à concevoir des algorithmes et à les mettre en œuvre offre une plus grande liberté. Cette facilité est d'autant plus grande qu'on a déjà par le passé résolu un problème similaire. Les exemples qui suivent sont inspirés de problèmes recontrés au cours de mon parcours professionnels. Ils consistent à apporter une solution algorithmique à un problème ou encore à automatiser une tâche répétitive.

Il est parfoirs utile de reproduire un problème rencontré au travail à partir d'un exemple simplifié, de garder une trace de la réflexion, un bout de programme informatique, de façon à ne pas perdre trop de temps lorsqu'un problème similaire apparaît de nouveau. Il m'est arrivé plusieurs fois de m'inspirer des solutions des programmes qui suivent pour résoudre un problème.

# Chapitre 21

# Périphériques

---

Jusqu'à présent, les exemples cités dans ce livre ont peu fait appel aux modules Python, qu'ils soient fournis avec le langage lui-même ou téléchargés sur Internet. C'est le cas des sujets qui suivent.

## 21.1 Copie de fichiers vers une clé USB

### 21.1.1 Enoncé

Lorsqu'on effectue une recherche de fichiers, on utilise souvent un filtre. Par exemple, le filtre $*.*$ désigne tous les fichiers possibles, $*$.doc désigne tous les fichiers créés par *Microsoft Word*. L'objectif de ce TD est de recopier un ensemble de fichiers d'un répertoire vers un autre en respectant l'arborescence d'origine. Ces fichiers seront inclus dans un répertoire racine ou un de ses sous-répertoires et leur nom devra vérifier deux filtres :

1. Le premier filtre sélectionne les fichiers à copier.
2. Le second permet d'exclure des fichiers sélectionnés par le premier filtre.

Ce petit programme pourra être utilisé par la suite pour recopier des fichiers vers une clé USB par exemple. Il regroupe deux tâches indépendantes, la copie de fichiers et l'adéquation entre un nom de fichier et un filtre. Le schéma proposé est de réaliser ces deux tâches à l'aide d'une classe `copie_usb`.

```
class copie_usb (object):
    """recopie des fichiers sur une clé USB"""

    def __init__(self,ch1,ch2,accept = ".*",refuse = "") :
        """initialisation,
        @param      ch1         répertoire source
        @param      ch2         répertoire destination
        @param      accept      filtre pour les fichiers acceptés pour la copie
        @param      refuse      pour refuser des fichiers parmi ceux déjà acceptés"""

    def accepter (self, fichier) :
        """dit si un fichier est accepté à la copie ou non,
        retourne un booléen"""

    def liste_fichier_repertoire (self,repertoire):
        """récupération de la liste des répertoires et celle des fichiers,
        inclus dans le répertoire repertoire"""

    def copie (self) :
        """effectue la copie"""

        # première étape
        # récupération de la liste des répertoires et fichiers

        # seconde étape
        # élimination des importuns

        # troisième étape
```

```
        # on créé les répertoires s'ils n'existent pas

        # quatrième étape
        # on recopie les fichiers


# petit exemple d'utilisation
c = copie_usb (ch1, ch2, filtre_accept, filtre_refuse)
c.copie ()
```

**1)** La première étape est d'écrire le code concernant le constructeur `__init__` de la classe `copie_usb`. Il s'agit d'initialiser les paramètres de la classe qui sont aussi ceux du constructeur.

**2)** Cette seconde question se concentre sur l'adéquation entre un nom de fichier et un filtre, c'est-à-dire l'implémentation de la méthode `liste_fichier_repertoire`. Pour ce faire, on utilise le module `re` qui traite les *expressions régulières*.

Une expression régulière est une sorte de schéma de construction pour les chaînes de caractères. Par exemple, le format .∗ désigne tous les chaînes de caractères. Le format ^[a].∗ désigne toutes les chaînes de caractères commençant par la lettre `a` minuscule. L'ensemble de ces règles est décrit à l'adresse *http ://docs.python.org/lib/re-syntax.html*. Le tableau qui suit regroupe les expressions nécessaires à la réalisation de ce TD.

| | |
|---|---|
| .∗[.].∗ | Cette expression régulière correspond à tous les noms de fichiers ayant une extension, c'est-à-dire au moins un point inclus dans le nom du fichier. |
| .∗[.]pdf$|.∗[.]html$ | Cette expression régulière correspond à tous les noms de fichiers ayant une extension `pdf` ou `html`. Le "ou" est traduit par le symbole |, il peut y en avoir plusieurs dans la même expression régulière. |
| programme\\.∗[.]zip$ | Cette expression régulière correspond à tous les noms de fichiers ayant une extension `zif` et inclus dans un répertoire `programme\` ou un de ses sous-répertoires. |

Le module `re` traite les expressions régulières. Parmi les fonctions proposées par ce module, deux seront utilisées.

| **Fonctions utiles :** | |
|---|---|
| re.compile(str, flag = 0) | Construit une expression régulière à partir d'une chaîne de caractères `str`. Si `flag` = re.IGNORECASE, l'expression régulière ne fait pas la distinction entre les minuscules et les majuscules. |
| re.match(ex, str) | Retourne une expression régulière si la chaîne de caractères `str` est en adéquation avec l'expression régulière `ex`, et `None` dans les cas contraire. |

Petit exemple de code :

```
import re
t = re.compile (".*[.]htm$", re.IGNORECASE)
if re.match (t, "inDex.htm") : print True
else : print False
if re.match (t, "inDex.html") : print True
else : print False
```

Le programme affiche `True` puis `False`. Le résultat de la fonction `match` est une expression régulière s'il y a adéquation ou un résultat vide s'il n'y a pas adéquation comme le montre l'exemple suivant.

```
print re.match (t, "inDex.htm")  # affiche <_sre.SRE_Match object at 0x008712C0>
print re.match (t, "inDex.html") # affiche None
```

**3)** La troisième partie du TD concerne la récupération des fichiers et des répertoires inclus dans un répertoire appelé *racine* qui se conclut par l'implémentation des fonctions `liste_fichier_repertoire` et `copie` de la classe `copie_usb`. Tout d'abord, le plus simple, la copie d'un fichier est réalisée au moyen de la fonction `copy` du module `shutil`.

| **Fonctions utiles :** | |
|---|---|
| `shutil.copy(source,destination)` | Copie le fichier `source`, ses nouveaux nom et emplacement sont donnés par `destination`. |

Ensuite, il existe une fonction `listdir` du module `os` qui permet d'obtenir la liste des fichiers et sous-répertoires inclus dans un répertoire. Ce même module propose une fonction `mkdir` qui permet de créer un répertoire.

| **Fonctions utiles :** | |
|---|---|
| `os.listdir(rep)` | Retourne la liste des fichiers et sous-répertoires inclus dans le répertoire `rep`. Les noms retournés sont relatifs au répertoire `rep`. Le nom complet d'un fichier `f` inclus dans la liste retournée est donc `rep` + "\\" + `f`. |
| `os.mkdir(rep)` | Crée le répertoire `rep`, si les répertoires intermédiaires n'existent pas, cette fonction génère une exception. |
| `os.makedirs(rep)` | Crée le répertoire `rep`, si les répertoires intermédiaires n'existent pas, cette fonction les crée également. |

Enfin, le module `os.path` propose entre autres trois fonctions qui permettent de savoir si un fichier ou un répertoire existe, et si un nom correspond à un fichier ou à un répertoire.

| **Fonctions utiles :** | |
|---|---|
| `os.path.exists(nom)` | Retourne `True` si le répertoire ou le fichier `nom` existe. |
| `os.path.isfile(nom)` | Retourne `True` si `nom` correspond à un nom de fichier. |
| `os.path.isdir(nom)` | Retourne `True` si `nom` correspond à un nom de répertoire. |

**Travail à faire :**

La liste des fichiers sélectionnés ne contient pas forcément tous les fichiers inclut dans un répertoire et ses sous-répertoires. Il est probable que la classe `copie_usb` crée alors des répertoires sur la clé USB qui ne contiennent aucun fichier. Comment faire pour éviter que la méthode `copie` ne crée ces répertoires vides ?

### 21.1.2 Correction

Le programme qui suit répond aux trois questions de l'énoncé de ce TD ainsi qu'à la question subsidiaire par l'intermédiaire de la méthode `repertoire_selectionne`. Cette dernière détermine si un fichier d'une liste `file_clean` est inclus dans un répertoire `r`. Ce programme recopie tous les fichiers inclus dans le répertoire *C :/Documents and Settings/Dupré/Mes documents/informatique/support/python_td* vers le répertoire *c :/temp/copie_usb*. Les fichiers dont les extensions sont *pdf*, *html* et *bmp* ne sont pas recopiés. Les fichiers dont l'extension est *zip* et inclus dans le répertoire *programme* ne sont pas non plus récopiés.

```
# -*- coding: cp1252 -*-
"""copie de fichiers sur une clé USB"""

import re       # pour les expressions régulières
import os       # pour les fichiers et répertoires
import os.path  # pour les noms de fichiers et noms de répertoires
import copy
import shutil   # pour la copie de fichiers

class copie_usb (object):
    """recopie des fichiers sur une clé USB"""

    def __init__(self,ch1,ch2,accept = ".*",refuse = "") :
        """initialisation,
        @param      ch1         répertoire source
        @param      ch2         répertoire destination
        @param      accept      filtre pour les fichiers acceptés pour la copie
        @param      refuse      pour refuser des fichiers parmi ceux déjà acceptés"""
        self.ch1     = ch1
        self.ch2     = ch2
        self.accept  = re.compile (accept, re.IGNORECASE) # création des motifs
        self.refuse  = re.compile (refuse, re.IGNORECASE) # création des motifs

    def accepter (self, fichier) :
        """dit si un fichier est accepté à la copie ou non"""
        r = re.match (self.accept, fichier)
        if not r : return False
        r = re.match (self.refuse, fichier)
        return not r

    def aide (self) :
        """retourne une aide sur les formats d'expression"""
        help (re.engine)

    def liste_fichier_repertoire (self,repertoire):
        """récupération de la liste des répertoires et celle des fichiers,
        inclus dans le répertoire repertoire"""
        rep     = []
        file    = []
        list    = os.listdir (repertoire)

        for cc in list :
            c = repertoire + "\\" + cc
            if os.path.isfile (c) : file.append (cc)
            if os.path.isdir (c) : rep.append (cc)

        rep2 = copy.copy (rep)
        for chemin in rep2 :
            r,f = self.liste_fichier_repertoire (repertoire + "\\" + chemin)
            for x in r :
                rep.append (chemin + "\\" + x)
            for x in f :
                file.append (chemin + "\\" + x)

        return rep,file

    def repertoire_selectionne (self, r, file_clean) :
        """dit si la liste file_clean contient au moins un fichier
        inclus dans le repertoire r"""
        t = re.compile ("^" + r + "\\\\.*", re.IGNORECASE)
        for l in file_clean :
            if re.match (t,l) : return True
        return False
```

```
    def copie (self) :
        """effectue la copie"""

        # récupération de la liste des répertoires et fichiers
        rep,file = self.liste_fichier_repertoire (self.ch1)
        # élimination des importuns
        file_clean = [ f for f in file if self.accepter (f) ]

        # facultatif, exclue les répertoires pour lesquels
        # aucun fichier n'est sélectionné
        rep_clean = [ r for r in rep if self.repertoire_selectionne (r, file_clean) ]

        # on créé les répertoires s'il n'existent pas
        if not os.path.exists (self.ch2) :
            print "création du répertoire ", self.ch2
            os.mkdir (self.ch2)

        for r in rep_clean :
            c = self.ch2 + "\\" + r
            if not os.path.exists (c) :
                print "création du répertoire ", r
                os.mkdir (c)

        # on recopie les fichiers
        for f in file_clean :
            s = self.ch1 + "\\" + f
            c = self.ch2 + "\\" + f
            print "copie du fichier ", f
            shutil.copy (s, c)


if __name__ == "__main__" :
    print "copie de fichiers vers une clé USB"
    ch1             = "C:\\Documents and Settings\\Dupré\\" \
                        "Mes documents\\informatique\\support\\python_td"
    ch2             = "c:\\temp\\copie_usb"
    filtre_accept   = ".*[.].*"
    filtre_refuse   = ".*[.]pdf$|.*[.]html$|.*[.]bmp|programme\\\\.*[.]zip$"

    # filtre_accept accepte tout type de fichier
    # filtre_refuse refuse tous les fichiers dont l'extension est pdf, html ou
    # inclus dans le répertoire programme et ayant l'extension zip

    c = copie_usb (ch1, ch2, filtre_accept, filtre_refuse)
    c.copie ()
```

**fin correction TD 21.1.1** □

## 21.2 Recherche sur Internet

### 21.2.1 Enoncé

Ce TD a pour but de rechercher une expression dans une page HTML et les pages que la première référence. Ce peut être par exemple pour rechercher tous les articles d'un journal relatifs à un même sujet.

**1)** La première étape consiste à décliner une version de la classe `HTMLParser` du module `HTMLParser` afin de conserver lors du déchiffrage d'une page HTML le contenu de cette page et les hyperliens qu'elle contient. Pour lire une page HTML à l'aide d'une classe de type `HTMLParser`. L'extrait suivant montre comment se servir d'une telle classe :

```
    p = HTMLParser ()
    p.feed (text)
    p.close ()
```

Il suffit ensuite de dériver cette classe `HTMLParser` pour surcharger les deux méthodes suivantes :

1. `handle_starttag` est appelée chaque fois qu'une nouvelle balise est lue dans un fichier HTML, en particulier la balise < a href = ... < /a > qui permet d'insérer un lien hypertexte. Surcharger cette méthode permet de balayer tous les hyperliens d'une page. Elle reçoit deux paramètres, `tag` qui est le nom de la balise et `attr` qui contient une liste de couples (attribut,valeur). La balise recherchée est `a` qui contient un attribut `href` comme celui de l'exemple qui suit :

   ```
   <a href="bresenham_ligne4.html"> tracé d'une ligne</a>
   ```

   Les quelques lignes qui suivent sont les premières notes de la méthodes `handle_starttag`.

   ```
       def handle_starttag (self, tag,attr) :
           print "balise ", tag
           for i,j in attr :
               print i,j
               if i == "href" :
                  # ...
   ```

2. `handle_data` est appelée chaque fois qu'un texte est inclus entre deux balises, il s'agit du texte dans lequel les mots-clés, les expressions, seront cherchés. Le programme concatène chaque chaîne de caractères que reçoit cette fonction. La méthode `handle_data` ne reçoit qu'un paramètre `data`, une chaîne de caractères dans laquelle il faudra chercher une expression régulière. Quelques notes encore...

   ```
       def handle_data (self, data) :
           d = data.replace ("\n", "")
   ```

Encore quelques notes pour la déclaration de la classe et son constructeur. `list` est une liste d'hyperliens à explorer. Lors de la lecture d'une page HTML, cette liste sera étendue des hyperliens rencontrés. De même, `text` contiendra après la lecture tout le contenu d'une page débarrassée de ses balises.

```
class HTML_explore (HTMLParser.HTMLParser) :

    def __init__ (self, list, url, text) :
        html_parser_script.HTMLParserScript.__init__(self)
        self.list           = list
        self.url            = url
        self.text           = text
```

Une dernière précision, le module `urlparse` permet de concaténer un lien absolu - par exemple *http ://www.lemonde.fr/* - et un hyperlien relatif à cette adresse :

```
adr = urlparse.urljoin ("http://www.lemonde.fr/", "web/article/")
```

**2)** Il faut encore ouvrir une page HTML, charger son contenu, le déchiffer, en extraire des hyperliens et rechercher une expression régulière à l'aide du module `re`. Tout d'abord, il faut charger le contenu d'une page comme le montre l'exemple suivant :

```
try :
    f       = urllib.urlopen (url)
    d       = f.read ()
    whole   = d
except Exception, exc:
    print "    exception lecture ", exc, "\t -- ", url
    error.append (url)
    continue
```

Une fois que la page HTML est chargée, il faut encore la déchiffrer à l'aide de la classe `HTML_explore` écrite lors de la première question. Cette classe hérite de la classe `HTMLParser` qui inclut une méthode `feed`. Pour déchiffer la chaîne HTML `whole`, il faut utiliser le code suivant :

```
p       = HTML_explore (adr, url, text)
p.feed (whole)
p.close ()
```

La récupération des hyperliens et du contenu d'une page sans ses balises est faite par la classe `HTML_explore`. Il reste encore à chercher une expression régulière à l'aide de l'exemple suivant (voir également le TD 21.1.1, page 457) :

```
s               = ".*Bush.*"
ex              = re.compile (s, re.IGNORECASE)
if ex.match (contenu) :
    # ...
```

Pour finir, la partie principale du programme doit ressembler aux lignes qui suivent.

```
url         = "http://www.lemonde.fr/"
# on appelle la fonction process_file pour récupérer les informations
# inscrites dans le fichier XML
s               = ".*ukraine.*"
ex              = re.compile (s, re.IGNORECASE)
li,error,nb     = process_file (url, ex, 3, -10)
```

**Remarque 21.1 : version du langage HTML**
La classe `HTML_explore` peut dériver indifféremment de la classe `HTMLParser` du module `HTMLParser` ou de la classe `HTMLParserScript` du module portant le même nom. Dans le premier cas, ce programme ne pourra lire que la version 2.01 du langage HTML. Dans le second cas, le programme sera capable de lire des versions plus récentes sans toutefois déchiffrer complètement la version 4.01, la plus répandue.

### 21.2.2 Correction

La classe `HTML_explore` peut dériver indifféremment de la classe `HTMLParser` du module `HTMLParser` ou de la classe `HTMLParserScript` du module portant le même nom. Dans le premier cas, ce programme ne pourra lire que la version 2.01 du langage HTML. Dans le second cas, le programme sera capable de lire des versions plus récentes sans toutefois déchiffrer complètement la version 4.01, la plus répandue.

La classe `HTML_explore` dérive donc d'une des deux classes précédemment citées dont elle surcharge deux méthodes :

1. `handle_starttag` est appelée chaque qu'une nouvelle balise est lue dans un fichier HTML, en particulier la balise < a href = ... < /a > qui permet d'insérer un lien hypertexte. Surcharger cette méthode permet de balayer tous les hyperliens d'une page.
2. `handle_data` est appelée chaque fois qu'un texte est inclus entre deux balises, il s'agit du texte dans lequel les mots-clés, les expressions, seront cherchés. Le programme concatène chaque chaîne de caractères que reçoit cette fonction.

La fonction `process_file` ouvre une page HTML, la place intégralement dans une chaîne de caractères qui est envoyée à la classe `HTML_explore`. Après avoir déchiffré cette page, la classe `HTML_explore` retourne une chaîne de caractères ne contenant que le texte sans mise en forme ainsi que la liste des hyperliens. `process_file` recherche une expression dans le contenu d'une page puis passe aux pages suivantes tant qu'il y en a. Le programme ne conserve pas les hyperliens si ceux-ci sont trop éloignés de la première page. La fonction retourne les pages sélectionnées et les pages pour lesquelles une erreur de lecture s'est produite.

```
# -*- coding: cp1252 -*-
"""recherche de mot-clés dans des pages HTML,
ce programme ne peut lire pour l'instant que le format HTML 2.0
et pas HTML 4.01, format utilisé par la plupart des sites."""

import urllib               # accéder une adresse internet
import urlparse             # manipuler les adresses internet
import re                   # expression à chercher
import html_parser_script   # classe de parser HTML modifiée pour éviter les scripts

class HTML_explore (html_parser_script.HTMLParserScript) :
    """parcour un fichier HTML"""

    def __init__ (self, list, text, url, niveau, niveau_max) :
        html_parser_script.HTMLParserScript.__init__(self)
        self.list           = list
        self.niveau         = niveau
        self.niveau_max     = niveau_max
        self.url            = url
        self.text           = text

    def cherche (self,url) :
        """recherche un url dans la liste"""
        for n,u in self.list :
            if u == url : return True
        return False

    def diese (self,adr) :
        """simplifie les liens avec le symboles #"""
        p = adr.find ("#")
        if p == -1 : return adr
        else : return adr [0 : p]

    def handle_starttag (self, tag,attr) :
        """nouveau tag ou balise"""
        if self.niveau >= self.niveau_max : return
        self.tag = tag
        if self.tag == "a" or self.tag == "area" :
            for i,j in attr :
                if (i == "aref" or i == "href") and len (j) > 0 and j [0] != "#" :
                    adr = self.diese (j)
                    if len (adr) > 4 and (adr [0:4] == "http" or adr == "ftp") :
                        if not self.cherche (adr) :
                            self.list.append ((self.niveau, adr))
                    elif len (adr) > 5 and adr [0:6] != "mailto" :
                        adr = urlparse.urljoin (self.url, adr)
                        if not self.cherche (adr) :
```

```
                        self.list.append ((self.niveau, adr))

    def handle_endtag (self, tag) :
        """fin d'un tag ou balise"""
        pass

    def handle_data (self, data) :
        """texte compris entre le début et la fin d'un tag"""
        d = data.replace ("\n", "")
        if len (d) > 3 :
            self.text.append (data)


def process_file (url, exp, niveau_max = 5, nbmax = -1) :
    """parcours un fichier HTML et retourne la liste des url
    qui contiennent l'expression, la liste des
    adresses qui n'ont pas pu être atteinte ainsi que le nombre d'adresses
    explorées, cette fonction n'explore pas plus de
    nbmax pages sauf si nbmax == -1"""

    res     = []
    error   = []

    # création d'une classe qui va parcourir le fichier XML
    adr     = [(0, url)]
    text    = []
    errfile = 0
    nb      = 0

    for niv,url in adr :

        nb     += 1
        if nb > nbmax and nbmax >= 0 : break
        whole   = ""
        print "open %d/%d" % (nb,len (adr)),
        print " found = ", len (res), " -- ", niv, " : ", url

        try :
            f       = urllib.urlopen (url)
            d       = f.read ()
            whole  += d
        except Exception, exc:
            print "    exception lecture ", exc, "\t -- ", url
            error.append (url)
            continue

        try :
            text    = []
            p       = HTML_explore (adr, text, url, niv+1, niveau_max)
            # on lit le fichier file ligne par ligne,
            # chaque ligne est envoyée au parser XML
            p.feed (whole)
            p.close ()

            t = ""
            for s in text : t += s + " "
            t = t.replace ("\n", " ")
            t = t.replace ("\r", " ")
            if exp.match (t) : res.append (url)

        except Exception, exc:
            print "    exception html ", exc, exc.__class__, "\t -- ", url
            error.append (url)
            f = open ("c:\\temp\\html_err" + str (errfile) + ".html", "w")
            f.write (whole)
```

```
            f.close ()
            errfile += 1
            continue

    return res, error, len (adr)



if __name__ == "__main__" :
    # choix d'un nom de fichier
    url         = "http://www.lemonde.fr/"
    # on appelle la fonction process_file pour récupérer les informations
    # inscrites dans le fichier XML
    s               = ".*Bush.*"
    ex              = re.compile (s, re.IGNORECASE)
    li,error,nb     = process_file (url, ex, 3, -10)

    print "-------------------------------------------------------------------"
    print "-------------------------------------------------------------------"
    print "nombre d'adresses explorées :\t", nb
    print "nombre d'adresses sélectionnées :\t", len (li)
    print "nombre d'adresses non ouvertes ou mal lues :\t", len (error)
    print "-------------------------------------------------------------------"
    print "-------------------------------------------------------------------"
    if len (li) > 0 :
        print "url contenant l'expression ", s
        for l in li :
            print " --------- " , l

    if len (error) > 0 :
        print "url n'ayant pas pu être ouverts ou mal lus "
        for l in error :
            print " --- erreur ", l
```

Ce fichier inclus la classe `HTMLParserScript`. Elle adapte la classe `HTMLParser` du module afin de lire un format HTML plus évolué que la version 2.01. En revanche, elle ne déchiffre pas tout script inclus dans une page HTML.

```
# -*- coding: cp1252 -*-
"""recherche de mot-clés dans des pages HTML,
ce programme ne peut lire pour l'instant que le format HTML 2.0
et pas HTML 4.01, format utilisé par la plupart des sites."""

import HTMLParser       # parcourir un fichier HTML
import re
import markupbase

starttagopen = re.compile('<[a-zA-Z]')

attrfind = re.compile(
    r'\s*([a-zA-Z_][-.:a-zA-Z_0-9]*)(\s*=\s*'
    r'(\"[a-z]+\"#[a-zA-Z. ă]+[.][a-zA-Z]+|\"[0-9]+\"\"|\'[^\']*\'|"[^"]*"'\
     '|([a-zA-Z_:.]+\s*\([0-9/a-zA-Z\"\',. ă&;éèàùûâêîô\-_\?:!=@]*\))'\
     '(;void\(0\);)?|\"\"|[-a-zA-Z0-9./,:;+*%?!&$\(\)_#=~]*))?')

# si locatestarttagend est modifié, il faut penser à modifier
# également attrfind
locatestarttagend = re.compile(r"""
  <[a-zA-Z][-.a-zA-Z0-9:_]*          # tag name
  (?:\s+                             # whitespace before attribute name
    (?:[a-zA-Z_][-.:a-zA-Z0-9_]*     # attribute name
      (?:\s*=\s*                     # value indicator
        (?:([a-zA-Z_:.]+\s*\([0-9/a-zA-Z\"',. ă&;éèàùûâêîô\-_\?:!=@]*\))(;void\(0\);)? #
```

```
          |\"[0-9]+\"\"              # colspan = "2""
          |\"[a-z]+\"\#[a-zA-Z. ă]+[.][a-zA-Z]+   # "rect"#Le Monde.fr
          |\"\"                     # alt = ""
          |\"[^\"]*\"               # LIT-enclosed value
          |'[^']*'                  # # LITA-enclosed value
          |[^'\">\s]+               # bare value
         )
       )?
     )
   )*
  \s*                               # trailing whitespace
""", re.VERBOSE)

entityref = re.compile('&([a-zA-Z][-.a-zA-Z0-9]*)[^a-zA-Z0-9]')
charref = re.compile('&#@(?:[0-9]+|[xX][0-9a-fA-F]+)[^0-9a-fA-F]')
incomplete = re.compile('&[a-zA-Z#]')

commentclose = re.compile(r'--\s*>|/\s*>|-- /\s*>|-\s*>')


class HTMLParserScript (HTMLParser.HTMLParser) :

    def parse_comment(self, i, report=1):
        rawdata = self.rawdata
        #assert rawdata[i:i+4] == '<!--', 'unexpected call to parse_comment()'
        match = commentclose.search(rawdata, i+4)
        if not match:
            print "----------------------------------------------------------"
            print rawdata[i:i+250]
            print "----------------------------------------------------------"
            print end
            print "----------------------------------------------------------"
            return -1
        if report:
            j = match.start()
            self.handle_comment(rawdata[i+4: j])
        j = match.end()
        return j

    def _scan_name(self, i, declstartpos):
        rawdata = self.rawdata
        n = len(rawdata)
        if i == n:
            return None, -1
        m = markupbase._declname_match(rawdata, i)
        if m:
            s = m.group()
            name = s.strip()
            if (i + len(s)) == n:
                return None, -1  # end of buffer
            return name.lower(), m.end()
        else:
            self.updatepos(declstartpos, i)
            print "----------------------------------------------------------"
            print rawdata[i:i+250]
            print "----------------------------------------------------------"
            print end
            print "----------------------------------------------------------"
            self.error("expected name token")

    def parse_starttag(self, i):
        self.__starttag_text = None
        endpos = self.check_for_whole_start_tag(i)
        if endpos < 0:
            return endpos
```

```
        rawdata = self.rawdata
        self.__starttag_text = rawdata[i:endpos]

        # Now parse the data between i+1 and j into a tag and attrs
        attrs = []
        match = HTMLParser.tagfind.match(rawdata, i+1)
        assert match, 'unexpected call to parse_starttag()'
        k = match.end()
        self.lasttag = tag = rawdata[i+1:k].lower()

        while k < endpos:
            m = attrfind.match(rawdata, k)
            if not m:
                break
            attrname, rest, attrvalue = m.group(1, 2, 3)
            if not rest:
                attrvalue = None
            elif attrvalue[:1] == '\'' == attrvalue[-1:] or \
                 attrvalue[:1] == '"' == attrvalue[-1:]:
                attrvalue = attrvalue[1:-1]
                attrvalue = self.unescape(attrvalue)
            attrs.append((attrname.lower(), attrvalue))
            k = m.end()

        end = rawdata[k:endpos].strip()
        if end not in (">", "/>"):
            lineno, offset = self.getpos()
            if "\n" in self.__starttag_text:
                lineno = lineno + self.__starttag_text.count("\n")
                offset = len(self.__starttag_text) \
                         - self.__starttag_text.rfind("\n")
            else:
                offset = offset + len(self.__starttag_text)
            print "----------------------------------------------------------"
            print rawdata[i:i+250]
            print "----------------------------------------------------------"
            print end
            print "----------------------------------------------------------"
            self.error("junk characters in start tag: %s"
                       % `rawdata[k:endpos][:20]`)
        if end.endswith('/>'):
            # XHTML-style empty tag: <span attr="value" />
            self.handle_startendtag(tag, attrs)
        else:
            self.handle_starttag(tag, attrs)
            if tag in self.CDATA_CONTENT_ELEMENTS:
                self.set_cdata_mode()
        return endpos

    def check_for_whole_start_tag(self, i):
        rawdata = self.rawdata
        m = locatestarttagend.match(rawdata, i)
        if m:
            j = m.end()
            next = rawdata[j:j+1]
            if next == ">":
                return j + 1
            if next == "/":
                if rawdata.startswith("/>", j):
                    return j + 2
                if rawdata.startswith("/", j):
                    # buffer boundary
                    return -1
                # else bogus input
                self.updatepos(i, j + 1)
```

```
            self.error("malformed empty start tag")
        if next == "":
            # end of input
            return -1
        if next in ("abcdefghijklmnopqrstuvwxyz=/"
                    "ABCDEFGHIJKLMNOPQRSTUVWXYZ"):
            # end of input in or before attribute value, or we have the
            # '/' from a '/>' ending
            return -1
        #if next in "'" :
        #    return -1
        print "-----------------------------------"
        print next
        print m.groups ()
        print "-----------------------------------"
        print rawdata [i: i + 250]
        print "..................................."
        print rawdata [j: j + 30]
        self.updatepos(i, j)
        self.error("malformed start tag")
    raise AssertionError("we should not get here!")

def goahead(self, end):
    if not self.__dict__.has_key ("_script") :
        self._script    = False         # ajout
        self._nb_script = 0             # ajout
    rawdata = self.rawdata
    i = 0
    n = len(rawdata)
    while i < n:
        match = self.interesting.search(rawdata, i) # < or &
        if match:
            j = match.start()
        else:
            j = n
        if i < j: self.handle_data(rawdata[i:j])
        i = self.updatepos(i, j)
        if i == n: break
        startswith = rawdata.startswith

        if self._script :                           # ajout
            if startswith('</script>', i):          # ajout
                k = i + len ('</script>')           # ajout
                self._script = False                # ajout
                i = self.updatepos(i, k)
            elif startswith('</SCRIPT>', i):       # ajout
                k = i + len ('</SCRIPT>')           # ajout
                self._script = False                # ajout
                i = self.updatepos(i, k)
            else:
                k = i + 1
                i = self.updatepos(i, k)
        elif startswith('<', i):
            if startswith("<script", i):
                self._nb_script += 1
                k = i + len ("<script")
                self._script = True
            elif startswith("<SCRIPT", i):
                self._nb_script += 1
                k = i + len ("<SCRIPT")
                self._script = True
            elif starttagopen.match(rawdata, i): # < + letter
                k = self.parse_starttag(i)
            elif startswith("</", i):
                k = self.parse_endtag(i)
```

```
            elif startswith("<!--", i):
                k = self.parse_comment(i)
            elif startswith("<!-", i):
                k = self.parse_comment(i)
            elif startswith("<!/", i):
                k = self.parse_comment(i)
            elif startswith("<!-- /", i):
                k = self.parse_comment(i)
            elif startswith("<?", i):
                k = self.parse_pi(i)
            elif startswith("<!", i):
                k = self.parse_declaration(i)
            elif (i + 1) < n:
                self.handle_data("<")
                k = i + 1
            else:
                break
            if k < 0:
                if end:
                    print "########################################## (1)"
                    print rawdata [i:i+250]
                    self.error("EOF in middle of construct")
                break
            i = self.updatepos(i, k)
        elif startswith("&#", i):
            match = charref.match(rawdata, i)
            if match:
                name = match.group()[2:-1]
                self.handle_charref(name)
                k = match.end()
                if not startswith(';', k-1):
                    k = k - 1
                i = self.updatepos(i, k)
                continue
            else:
                break
        elif startswith('&', i):
            match = entityref.match(rawdata, i)
            if match:
                name = match.group(1)
                self.handle_entityref(name)
                k = match.end()
                if not startswith(';', k-1):
                    k = k - 1
                i = self.updatepos(i, k)
                continue
            match = incomplete.match(rawdata, i)
            if match:
                # match.group() will contain at least 2 chars
                if end and match.group() == rawdata[i:]:
                    print "########################################## (2)"
                    print rawdata [i:i+250]
                    self.error("EOF in middle of entity or char ref")
                # incomplete
                break
            elif (i + 1) < n:
                # not the end of the buffer, and can't be confused
                # with some other construct
                self.handle_data("&")
                i = self.updatepos(i, i + 1)
            else:
                break
        else:
            assert 0, "interesting.search() lied"
    # end while
```

```
        if end and i < n:
            self.handle_data(rawdata[i:n])
            i = self.updatepos(i, n)
        self.rawdata = rawdata[i:]
```

**fin correction TD 21.2.1** □

## 21.3 Python, Excel, Latex

### 21.3.1 Enoncé

On n'imagine plus un ordinateur sans un tableur comme Excel installé dessus.

### 21.3.2 Correction

**fin correction TD 21.3.1** □

## 21.4 Fond d'écran

### 21.4.1 Enoncé

### 21.4.2 Correction

```
"""panorama avec images"""
import pygame
import os
import time
from PIL import Image
import random

def attendre_touche () :
    for event in pygame.event.get():
        if event.type == pygame.MOUSEBUTTONUP : return "clic"
        elif event.type == pygame.KEYDOWN :
            if event.key == 275 : return "right"
            elif event.key == 276 : return "left"
            elif event.key == 273 : return "up"
            elif event.key == 274 : return "down"
            elif event.key == 27 : return "quit"
            elif event.key == 32 : return "pause"
            else :
                pass
                #print "key ", event.key
        else :
            #print event
            pass
    return ""
```

```
def get_image_list (dir = ".") :
    li = os.listdir (dir)
    res = []
    for l in li :
        s = l.lower ()
        if "jpg" in s or "png" in s or "jpeg" in s :
            res.append (l)
    res.sort ()
    return res

def image_resize (im, size, pil = True) :
    s = im.get_size ()
    rapx = float (size [0]) / s [0]
    rapy = float (size [1]) / s [1]

    rap = min (rapx, rapy)
    rap = min (rap, 2.0)

    size2 = ( int (s [0] * rap), int (s [1] * rap) )

    if not pil :
        res = pygame.transform.scale (im, size2)
    else :
        # scale from pygame is not good, use PIL's one
        s = pygame.image.tostring (im, "RGBX")
        temp = Image.fromstring ("RGBX", im.get_size (), s)
        tu = (0,0, im.get_size () [0]-1, im.get_size () [1] - 1)
        temp = temp.transform (size2, Image.EXTENT,  tu, Image.BICUBIC)
        mode = temp.mode
        size = temp.size
        data = temp.tostring()
        res = pygame.image.fromstring (data, size, mode)
    return res

def load_image (file) :
    try :
        im = pygame.image.load (file)
    except Exception, e :
        print "unable to load image ", file, " error ", e
        return None
    return im

def display (sur, im) :
    im = image_resize (im, sur.get_size ())
    sur.fill ( (0,0,0) )

    x = (sur.get_size () [0] - im.get_size () [0]) / 2
    y = (sur.get_size () [1] - im.get_size () [1]) / 2

    sur.blit (im, (x,y))

def loop (sur, li, d = 500, rnd = True, transition = True) :

    already = { }
    for i in range (0, len (li)) : already [i] = - len (li)

    print "number of images ", len (li)
    lo = 0
    i = 0
    while i < len (li) or rnd :

        lo += 1
        if rnd :
            i = random.randint (0, len (li)-1)
            while lo - already [i]  < len (li) / 2 :
```

```
                i = random.randint (0, len (li)-1)
        im = li [i]
        already [i] = lo


        ti = pygame.time.get_ticks ()

        s = load_image (im)
        if s == None : continue

        display (sur, s)

        if not transition :
            pygame.display.flip ()
        else :
            ts = sur.get_size ()
            for i in xrange (0, ts [0], 20) :
                r = pygame.Rect (i, 0, i+100, ts [1])
                pygame.display.update (r)
                pygame.time.delay (10)


        ti = ti - pygame.time.get_ticks ()

        ti = max (0, d - ti)
        pygame.time.delay (ti)

        if pygame.event.peek () :
            t = attendre_touche ()
            if t == "quit" : return
            elif t == "left" : i -= 1
            elif t == "right" : i += 1
            elif t == "up" : i += 10
            elif t == "down" : i += 10
            elif t == "pause" :
                while attendre_touche () != "pause" :
                    pass
        i += 1
        i = max (i, 0)

    print "stop at ", i


if __name__ == "__main__" :
    li = get_image_list ()

    pygame.init ()
    pygame.display.set_mode ((1280,800), pygame.FULLSCREEN)
    #pygame.display.set_mode ((1280,800)) #, pygame.FULLSCREEN)
    s = pygame.display.get_surface ()

    loop (s, li)

    pygame.time.delay (1000)
```

**fin correction TD 21.4.1** □

# Chapitre 22

# Logique et jeux

---

Il est sans doute plus long de concevoir un programme informatique pour résoudre un problème qui nécessite seulement quelques minutes de réflexion à une personne pour le résoudre. Il n'est pas évident de reproduire un raisonnement logique sous forme d'algorithme mais s'y essayer au moins une fois permet d'évaluer la complexité de problèmes plus difficiles.

Rédiger un algorithme est aussi un bon moyen de formaliser, de décrire très précisément un raisonnement, c'est le cas des trois premiers exemples. C'est aussi un moyen d'appréhender un problème mathématique, de confronter ses idées aux simulations réalisées par ordinateur. C'est le cas du dernier exemple.

## 22.1 Résolution d'un sudoku

### 22.1.1 Enoncé

### 22.1.2 Correction

```
s = [[0, 0, 0, 3, 0, 0, 8, 0, 0], \
     [0, 0, 7, 9, 0, 8, 0, 0, 5], \
     [0, 0, 0, 2, 0, 4, 1, 0, 0], \
     [0, 9, 0, 8, 1, 0, 0, 4, 7], \
     [0, 4, 0, 0, 0, 0, 0, 0, 6], \
     [0, 0, 0, 0, 0, 0, 0, 0, 0], \
     [0, 1, 0, 0, 0, 5, 0, 2, 0], \
     [5, 3, 4, 0, 0, 0, 0, 0, 0], \
     [0, 0, 0, 7, 0, 0, 0, 0, 0]]




i = 0
j = 1

def chiffre_barre_ligne (s,r,i) :
    for k in range (0,9) :
        if s [i][k] > 0 :
            r [ s [i][k]-1 ] = 0

def chiffre_barre_colonne (s,r,j) :
    for k in range (0,9) :
        if s [k][j] > 0 :
            r [ s [k][j]-1 ] = 0


def chiffre_barre_carre (s,r,i,j) :
    a = i/3 * 3
    b = j/3 * 3
    for k in range (a,a+3) :
        for l in range (b,b+3) :
```

```
            if s [k][l] > 0 :
                r [ s [k][l]-1 ] = 0

def nombre_possible (s,i,j ) :
    r = [ 1,1,1,1,1,1,1,1,1]
    chiffre_barre_ligne (s,r,i)
    chiffre_barre_colonne (s,r,j)
    chiffre_barre_carre (s,r,i,j)
    return r

def meilleure_case (s ) :
    dmin = 10
    imin = jmin = -1
    rmin = []
    for i in range (0,9) :
        for j in range (0,9) :
            if s [i][j] == 0 :
                r = nombre_possible (s,i,j)
                d = sum (r)
                if d < dmin :
                    dmin = d
                    imin = i
                    jmin = j
                    rmin = r
    return imin,jmin,rmin

def resolution (s) :
    imin,jmin,rmin = meilleure_case (s)
    if imin == -1 :
        return "jackpot"
    d = sum (rmin)
    if d == 1 :
        p = rmin.index (1)
        s [imin][jmin] = p+1
        print "s modifie ",imin,jmin,p+1
        a = resolution (s)
        if a == 0 :
            s [imin][jmin] = 0
            return 0
        else :
            return 1
    elif d == 0 :
        print "on s'arrete"
        return 0
    elif d > 1 :
        print "on continue"
        for n in range (0, 9) :
            if rmin [n] == 1 :
                s [imin][jmin] = n+1
                a = resolution (s)
                if a == 0 :
                    s [imin][jmin] = 0
                else :
                    return 1
        return 0

resolution (s)

for i in range (0,9) :
```

```
    print s[i]
```

```
import copy
import psyco

psyco.full ()

class Sudoku :
    """definition d'un sudoku"""
    def __init__ (self, tab) :
        """tab est soit un entier indiquant la dimension du sudoku ou
        le sudoku lui-même"""
        try :
            i = tab [0][0]
            self.tab = tab
            self.dim = len (tab)
        except :
            self.dim = tab
            self.tab = [ [ 0 for i in range (0,self.dim) ] \
                           for j in range (0,self.dim) ]


    def __str__ (self) :
        """affiche un sudoku"""
        s = ""
        for i in range (0,self.dim) :
            if i % 3 == 0 and i > 0 : s += "\t-----------------------------------------\n"
            s += "\t"
            for j in range (0,self.dim) :
                if j % 3 == 0 and j > 0 : s += "|\t"
                if self.tab [i][j] == 0 : s += "."
                else : s += str (self.tab [i][j])
                s += "\t"
            s += "\n"
        return s

    def __getitem__ (self, p) :
        """retourne l'élément p [0], p [1] du sudoku"""
        return self.tab [p [0]][p [1]]

    def __setitem__ (self, p, v) :
        """change l'élément p [0], p [1] du sudoku"""
        self.tab [p [0]][p [1]] = v

    def filled (self) :
        """retourne le nombre de cases non vides"""
        n = 0
        for i in range (0, self.dim) :
            for j in range (0, self.dim) :
                if self [ (i,j) ] > 0 : n += 1
        return n

    def possible (self, i, j) :
        """retourne l'ensemble des nombres possibles pour
        pour la case i,j"""

        # vérification des contraintes sur les lignes et les colonnes
        if self.tab [i][j] > 0 : return []
        se = []
        for k in xrange (0, self.dim) :
            if self.tab [i][k] > 0 :
                se.append (self.tab [i][k])
```

```
            if self.tab [k][j] > 0 :
                se.append (self.tab [k][j])

        # vérification des contraintes dans chaque petit carré
        ii = int (i / 3)
        jj = int (j / 3)
        for iii in range (ii * 3, (ii+1)*3) :
            for jjj in range (jj * 3, (jj+1)*3) :
                if self.tab [iii][jjj] > 0 :
                    se.append (self.tab [iii][jjj])


        sol = []
        for n in xrange (1, self.dim+1) :
            if n not in se : sol.append (n)

        return sol

    def get_best_solution (self, error = []) :
        """retourne la meilleure case à essayer, celle pour laquelle
        l'ensemble des nombres possibles est le plus petit :
         - 0 chiffre  : configuration impossible, résultat = None
         - 1 chiffre  : on avance, résultat = (i,j,[ c] )
         - n > 1 chiffres : on essaye l'ensemble le moins grand,
                    résultat = (i,j,[ c1, c2, c3, ...] )
        """

        # on regarde d'abord si c'est impossible
        for i in xrange (0,self.dim) :
            for j in xrange (0,self.dim) :
                if self.tab [i][j] > 0 : continue
                b = self.possible (i,j)
                if len (b) == 0 :
                    return None

        # sinon
        best = None
        pos = None

        for i in xrange (0,self.dim) :
            for j in xrange (0,self.dim) :
                if (i,j) in error : continue
                b = self.possible (i,j)
                if len (b) == 1 :
                    # une case où il y a une seule possibilité
                    return (i,j,b)
                elif len (b) > 1 :
                    # plusieurs possibilités, on prend l'ensemble le plus petit
                    if best == None or len (best) > len (b) :
                        pos = (i,j)
                        best = b

        # résultat
        if best == None :
            return None
        else : return (pos [0], pos [1], best)




class EnigmeSudoku :
    """résolution d'un Sudoku"""

    def __init__ (self, sudoku, display = True) :
        """définition de la classe qui résoud le Sudoku"""
```

```
        self.sudoku = sudoku

    def __str__ (self) :
        return str (self.sudoku)

    def resoud (self, solutioni = None) : # solution = [ ]) :
        """resolution, exploration en profondeur,
        fonction recursive"""

        if solutioni == None :
            # cas simple : première itération
            solutioni = copy.deepcopy (self.sudoku)
            self.count = 0
        elif solutioni.filled () == solutioni.dim ** 2 :
            # second simple : le sudoku est terminé
            return solutioni

        solution = copy.deepcopy (solutioni)

        # pour compter le nombre d'itérations
        self.count += 1

        # retourne la case la plus sympathique
        next = solutioni.get_best_solution ()

        # s'il existe une case impossible
        if next == None : return None

        # sinon on essaye toutes les solutions pour cette case
        for n in next [2] :
            solution [ (next [0], next [1]) ] = n
            res = self.resoud (solution)
            if res != None :
                # réussi
                return res

        # toutes les solutions pour cette case mènent à une impasse
        return None


if __name__ == "__main__" :

    sudoku_exemple = [ \
              [5,3,0,0,7,0,0,0,0], \
              [6,0,0,1,9,5,0,0,0], \
              [0,9,8,0,0,0,0,6,0], \
              [8,0,0,0,6,0,0,0,3], \
              [4,0,0,8,0,3,0,0,1], \
              [7,0,0,0,2,0,0,0,6], \
              [0,6,0,0,0,0,2,8,0], \
              [0,0,0,4,1,9,0,0,5], \
              [0,0,0,0,8,0,0,7,9]]

    s = Sudoku (sudoku_exemple)
    print s
    print "------------------------------\n"


    en = EnigmeSudoku (s)
    print "------------------------------\n"
    sol = en.resoud ()
    print "------------------------------\n"
    print "iteration ", en.count
    print sol
    print "fin"
```

## 22.2 Résolution d'un puzzle

Le puzzle en question est celui de la figure 22.1, il ne contient que neuf pièces, chacune d'elle est carrée et chaque côté est soit la tête, soit les jambes d'une girafe pour lesquelles il y a quatre couleurs possibles. L'appariement des pièces ne dépend que des motifs sur les bords de chaque pièce et non de leur forme. La forme finale du puzzle - un grand carré - est connue. Même s'il paraît simple, il existe un grand nombre d'arrangements possibles : il existe 9! façons possibles de placer chaque pièce et à chaque position, chaque pièce peut être tournée 4 fois. Nous oublierons volontairement les symétries car il plus difficile d'en tenir compte dans un programme informatique que de faire l'impasse dessus.

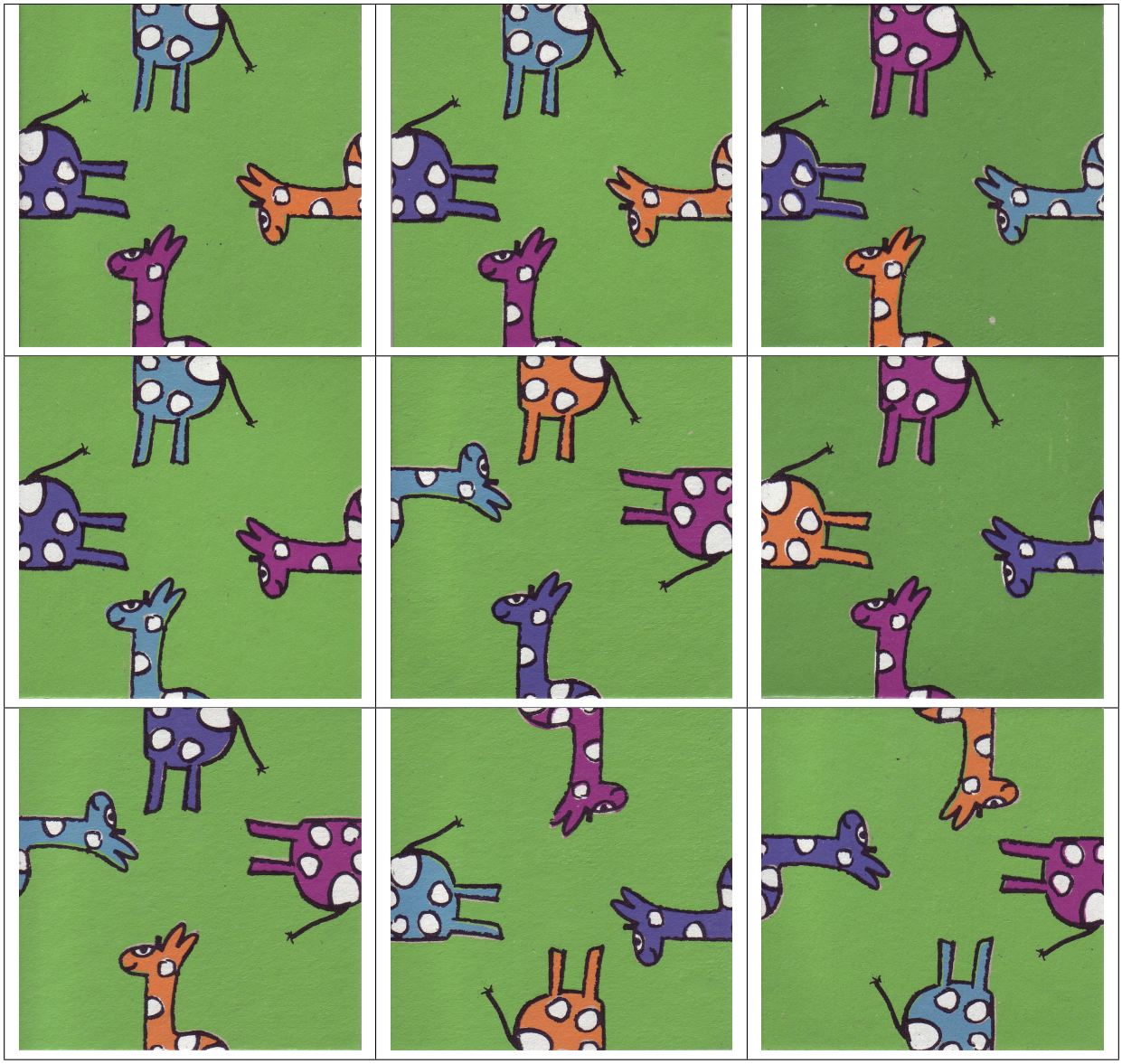


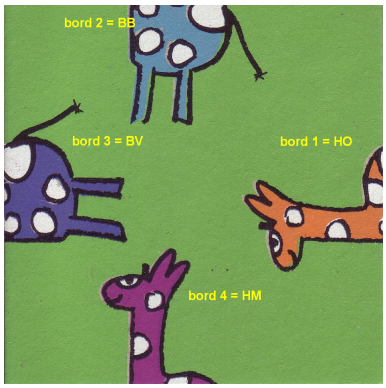
**Figure 22.1 :** *Les neuf pièces du puzzle (voir la figure 22.8 pour les voir assemblées).*

Avant de pouvoir construire la résolution du puzzle, il faut tout d'abord décrire les éléments qui le compose afin de pouvoir les manipuler. C'est l'objet du paragraphe suivant.

### 22.2.1 Description des données

La description des données suggère deux problèmes. Le premier est l'acquisition et le second leur représentation au sein du programme informatique. Un rapide coup d'œil sur les images et on détecte facilement qu'il existe 2x4=8 différents bords possibles et qu'une pièce peut simplement être décrite par la donnée de ces quatre bords. Mais les obtenir depuis l'image directement fait appel à des techniques de traitements d'images qui seraient trop longues à mettre en place. Il est beaucoup plus simple de construire un fichier texte dans lequel seront stockées les informations utiles concernant chaque pièce.

Il n'existe pas de manière unique de décrire un bord, le moyen envisagé est le suivant : chaque pièce va être décrite par une chaîne de huit caractères, 4x2 pour être exacte. Chaque bord est décrit par deux lettres, la première H ou B désigne le haut ou le bas de la girafe. La seconde lettre M, O, B ou V désigne la couleur mauve, orange, bleu ou violet. Par exemple, la première pièce HOBBBVHM possède quatre bords définis comme suit :




1. HO = haut orange
2. BB = bas bleu
3. BV = bas violet
4. HM = haut mauve

Mises bout à bout, ces descriptions aboutissent au contenu de la figure 22.4 (page 483). Il ne reste plus qu'à écrire un petit bout de code capable de lire ce fichier puis de construire chaque pièce. Mais avant cela, il faut imaginer une représentation informatique de ces données qui va recevoir l'information lue depuis les chaînes de caractères.

Les classes ne sont pas indispensables d'autant plus que aucun héritage ne sera nécessaire dans ce cas présent. Toutefois, elles simplifient la description de chaque pièce : le puzzle comporte neuf pièces qui chacune possède quatre bords. Le programme contiendra trois classes : `Bord`, `Piece`, `Puzzle`. Ces trois classes contiendront à la fois des données et des méthodes qui s'appliquent à ces données.

#### 22.2.1.1 classe `Bord`

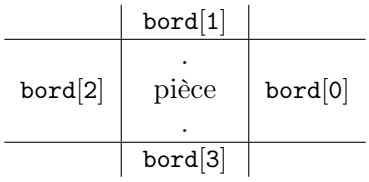
La classe `Bord` ne contient que deux attributs, `partie` qui désigne la partie du corps de la girafe et `couleur`, sa couleur. Ce sont les seules données. Il n'y a pas besoin de données supplémentaires pour dire si deux bords peuvent être appariés, ce sera la méthode `compatible` et incluse dans la classe `Bord`. Deux bords pourront seront compatibles s'ils ont la même couleur et des parties opposées.

```
    def compatible (self, bord) :
        return self.couleur == bord.couleur and self.partie != bord.partie
```

Les autres méthodes de cette classe sont le constructeur `__init__` permettant de renseigner les attributs de la classe depuis une chaîne de caractères et la méthode `__str__` chargée d'afficher le bord.

#### 22.2.1.2 classe `Piece`

Une pièce est tout d'abord composée de quatre bords, la classe `Piece` doit nécessairement contenir une liste de quatre instances de la classe `Bord`. La seule difficulté consiste à préciser à quel bord de la pièce correspond le premier bord de la liste. On décide de tourner dans le sens trigonométrique.




La description d'une pièce fait également intervenir sa position et son orientation (le fait de tourner la pièce de 0,90,180 ou 270 degrés) par rapport à sa description d'origine. Ce ne sont pas des informations immuables, elles seront le résultat de la résolution du puzzle mais il faut décider où placer ces informationsé : au sein de la classe `Piece` ou au sein de la classe `Puzzle`. Il n'y a pas de meilleur choix, tout dépend de la manière de rédiger l'algorithme ou de comment ces classes seront utilisées par la suite. Dans notre cas, le choix est de les insérer dans la classe `Piece`, ce seront les attributs `orientation` et `position`. Ce dernier mérite un peu plus d'attention car son type sera entier et non un t-uple de deux entiers.

La position‘ [1] pourrait être décrite grâce à deux coordonnées entières mais le sera grâce à un seul entier. On obtient les coordonnées comme ceci d'après la position $n \in \{1, ..., 0\}$ :

$$x = \{(n-1) \mod 3\} + 1 \tag{22.1}$$

$$y = \left[\frac{n-1}{3}\right] + 1 \tag{22.2}$$

Réciproquement, la position $n$ est obtenue à partir des coordonnées $(x, y) \in \{1, ..., 3\}^2$ :

$$n = (y-1) * 3 + x \tag{22.3}$$

La position est un entier dans l'ensemble $\{1, ...9\}$. Lors de la résolution, certaines pièces seront placées et d'autres pas encore. La convention choisie pour une pièce non encore placée est de lui affecter une position nulle. Dans ce cas, l'orientation n'a pas d'importance.

La connaissance de l'orientation et de la position de deux pièces permet également de savoir si elles peuvent être voisines étant donné le motif de leur bord conjoint s'il existe. Ce sera l'objectif de la méthode `voisin_possible` qui retourne `True` si les deux pièces font bon ménage ensemble. Le premier test porte sur les positions des pièces et de savoir si elles sont effectivement voisines. Pour cela, on calcule la différence $d$ des positions. Les deux pièces sont voisines si $d$ est égale à 1 ou -1 et si les pièces sont sur la même ligne ou que $d$ est égale à 3 ou -3.

```
    def voisin_possible (self, p, a) :
        d = p.position - self.position
        if abs (d) == 1 and (p.position - 1) / 3 == (self.position - 1) / 3 :
            # voisin en x
            return True
        elif abs (d) == 3 :
            # voisin en y
            return True
        else :
            # pas voisin
            return False
```

1. La vérification des affirmations qui suivent est laissée au soin du lecteur. Le procédé est identique à celui utilisé pour la représentation des matrices par des tableaux d'une seule dimension.

Une fois cette condition vérifiée, il reste à vérifier que le bord qu'elles ont en commun ont des motifs appariés. Il faut donc dans un premier temps savoir quels sont les deux bords en contact. Il faut d'abord les nommer, ils seront appelés bord 0, 90, 180, 270 en tournant dans le sens trigonométrique. Lorsqu'on tourne la pièce de 90 degrés, le 0 est à la place du bord 90. En tenant compte de l'orientation, on peut calculer l'indice $a'$ qui tient compte de l'orientation $o$ (voir la figure 22.2). Soit $a \in \{0, 90, 180, 270\}$ le nom du bord cherché et $o \in \{0, 90, 180, 270\}$ l'orientation de la pièce, dans ce cas :

$$a' = \frac{(a - o + 360) \mod 360}{90} \in \{0, 1, 2, 3\} \text{ est l'indice du bord cherché} \tag{22.4}$$

| | | |
|---|---|---|
| | $0 = 90 + 90$<br>$a' = a - o$ | |
| $90 = 180 + 90$<br>$a' = a - o$ | pièce tournée de 90 degré | $270 = 360 + 90$<br>$a' = a - o$ |
| | $180 = 270 + 90$<br>$a' = a - o$ | |

**Figure 22.2 :** *Correspondance entre a, o, $a'$ lorsqu'une pièce est tournée de 90 degré dans le sens trigonométrique.*

Deux pièces sont voisines si elles sont à gauche et droite l'une de l'autre (premier cas) ou en haut et bas l'une de l'autre (second cas). Le dessin de la figure 22.3 permet de trouver les paires d'indices $a'$ qui correspondent aux bords en contact, et connaissant l'orientation de chaque pièce, les indices $a$ des bords en contact. La méthode `compatible` de l'un des deux bords `dira` si effectivement les deux bords en contact sont appariés.

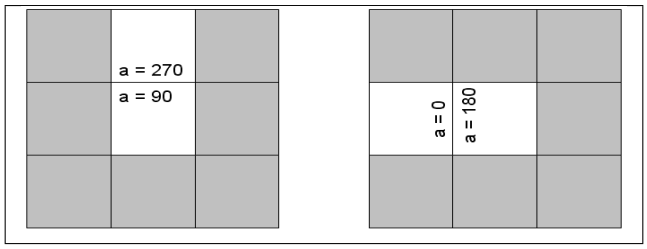



**Figure 22.3 :** *Deux pièces sont voisines en x ou en y. Les bords voisins sont dans le premier cas* 0 *et* 180*, dans le second,* 90 *et* 270 *(indices $a'$).*

Les autres méthodes de la classe `Piece` concernent l'affichage et la création des pièces à partir du format décrit au paragraphe 22.2.1 et stocké dans un fichier texte (voir figure 22.4). Les autres attributs sont également liés à l'affichage, un nom de fichier, et deux images. Leur utilisation sera décrite au paragraphe 22.2.2.

#### 22.2.1.3 classe `Puzzle`

Avant de construire de chercher à résoudre le problème, il faut s'interroger sur la forme du résultat que l'on cherche. Dans le cas de ce puzzle, ce peut être une liste de 9 `Piece` et pour chacune d'entres elles, les attributs `position` et `orientation` doivent être renseignés ; c'est à dire être compris en 1 et 9 pour la position et dans l'ensemble $\{0, 90, 180, 270\}$ pour l'orientation.

La classe `Puzzle` contiendra donc une liste de `Piece` précédemment décrite et incluera les méthodes nécessaires à la résolution du puzzle. La description du principe général de la résolution interviendra après celles de méthodes simples de la classe `Puzzle` [2].

**piece_position**

2. On utilise parfois le terme de *briques élémentaires*, l'algorithme jouant le rôle de la charpente.

```
HOBBBVHM
HOBBBVHM
HBBMBVHO
HMBBBVHB
BMBOHBHV
HVBMBOHM
BMBVHBHO
HVHMBBBO
BMHOHVBB
```

**Figure 22.4 :** *Description des neuf pièces du puzzle au format texte, une pièce par ligne. Chaque ligne est une séquence de quatre groupes de deux lettres. Chaque groupe désigne un bord d'une pièce. La première lettre (H ou B) désigne le haut ou le bas d'une girafe. La seconde lettre (O, M, V, B) désigne sa couleur (orange, mauve, violet, bleu).*

La liste résultante n'est pas forcément ordonnée en ce sens que la pièce en première position dans la liste n'est pas forcément la première pièce du puzzle. C'est l'attribut `position` de chaque pièce qui détermine la position de celle-ci. La méthode `piece_position` retourne la pièce à une certaine position donnée ou `None` si l'algorithme n'a pas encore posé de pièce à cet endroit.

```
def piece_position (self, pos) :
    for p in self.piece :
        if p.position == pos :
            return p
    return None
```

**ensemble_voisin**

Cette fonction a pour objectif de retourner la liste des positions voisines à une position donnée. Le résultat est illustré par la figure 22.5.

| | | |
|---|---|---|
| **1** 2,4 | **2** 1,3,5 | **3** 2,6 |
| **4** 1,5,7 | **5** 2,4,6,8 | **6** 3,5,9 |
| **7** 4,8 | **8** 5,7,9 | **9** 6,8 |

```
def ensemble_voisin (self, i) :
    i -= 1
    res = []
    for x in [-1,0,1] :
        for y in [-1,0,1] :
            if abs (x) == abs (y) : continue
            if x == -1 and i % 3 == 0 : continue
            if x == 1  and i % 3 == 2 : continue
            if y == -1 and i / 3 == 0 : continue
            if y == 1  and i / 3 == 2 : continue
            j = i + x + y * 3
            if j in range (0,9) :
                res.append (j)
    return [ j+1 for j in res ]
```

**Figure 22.5 :** *Ensemble des positions voisines pour chacune des positions.*

**angle_possible**

Il s'agit ici de déterminer une orientation possible pour la pièce `p` sachant qu'elle est à la position `p.position`. Dans ce cas, on détermine l'ensemble des positions voisines à `p.position`, puis l'ensemble des pièces posées sur ces cases voisines. Il ne reste plus qu'à déterminer pour une pièce toutes les orientations qui ne dérogent pas à la règle des appariements de côtés. La fonction retourne une liste d'au plus quatre éléments et vide si une pièce ne peut être placée à la position `p.position` dans quelque orientation que ce soit.

```
def angle_possible (self, p, display = False) :
    voisin = self.ensemble_voisin (p.position)
    if display :
        print "voisin = ", voisin
    res = []
    for a in [0,90,180,270] :
        r = True
        for v in voisin :
            piece = self.piece_position (v)
            if piece != None :
                r = r and piece.voisin_possible (p, a)
        if r :
            res.append (a)
    return res
```

### 22.2.2 Résolution

La description des éléments du puzzle est terminée ainsi que quelques manipulations simples. Avant d'entamer la résolution proprement dite, il est sans doute utile de préciser la complexité du problème souvent exprimée par l'étendue de l'ensemble des solutions. Chaque pièce doit être placée à une position donnée parmi 9 dans une orientation donnée parmi 4. Avec 9! permutations, il y a donc potentiellement 9! x $4^9$ = 18.874.368 solutions envisageables et pour chacune d'elles, il suffit qu'un appariement ne soit pas possible pour que la solution n'en soit pas une.

Il est possible d'explorer l'ensemble des solutions avec les ordinateurs de bureau actuels. Avec 16 pièces, ce serait encore très long (de l'ordre de $10^{23}$). En théorie, l'algorithme proposé ne réduit pas forcément le nombre de solutions à explorer, ceci dépend entre autres de l'ordre initial des pièces. En pratique, il permet souvent d'éliminer des sous-ensembles de solutions. Par exemple, supposons que cinq pièces soient déjà placées aux positions 1 à 5. S'il n'est pas possible de placer aucune des autres pièces à la position 6 dans quelque orientation que ce soit, aucune des solutions avec ces cinq pièces ne marchera, soit 6144 solutions non essayées grâce à quelques tests.

La résolution du puzzle passe par la construction d'une fonction récurrente qui s'inspire de celle qu'on utilise pour construire l'ensemble des permutations de $\{1, ..., n\}$.

#### 22.2.2.1 Permutations

Il est assez intuitif de construire par récurrence l'ensemble des permutations de $\{1, ..., n\}$ à l'instar de la fonction factorielle. Parmi les $n!$ permutations, il y a extactement $(n-1)!$ permutations avec le nombre 1 en première position, il y a aussi $(n-1)!$ avec le nombre 2 en première position et ainsi de suite. En s'aidant de cette constatation, on cherche à construire la fonction `permutation(n)` qui retourne la liste des permutations de l'ensemble $\{1, ..., n\}$ par récurrence.

On note $P_n$ les permutations de l'ensemble $\{1, ..., n\}$. Les premières $(n-1)!$ permutations sont celles où le premier nombre 1 est fixé en première position et où on applique les permutations $P_{n-1}$ à l'ensemble restant $\{2, ..., n\}$. Les permutations suivantes sont celles où on permute 1 avec 2 et où on applique les permutations à l'ensemble $\{1, 3, 4, ..., n\}$. On obtient en continuant ce raisonnement :

| | |
|---|---|
| première vague de permutations : | 1 + permutations appliquées à $\{2, 3, 4, 5, ..., n\}$ |
| seconde vague de permutations : | 2 + permutations appliquées à $\{\mathbf{1}, 3, 4, 5, ..., n\}$ |
| troisième vague de permutations : | 3 + permutations appliquées à $\{2, \mathbf{1}, 4, 5..., n\}$ |
| quatrième vague de permutations : | 4 + permutations appliquées à $\{2, 3, \mathbf{1}, 5..., n\}$ |
| ... | |

Le programme complet aboutit à :

```
# -*- coding: iso-8859-15 -*-
def echange (l, i, j) :
    """permutation des éléments i et j de la liste l"""
    t       = l [i]
    l [i]   = l [j]
    l [j]   = t

def applique_permutation (set, p) :
    """applique la permutation p à l'ensemble set"""
    return [ set [k-1] for k in p ]

def permutation (n) :
    if n == 1 : return [ [ 1 ] ]
    else :
        per = permutation (n-1)             # permutation sur {1,...,n-1}
        set = [ i+1 for i in range (0,n) ]  # ensemble de départ
        res = []                            # résultat

        for i in range (0, n) :
            echange (set, 0, i)             # échange élément 0 et i
            for p in per :                  # boucle sur les permutations de {1,...,n-1}
                pr  = [ set [0] ] + applique_permutation ( set [1:], p )
                res.append (pr)             # on l'ajoute au résultat
            echange (set, 0, i)             # échange élément 0 et i

        return res

print permutation (1) # retourne [ [ 1 ] ]
print permutation (2) # retourne [[1, 2], [2, 1]]
print permutation (3) # retourne [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 2, 1], [3, 1, 2]]
```

#### 22.2.2.2 Application au puzzle

Il serait simple d'utiliser le programme précédent pour construire l'ensemble des permutations puis de le parcourir en changeant pour chacune d'entre elles les orientations de chaque pièce mais ce système nous ferait évaluer l'ensemble des solutions possibles sans possibilité de le réduire comme suggéré au paragraphe 22.2.2. Il vaut mieux reprendre le programme précédent et remplacer la fonction `permutation` de sorte qu'elle explore les solutions au lieu de les retourner comme résultat.

Le principe de la résolution repose sur la récurrence de la méthode `solution`. Il y a 4x9 possibilités pour le choix en première position. La méthode `solution` en choisit une puis appelle la méthode `solution` sur la seconde position, si celle-ci retourne `True` alors ce premier choix était le bon, sinon on passe au choix suivant. Au second appel, une pièce est choisie pour la seconde position puis on appelle encore la méthode `solution`. Le schéma est le suivant :

1. On calcule la liste des choix pour la position $p$.
2. Si cet ensemble est vide, on retourne `False`, il n'y a pas de solution possible dans cette voie.
3. Sinon on choisit la première possibilité.
4. On appelle la méthode `solution` pour la position $p + 1$.
5. Si cette méthode retourne `True`, une solution a été trouvée.
6. Sinon, on passe au choix suivant jusqu'à ce qu'il n'y en ait plus.

L'implémentation proposée ne suit pas tout-à-fait ce schéma puisqu'on ne calcule pas l'ensemble des choix possibles au début de la méthode `solution` mais on calcule un premier choix (méthode `meilleur_choix` en tenant des compte des pièces déjà placées (dans la liste `free`). Si la solution finale utilise cette pièce, c'est terminé sinon, la fonction `meilleur_choix` est appelée une seconde fois pour obtenir un second choix de pièce en tenant compte des pièces placées et essayées. Le paragraphe 22.2.4 reprend le code de la méthode `solution`. La figure 22.8 montre le puzzle fini.

Il convient de rapprocher l'utilisation de méthodes récursives au concept de graphe ou plus précisément d'arbre. La figure 22.6 illustre ce que peut être un tel arbre dans le cas de ce puzzle. Chaque nœud représente un nouvel appel à la méthode `solution`. Chaque nœud est relié par un arc au nœud précédent l'ayant appelé. Une branche (depuis la racine jusqu'à une feuille) longue de moins de 9 nœuds signifie qu'une contradiction est apparue avant de placer toutes les pièces, ceci signifie qu'il n'était pas nécessaire d'explorer toutes les branches passant par les derniers nœuds intermédiaires. Une branche de 9 nœuds signifie que la contradition est apparue sur la dernière pièce et une seule branche de 9 nœuds a mené à la solution trouvée.

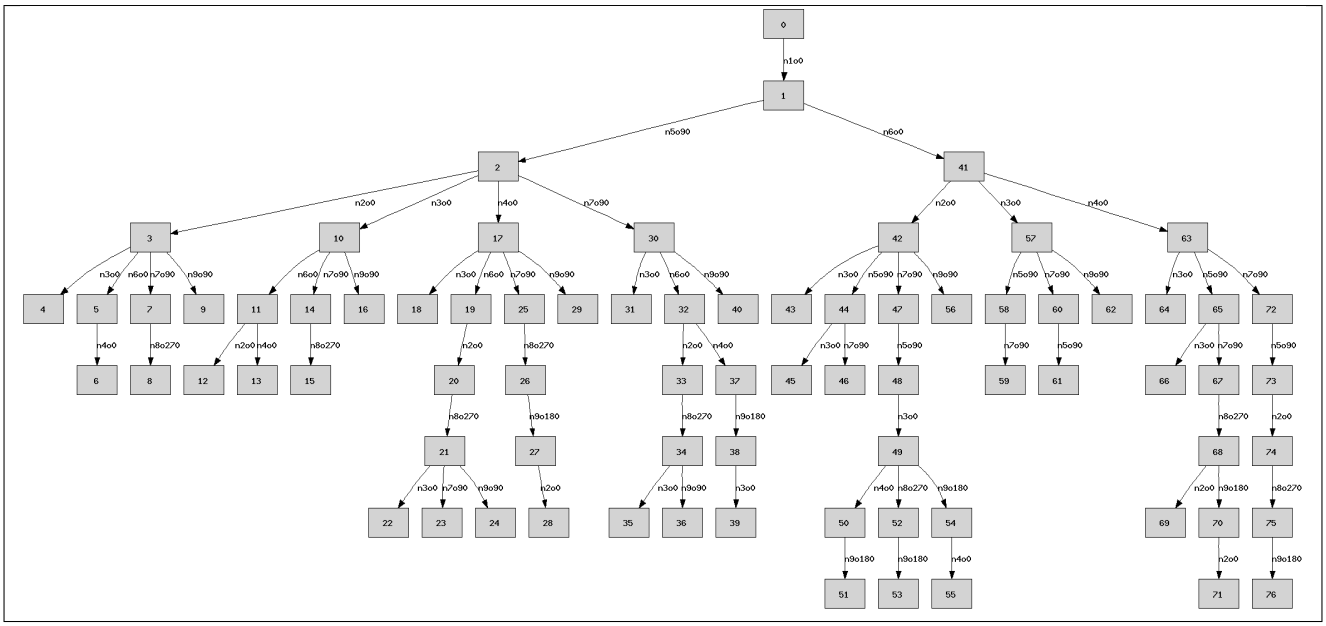


**Figure 22.6 :** *Dépliement des appels récursifs de la méthode* `solution` *sous forme d'arbre. Le numéro de chaque nœud désigne le* $n^{ième}$ *appel à la méthode* `solution`*. Sur chaque arc est superposé un label du type* `n3o90`*, ceci signifie que l'algorithme a placé la pièce 3 tournée de 90 degrés à une position indiquée par la profondeur du nœud dans l'arbre.*

### 22.2.3 Prolongements

On retrouve des puzzles carrés comme celui-là avec des photos aériennes (voir figure 22.7). Une zone géographique a été balayée avec un avion ou un satellite et on dispose d'une multitude de photos carrées disjointes recouvrant la zone. Le nombre de pièces est bien souvent plus grand que 9, une autre difficulté est que parfois, on ne connaît pas les dimensions du puzzle final et il faut trouver le meilleur arrangement. Enfin, autant dans l'exemple des girafe, il est facile de dire si deux côtés sont appariés ou non, autant avec des photos aériennes, ce n'est pas toujours un problème facile. Cela est dû au fait que les photos ne sont pas toujours disjointes, que certaines photos ne sont pas prises dans les mêmes conditions (un nuage passe et les couleurs sont assombries), la compression à une faible résolution puis la décompression déteriore la qualité des images. Dans ces conditions, la notion d'appariement est plus une notion floue.

### 22.2.4 Programme final

Le programme s'ouvre sur les neuf pièces disposées à une position correspondant à leur ordre d'apparition et dans leur orientation d'origine. Un clic de souris fait démarrer le programme. La recherche de la solution fait apparaître les choix effectués par l'algorithme jusqu'à ce qu'une solution se dégage. Un dernier clic de



**Figure 22.7 :** *Prises de vue aériennes recouvrant une zone géographique. Le puzzle reconstruit n'a pas une forme déterminée à l'avance.*

souris termine le programme. La fonction `sleep` permet de ralentir l'exécution du programme et de rendre le défilement des pièces humainement visible.

Les images sont chargées une fois puis recopiées et réduites. Une image de petite taille signifie que la pièce n'a pas encore été placée par la méthode de résolution.

```
# -*- coding: cp1252 -*-
import pygame
import random
import copy
import time

class Bord :
    """définition d'un bord ou côté d'une pièce, il possède :
       - partie : une partie de la girafe (haut ou bas)
       - une couleur : la couleur de cette partie, (orange, violet, bleu clair, bleu foncé)
    """

    def __init__ (self, definition) :
        """definition est une chaîne de 2 caractères qui définit un bord, exemple :
            HO pour haut orange
```

```
            BB pour bas bleu
            BV pour bas violet
            HM pour haut mauve
        """

        if   definition [0] == "H" : self.partie = "haut"
        elif definition [0] == "B" : self.partie = "bas"
        else :                       self.partie = definition + "###"

        if   definition [1] == "O" : self.couleur = "orange"
        elif definition [1] == "B" : self.couleur = "bleu clair"
        elif definition [1] == "M" : self.couleur = "violet"
        elif definition [1] == "V" : self.couleur = "bleu fonce"
        else :                       self.couleur = definition + "##"

    def __str__ (self) :
        """cette méthode est appelée lorsqu'on exécute l'instruction print
        avec un objet de type Bord"""
        s = self.partie + " " + self.couleur
        return s

    def compatible (self, bord) :
        """dit si deux bords sont compatibles, c'est à dire
        de la même couleur et de partie différente"""
        return self.couleur == bord.couleur and self.partie != bord.partie


class Piece :
    """définition d'une pièce du puzzle, celle-ci inclut :
      - bord : cette liste contient quatre objets de type Bord, cette liste ne changera plus
      - position    : c'est la position de la pièce dans le puzzle, ce qui nous intéresse,
                      c'est la position finale de la pièce dans le puzzle, cette information
                      va donc bouger au fur et à mesure que nous allons essayer de
                      résoudre le puzzle
      - orientation : de même que pour la position, une pièce peut être tournée sans changer de
                      position, c'est le résultat final qui nous intéresse

    pour l'affichage, on ajoute deux informations :
      - name  : le nom de l'image de la pièce
      - image : c'est la représentation de l'image dans la mémoire de
                l'ordinateur pour le module pygame
    """

    def __init__ (self, name, definition, position, numero) :
        """ on définit la pièce
           - name : nom de l'image représentant la pièce
           - definition : chaîne de 8 caractères, c'est une suite de 4 x 2 caractères définissant
                          chaque bord, voir la classe bord pour leur signification
           - position   : c'est la position initiale de la pièce, on suppose que
                          l'orientation est nulle pour commencer

        à partir de ces informations, on construit :
          - image       : c'est la représentation en mémoire de l'image de la pièce
          - bord        : c'est une liste qui définit les 4 bords de la pièce
          - orientation : c'est l'orientation de la pièce, au début de la résolution, elle est nulle
        """

        self.name           = name
        image               = pygame.image.load (name)
        self.image          = pygame.transform.scale (image, (250,250))
        s                   = self.image.get_size ()
        self.image_petite   = pygame.transform.scale (self.image,  (int(s[0]*0.7), int(s[1]*0.7)))
        self.bord           = []

        for i in range (0,4) :
```

```
            self.bord.append ( Bord (definition [i*2:i*2+2] ) )

        self.orientation    = 0
        self.position       = position
        self.numero         = numero

    def __str__ (self) :
        """définition ce qu'on doit afficher lorsqu'on exécute
        l'instruction print avec un objet de type Piece"""
        s = str (self.position) + " : "
        for b in self.bord :
            s += str (b) + " - "
        s += " orientation " + str (self.orientation)
        s += " numero " + str (self.numero)
        return s

    def affiche (self, screen, position) :
        """affiche la pièce à l'écran en tenant compte de sa position et de son orientation"""
        if "darker" in self.__dict__ and self.darker :
            position = ( position [0] + 20, position [1] + 20 )
            image    = pygame.transform.rotate (self.image_petite, self.orientation)
            screen.blit (image, position)
        else :
            image = pygame.transform.rotate (self.image, self.orientation)
            screen.blit (image, position)

    def bord_angle (self, angle, orientation = None) :
        """retourne le bord connaissant l'orientation de la pièce,
        le bord demandé est celui correspondant à :
           - 0  bord droit
           - 90 bord haut
           - 180 bord gauche
           - 270 bord bas"""
        if orientation == None : return self.bord_angle (angle, self.orientation)
        else :
            dif = (angle - orientation + 360) % 360 / 90
            return self.bord [dif]

    def voisin_possible (self, p, a) :
        """détermine si la pièce self peut être voisine avec la pièce p tournée de l'angle a"""
        d = p.position - self.position
        if abs (d) == 1 and (p.position - 1) / 3 == (self.position - 1) / 3 :
            # voisin en x
            if d == 1 :
                a1 = 0
                a2 = a1 + 180
            else :
                a1 = 180
                a2 = 0
        elif abs (d) == 3 :
            # voisin en y
            if d == 1 :
                a1 = 90
                a2 = 270
            else :
                a1 = 270
                a2 = 90
        else :
            # pas voisin
            return False

        b1 = self.bord_angle (a1)
        b2 = p.bord_angle (a2, a)
        return b1.compatible (b2)
```

```
class Puzzle :
    """définition d'une classe puzzle, elle contient simplement
    une liste de 9 pièces dont les positions sont :
    1 2 3
    4 5 6
    7 8 9
    et les orientations choisies dans l'ensemble { 0,90,180,270 }
    """

    def __init__ (self, dir = ".") :
        """on définit le puzzle à partir des informations contenus
        dans le répertoire dir qui doit contenir :
        - 9 images appelées piece0.png, ..., piece1.png
        - un fichier definition.txt contenant la définition de
          chacun des 4 bords de chacune des 9 pièces
                            HOBBBVHM
                            HOBBBVHM
                            HBBMBVHO
                            HMBBBVHB
                            BMBOHBHV
                            HVBMBOHM
                            BMBVHBHO
                            HVHMBBBO
                            BMHOHVBB
        """

        # on lit définition.txt
        f = open (dir + "/definition.txt")
        bo = f.readlines ()
        f.close ()

        # on définit chaque pièce
        self.piece = []
        for i in range (1,10) :
            name    = dir + "/piece" + str (i) + ".png"
            d       = bo [i-1]
            p       = Piece (name, d, 0, i)
            self.piece.append ( p )

    def __str__ (self) :
        """ce qu'on doit afficher lorsqu'on exécute
        l'instruction print avec un objet de type Puzzle"""
        s = """1 2 3
4 5 6
7 8 9
"""
        for p in self.piece :
            s += str (p) + "\n"
        return s

    def pixel (self, position) :
        """retourne en fonction de la position (1 à 9) de la pièce sa position sur l'écran,
        soit deux coordonnées"""
        p       = position - 1
        ligne   = p / 3
        colonne = p % 3
        return (colonne * 250, ligne * 250)

    def affiche (self, screen, petite = False) :
        """affiche les pièces sur l'écran,
        en plus petit pour celles qui ne sont pas encore placées"""
        screen.fill ((0,0,0))
        free = [0 for i in self.piece]
```

```
        for p in self.piece :
            if p.position > 0 :
                p.darker = False
                p.affiche (screen, self.pixel (p.position))
                free [p.position-1] = 1

        if petite :
            em = []
            for i in range (0, len (free)) :
                if free [i] == 0 : em.append (i+1)
            i = 0
            for p in self.piece :
                if p.position == 0 :
                    p.darker = True
                    p.affiche (screen, self.pixel (em [i]))
                    i += 1

        pygame.display.flip ()

    def meilleure_piece (self, free, pos) :
        """retourne la prochaine pièce à placer sur le puzzle,
        dans un premier temps, on peut prend la première qui vient,
        ensuite, on peut essayer un choix plus judicieux"""
        if len (free) == 0 : return None
        else : return free [0]

    def piece_position (self, pi) :
        """recherche la piece associée à la position pi"""
        for p in self.piece :
            if p.position == pi :
                return p
        return None

    def ensemble_voisin (self, i) :
        """retourne les positions voisins de la position i"""
        i -= 1
        res = []
        for x in [-1,0,1] :
            for y in [-1,0,1] :
                if abs (x) == abs (y) : continue
                if x == -1 and i % 3 == 0 : continue
                if x == 1  and i % 3 == 2 : continue
                if y == -1 and i / 3 == 0 : continue
                if y == 1  and i / 3 == 2 : continue
                j = i + x + y * 3
                if j in range (0,9) :
                    res.append (j)
        return [ j+1 for j in res ]

    def nb_place (self) :
        """retourne le nombre de places vides"""
        i = 0
        for p in self.piece :
            if p.position == 0 : i += 1
        return i

    def angle_possible (self, p, display = False) :
        """retourne l'ensemble des angles possibles pour une pièce donnée"""
        voisin = self.ensemble_voisin (p.position)
        if display :
            print "voisin = ", voisin
        res = []
        for a in [0,90,180,270] :
            r = True
            for v in voisin :
```

```
                piece = self.piece_position (v)
                if piece != None :
                    r = r and piece.voisin_possible (p, a)
            if r :
                res.append (a)
        return res

    def solution (self, pos = 1, screen = None) :
        if pos == 1 :
            for p in self.piece :
                p.position = 0
            self.nb_position = 0
            self.essai = 0

        self.essai += 1

        if self.nb_position == len (self.piece) :
            time.sleep (0.2)
            return

        # le tableau free mémorise l'ensemble des pièces non encore placées
        free = []
        for p in self.piece :
            if p.position == 0 :
                free.append (p)

        if screen != None :
            self.affiche (screen, True)
            pygame.display.flip ()

        p = self.meilleure_piece (free, pos)
        # si p == None, on n'étudie pas les solutions avec les pièces placées
        # avant aux positions 1 à pos --> on n'entrera pas dans la boucle suivante
        while p != None :

            p.position     = pos
            angle          = self.angle_possible (p)

            # p est la pièce choisie, pour cette pièce
            # on passe en revue toutes les orientations
            for a in angle :
                p.orientation = a

                time.sleep (0.2)  # temporisation

                if self.nb_place () == 0 :  return True
                else :
                    r = self.solution (pos+1, screen = screen)
                    if r : return True

            p.position     = 0

            # on réactualise le tableau free qui aura été modifié par
            # l'appel à self.solution et on enlève le choix précédemment testé
            free2 = free
            free  = []
            for f in free2 :
                if f.numero != p.numero : free.append (f)

            # on passe au choix suivant avec free contenant les pièces
            # placées et les pièces essayées
            p = self.meilleure_piece (free, pos)


def attendre_clic ():
```

```
    """attend la pression d'un clic de souris"""
    reste = True
    while reste:
        for event in pygame.event.get():
            if event.type == pygame.MOUSEBUTTONUP :
                reste = False
                break

if __name__ == "__main__" :

    # taille de l'écran
    size = (750,750)

    # initialisation du module pygame
    pygame.init ()
    screen  = pygame.display.set_mode (size)

    # on définit le puzzle
    p = Puzzle ()

    # on affiche le puzzle avec print (format texte)
    print p

    # on affiche le puzzle à l'écran
    p.affiche (screen, petite = True)
    attendre_clic ()

    # on rafraîchit l'écran pour que le puzzle apparaissent
    pygame.display.flip ()

    # on trouve la solution
    r = p.solution (screen = screen)
    print "résolution ", r
    print "appelle à la méthode solution ", p.essai

    # on attend la pression d'un clic de souris avant de terminer le programme
    p.affiche (screen, petite = True)
    attendre_clic ()
```

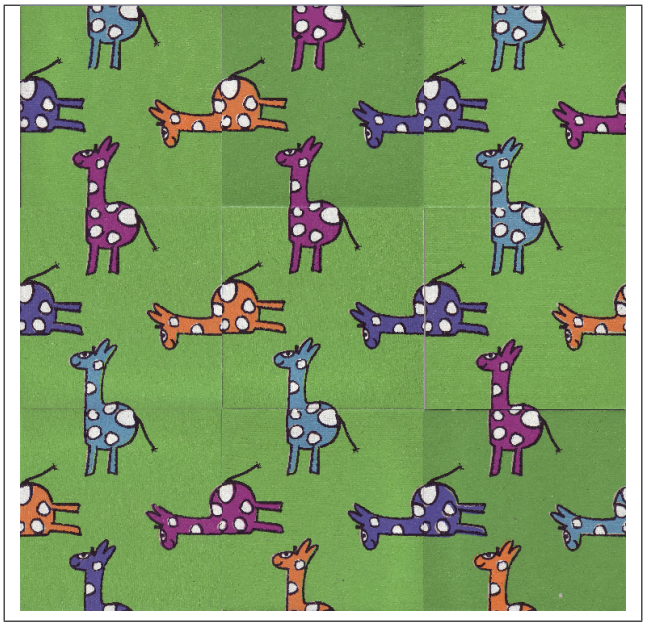


**Figure 22.8 :** *Le puzzle fini.*

## 22.3 Résolution d'une énigme

### 22.3.1 Enoncé

### 22.3.2 Correction

```
import copy

ttcouleur     = ["jaune", "bleu", "rouge", "blanc", "vert"]
ttnationalite = ["danois", "norvegien", "anglais", "allemand", "suedois"]
ttboisson     = ["eau", "the", "lait", "cafe", "biere"]
ttcigare      = ["Dunhill", "Blend", "Pall Mall", "Prince", "Bluemaster"]
ttanimal      = ["chats", "cheval", "oiseaux", "poisson", "chiens"]



ensemble      = [ ttcouleur, ttnationalite, ttboisson, ttcigare, ttanimal ]

def permutation (nb) :
    per = []
    p = [i for i in range (0,nb)]
    while p [0] < nb :
        next = False
        for i in range (1,nb) :
            if p [i] in p [0:i] :
                next = True
                break

        if not next :
            per.append (copy.copy (p))

        p [nb-1] += 1
        for j in range (nb-1, 0, -1) :
            if p [j] >= nb :
                p [j] = 0
                p [j-1] += 1

    return per

class Regle :
    def __init__ (self) :
        self.name = None
        self.set  = None

    def genere (self) :
        """genere l'ensemble des clauses possibles,
        listes de listes, (l [0][0] et l [0][1]) ou (l [1][0] et l [1][1]),
        une clause est un triplet (individu, (propriete, categorie) )"""
        return None

    def __str__ (self) :
        if self.name != None :
            if "clauses" not in self.__dict__ :
                s = self.name + " \t: "
                a = self.genere ()
                for al in a :
                    st = "\n       ou  " + str (al)
                    if len (st) > 260 : st = st [:260] + "..."
                    s += st
                    if len (s) > 1000 : break
                return s
            else :
                s = self.name + " \t: " + str (self.set)
                for al in self.clauses :
```

```
                    st = "\n       ou   " + str (al)
                    if len (st) > 260 : st = st [:260] + "..."
                    s += st
                    if len (s) > 1000 : break
                return s
        else :
            return "None"

    def combine (self, cl1, cl2) :
        """combine deux clauses, deux cas :
            1. rien en commun ou tout en commun --> concatenation des clauses
            2. une position ou une propriete en commun --> clause nulle"""
        # incompatibilite
        for p1 in cl1 :
            for p2 in cl2 :
                if p1 [1][0] == p2 [1][0] : # meme propriete
                    if p1 [0] != p2 [0] :  # mais positions differentes
                        return None
                if p1 [0] == p2 [0] : # meme individu
                    if p1 [1][1] == p2 [1][1] and p1 [1][0] != p2 [1][0] :
                        # meme categorie mais propriete differente
                        return None
        # compatibilite
        r = copy.deepcopy (cl1)
        for c in cl2 :
            if c not in r : r.append (c)
        return r

    def combine_croise (self, set1, set2) :
        """combines deux ensembles de clauses"""
        if len (set1) == 0 : return copy.deepcopy (set2)
        if len (set2) == 0 : return copy.deepcopy (set1)
        res = []
        for cl1 in set1 :
            for cl2 in set2 :
                r = self.combine (cl1, cl2)
                if r != None :
                    res.append (r)
        return res


class ReglePosition (Regle) :
    def __init__ (self, p1, pos) :  # p1 a la position
        self.set = [p1]
        self.name = "position"
        self.position = pos

    def genere (self) :
        return [ [ ( self.position, self.set [0]) ] ]

class RegleEquivalence (Regle) :
    def __init__ (self, p1, p2) :  # p1 equivalent a p2
        self.set = [p1, p2]
        self.name = "equivalence"

    def genere (self) :
        l = []
        for i in range (0,5) :
            l.append ( [ (i, self.set [0]), (i, self.set [1]) ] )
        return l

class RegleVoisin (Regle) :
    def __init__ (self, p1, p2) :  # p1 voisin de p2
        self.set = [p1, p2]
        self.name = "voisin"
```

```
    def genere (self) :
        l = []
        for i in range (0,4) :
            l.append ( [ (i, self.set [0]), (i+1, self.set [1]) ] )
            l.append ( [ (i+1, self.set [0]), (i, self.set [1]) ] )
        return l

class RegleAvant (Regle) :
    def __init__ (self, p1, p2) :  # p1 avant p2
        self.set = [p1, p2]
        self.name = "avant"

    def genere (self) :
        l = []
        for j in range (1,5) :
            for i in range (0,j) :
                l.append ( [ (i, self.set [0]), (j, self.set [1]) ] )
        return l

class RegleEnsemble (Regle) :
    def __init__ (self, set, categorie) : # permutation sur les elements d'une categorie
        self.set = [ (s,categorie) for s in set ]
        self.name = "ensemble"

    def genere (self) :
        l = []
        per = permutation (5)
        for p in per :
            tl = []
            for i in range (0,len (p)) :
                tl.append ( (i, self.set [p [i]]) )
            l.append (tl)
        return l


class Enigme :

    def __init__ (self, display = True) :
        self.regle = []

        self.regle.append ( ReglePosition (self.find ("lait") , 2) )
        self.regle.append ( ReglePosition (self.find ("norvegien"), 0) )

        self.regle.append ( RegleEquivalence (self.find ("Pall Mall"),  self.find ("oiseaux")) )
        self.regle.append ( RegleEquivalence (self.find ("anglais"),    self.find ("rouge")) )
        self.regle.append ( RegleEquivalence (self.find ("suedois"),    self.find ("chiens")) )
        self.regle.append ( RegleEquivalence (self.find ("danois"),     self.find ("the")) )
        self.regle.append ( RegleEquivalence (self.find ("vert"),       self.find ("cafe")) )
        self.regle.append ( RegleEquivalence (self.find ("jaune"),      self.find ("Dunhill")) )
        self.regle.append ( RegleEquivalence (self.find ("biere"),      self.find ("Bluemaster")) )
        self.regle.append ( RegleEquivalence (self.find ("allemand"),   self.find ("Prince")) )

        self.regle.append ( RegleVoisin (self.find ("Dunhill"),     self.find ("cheval")) )
        self.regle.append ( RegleVoisin (self.find ("norvegien"),   self.find ("bleu")) )
        self.regle.append ( RegleVoisin (self.find ("Blend"),       self.find ("eau")) )
        self.regle.append ( RegleVoisin (self.find ("Blend"),       self.find ("chats")) )

        self.regle.append ( RegleAvant (self.find ("vert"), self.find ("blanc")) )

        self.regle.append ( RegleEnsemble (ttcouleur, 0 ))
        self.regle.append ( RegleEnsemble (ttnationalite, 1) )
        self.regle.append ( RegleEnsemble (ttboisson, 2) )
        self.regle.append ( RegleEnsemble (ttcigare, 3 ))
        self.regle.append ( RegleEnsemble (ttanimal, 4 ))
```

```
    for r in self.regle :
        r.clauses = r.genere ()
        r.utilise = False

    self.count = 0

def find (self, p) :
    for i in range (0, len (ensemble)) :
        if p in ensemble [i] :
            return (p, i)
    stop
    return None

def __str__ (self) :
    if "solution" not in self.__dict__ or self.solution == None or len (self.solution) == 0 :
        if self.count > 0 : s = "solution impossible apres " + str (self.count) + " iterations \n"
        else : s = ""
        for r in self.regle :
            s += str (r) + "\n"
        return s
    else :
        sr = "solution, iteration " + str (self.count) + "\n"
        sr += self.display (self.solution)
        return sr

def display (self, solution) :
    sr = ""
    ind = [ [] for i in range (0,5) ]
    for s in solution :
        ind [ s [0] ].append ( s [1] )
    for i in ind :
        sr += "     " + str (i) + "\n"
    return sr

def resoud (self, solution = []) : # solution = [ ]) :
    """resolution, exploration en profondeur,
    fonction recursive"""

    self.count += 1
    if self.count % 10 == 0 :
        print self.count, " - proprietes en place : ", len (solution)-1

    if len (solution) == 25 :
        # c'est presque fini : on verifie que toutes les regles ont ete utilisees
        for r in self.regle :
            cl = r.combine_croise ([ solution ], r.clauses)
            if cl == None or len (cl) == 0 :
                # la solution est incompatible avec une regle
                return None
        self.solution = solution
        return solution

    # on cherche la regle celle qui engendre le moins de clauses
    # de facon a elargir la recherche le plus tard possible
    best = None
    rule = None

    for r in self.regle :

        cl = r.combine_croise ([ solution ], r.clauses)

        if cl == None :
            # la solution est incompatible avec une regle
            return None
```

```
            # on regarde si la regle r apporte quelque chose
            for c in cl :
                if len (c) > len (solution) :
                    break
            else :
                cl = None

            if cl != None and (best == None or len (best) > len (cl)) :
                best = cl
                rule = r

        if best == None :
            # regle incompatible avec la solution
            return None

        rule.utilise = True

        # on teste toutes les clauses
        for c in best :
            r = self.resoud (c)
            if r != None :
                # on a trouve
                return r

        rule.utilise = False # impossible
        return None


en = Enigme ()
print en
print "-----------------------------\n"
en.resoud ()
print "-----------------------------\n"
print en
```

### 22.3.3 Enoncé

Lors du premier tome de Harry Potter, les trois héros doivent résoudre une énigme - qui ne nécessite aucune magie - afin d'accéder à la salle où est cachée la pierre philosophale. Ce problème, dont l'auteur serait le professeur Rogue (Professor Snape pour les anglophones), consiste à trouver deux potions parmi les sept qui se trouvent devant eux : celles permettent d'avancer et de reculer. Ils sont aidés de quelques indices :

1. Il y a trois fioles de poison, deux fioles de vin d'ortie, une fiole permettant d'avancer et une fiole permettant de reculer.
2. Immédiatement à gauche de chacune des deux fioles de vin se trouve une fiole de poison.
3. Les fioles 1 et 7 ont des contenus différents ; ni l'une ni l'autre n'est la fiole qui permet d'avancer.
4. Ni la fiole la plus grande (fiole 6) ni la plus petite (fiole 3) ne contient du poison.
5. Les contenus des fioles 2 et 6 sont identiques.

**1)** Combien y a-t-il de configurations possibles sans tenir compte des indices ?

**2)** Construire un programme en s'inspirant de la troisième séance de cours qui permette de résoudre cette énigme ?

**3)** Y a-t-il trop d'indices ? Comment modifier votre programme pour répondre à cette question ?

**4)** On désire introduire un nouvel indice à l'énoncé de l'énigme, il sera choisi à partir de la solution construite avec le programme de la question 2 et sera du même type que le dernier des indices initiaux : les

contenus des fioles $i$ et $j$ sont identiques. Parmi toutes les solutions possibles, y en a-t-il une qui permettrait de se passer du second indice ?

### 22.3.4 Correction

Une première solution avec des fonctions.

```
def solution_correcte (sol) :
    """cette fonction recoit un tableau de 7 cases,
    chaque case contient un entier compris entre 0 et 3 inclus :
    0 : poison, 1 : vin, 2 : reculer, 3 : avancer
    la fonction determine si l'agencement propose dans sol
    verifie les cinq regles de l'enonce,
    retourne True si toutes les regles sont verifiees
    ou False si l'une des regles n'est pas verifiee,

    rappel : les indices vont de 0 a 6 inclus car il y a 7 cases"""

    # regle 1
    nb = [0,0,0,0]
    for s in sol :
        nb [s] += 1
    if nb [0] != 3 : return False # 3 poison
    if nb [1] != 2 : return False # 2 vin
    if nb [2] != 1 : return False # 1 reculer
    if nb [3] != 1 : return False # 1 avancer

    # regle 2
    for i in range (1,len (sol)) :
        if sol [i] == 1 and sol [i-1] != 0 : return False

    # regle 3
    if sol [0] == sol [6] : return False
    if sol [0] == 3 : return False
    if sol [6] == 3 : return False

    # regle 4
    if sol [2] == 0 : return False
    if sol [5] == 0 : return False

    # regle 5
    if sol [1] != sol [5] : return False

    # si on arrive ici, c'est que toutes les regles sont verifiees
    return True

def affiche_solution (sol) :
    a = ["poison", "vin", "reculer", "avancer"]
    for s in sol :
        print a [s], ", ",
    print " "


# premier idee : 7 boucles imbriqees
sol = [ 0,0,0,0,0,0,0]
for a in range (0,4) :
    sol [0] = a
    for b in range (0,4) :
        sol [1] = b
        for c in range (0,4) :
            sol [2] = c
```

```
            for d in range (0,4) :
                sol [3] = d
                for e in range (0,4) :
                    sol [4] = e
                    for f in range (0,4) :
                        sol [5] = f
                        for g in range (0,4) :
                            sol [6] = g
                            #print sol
                            r = solution_correcte (sol)
                            if r :
                                affiche_solution (sol)

# seconde idee : 2 boucles seulement pour
# parcourir toutes les solutions
sol = [ 0,0,0,0,0,0,0]
while sol [0] < 4 :
    r = solution_correcte (sol)
    if r :
        affiche_solution (sol)

    sol [6] += 1
    for i in range (len (sol)-1, 0, -1) :  # on parcourt les indices en
                                           # allant de 6 a 1 inclus
        if sol [i] >= 4 :
            sol [i]    = 0
            sol [i-1] += 1
```

Une seconde solution avec des classes, ce programme est de longueur équivalente au premier mais il est plus facilement plus adaptable à la résolution d'une autre énigme.

```
contenu_case = ["poison", "vin", "reculer", "avancer"]

class Case :
    def __init__ (self, contenu) :
        self.contenu = contenu_case.index (contenu)

    def __str__ (self) :
        return contenu_case [self.contenu]

class Regle1 :
    def correcte (self, cases) :
        nb = [0,0,0,0]
        for s in cases :
            nb [s.contenu] += 1
        if nb [0] != 3 : return False # 3 poison
        if nb [1] != 2 : return False # 2 vin
        if nb [2] != 1 : return False # 1 reculer
        if nb [3] != 1 : return False # 1 avancer
        return True

class Regle2 :
    def correcte (self, cases) :
        for i in range (1,len (cases)) :
            if cases [i].contenu == 1 and cases [i-1].contenu != 0 : return False
        return True

class Regle3 :
    def correcte (self, cases) :
        if cases [0].contenu == cases [6].contenu : return False
        if cases [0].contenu == 3 : return False
        if cases [6].contenu == 3 : return False
        return True
```

```
class Regle4 :
    def correcte (self, cases) :
        if cases [2].contenu == 0 : return False
        if cases [5].contenu == 0 : return False
        return True

class Regle5 :
    def correcte (self, cases) :
        if cases [1].contenu != cases [5].contenu : return False
        return True

class Enigme :
    def __init__ (self) :
        self.regle = [ Regle1 (), Regle2 (), Regle3 (), Regle4 (), Regle5 ()]
        self.cases = [ Case ("poison") for i in range (0,7) ]

    def __str__ (self) :
        for s in self.cases :
            print s, ", ",
        print ""

    def solution_correcte (self) :
        for r in self.regle :
            if not r.correcte (self.cases) :
                return False
        return True

    def resoud (self) :

        for c in self.cases :
            c.contenu = 0

        while self.cases [0].contenu < 4 :
            r = self.solution_correcte ()
            if r :
                print self.__str__ ()

            self.cases [6].contenu += 1
            for i in range (len (self.cases)-1, 0, -1) :  # on parcourt les indices en
                                                         # allant de 6 a 1 inclus
                if self.cases [i].contenu >= 4 :
                    self.cases [i].contenu    = 0
                    self.cases [i-1].contenu += 1


e = Enigme ()
e.resoud ()
```

**fin correction TD 22.3.3** □

## 22.4 Simulation informatique

### 22.4.1 Enoncé

Harmonie a décidé d'inviter ses amis autour d'une seule grande table. Pour que tout soit parfait, elle veut qu'un homme ait au moins une femme à ses côtés et qu'une femme ait au moins un homme à ses côtes.

Harmonie a beaucoup d'amis, ils viennent pour la plupart accompagnés. Pour ne pas se torturer l'esprit, elle choisit la méthode informatique pour vérifier que sa table respecte bien ses conditions.

**1)** Ecrire une fonction qui lit un fichier texte et place toutes ses lignes dans une liste.

**2)** Utiliser cette fonction pour récupérer les fichiers :

1. les prénoms masculins (homme.txt)
2. les prénoms féminins (femme.txt)
3. la configuration de la table (table.txt)

**3)** On veut écrire une fonction qui détermine si un prénom est masculin ou féminin à l'aide des listes extraites des fichiers "homme.txt", et "femme.txt". Cette fonction prendra donc trois paramètres, ces deux listes de prénoms et un prénom dont il faudra déterminer le genre. La fonction pourra par exemple retourner `True` si c'est un prénom féminin, `False` si c'est un prénom masculin.

**4)** Vérifier que la table de Harmonie suit les contraintes qu'elle s'est imposée. On précise que la table est ronde.

**5)** Un de ses amis l'a prévenue qu'il viendrait accompagné d'un ou une Dominique. Harmonie ne sait pas si il ou elle est un homme ou une femme. Que va-t-elle faire ? Harmonie décide que Dominique aura un homme et une femme à ses côtés et que qui que soit Dominique, ses voisins seront bien entourés. Y a-t-il une place qui conviendrait à Dominique ? Nous précisons que la table est toujours ronde.

**6)** Avec sa table ainsi conçue, combien de Dominique Harmonie pourrait-elle encore accepter sans avoir à tout recomposer ?

**7)** Si Harmonie a invité $n$ femmes, combien peut-elle inviter d'hommes au maximum sans déroger à ses règles ? Au minimum ?

**8)** Pour aller plus loin : admettons qu'il n'y ait $n$ Dominique autour de la table, quelle est la probabilité pour que la table suive les règles d'Harmonie ?

### 22.4.2 Correction

On revient ici sur les questions 7 et 8.

**7)**

S'il y a $n$ femmes autour de la table, il est au plus possible d'insérer deux hommes entre deux femmes, soit $2n$ hommes (FHH FHH FHH ...) . A l'inverse, il est au plus possible d'avoir deux femmes entre deux hommes (HFF HFF HFF ...), soit $\frac{n}{2}$ si $n$ est pair ou $\frac{n+1}{2}$ si $n$ est impair. Il ne faut oublier pour ce nombre minimum que la table est ronde.

**8)**

### 22.4.3 Comment observer un problème à l'aide d'un programme informatique

On suppose maintenant qu'on ne connaît plus la personne assise à chaque place de la table, on sait que la probabilité que ce soit une femme ou un homme est égale à $\frac{1}{2}$. On se propose dans un premier temps de répondre à la même question mais sans tenir compte du fait que la table est ronde, c'est-à-dire sans tenir compte des extrémités.

Le premier problème est de savoir s'il est plus facile de déterminer la probabilité que la table vérifie les conditions d'Harmonie ou celle qu'elle ne les vérifie pas, c'est-à-dire la probabilité de ne pas avoir ou d'avoir trois hommes ou trois femmes consécutifs. Afin de répondre à cette question, on écrit un programme informatique comptant le nombre de dispositions bonnes ou mauvaises en fonction de la taille de la table.

```
def trois_pareil (table) :
    """table est une suite booleens, on retourne True
```

```
    s'il n'y en a pas trois consecutifs"""
    for i in xrange (len (table)-2) :
        if table [i] == table [i+1] == table [i+2] :
            return False
    return True

def compte (n) :
    """on parcours toutes les tables possibles et on compte les
    bonnes configurations"""
    table = [ False for i in range (0,n) ]
    nb = 0
    bon = 0
    while table [0] != None :
        if n == 5 and trois_pareil (table) : print table
        nb += 1
        if trois_pareil (table) : bon += 1
        for i in xrange (len (table)-1, -1, -1) :
            if not table [i] :
                table [i] = True
                break
            else :
                table [i] = False
                if i == 0 : table [0] = None

    return nb, bon

for n in range (3, 15) :
    nb, bon = compte (n)
    print n, " nb = ", nb, "  bon = ", bon, " mauvais = ",
    print " proba = ",
    print float (bon) / float (nb), "          \t",
    print bon, "/", nb
```

Le programme permet d'obtenir les résultats suivants :

| $n$ | **dispositions** | **bonnes** | **mauvaises** |
|---|---|---|---|
| 3 | 8 | 6 | 2 |
| 4 | 16 | 10 | 6 |
| 5 | 32 | 16 | 16 |
| 6 | 64 | 26 | 38 |
| 7 | 128 | 42 | 86 |
| 8 | 256 | 68 | 188 |
| 9 | 512 | 110 | 402 |
| 10 | 1024 | 178 | 846 |
| 11 | 2048 | 288 | 1760 |
| 12 | 4096 | 466 | 3630 |
| 13 | 8192 | 754 | 7438 |
| 14 | 16384 | 1220 | 15164 |

La suite des mauvaises dispositions ne semble pas obéir à une règle particulière tandis que la suite des bonnes dispositions est une suite de Fibonacci. Si on note $u_n$ le nombre de bonnes dispositions, alors cette suite vérifie :

$$\left\{ \begin{array}{lcl} u_3 & = & 6 \\ u_4 & = & 10 \\ u_n & = & u_{n-1} + u_{n-2} \ \ \forall n \geqslant 5 \end{array} \right. \tag{22.5}$$

Il reste à comprendre pourquoi cette suite vérifie cette propriété. Comme l'indique la suite, on cherche un raisonnement par récurrence qui permettent de prolonger soit une table de longueur $n$, soit une table de

longueur $n+1$ pour obtenir une table de longueur $n+2$. Il faut prolonger deux tables en prenant soin qu'aucune configuration ne soit identique à une autre.

Si une table se termine par une femme, on est sûr que si on la continue avec un ou deux hommes, elle vérifiera toujours les contraintes. On applique la règle opposée si la table se termine par un homme.

| | | | | |
|---|---|---|---|---|
| table de longueur $n$ | ...F | H | H | il y en a $\frac{u_n}{2}$ |
| table de longueur $n$ | ...H | F | F | il y en a $\frac{u_n}{2}$ |
| table de longueur $n+1$ | | ...F | H | il y en a $\frac{u_{n+1}}{2}$ |
| table de longueur $n+1$ | | ...H | F | il y en a $\frac{u_{n+1}}{2}$ |

On vérifie bien qu'aucune disposition n'est identique à une autre. Il faut maintenant vérifier qu'on n'en a pas oublié [3]. Pour répondre à cette question, on retourne chacune des tables. On vérifie au préalable que pour tout $n$, $u_n$ est forcément pair puisque si une disposition est bonne, en inversant homme et femme, cette autre disposition est aussi bonne. Si on retourne les fins trouvées précédemment, on obtient des débuts de table :

| | | |
|---|---|---|
| H | H | F... |
| F | F | H... |
| H | F... | F ou H... |
| F | H... | F ou H... |

Il y a au total 6 débuts différents ce qui correspond exactement à $u_3$. On est donc sûr de n'avoir pas oublié de terminaison. La suite $(u_n)$ des bonnes dispositions vérifie donc bien :

$$\begin{cases} u_3 &= 6 \\ u_4 &= 10 \\ u_n &= u_{n-1} + u_{n-2} \quad \forall n \geqslant 5 \end{cases} \tag{22.6}$$

On peut aussi exprimer $u_n$ en fonction de $n$ seulement :

$$u_n = A\left(\frac{1-\sqrt{5}}{2}\right)^{n-3} + B\left(\frac{1+\sqrt{5}}{2}\right)^{n-3} \tag{22.7}$$

Avec :

$$\begin{cases} A+B &= 6 \\ A\left(\frac{1-\sqrt{5}}{2}\right) + B\left(\frac{1+\sqrt{5}}{2}\right) &= 10 \end{cases} \tag{22.8}$$

$$\Longrightarrow \begin{cases} A+B &= 6 \\ \sqrt{5}(-A+B) &= 14 \end{cases} \tag{22.9}$$

$$\Longrightarrow \begin{cases} A &= -1-3\sqrt{5} \\ B &= 3\sqrt{5}+7 \end{cases} \tag{22.10}$$

La probabilité $P_n$ qu'une table droite (pas ronde) vérifie les contraintes est donc :

$$P_n = \frac{-\left(1+3\sqrt{5}\right)\left(\frac{1-\sqrt{5}}{2}\right)^{n-3} + \left(7+3\sqrt{5}\right)\left(\frac{1+\sqrt{5}}{2}\right)^{n-3}}{2^n} \tag{22.11}$$

3. Apparemment oui, car la somme des dispositions trouvées est égale au résultat escompté.

Il reste à répondre à la même question lorsque la table est ronde. On modifie le programme de manière à compter les bonnes tables rondes (voir à la fin de cette section).

| $n$ | **dispositions** | **bonnes droites** | **bonnes rondes** |
|---|---|---|---|
| 3 | 8 | 6 | 6 |
| 4 | 16 | 10 | 6 |
| 5 | 32 | 16 | 10 |
| 6 | 64 | 26 | 20 |
| 7 | 128 | 42 | 28 |
| 8 | 256 | 68 | 46 |
| 9 | 512 | 110 | 78 |
| 10 | 1024 | 178 | 122 |
| 11 | 2048 | 288 | 198 |
| 12 | 4096 | 466 | 324 |
| 13 | 8192 | 754 | 520 |
| 14 | 16384 | 1220 | 842 |

Prolonger la table n'est plus aussi évident qu'avant. Il y a les bonnes configurations :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| table de longueur $n$ | F | ... | ... | F | H | H | il y en a $\alpha_n$ |
| table de longueur $n$ | H | ... | ... | H | F | F | il y en a $\alpha_n$ |
| table de longueur $n+1$ | H | F | ... | ... | F | H | il y en a $\beta_n$ |
| table de longueur $n+1$ | F | H | ... | ... | H | F | il y en a $\beta_n$ |
| table de longueur $n+1$ | F | H | ... | ... | F | H | il y en a $\gamma_n$ |
| table de longueur $n+1$ | H | F | ... | ... | H | F | il y en a $\gamma_n$ |
| table de longueur $n+1$ | F | F | ... | ... | F | H | il y en a $\delta_n$ |
| table de longueur $n+1$ | H | H | ... | ... | H | F | il y en a $\delta_n$ |

Et les mauvaises :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| table de longueur $n$ | H | ... | ... | F | H | H | il y en a $\eta_n$ |
| table de longueur $n$ | F | ... | ... | H | F | F | il y en a $\eta_n$ |
| table de longueur $n+1$ | H | H | ... | ... | F | H | il y en a $\zeta_n$ |
| table de longueur $n+1$ | F | F | ... | ... | H | F | il y en a $\zeta_n$ |

On pourrait chercher à estimer les nombres $\alpha_n$, $\beta_n$, ... Mais cette approche paraît plutôt fastidieuse, d'autant plus qu'on ne sait pas si on aura la chance de retomber sur une suite de Fibonacci ou une autre suite aussi simple. Peut-être vaut-il mieux aborder le problème selon un angle différent et avec des mathématiques différentes.

### 22.4.4 Chaînes de Markov

Les chaînes de Markov sont souvent utilisées pour étudier des séquences de nombres, des séquences d'ADN. On peut calculer la probabilité d'une lettre dans une séquence d'ADN. On peut aussi calculer la probabilité d'un couple de lettres consécutives, d'un triplet, ... En langue anglaise, c'est souvent l'association d'un verbe et de sa préposition qui a du sens et non l'un ou l'autre pris séparément.

On considère tout d'abord une table droite de sept personnes dont on ne retient que les genres des prénoms : HFFHHHF. Et au lieu de la représenter comme une succession de lettres, on la représente comme une succession de cinq triplets de lettres. Chaque triplet reprend les deux dernières lettres du triplet précédent et en ajoute une nouvelle :

```
HFF
 FFH
  FHH
   HHH
    HHF
```

Dans cette séquence de triplets, une table d'Harmonie ne doit pas contenir un seul triplet $HHH$ ou $FFF$. L'objectif est donc de calculer le nombre de tables ne contenant pas ces deux triplets. L'inconvénient maintenant est qu'il n'est pas possible d'avoir n'importe quel triplet après un autre : il faut que les deux dernières lettres du premier soient égales aux deux premières lettres du suivant. On note $t_1$ le premier triplet, $t_2$ le second triplet et $\mathbb{P}(t_2 \mid t_1)$ la probabilité que $t_2$ suivent $t_1$. Par exemple :

$$\begin{array}{ll} \mathbb{P}(t_2 = FFF \mid t_1 = FFH) = 0 & \mathbb{P}(t_2 = FFH \mid t_1 = FFH) = 0 \\ \mathbb{P}(t_2 = FHF \mid t_1 = FFH) = \frac{1}{2} & \mathbb{P}(t_2 = FHH \mid t_1 = FFH) = \frac{1}{2} \\ \mathbb{P}(t_2 = HFF \mid t_1 = FFH) = 0 & \mathbb{P}(t_2 = FHH \mid t_1 = FFH) = 0 \\ \mathbb{P}(t_2 = HHF \mid t_1 = FFH) = 0 & \mathbb{P}(t_2 = HHH \mid t_1 = FFH) = 0 \end{array}$$

Ce tableau montre qu'il n'y a que deux transitions possibles entre deux triplets consécutifs car ils doivent avoir deux lettres en commun. De plus, ces probabilités sont les mêmes qu'on soit au milieu de la table ou au début : elles ne dépendent pas de la position $n$. On les notera dans la suite $\mathbb{P}(t_{n+1} \mid t_n)$.

On construit ensuite l'ensemble $E = \{e_i \mid 1 \leqslant i \leqslant 8\}$ où :

$$\begin{array}{ll} e_1 = HHH & e_2 = HHF \\ e_3 = HFH & e_4 = HFF \\ e_5 = FHH & e_6 = FHF \\ e_7 = FFH & e_8 = FFF \end{array}$$

On construit aussi la matrice $M = \left(\mathbb{P}(t_{n+1} = e_j \mid t_{n+1} = e_i)\right)_{\substack{1 \leqslant i \leqslant 8 \\ 1 \leqslant j \leqslant 8}}$, on obtient [4] :

$$M = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & . & . & . & . & . & . \\ . & . & \frac{1}{2} & \frac{1}{2} & . & . & . & . \\ . & . & . & . & \frac{1}{2} & \frac{1}{2} & . & . \\ . & . & . & . & . & . & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & . & . & . & . & . & . \\ . & . & \frac{1}{2} & \frac{1}{2} & . & . & . & . \\ . & . & . & . & \frac{1}{2} & \frac{1}{2} & . & . \\ . & . & . & . & . & . & \frac{1}{2} & \frac{1}{2} \end{pmatrix} \tag{22.12}$$

Tout l'intérêt de cette modélisation tient dans le calcul des puissances de la matrice $M$. A quoi correspond $M^2$ ? Par exemple, on note $m_{ij}^d$ le coefficient $ij$ de la matrice $M^d$, alors :

4. Afin de rendre la lecture plus facile, les "." signifie 0.

$$
\begin{aligned}
m^2_{ij} &= \sum_{k=1}^{8} m^1_{ik}\, m^1_{kj} \\
&= \sum_{k=1}^{8} \mathbb{P}\left(t_{n+1} = e_k \mid t_n = e_i\right)\ \mathbb{P}\left(t_{n+1} = e_j \mid t_n = e_k\right) \\
&= \sum_{k=1}^{8} \mathbb{P}\left(t_{n+2} = e_j \mid t_{n+1} = e_k\right)\ \mathbb{P}\left(t_{n+1} = e_k \mid t_n = e_i\right) \\
&= \mathbb{P}\left(t_{n+2} = e_j \mid t_n = e_i\right)
\end{aligned}
\tag{22.13}
$$

La matrice $M^2$ contient les probabilités d'apparition non pas du triplet suivant mais du second triplet suivant. De la même manière, on démontre que $m^d_{ij}$ est la probabilité d'apparition du triplet $e_j$ à la position $n+d$ sachant que le triplet à la position $n$ est $e_i$.

Est ce que cela nous permet de résoudre notre problème ? Pas encore, il reste à écrire d'une manière ou d'une autre qu'on ne doit pas rencontrer à aucune position le triplet $e_1 = HHH$ ou $e_8 = FFF$. Tous les triplets de la table doivent être différents de $e_1$ et $e_8$, autrement dit, on doit aller du premier triplet au dernier triplet sans passer par $e_1$ et $e_8$. Imaginons que si jamais, on passe par un des ces deux triplets, on reste bloqué, comme si les triplets $e_1$ et $e_8$ étaient des culs-de-sac : une fois qu'on passe par le triplet $e_1$, on y reste. La probabilité de rester dans ce triplet est égale à 1. Ceci aboutit à la matrice de transition $M'$ suivante :

$$
M' = \begin{pmatrix}
1 & . & . & . & . & . & . & . \\
. & . & \frac{1}{2} & \frac{1}{2} & . & . & . & . \\
. & . & . & . & \frac{1}{2} & \frac{1}{2} & . & . \\
. & . & . & . & . & . & \frac{1}{2} & \frac{1}{2} \\
\frac{1}{2} & \frac{1}{2} & . & . & . & . & . & . \\
. & . & \frac{1}{2} & \frac{1}{2} & . & . & . & . \\
. & . & . & . & \frac{1}{2} & \frac{1}{2} & . & . \\
. & . & . & . & . & . & . & 1
\end{pmatrix}
\tag{22.14}
$$

De la même manière que pour $M^d$, $M'^d$ contient les probabilités d'apparition du triplet $d$ connaissant le triplet de départ. Et si ce dernier triplet est différent de $e_1$ et $e_8$, alors il est impossible qu'on y soit passé puisque dans ce cas, on y serait resté jusqu'à la fin. Tous les triplets acceptables pour une table d'Harmonie sont dans l'ensemble $E' = \{e_2, ..., e_7\}$. Toutes les tables droites de longueur $n$ vérifiant les contraintes d'Harmonie partent d'un triplet de $E'$ pour arriver dans $E'$. La probabilité d'une table d'Harmonie de longueur $n$ constituée des triplets $\{t_1, \ldots, t_{n-2}\}$ s'exprime comme suit :

$$
\begin{aligned}
P_n &= \mathbb{P}\left(t_1 \in E'\right) \mathbb{P}\left(t_{n-2} \in E' \mid t_1 \in E'\right) \\
&= \sum_{i=2}^{7} \mathbb{P}\left(t_1 = e_i\right) \left[ \sum_{j=2}^{7} \mathbb{P}\left(t_{n-2} = e_j \mid t_1 = e_i\right) \right] \\
&= \sum_{i=2}^{7} \frac{1}{8} \left[ \sum_{j=2}^{7} m'^{n-3}_{ij} \right] = \frac{1}{8} \sum_{i=2}^{7} \sum_{j=2}^{7} m'^{n-2}_{ij}
\end{aligned}
\tag{22.15}
$$

Il ne reste plus qu'à en déduire la probabilité d'une table d'Harmonie ronde en introduisant la probabilité de passer du dernier triplet au premier. Par exemple, pour la séquence HFFHHFF. On avait construit pour une table droite la séquence :

```
HFF
 FFH
  FHH
   HHF
    HFF
```

Mais pour une table ronde, il faut ajouter le fait de revenir au début tout en respectant les règles de passages entre deux triplets : les deux dernières d'un triplet sont les deux premières du triplet suivant. On est alors obligé d'ajouter deux triplets qui permettent de faire la liaison entre le début et la fin.

```
HFF
 FFH
  FHH
   HHF
    HFF
     FFH      ajouté
      FHF     ajouté
```

Ensuite seulement, il est possible de passer du dernier triplet au premier, ce qui donne une probabilité $P^r$ d'avoir une table ronde de longueur $n$ respectant les critères d'Harmonie :

$$P_n^r = \sum_{i=2}^{7}\sum_{j=2}^{7} m_{ij}^{\prime n-1} m'_{ji} \tag{22.16}$$

Dans cette expression (22.16), par rapport la probabilité (22.15), le terme $\mathbb{P}(t_1 = e_i)$ a disparu puisqu'il n'y a plus de début de la table, le premier triplet dépend du dernier, ce que dit le terme $m'_{ji}$. De même, comme la séquence contient deux triplets de plus, on utilise les coefficients de la matrice $M'^{n-1}$.

L'expression obtenue n'est pas aussi aisément calculable que celle (22.11) obtenue à l'aide des suites de Fibonacci mais pour une table droite. Il faudrait pour cela obtenir explicitement la forme de la matrice $M'^n$ pour tout $n$. On pourrait pour cela chercher une récurrence entre les coefficients ou alors appliquer une méthode plus systématique comme la diagonalisation de la matrice $M'$.

### 22.4.5 Le programme

```
##############################################################################
#                              questions 1 a 5
##############################################################################

def lit_fichier (nom) :
    """cette fonction lit le contenu d'un fichier
    et retourne une liste contenant ses lignes"""
    f = open (nom, "r")
    li = f.readlines ()
    f.close ()
    r =[]
    for l in li :
        s = l.replace ("\n", "")
        s = s.replace ("\r", "")
        r.append (s)
    return r

def genre (prenom, femme, homme) :
    """retourne le genre d'un prenom"""
    if prenom in femme : return True
    if prenom in homme : return False
```

```
    return None

def contrainte (table, femme, homme) :
    """cette fonction verifie que la table
    obeit aux contrainte d'Harmonie :
    il ne peut y avoir trois personnes voisines du meme sexe
    qui se suivent"""
    ge = [ genre (t,femme, homme) for t in table ]
    for i in range (0,len (ge)) :
        if ge [i] == ge [ (i+1) % len (ge) ] == ge [(i+2) % len (ge) ] :
            return False
    return True

def dominique (table, femme, homme) :
    """cette fonction dit si on peut placer dominique,
    deux schemas de figures possibles :
      1-  H F dominique H F
      2-  F H dominique F H
    """
    ge = [ genre (t,femme, homme) for t in table ]
    for i in range (0,len (ge)) :
        if  ge [i]                != ge [ (i+1) % len (ge) ] and \
            ge [(i+2) % len (ge)] != ge [ (i+3) % len (ge) ] and \
            ge [i]                == ge [ (i+2) % len (ge) ] :
            return True
    return False

femme = lit_fichier ("femme.txt")
homme = lit_fichier ("homme.txt")
table = lit_fichier ("table.txt")
r = contrainte (table, femme, homme)
print "la table suit les desirs d'Harmonie : ", r
r = dominique (table, femme, homme)
print "peut-on ajouter Dominique : ", r


#############################################################################
#                             question 6
#############################################################################

def contrainte_dominique (table, femme, homme) :
    """meme fonction que contrainte mais on tient aussi du fait que
    le genre peut etre indetermine (None), toute configuration
    incertaine est eliminee"""
    ge = [ genre (t,femme, homme) for t in table ]
    for i in range (0,len (ge)) :
        if ge [i] == ge [ (i+1) % len (ge) ] == ge [(i+2) % len (ge) ] :
            return False
        if ge [i] == None and ge [ (i+1) % len (ge) ] == ge [(i+2) % len (ge) ] :
            return False
        if ge [i] ==  ge [ (i+1) % len (ge) ] and ge [(i+2) % len (ge) ] == None :
            return False
        if ge [i] ==  ge [ (i+2) % len (ge) ] and ge [(i+1) % len (ge) ] == None :
            return False
        if ge [i] == None and ge [ (i+1) % len (ge) ] == None :
            return False
        if ge [i] == None and ge [ (i+2) % len (ge) ] == None :
            return False
        if ge [ (i+2) % len (ge) ] == None and ge [ (i+1) % len (ge) ] == None :
            return False
    return True

def place_dominique (table, femme, homme) :
    """on cherche a inserer autant de dominique que possible"""
    nb = len (table)
```

```
    fin = False
    while not fin :
        fin = True
        for i in range (0, len (table)) :
            table.insert (i, "DOMINIQUE")
            r = contrainte_dominique (table, femme, homme)
            if r :
                fin = False
                break
            else :
                del table [i]
    return len (table) - nb

print "----------------------------------------------------------------"
nb = place_dominique (table, femme, homme)
print "nombre de dominique ", nb
for p in table :
    print p


################################################################################
#                               question 7
################################################################################
#
#  nombre minimal : partie entiere de (n+1)/2
#  nombre maximal : 2n
#
################################################################################
#                               question 8
################################################################################

def trois_pareil (table) :
    """table est une suite booleens, on retourne True
    s'il n'y en a pas trois consecutifs"""
    for i in xrange (len (table)-2) :
        if table [i] == table [i+1] == table [i+2] :
            return False
    return True

def trois_pareil_rond (table) :
    """table est une suite booleens, on retourne True
    s'il n'y en a pas trois consecutifs"""
    for i in xrange (len (table)) :
        if table [i] == table [ (i+1) % len (table)] == table [(i+2) % len (table)] :
            return False
    return True

def compte (n) :
    """on parcours toutes les tables possibles et on compte les
    bonnes configurations"""
    table = [ False for i in range (0,n) ]
    nb = 0
    bon = 0
    while table [0] != None :
        nb += 1
        if trois_pareil_rond (table) : bon += 1
        for i in xrange (len (table)-1, -1, -1) :
            if not table [i] :
                table [i] = True
                break
            else :
                table [i] = False
                if i == 0 : table [0] = None
    return nb, bon
```

```
print "-------------------------------------------------------------------"
print "calcul en parcourant toutes les tables possibles"
print "-------------------------------------------------------------------"
for n in range (3, 15) :
    nb, bon = compte (n)
    print n, " nb = ", nb, "  bon = ", bon,
    print " proba = ", float (bon) / float (nb), "          \t",
    print bon, "/", nb

##############################################################################
#                              question 8
#                                 markov
##############################################################################
# module pour le calcul matriciel
import scipy_base as SCI
import copy


def construit_matrice () :
    """construit la matrice de transition,
    le vecteur des probabilites d'entrees dans chaque etat"""
    # construction des etats
    etat = []
    for i in range (0,2) :
        for j in range (0,2) :
            for k in range (0,2) :
                etat.append ( (i,j,k) )
    # construction de la matrice
    mat = []
    for i in range (0, len (etat)) :
        l = []
        for j in range (0, len (etat)) :
            if etat [i] [1:3] == etat [j] [0:2] :
                # (i,j,k) --> (j,k,0) ou (j,k,1)
                l.append (1)
            else : l.append (0)
        # on renormalise sur chaque ligne
        s = sum (l)
        for j in range (0, len (etat)) : l[j] /= float (s)
        mat.append (l)

    entree = [ 0.125 for i in range (0,8) ]
    return etat, SCI.mat (mat), SCI.mat (entree)

def calcul_probabilite_droite (n, etat, mat_, entree) :
    """calcule les probabilites pour une table droite"""
    pos     = [ etat.index ( (0,0,0) ), etat.index ( (1,1,1) ) ]
    temp    = copy.deepcopy (entree)
    mat     = copy.deepcopy (mat_)
    for p in pos :
        for i in range (0,8) :
            mat [ (p,i) ] = 0.0
        mat [ (p,p) ] = 1.0

    m = mat ** (n-1)

    sum = 0.0
    for i in range (0,8) :
        for j in range (0,8) :
            if i not in pos and j not in pos :
                sum += m [ (i,j) ]  * mat [ (j,i) ]
    return sum
```

```
print "-----------------------------------------------------------------"
print "calcul avec les chaines de Markov"
print "-----------------------------------------------------------------"
etat,mat,entree = construit_matrice ()

for n in range (3,15) :
    proba = calcul_probabilite_droite (n, etat, mat, entree)
    print n, "  bon ", proba
```

**fin correction TD 22.4.1** □

# Chapitre 23

# Mathématique, recherche opérationnelle

---

En dix ans, Internet est devenu un outil quotidien. On y trouve une telle multitude de programmes au point de penser que la réponse que l'on cherche y est forcément accessible. Quelqu'un a plus ou moins déjà fait ce que vous avez à faire. Il suffit juste de savoir les mots-clé qui permettront de le retrouver. Lorsqu'un livreur de meubles doit organiser sa tournée, il résoud un problème un problème complexe qui est celui de relier un certain nombre de destinations le plus rapidement possible. Il minimise la distance reliant toutes ces destinations. Ce problème, simple en apparence, est un des plus compliqués car il n'existe pas d'algorithmes qui retourne la solution optimale en un temps raisonnable et il est beaucoup mieux connu sous le nom du problème du voyageur de commerce [1].

Les sujets abordés dans les paragraphes qui suivent ne sont pas indispensables à la formation d'un ingénieur mais ils concourrent à sa culture scientifique. La conception d'un programme revient souvent à casser un problème complexe en une multitude de problèmes simples. Les problèmes simples sont d'abord des sommes, des calculs de maxima, des tris. Ils deviennent ensuite ceux exposés dans la suite.

## 23.1 Plus court chemin dans un graphe

### 23.1.1 Enoncé

La figure 23.1 représente un mini réseau routier. Chaque nœud désigne une ville tandis que chaque arc dissimule une route reliant deux villes ainsi que sa longueur. Comment définir une méthode qui permette dans tous les cas de déterminer le chemin entre deux nœuds d'un graphe ?

On définit un graphe $G$ par le couple $G = (X, E)$ où $X$ est l'ensemble des sommets du graphe, et $E$ celui de ses arêtes. Appliqué au graphe de la figure 23.1, cela donne :

$$\begin{array}{rl} X = & \{0, 1, ..., 14\} \\ E = & \{(0,1),(0,2),(0,8),\ (1,2),(1,3),\ (2,3),\ (3,7),(3,8),(3,10),\ (4,5),(4,12),(4,13) \\ & (5,14),\ (6,10),(6,11),(6,13),(6,14),\ (7,8),(7,10),(7,11),\ (8,9) \\ & (9,12),\ (11,12),(11,14)\} \end{array} \tag{23.1}$$

Les arcs sont souvent pondérés comme ici puisque à chaque arc est associée la distance séparant deux nœuds ou plus généralement la valeur d'une fonction $f : E \longrightarrow \mathbb{R}$. Il s'agit de déterminer le chemin le plus court ou celui de poids minimum, c'est-à-dire que la somme des valeurs de la fonction $f$ associées à chaque arc du chemin doit être minimum.

Le graphe de la figure 23.1 est supposé non-orienté puisque chaque route qui relie deux villes peut être parcourue dans un sens ou dans l'autre. Un graphe orienté n'autorise le parcours d'un arc que dans un sens, comme les rues en sens interdit dans une ville. Les arcs orientés sont représentés graphiquement par des flèches, qui sont absentes du graphe de la figure 23.1 puisque le graphe est non-orienté.

L'ensemble $E$ (23.1) est décrit par une suite de couples de sommets : $E \subset X^2$. Il existe toutefois d'autres manières de représenter l'ensemble des arcs. La première consiste à associer à chaque sommet de $X$ l'ensemble de ses successeurs :

1. *Travelling Salesman Problem* en anglais

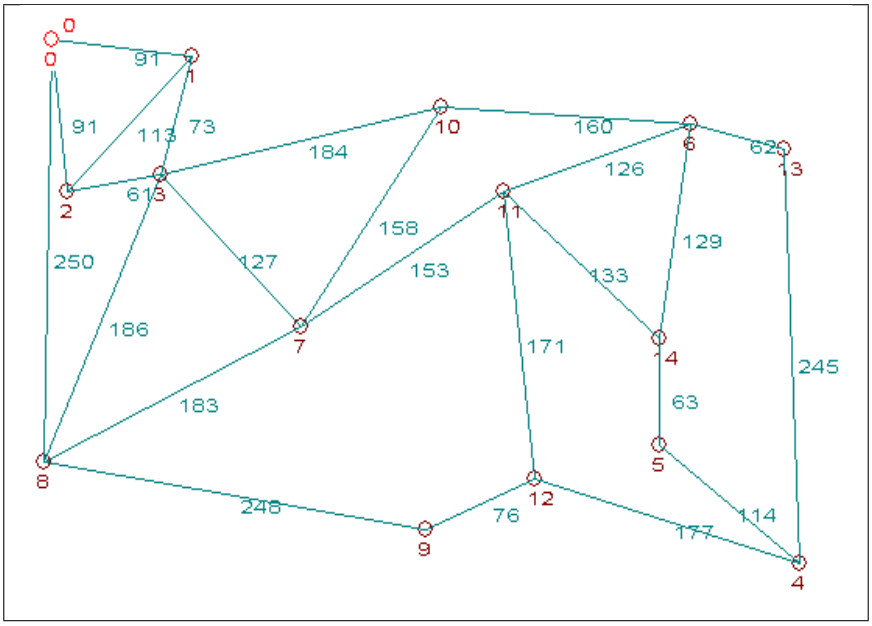



**Figure 23.1 :** *Chaque nœud de ce graphe symbolise une ville tandis que chaque arc symbolise une route et sa distance. On cherche à savoir quelle est le chemin de plus court pour relier les villes numéro 0 et numéro 4.*

$$S : x \in X \longrightarrow S(x) = \{y \in X \mid (x, y) \in E\}$$

Par exemple, pour le sommet 7 de la figure 23.1, $S(7) = 3, 8, 10, 11$. Cette représentation est utilisée dans le TD 4 et l'est généralement lorsque les graphes à manipuler contiennent peu d'arcs. Une autre façon de représenter l'ensemble $E$ est une matrice appelée *matrice d'adjacence* notée $A$. Il suffit pour cela d'indexer les sommets de $X$ par des entiers :

$$A = (a_{ij})_{ij}$$
$$\text{avec } \forall i, j,\ a_{ij} = \begin{cases} 1 & \text{si } (i, j) \in E \\ 0 & \text{sinon} \end{cases}$$

Cette matrice est souvent utilisée pour de petits graphes tels que ceux employés dans cet exercice. Elle permet également d'associer facilement un poids à chaque arc :

$$a_{ij} = \begin{cases} \text{poids de l'arc } i \longrightarrow j & \text{si } (i, j) \in E \\ 0 \text{ ou } \infty & \text{sinon} \end{cases}$$

La valeur associée à un arc inexistant ($\notin E$) dépend du problème à résoudre. Dans le cas de la recherche du plus court chemin, on prendra comme valeur $\infty$. Tout chemin passant par un arc inexistant est de longueur infinie. Dans le cas de la recherche du plus long chemin entre deux nœuds, les arcs inexistant auront un poids nul. La matrice d'adjacence possède quelques propriétés intéressantes :

1. Si elle est symétrique, $A = A'$, alors le graphe est non-orienté.
2. Si les arcs inexistants sont pondérés par la valeur 0 et les arcs existants par des valeurs strictement positives, si $(A^n)_{ij} > 0$, alors il existe un chemin de longueur $n$ entre les nœuds $i$ et $j$.

Cette seconde propriété se démontre par récurrence. Pour $n = 2$, on a $(A^2)_{ij} = \sum_k a_{ik}a_{kj}$. Par conséquent, si $(A^2)_{ij} > 0$ alors il existe un sommet $k$ tel que les arêtes $(i, k)$ et $(k, j)$ existent. Il est facile de continuer la démonstration pour $n > 2$.

Les matrices d'adjacence sont peu adaptées pour des graphes ayant un grand nombre de sommets et un petit nombre d'arêtes. D'un point de vue informatique, beaucoup de mémoire est utilisée pour stocker une information inutile puisque la matrice est presque uniforme sauf pour un petit nombre d'arêtes. Presque tous ces éléments sont nuls ou égaux à $\infty$ selon la convention choisie. D'un point de vue informatique, la matrice d'adjacence est rarement une représentation optimale mais c'est pour sa simplicité qu'elle sera utilisée dans ce programme d'exemple chargé de déterminer le plus court chemin dans un graphe.

Afin de se concentrer sur l'algorithme de recherche du plus court chemin, une partie du programme est déjà développée. Elle contient différentes fonctions permettant de constuire aléatoirement un certain nombre de villes et de routes puis d'afficher le résultat dans une image. Il ne reste plus qu'à lui ajouter la fonction qui détermine le plus court chemin entre deux villes. Ce programme utilise le type `Matrix` fourni avec les sous-modules `Matrix` et `UserArray` du module `Numeric`. Mis à part les deux lignes qui servent à créer une matrice d'adjacence carrée et nulle de taille $nb$, pour comprendre le programme, il suffit de savoir qu'un élément d'indice $(i, j)$ de la matrice d'adjacence `m` est désigné par `m[i,j]`.

```
# -*- coding: cp1252 -*-
import random                 # pour tirer aléatoirement des nombres
import Matrix as mat          # pour les matrices
import UserArray as ua        # pour les matrices
import math                   # fonction sqrt
import PIL.Image as Im        # pour les images
import PIL.ImageDraw as Id    # pour dessiner

def construit_ville(n, x =500, y = 500):
    """tire aléatoirement n villes dans un carrée x * y"""
    l = []
    for i in range(0,n):
        xx = x * random.random ()
        yy = y * random.random ()
        l.append ((xx,yy))
    return l

def distance_ville (l,i,j):
    """calcule la distance entre deux villes i et j de la liste l"""
    x = l [i][0] - l [j][0]
    y = l [i][1] - l [j][1]
    return math.sqrt (float (x*x+y*y))

def construit_arete (l,part = 0.15):
    """tire aléatoirement part * len (l) arêtes et construit la matrice
    d'adjacence"""
    nb  = len (l)
    m   = mat.Matrix ( [ 0 for i in range(0,nb) ]) # crée un vecteur de nb zéros
    m   = ua.transpose (m) * m                     # effectue une multiplication du vecteur
                                                   # précédent avec son vecteur transposé
                                                   # pour obtenir une matrice carrée nulle
    are = int (part * nb * nb)
    while are > 0:
        i = random.randint (0,nb-1)
        j = random.randint (0,nb-1)
        if m [i,j] > 0: continue               # si l'arête existe déjà, on recommence
        m [i,j] = int (distance_ville (l,i,j)) # on affecte comme poids à l'arête
                                               # la distance entre les deux villes
        are -= 1
    return m

def dessin_ville_arete (l,m,chemin):
    """dessine la ville et les routes dans une image"""
    mx, my = 0,0
```

```
    for i in l:
        mx = max (mx, i [0])
        my = max (my, i [1])
    mx += 25
    my += 25
    mx, my = int (mx), int (my)
    im = Im.new ("RGB", (mx, my), (255,255,255)) # création d'une image blanche
    draw = Id.Draw(im)

    # dessin des villes
    for i in l:
        j  = (int (i [0]), int (i[1]))
        j2 = (j [0] + 10, j [1] + 10)
        draw.ellipse ((j,j2), fill = (0,0,0))

    # dessin des arêtes
    for i in range (0,len(l)):
        for j in range (0,len(l)):
            if m [i,j] > 0:
                a = (int (l [i][0]+5), int (l [i][1]+5))
                b = (int (l [j][0]+5), int (l [j][1]+5))
                draw.line ((a,b),fill=(255,0,0))

    # dessin des villes de départ et d'arrivée
    v1 = chemin [0]
    v2 = chemin [ len (chemin)-1]
    a = (int (l [v1][0]), int (l [v1][1]))
    b = (int (l [v1][0]+10), int (l [v1][1]+10))
    draw.ellipse ((a,b), fill = (0,255,0))
    a = (int (l [v2][0]), int (l [v2][1]))
    b = (int (l [v2][0]+10), int (l [v2][1]+10))
    draw.ellipse ((a,b), fill = (0,255,0))

    # dessin du chemin
    for c in range (0,len(chemin)-1):
        i = chemin [c]
        j = chemin [c+1]
        print i,j
        if m [i,j] > 0:
            a = (int (l [i][0]+5), int (l [i][1]+5))
            b = (int (l [j][0]+5), int (l [j][1]+5))
            draw.line ((a,b),fill=(0,0,255))
        else:
            a = (int (l [i][0]+5), int (l [i][1]+5))
            b = (int (l [j][0]+5), int (l [j][1]+5))
            draw.line ((a,b),fill=(0,0,50))

    # on retourne l'image
    return im

# programme principal
# construction des villes
l = construit_ville (10)
print l

# construction des arêtes
m = construit_arete (l)
print m

# choix de la ville de départ de d'arrivée
a,b = 0,0
while a == b:
    a = random.randint (0,len(l)-1)
    b = random.randint (0,len(l)-1)
```

```
print "ville de départ et d'arrivée : ",a,b

# construction de l'image du résultat
im = dessin_ville_arete(l,m,[a,b])
im.show ()  # on affiche l'image
```

### 23.1.2 Correction

### 23.1.3 Théorie des graphes, un peu d'histoire

Cette brève histoire de la théorie des graphes est extraite du site *http ://www.sciences.ch/*.

L'histoire de la théorie des graphes (ou des "complexes cellulaires") débute peut-être avec les travaux d'Euler au 18e siècle et trouve son origine dans l'étude de certains problèmes, tels que celui des ponts de Königsberg - les habitants de Königsberg se demandaient s'il était possible, en partant d'un quartier quelconque de la ville, de traverser tous les ponts sans passer deux fois par le même et de revenir à leur point de départ, voir figures 23.2 et 23.3 -, la marche du cavalier sur l'échiquier ou le problème de coloriage de cartes.

La théorie des graphes s'est alors développée dans diverses disciplines telles que la chimie (isomères), la biologie, les sciences sociales (réseaux de transports), gestion de projets (C.P.M.), informatique (topologie des réseaux), etc.. Depuis le début du 20e siècle, elle constitue une branche à part entière des mathématiques, grâce aux travaux de König, Menger, Cayley puis de Berge et d'Erdös.

De manière générale, un graphe permet de représenter la structure, les connexions d'un ensemble complexe en exprimant les relations entre ses éléments : réseau de communication, réseaux routiers, interaction de diverses espèces animales, circuits électriques, ... Les graphes constituent donc une méthode de pensée qui permet de modéliser une grande variété de problèmes en se ramenant à l'étude de sommets et d'arcs.

Les derniers travaux en théorie des graphes sont souvent effectués par des informaticiens, du fait de l'importance qu'y revêt l'aspect algorithmique. La théorie des graphes connaît un assez grand engouement ces trente dernières, peut-être est-ce parce qu'elle ne nécessite pas dans ses concepts élémentaires de bagage mathématique considérable.

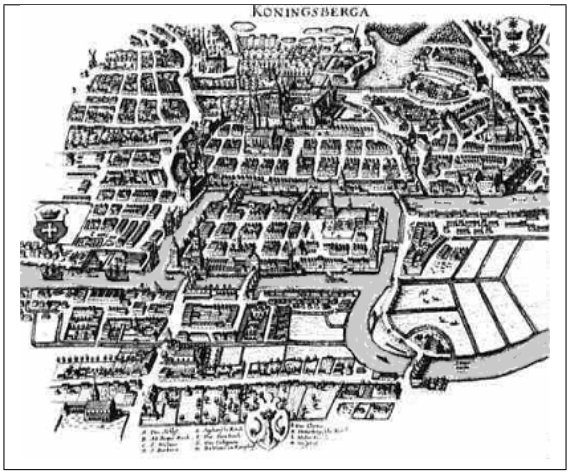



**Figure 23.2 :** *Les sept ponts de Könisberg, existe-t-il un chemin qui permettent revenir à son point de départ en ne parcourant chacun des sept ponts qu'une seule fois ?*

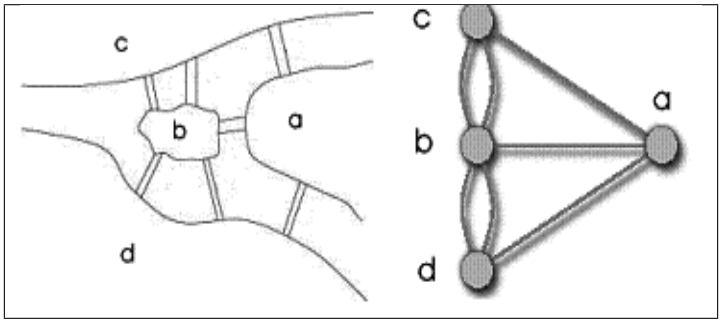



**Figure 23.3 :** *Les sept ponts de Könisberg de la figure 23.2 et un graphe qui les représente. Chaque arc correspond à un pont. Chaque nœud correspond à un quartier. Il est impossible de revenir au nœud de départ en ne parcourant toutes les arêtes qu'une seule fois. Pour que cela fût possible, il eut fallu que ce graphe ne comportât au plus que deux nœuds reliés à un nombre impair d'arcs.*

### 23.1.4 Principe d'optimalité

Déterminer le plus court chemin dans un graphe fait partie d'une classe de problème résolus par ce qu'on appelle la programmation dynamique qui découle elle-même du principe d'optimalité défini dans l'œuvre du mathématicien Pierre de Fermat. Le principe de Fermat contient les postulats fondamentaux de l'optique géométrique et permet de retrouver le principe de propagation rectiligne, le principe de retour inverse et les lois de Snell-Descartes sur la réfraction.

> Le trajet réellement suivi par la lumière pour se rendre d'un point A à un point B, correspond à un temps de parcours minimum ou, si l'on tient compte de la nature des différents milieux traversés, à un chemin optique minimum.

En d'autres termes, appliqué à la recherche du plus court chemin dans un graphe comme celui de la figure 23.4, ce principe peut être interprété selon une récurrence de la forme :

> Toute partie d'un chemin optimal est elle-même optimale : si le trajet le plus court pour se rendre d'un point A à un point C passe par le point B, alors ce trajet est aussi le plus court pour aller de A à B et de B à C.

### 23.1.5 Plus court chemin dans un graphe

On désigne par $G = (X, S)$ un graphe dont l'ensemble des sommets est $X$ et l'ensemble des arêtes est $S$. On note également $d(i, j)$ le poids associé à l'arc $(i, j) \in S$. Pour le graphe de la figure 23.4, $d(i, j)$ est la longueur de la route reliant les villes $i$ et $j$ ou plus simplement la distance à vol d'oiseau. La fonction $d$ est symétrique si seulement le graphe est non-orienté, comme celui de la figure 23.4.

A partir du principe d'optimalité, il est possible de construire un algorithme fonctionnant par récurrence permettant de trouver le plus court chemin entre deux nœuds. On suppose que $v^*(i, j)$ représente la distance minimale entre deux nœuds. Le principe précédent signifie que :

$$v^*(i, j) = \min \{v^*(i, k) + d(k, j) \mid (k \to j) \in S\}$$

Autrement dit, le plus court chemin entre les nœuds $i$ et $j$ correspond au meilleur des plus courts chemins entre le nœud $i$ et ses voisins $k$ auxquels est ajoutée la distance $d(k, j)$. Plusieurs algorithmes équivalents existent pour calculer la fonction $v^*$ (algorithmes de Bellman, Dijkstra ou Ford).

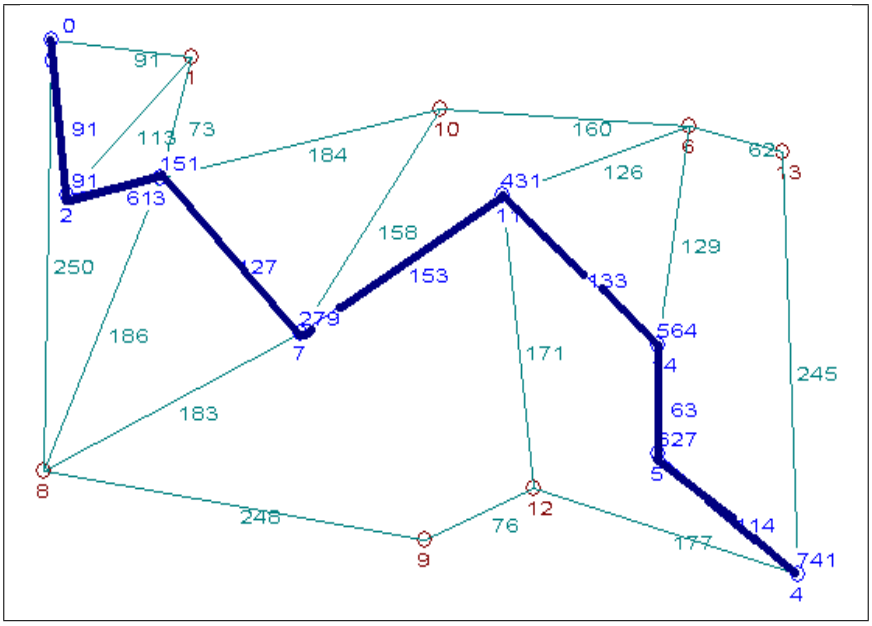



**Figure 23.4 :** *Ce graphe est identique à celui de la figure 23.1. Le chemin bleu est le chemin le plus court qui permette de relier les nœuds numéro 0 et numéro 4.*

L'algorithme de Bellman propose de calculer tous les chemins les plus courts. On suppose qu'il y a $n$ nœuds notés $(N_1, \ldots, N_n)$. Les distances entre chaque nœud sont données par la matrice $D = (d_{ij})$ où $d_{ij}$ est la distance du chemin direct qui relie les nœuds $n_i$ et $n_j$. On cherche alors la distance de tous les chemins les plus courts entre deux nœuds qu'on note également par le biais d'une matrice $M = (m_{ij})$.

**Algorithme 23.1 : algorithme du plus court chemin (Bellman)**
On reprend les notations décrites au paragraphe précédent.
**Etape A : initialisation**
$\forall i, j,\ M_{ij} = d_{ij}$

**Etape B : mise à jour**
$m \longleftarrow 0$
**pour** $l = 1$ **à** $N$ **faire**
  **pour** $i = 1$ **à** $N$ **faire**
    **pour** $j = 1$ **à** $N$ **faire**
      **pour** $k = 1$ **à** $N$ **faire**
        $M_{ij} \longleftarrow \min\{M_{ij}, M_{ik} + d_{kj}\}$
      **fin pour**
    **fin pour**
  **fin pour**
**fin pour**

L'algorithme de Ford est le moins restrictif puisqu'il suppose seulement que le graphe n'inclut pas de cycle de poids négatif, condition également nécessaire aux autres algorithmes. L'algorithme de Dijkstra suppose quant à lui que les poids sont strictement positifs et utilise cette hypothèse supplémentaire pour trouver

ce meilleur chemin plus rapidement.

**Définition 23.2 : cycle dans un graphe**
Un cycle est un chemin qui part d'un nœud $i$ et qui y revient. Le poids d'un chemin est la somme des poids des arêtes qu'il emprunte.

**Algorithme 23.3 : algorithme du plus court chemin (Ford)**
On suppose qu'un graphe est défini par un ensemble de $N$ sommets indicés de 1 à $N$. $X = \{1, \ldots, N\}$. Ces arêtes et leurs poids sont définis par une matrice d'adjacence $A = (a_{ij})_{(i,j)\in X^2}$ de telle sorte que $a_{ij}$ soit égal au poids de l'arête si elle existe, égal à l'infini si l'arête n'existe pas. On suppose que le graphe ne contient pas de cycle de poids négatifs. Dans le cas contraire, l'algorithme est sans fin. On cherche le chemin le plus court du nœud $i_0$ au nœud $i_1$.

**Etape A : initialisation**

$$\forall i \in X,\ v(i_0, i) = \begin{cases} 0 & \text{si } i = i_0 \\ \infty & \text{sinon} \end{cases}$$

$$\forall i \in X,\ p(i_0, i) = \emptyset$$

**Etape B : mise à jour**

$m \longleftarrow 0$
**pour** $i = 1$ **à** $N$ **faire**
  **pour** $j = 1$ **à** $N$ **faire**
    $t \longleftarrow v(i_0, i) + d(i, j)$
    **si** $t < v(i_0, j)$ **alors**
      $v(i_0, j) \longleftarrow t$
      $p(i_0, j) \longleftarrow \{i\}$
      $m \longleftarrow m + 1$
    **fin si**
  **fin pour**
**fin pour**

**Etape C : récurrence**
Si $m > 0$, cela signifie qu'une meilleure solution a été trouvée. On retourne à l'étape B.

**Etape D : meilleur chemin**
La longueur du meilleur chemin est donnée par $v(i_0, i_1)$. Le meilleur chemin utilise les valeurs $(p(i_0, i))_i$ qui mémorisent les prédécesseurs, ceux qui ont permis d'obtenir le meilleur chemin. $p(i_0, i_1)$ est le prédécesseur de $i_1$, $p\left(i_0, p(i_0, i_1)\right)$ est le prédécesseur $p(i_0, i_1)$. Le plus court chemin à l'envers correspond donc à la suite définie par récurrence $u_0 = i_1$ et $u_n = p\left(i_0, u_{n-1}\right)$. La construction de cette suite s'arrête lorsque $u_n = \emptyset$.

Lorsqu'un graphe contient un cycle de poids négatif, il est toujours possible de trouver des valeurs meilleures pour la fonction $v$. La dernière étape D n'arrive jamais.

### 23.1.6 Coût de l'algorithme

**Définition 23.4 : coût d'un algorithme**
Le coût d'un algorithme est le nombre d'opérations élémentaires (opération numérique, test, affectation, ...) nécessaire à sa complète réalisation. Ce coût est généralement exprimé comme un multiple constant d'une fonction tenant compte des dimensions des données manipulées par l'algorithme.

Par exemple, l'étape A de l'algorithme 23.3 effectue pour chaque sommet du graphe un test et deux affectations. Le coût de cette étape est donc un multiple de $N$. On dit que son coût est $O(N)$.

L'étape B effectue d'abord une affectation, puis se lance dans deux boucles imbriquées qui visitent les $N$ sommets du graphe. A chaque itération, l'algorithme effectue une additation, un test et selon la valeur de ce test, une ou quatre affectations. Le coût de cette étape est un multiple de $N^2$ auquel il faut ajouter 1 pour la première affectation. Cette affectation étant négligeable devant les $N^2$ autres opérations élémentaires, on dit que le coût de cette étape est $O(N^2)$.

Le coût de la dernière étape est fonction de la longueur du meilleur chemin qui ne peut inclure plus d'arêtes que n'en peut contenir le graphe, soit $N^2$. Le coût de l'étape D est au plus $O(N^2)$.

Si le graphe ne contient pas de poids négatifs, il est alors possible d'affirmer que le plus court chemin inclut au plus $N$ arêtes où $N$ est le nombre de sommets du graphe. L'étape B est visitée au plus $N$ fois.

Finalement, le coût de l'algorithme est au plus : $C = O(N) + NO(N^2) + O(N^2)$. On élimine de cette expression les termes négligeables pour obtenir un coût au plus égal à $C = O(N^3)$. Dans ce cas, le coût est fonction d'une puissance de $N$, on dit que le coût est *polynômial*.

**Remarque 23.5 : coût dépendant du contenu des données**
Il n'est pas toujours possible d'obtenir un coût dépendant seulement de la dimension des données manipulées. L'exemple le plus courant est l'algorithme de tri rapide (ou tri quicksort) dont le coût dépend de l'ordre du tableau à trier. Le coût de l'algorithme est alors défini comme un coût moyen sur l'ensemble des configurations possibles, ou des ordres possibles en ce qui concerne le tri rapide ($N!$).

### 23.1.7 A propos des graphes

Les graphes sont également utilisés pour résoudre les problèmes d'ordonnancement par l'intermédiaire de la méthode des potentiels (ou MPM). Cette méthode fait intervenir les graphes pour modéliser les dépendances entre des tâches et définir la date à laquelle chacun d'elles peut être commencée. Cette méthode combinent des chemins les plus courts et des chemins les plus longs afin de déterminer une plage temporelle pendant laquelle une tâche doit être exécutée.

Les graphes sont également utilisés pour optimiser les débits dans un réseau par l'intermédiaire de l'algorithme de Ford-Fulkerson. Ce problème est souvent désigné par "problème de flot maximal".

La théorie des graphes a permis aussi de résoudre le problème de coloriage des cartes : combien faut-il de couleurs au minimum pour colorier une carte sans que deux régions voisines soient de la même couleur ?

Les graphes apparaissent encore comme une possible représentation de modèles probabilistes tels que les chaînes de Markov. La matrice d'adjacence équivaut pour ces modèles à la matrice de transition qui contient des probabilités de transitions d'un état à un autre.

### 23.1.8 Programme

Le programme suivant implémente la recherche du meilleur chemin dans un graphe. La fonction `meilleur_chemin` est la trascription de l'algorithme 23.3 en langage *Python*. Les autres fonctions ne servent qu'à générer un graphe décrivant un réseau routier aléatoire. Les villes sont placées de telle sorte qu'elles soient suffisamment espacées pour rendre l'affichage graphique lisible. Pour la même raison, le graphe ne contient pas trop d'arêtes. La figure 23.5 montre un exemple de l'image construite par le programme.

Après avoir généré les villes et les arêtes de manière aléatoire grâce aux fonctions `construit_ville` et `construit_arete`, la fonction `choix_villes_depart_arrive` détermine deux villes de départ et d'arrivée. Cette fonction utilise la fonction `meilleur_chemin` afin d'éviter les chemins trop courts (moins de quatre villes). Enfin, la fonction `dessin_ville_arete` termine le programme par la construction d'une image contenant le graphe et le meilleur chemin trouvé. Toutes ces fonctions ne servent qu'à planter un décor pour la fonction `meilleur_chemin` et ainsi vérifier visuellement que la solution trouvée est bien la bonne. Ceci explique la longueur du programme alors que la fonction `meilleur_chemin` n'en forme qu'une petite partie.

Ce genre de problème facile - appelé *problème jouet* ou *toy problem* - est souvent présent dans les programme en tant que fonction de test. Il permet de vérifier qu'un algorithme a été implémenté correctement et

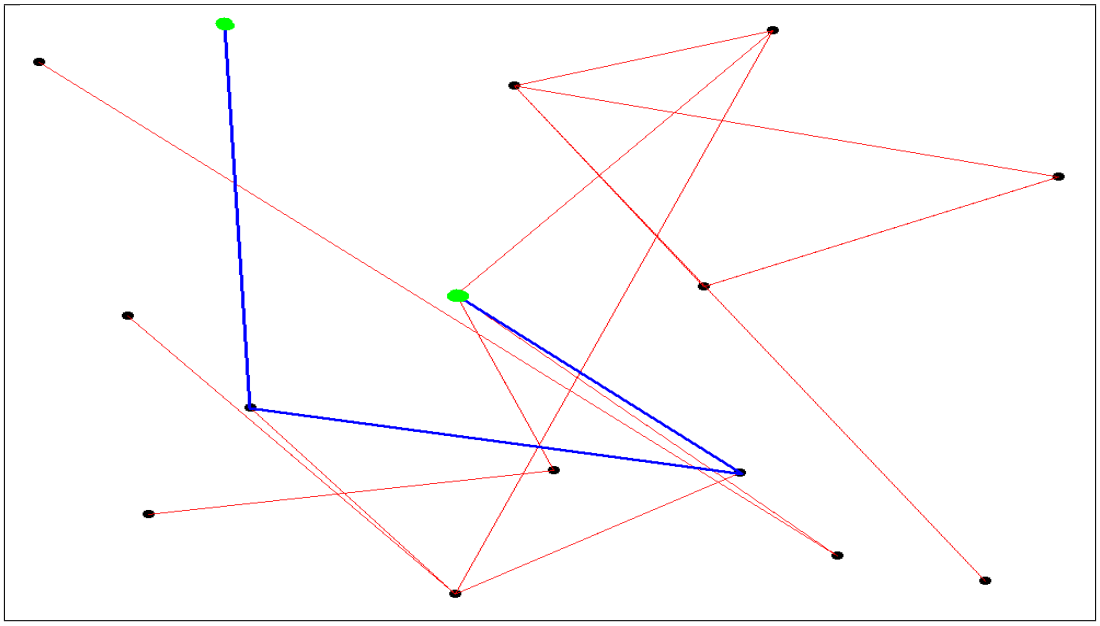


**Figure 23.5 :** *Exemple d'image construite par le programme cherchant le plus court chemin dans un graphe. Les villes vertes sont les points de départ et d'arrivée. Les arêtes bleues et épaisses sont les arêtes faisant partie du meilleur chemin.*

fonctionne pour une configuration dans laquelle le programmeur peut rapidement évaluer la validité d'un résultat. Il peut ainsi valider ou invalider une portion de son code.

```
# -*- coding: cp1252 -*-
import random                 # pour tirer aléatoirement des nombres
import Matrix as mat          # pour les matrices
import UserArray as ua        # pour les matrices
import math                   # fonction sqrt
import PIL.Image as Im        # pour les images
import PIL.ImageDraw as Id    # pour dessiner

infini = 10000000 # l'infini est égal à dix millions, c'est une variable globale

def construit_ville(n, x =1000, y = 800):
    """tire aléatoirement n villes dans un carrée x * y, on choisit
    ces villes de sortent qu'elles ne soient pas trop proches"""
    # deux villes ne pourront pas être plus proches que mind
    mind = math.sqrt (x*x+y*y) / (n * 0.75)
    # liste vide
    l = []
    while n > 0:
        # on tire aléatoirement les coordonnées d'une ville
        xx = x * random.random ()
        yy = y * random.random ()
        # on vérifie qu'elle n'est pas trop proche d'aucune autre ville
        ajout = True
        for t in l :
            d1 = t [0] - xx
            d2 = t [1] - yy
            d  = math.sqrt (d1*d1+d2*d2)
            if d < mind :
                ajout = False  # ville trop proche
        # si la ville n'est pas trop proche des autres, on l'ajoute à la liste
        if ajout:
            l.append ((xx,yy))
            n -= 1  # une ville en moins à choisir
```

```
    return l

def distance_ville (l,i,j):
    """calcule la distance entre deux villes i et j de la liste l"""
    x = l [i][0] - l [j][0]
    y = l [i][1] - l [j][1]
    return math.sqrt (float (x*x+y*y))

def construit_arete (l,part = 0.15):
    """tire aléatoirement part * len (l) arêtes et construit la matrice
    d'adjacence"""
    global infini
    nb  = len (l)
    m   = mat.Matrix ( [ 0 for i in range(0,nb) ]) # crée un vecteur de nb zéros
    m   = ua.transpose (m) * m                     # effectue une multiplication du vecteur
                                                   # précédent avec son vecteur transposé
                                                   # pour obtenir une matrice carrée nulle

    are = int (part * nb * nb)
    while are > 0:
        i = random.randint (0,nb-1)            # première ville
        j = random.randint (0,nb-1)            # seconde ville
        if i == j : continue                   # pas besoin d'arête entre i et i
        if m [i,j] > 0: continue               # si l'arête existe déjà, on recommence
        m [i,j] = int (distance_ville (l,i,j)) # on affecte comme poids à l'arête
                                               # la distance entre les deux villes
        m [j,i] = m [i,j] # symétrie de la matrice car le graphe est non orienté
        are -= 2          # deux cases de la matrice ne sont plus nulles

    # on associe à toutes les arêtes nulles de poids nul, donc inexistantes,
    # une valeur égale à l'infini pour signifier qu'elles ne sont pas reliées
    global infini
    for i in range (0, nb):
        for j in range (0, nb):
            if m [i,j] == 0:
                m [i,j] = infini

    return m

def dessin_ville_arete (l,m,chemin):
    """dessine la ville et les routes dans une image"""

    # on prend les coordonnées maximales
    mx, my = 0,0
    for i in l:
        mx = max (mx, i [0])
        my = max (my, i [1])
    mx += 25
    my += 25
    mx, my = int (mx), int (my)
    im = Im.new ("RGB", (mx, my), (255,255,255)) # création d'une image blanche
    draw = Id.Draw(im)

    # dessin des villes
    for i in l:
        j  = (int (i [0]), int (i[1]))
        j2 = (j [0] + 10, j [1] + 10)
        draw.ellipse ((j,j2), fill = (0,0,0))

    # dessin des arêtes
    global infini
    for i in range (0,len(l)):
        for j in range (0,len(l)):
            if m [i,j] > 0 and m [i,j] < infini :
                a = (int (l [i][0]+5), int (l [i][1]+5))
```

```
                b = (int (l [j][0]+5), int (l [j][1]+5))
                draw.line ((a,b),fill=(255,0,0))

    # dessin des villes de départ et d'arrivée en vert
    v1 = chemin [0]
    v2 = chemin [ len (chemin)-1]
    a = (int (l [v1][0]), int (l [v1][1]))
    b = (int (l [v1][0]+10), int (l [v1][1]+10))
    draw.ellipse ((a,b), fill = (0,255,0))
    a = (int (l [v2][0]), int (l [v2][1]))
    b = (int (l [v2][0]+10), int (l [v2][1]+10))
    draw.ellipse ((a,b), fill = (0,255,0))

    # dessin du chemin, arêtes en bleu
    for c in range (0,len(chemin)-1):
        i = chemin [c]
        j = chemin [c+1]
        if m [i,j] > 0 and m [i,j] < infini :
            a = (int (l [i][0]+5), int (l [i][1]+5))
            b = (int (l [j][0]+5), int (l [j][1]+5))
            draw.line ((a,b),fill=(0,0,255))
        else:
            a = (int (l [i][0]+5), int (l [i][1]+5))
            b = (int (l [j][0]+5), int (l [j][1]+5))
            draw.line ((a,b),fill=(0,0,50))

    # on retourne l'image
    return im

def meilleur_chemin (n,m,a,b):
    """détermine le meilleur chemin,
    n est le nombre de villes,
    m est la matrice d'adjacence,
    a est la ville de départ,
    b la ville d'arrivée"""

    # création d'un tableau, d [i] contient la meilleure distance minimale actuelle
    # séparant la ville i de la ville a
    d = [ 10000000 for i in range(0,n) ]

    # p [i] contient la ville prédécesseur qui permet d'atteindre la ville i
    # avec la distance d [i]
    p = [ -1 for i in range(0,n) ]

    # au départ, seul la distance d[a] est nulle
    d [a] = 0

    # cette boucle s'exécute tant qu'on effectue des mises à jour
    # dans le tableau d[i]
    modif = 1
    while modif > 0:
        modif = 0
        # on parcourt toutes les arêtes
        for i in range(0,n):
            for j in range (0,n):
                # nouvelle distance
                t = d [i] + m [i,j]
                if t < d [j] :  # si on a trouvé une meilleure distance minimale
                    d [j] = t   # on met à jour
                    p [j] = i
                    modif += 1  # une mise à jour de plus

    # on récupère le meilleur chemin
    l = []
    while b != -1:
```

```
        l.append (b)
        b = p [b]
    # on le retourne
    l.reverse ()
    return l

def choix_villes_depart_arrive(nb,m):
    """cette fonction choisit deux villes aléatoirement, départ et arrivée,
    elle évite que la ville et départ et d'arrivée soient les mêmes,
    elle évite que ces deux villes soient reliés par un seul arc,
    elle choisit deux villes pour lesquelles il existe un meilleur
    chemin"""
    global infini
    a,b = -1,-1
    tour = 0
    while True:
        a = random.randint (0,nb-1)     # première ville au hasard
        b = random.randint (0,nb-1)     # seconde ville au hasard
        if a == b: continue             # villes identiques, on recommence
        if m [a,b] != infini : continue # villes reliées, on recommence
        l = meilleur_chemin (nb,m,a,b)
        if l != None and len(l) > 3 :
            return a,b  # si le meilleur chemin existe,
                        # et n'est pas trop court (4 villes minimum,
                        # soit deux étapes entre a et b), alors
                        # on retourne le résultat
        else:
            tour += 1
            if tour > 120 : return 0,0       # au bout de 120 essais, on s'arrête

################################################################################
# programme principal
# construction des villes
l = construit_ville (15)

# construction des arêtes
print "adjacence"
m = construit_arete (l)

# choix de la ville de départ de d'arrivée
print "départ"
a,b = choix_villes_depart_arrive(len(l),m)

print "recherche du meilleur chemin"
chemin  = meilleur_chemin (len(l), m, a,b)

if chemin != None and len(chemin) > 0:
    print "meilleur chemin ", a, " --> ", b, " : ", chemin
    # construction de l'image du résultat
    im = dessin_ville_arete(l,m,chemin)
    im.show ()  # on affiche l'image
else :
    print "il n'existe pas de meilleur chemin"
```

### 23.1.9 Distance d'édition

Il est possible de définir une distance entre deux mots qu'on nomme le plus souvent une distance d'édition ou distance de Levenstein. On définit tout d'abord une distance entre caractère $C(x,y)$. $C(x,y) = 0 \Longleftrightarrow x = y$. Chaque mot est une séquence de caractères, $m = (c_1, \ldots, c_l)$ et $m' = \left(c'_1, \ldots, c'_{l'}\right)$. On définit la fonction $d(i,j)$ qui est égale à la distance entre les mots $m = (c_1, \ldots, c_i)$ et $m' = \left(c'_1, \ldots, c'_j\right)$. La distance

d'édition est définie par $d(l, l')$. Cette fonction se construit par récurrence :

$$d(0,0) = 0 \tag{23.2}$$
$$d(i,0) = C(c_i, \emptyset) \tag{23.3}$$
$$d(0,j) = C(\emptyset, c_j) \tag{23.4}$$
$$d(i,j) = \min \left\{ \begin{array}{l} d(i-1,j) + C(\emptyset, c_j), \\ d(i,j-1) + C(c_j, \emptyset), \\ d(i-1,j-1) + C(c_i, c_j) \end{array} \right\} \tag{23.5}$$

Cette distance peut se concevoir également comme la recherche d'un plus court chemin dans un graphe. Ici, les nœuds sont constitués des couples de positions $(i, j)$. Chaque nœud $(i, j)$ a trois prédécesseur $(i-1, j), (i, j-1), (i-1, j-1)$ et le coût associé à chaque arc correspondant aux valeurs de la fonction $C$.

On peut par exemple appliquer ce type d'algorithme pour déterminer les différences entre deux fichiers textes, quelles ont été les lignes ajoutées, supprimées comme le montre l'exemple suivant.

```
# -*- coding: latin-1 -*-
def get_lines (file) :
    """retourne toutes les lignes d'un fichier, nettoie les fins de lignes et les espaces"""
    f = open (file, "r")
    li = f.readlines ()
    f.close ()
    return [ l.strip ("\r\n") for l in li ]

def distance (line1, line2) :
    """construit une distance entre deux tableaux de lignes"""
    d = { (-1,-1):(0,(-1,-1), "") }
    for i in xrange (0, len (line1)) :
        d [ i,-1 ] = (i+1, (i-1,-1), "+ " + line1 [i])
    for j in xrange (0, len (line2)) :
        d [ -1,j ] = (j+1, (-1,j-1), "- " + line2 [j])

    for i in xrange (0, len (line1)) :
        l1 = line1 [i]
        for j in xrange (0, len (line2)) :
            l2 = line2 [j]
            c  = abs (cmp (l1, l2))
            i1 = d [i-1,j][0] + 1
            i2 = d [i,j-1][0] + 1
            i3 = d [i-1,j-1][0] + 2*c
            if i1 <= min (i2, i3) :
                d [i,j] = (i1, (i-1,j), "+ " + l1)
            elif i2 <= min (i1, i3) :
                d [i,j] = (i2, (i,j-1), "- " + l2)
            else :
                d [i,j] = (i3, (i-1,j-1), "  " + l1)

    last = (len (line1)-1, len (line2)-1)
    pos  = [d [last]]
    pn   = pos [0][1]
    while pn != (-1,-1) :
        p  = pos [len (pos)-1]
        pn = p [1]
        pos.append (d [pn])
    pos.pop ()
    pos.reverse ()
    return [ p [2] for p in pos ]

def distance_file (file1, file2) :
    line1 = get_lines (file1)
    line2 = get_lines (file2)
    return distance (line1, line2)
```

```
if __name__ == "__main__" :
    file1 = "filedistance.py"
    file2 = "filedistance2.py"
    res   = distance_file (file1, file2)
    for r in res :
        print r
```

Ce programme compare deux versions d'un même programme, il fait précéder chaque ligne de − ou + selon qu'elle a été supprimée ou insérée.

```
def distance (line1, line2) :
      """construit une distance entre deux tableaux de lignes"""
-     d = { (-1,-1):(0,(-1,-1)) }
+     d = { (-1,-1):(0,(-1,-1), "") }
+     for i in xrange (0, len (line1)) :
+         d [ i,-1 ] = (i+1, (i-1,-1), "+ " + line1 [i])
+     for j in xrange (0, len (line2)) :
+         d [ -1,j ] = (j+1, (-1,j-1), "- " + line2 [j])

      for i in xrange (0, len (line1)) :
          l1 = line1 [i]
          for j in xrange (0, len (line2)) :
```

## 23.2 Problème du voyageur de commerce

### 23.2.1 Enoncé

Le problème du voyageur de commerce est un problème classique, aussi connu sous le nom de *travevelling salesman problem*. Le voyageur de commerce doit effectuer la tournée de ses magasins le plus rapidement possible. Il doit donc déterminer un chemin passant par toutes les villes qui soit le plus court possible. La figure 23.6 donne un exemple de chemin passant par six villes.

Cette problématique se retrouve également dans d'autres domaines comme la plannification de tâches. Par exemple, une machine doit effectuer un certain nombre de travaux, l'ordre dans lequel elle les traite importe peu, on demande simplement que le temps de traitement de l'ensemble des tâches soit le plus court possible. Cependant, avant toute tâche, cette machine doit être préparée et le temps de préparation dépend des tâche précédente et suivante. Ce problème ressemble à celui du voyageur de commerce, les temps de préparation entre deux tâches jouent le rôle de distance.

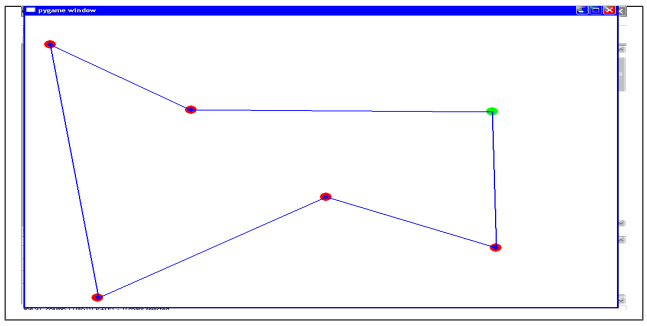


**Figure 23.6 :** *Plus court chemin passant par les six villes.*

Il existe plusieurs algorithmes pour déterminer ce plus court chemin. Deux seront présentés. Le premier est celui des cartes de Kohonen, il s'applique de préférence à un nombre de villes inférieur à un ou deux centaines. Au delà, lorsque les villes sont trop proches des unes des autres, cet algorithme retourne une

solution où les erreurs apparaissent de manière évidente (voir figure 23.7). La seconde méthode est plutôt utilisée pour la recherche du plus court chemin pour des graphes contenant un grand nombre de villes. Pour ces grands graphes, les cartes de Kohonen convergent lentement. Le plus court chemin est en fait une permutation de l'ensemble des villes, les deux méthodes proposées sont deux manières différentes d'explorer l'ensemble des permutations possibles.

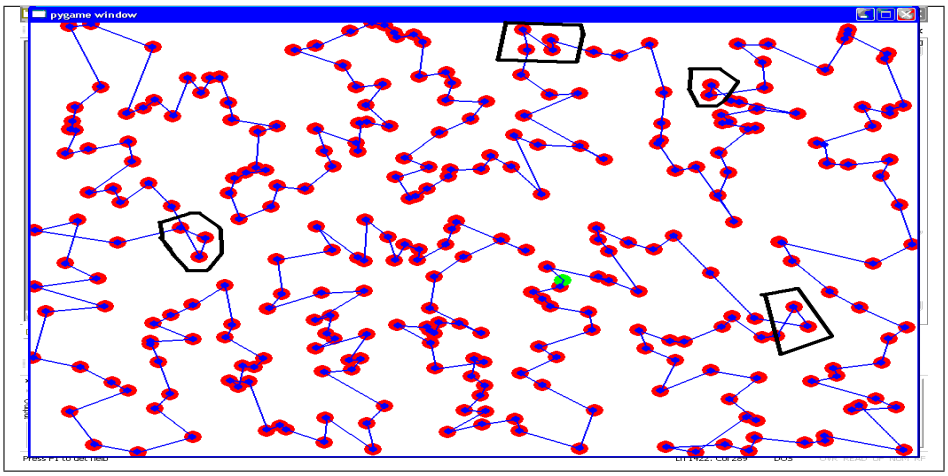



**Figure 23.7 :** *Plus court chemin passant par les 300 villes, solution retournée par l'algorithme de Kohonen. Certaines erreurs cerclées de noir sont évidentes comme deux segments du chemin qui se croisent. Celles-ci sont néanmoins facile à corriger une fois que la solution a convergé.*

Obtenir le plus court chemin passant par tous les sommets d'un graphe est un problème d'optimisation avec contrainte. C'est un problème dit *NP-complet* (voir définition 23.6) [2] est dédié à ce problème. Il référence des articles, quelques solutions à des problèmes géographiques comme le plus court chemin reliant toutes les villes de Suède. Il référence également une librairie écrite en C proposant divers algorithmes permettant de résoudre le problème du voyageur de commerce.

> **Définition 23.6 : problème NP-complet**
> Un problème est dit NP-complet s'il n'existe aucun algorithme de coût polynômial permettant d'obtenir une solution optimale.

Cette complexité peut se traduire en nombre, le document [Helsgaun2000] cite un problème d'optimisation de 7397 éléments [3] pour lequel il y a plus de $10^{25000}$ solutions possibles. Le problème du voyageur de commerce appartient à une classe de problème plus vaste appelée *optimisation combinatoire*. Pour ce type de problème, l'ensemble des solutions possibles est discret et non continu.

### 23.2.2 Cartes de Kohonen

Les cartes de Kohonen sont utilisées pour réaliser la projection d'un nuage de points sur un espace de dimension moindre. Dans le cas du problème du voyageur de commerce, il s'agit de projeter un nuage de points inclus dans un espace à deux dimensions vers un espace à une dimension. Chaque ville recevra une coordonnée : son ordre dans le chemin le plus court. Les cartes de Kohonen sont plus fréquemment utilisées pour projeter un nuage de points inclus dans un espace à $n > 2$ dimensions vers un espace à deux dimensions, soit un espace qui puissent être représenté graphiquement (voir figure 23.8). Ceci explique leur désignation de cartes de Kohonen ou en anglais *Self Organized Map* (SOM). Le document [Koivisto1999] (voir aussi [Vesanto2000]) regroupe plusieurs articles à propos de diverses techniques et diverses utilisations des cartes de Kohonen dans des problèmes tels que la reconnaissance de la parole, les modèles de Markov

2. Le site http://www.tsp.gatech.edu/index.html
3. recensé par le site http://www.tsp.gatech.edu/index.html

cachés, la recherche de mot-clé dans un texte, l'analyse en composantes principales (ACP).

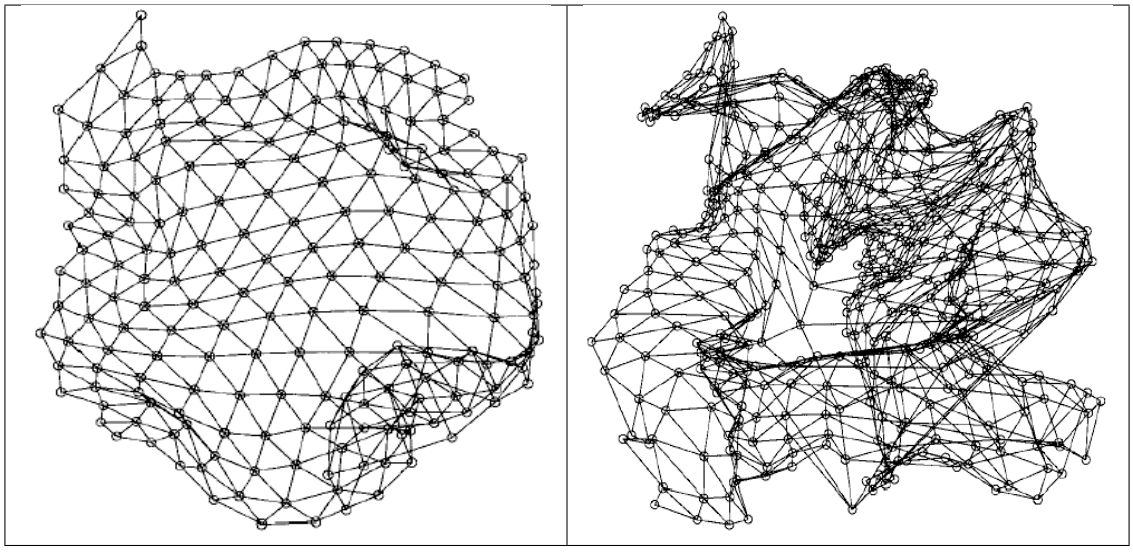


**Figure 23.8 :** *Exemple de cartes de Kohonen après convergence. Il s'agit de projeter dans un plan un nuage de points en trois dimensions. Chaque neurone possède six voisins. Plane au départ, la carte se déforme pour s'adapter au nuage de points qu'elle doit modéliser. La seconde carte est plus déformée que la première et présente quelques torsions qui ressemblent à un papillon. Ces figures sont tirées de l'article [Vesanto2000].*

On suppose qu'un chemin est constitué de points ou neurones, différents ou non des villes, chacun étant reliés à ses deux voisins. L'idée principale de l'algorithme consiste à faire en sorte que chaque villes tire vers elle un neurone et ses voisins. Au cours des itérations successives, la longueur du chemin s'agrandit

et il se rapproche peu à peu des villes jusqu'à que chaque ville soit rejointe par un neurone.

**Algorithme 23.7 : carte de Kohonen**
On note $n$ villes par $(v_1,\dots,v_n) \in \left(\mathbb{R}^k\right)^n$ avec $k \geqslant 2$. Un chemin de longueur $l$ est noté $(p_1,\dots,p_l) \in \left(\mathbb{R}^k\right)^l$, on suppose que les voisins du neurone $p_i$ sont les neurones d'indices $(i-1) \mod l$ et $(i+1) \mod l$. La fonction $d : \mathbb{R}^k \times \mathbb{R}^k \longrightarrow \mathbb{R}$ désigne une distance.

**Etape A : initialisation**
Le chemin est constitué de trois neurones placés autour du barycentre de l'ensemble $(v_1,\dots,v_n)$.
$t \longleftarrow 0$, $t$ compte le nombre d'itérations.

**Etape B : itération**
On définit le vecteur $(c_1,\dots,c_l) = 0$.
$t \longleftarrow t+1$
**pour** $i = 1$ **à** $n$ **faire**
On détermine l'indice $j^*$ du neurone le plus proche de la ville $i$.
$c_{j^*} \longleftarrow c_{j^*} + 1$
$p_{j^*} \longleftarrow p_{j^*} + \epsilon\left(t, d\left(v_i, p_{j^*}\right)\right) \ [v_i - p_{j^*}]$
On met à jour chacun des voisins $q$ de $p_{j^*}$ de la façon suivante :
$q \longleftarrow q + \eta\left(t, d\left(v_i, p_{j^*}\right), d\left(q, p_{j^*}\right)\right) \ [v_i - p_{j^*}]$
**fin pour**

**Etape C : ajout, suppression de neurones**
Pour chaque neurone d'indice $i$ du chemin, si $c_i = 0$ alors il est supprimé. Ceci signifie qu'aucune ville ne l'a choisi. Dans le cas contraire, si $c_i \geqslant 2$, plus d'une ville l'a choisi, le neurone est dupliqué. C'est-à-dire qu'un autre neurone est inséré dans le chemin et placé à proximité du neurone dupliqué.

**Etape D : terminaison**
Tant que les neurones bougent encore, on retourne à l'étape A.

Afin que l'algorithme converge, on choisit des fonctions $\epsilon(t,x)$ et $\eta(t,x)$ décroissantes par rapport à $t$. Il est conseillé de choisir des fonctions qui vérifient :

$$\forall x \in \mathbb{R}^+, \quad \sum_{t=1}^{\infty} \epsilon(t,x) = \infty \text{ et } \sum_{t=1}^{\infty} \epsilon^2(t,x) < \infty \tag{23.6}$$

$$\forall t,x,y \in \mathbb{N} \times \mathbb{R}^+ \times \mathbb{R}^+, \quad \eta(t,x,y) < \epsilon(t,x) \tag{23.7}$$

On peut choisir par exemple :

$$\epsilon(t,x) = \frac{\epsilon_0}{1+t} \frac{d_0}{d_0+x} \tag{23.8}$$

$$\eta(t,x) = \frac{\eta_0}{1+t} \epsilon(t,x) \tag{23.9}$$

Il est souvent utile d'effectuer plusieurs essais pour déterminer les valeurs $\epsilon_0$, $\eta_0$, $d_0$. Ces fonctions peuvent également dépendre d'autres paramètres tels que l'angle formé par un neurone et ses deux voisins. Il est aussi possible qu'une ville attire un neurone, ses deux neurones les plus proches puis encore d'autres voisins plus éloignés. Il n'existe pas de fonction optimale, elles dépendent du problème à résoudre.

Pour améliorer la convergence de l'algorithme, plutôt que de considérer à l'étape A les villes toujours dans le même ordre, elles sont toutes passées en revue dans un ordre aléatoire et différent à chaque itération. Cette modification permet de relancer plusieurs fois l'algorithme pour choisir au final le chemin de plus courte longueur.

La suppression et l'insertion de neurones est facile à concevoir lorsque la carte de Kohonen n'est qu'un chemin à une dimension. Lorsqu'il s'agit d'un maillage à deux dimensions, cette étape n'est pas incluse. L'initialisation consiste alors à placer un nombre de neurones fixe puis à appliquer l'étape A. Le voisinage est simplement plus étendu.

Le coût de cet algorithme dépend de la vitesse de convergence. Toutefois, l'étape B a un coût de $n^2$ où $n$ est le nombre de villes, coût à multiplier par le nombre d'itérations. Les paragraphes qui suivent proposent un autre algorithme permettant de construire une solution pour le problème du voyageur de commerce. Il est beaucoup plus rapide mais il explore davantage l'ensemble des solutions.

### 23.2.3 Arbre de poids minimal et circuit hamiltonien

Ce paragraphe décrit un algorithme permettant de construire un chemin dont on espère qu'il sera le plus court. Ilpart d'un ensemble de villes pour construire un *circuit hamiltonien* (voir définition 23.9). Cet algorithme utilise deux chemins d'un type particulier, eulérien et hamiltonien.

**Définition 23.8 : circuit eulérien**
Un chemin eulérien d'un graphe est un chemin passant par tous les arcs de ce graphe. Un circuit eulérien est un chemin eulérien dont les nœuds de départ et d'arrivée sont identiques.

**Définition 23.9 : circuit hamiltonien**
Un chemin hamiltonien d'un graphe est un chemin passant par tous les nœuds de ce graphe. Un circuit hamiltonien est un chemin hamiltonien dont les nœuds de départ et d'arrivée sont identiques.

La première étape de l'algorithme consiste à construire un arbre de poids minimum (voir définition 23.12). Si le chemin doit parcourir $n$ villes, cette première étape permet de réduire l'ensemble des arcs possibles ($\frac{n(n-1)}{2}$) à un multiple de $n$.

**Algorithme 23.10 : plus court chemin à partir d'un arbre de poids minimum**
On suppose qu'il faut déterminer le plus court chemin passant par $n$ villes. L'algorithme se décompose en quatre étapes :

1. construction de l'arbre de poids minimum (algorithme 23.13)
2. construction d'un circuit eulérien (algorithme 23.14)
3. construction d'un circuit hamiltonien (algorithme 23.15)
4. simplification du circuit hamiltonien (algorithme 23.16)

Le circuit eulérien parcourt tous les arcs de poids minimal et passe donc plusieurs fois par le même sommet. En enlevant les sommets redondants, on obtient un circuit hamiltonien qui est une solution pour le problème du voyageur de commerce. L'avantage de cet algorithme est son coût, multiple du nombre de villes.

### 23.2.4 Arbre de poids minimum

Obtenir l'arbre de poids minimum est encore un problème d'optimisation avec contrainte. Il s'agit de réduire l'ensemble des arcs tout en conservant un graphe composé d'une seule composante connexe.

**Définition 23.11 : composante connexe**
Une composante connexe $C$ d'un graphe est un ensemble de nœuds tel que pour tout couple de nœuds $(s,t) \in C^2$, il existe un chemin allant de $s$ à $t$ dont tous les nœuds intermédiaires sont inclus dans $C$.

Il est maintenant possible d'exprimer la contrainte liée au problème de l'arbre de poids minimum.

**Définition 23.12 : arbre de poids minimum**
Soit un graphe $G = (X, E)$, un arbre de poids minimum $G^* = (X^*, E^*)$ est un sous-arbre de $G$ qui vérifie :

1. $X^* = X$, le graphe de poids minimum partage les mêmes sommets que le graphe initial.
2. $E^* \subset E$, les arcs graphe de poids minimum possède forme un sous-ensemble de l'ensemble des arcs du graphe initial.
3. La somme des poids des arcs de $E^*$ est minimale.
4. Le graphe $G^*$ contient autant de composantes connexes que le graphe initial.

L'algorithme qui suit n'est pas le seul permettant d'obtenir une solution approchée à ce problème d'optimisation. Il ne considère que le cas où le graphe ne contient qu'une seule composante connexe. Il consiste à partir d'un graphe débarrassé de tous ses arcs puis à ajouter des arcs reliant deux composantes connexes différentes jusqu'à ce qu'il n'en reste plus qu'une.

**Algorithme 23.13 : algorithme de Kruskal**
On suppose que $G(X, E)$ est un graphe, $X$ est l'ensemble des sommets, $E$ est l'ensemble des arcs. Si $e \in E$, $f(e)$ désigne son poids. Cet algorithme a pour but d'obtenir un sous-ensemble $F \subset E$ de poids minimal tel que le graphe $G(X, F)$ ne forme qu'une seule composante connexe. Le nombre de sommets est $N$. L'arc $e$ relie les sommets $a(e) \longrightarrow b(e)$.

**Etape A : initialisation**

On trie les arcs par ordre croissant de poids, on obtient la suite $(e_1, \ldots, e_n)$. A chaque sommet $x$, on associe la composante connexe $c(x)$. $n \longleftarrow N$, $n$ désigne le nombre de composantes connexes. $F \longleftarrow \emptyset$

**Etape B : itération**

$i \longleftarrow 1$
**tant que** $(N > 1)$ **faire**
  **si** $c(a(e_i)) \neq c(b(e_i))$ **alors**
    $F \longleftarrow F \cup \{e_i\}$
    **pour chaque** $x \in X$ **faire**

$$c(x) = \begin{cases} c(x) & \text{si } c(x) \neq c(b(e_i)) \\ c(a(e_i)) & \text{si } c(x) = c(b(e_i)) \end{cases}$$

    **fin pour**
  **fin si**
  $i \longleftarrow i + 1$
**fin tant que**

Cet algorithme ne retourne pas la solution optimale mais une solution approchée. Son coût est proportionnel au cardinal de l'ensemble $E$. A priori, pour le problème du voyageur de commerce, si le graphe contient $n$ villes, le nombre d'arcs possibles est $\frac{n(n-1)}{2}$. Dans cet ensemble, la majeure partie des arcs ne sera pas utilisée pour le meilleur chemin puisque les arcs entre plus proches voisins leur seront préférés. Afin de réduire cet ensemble, il est possible de quadriller le plan en zones et d'associer à chaque ville la zone à laquelle il appartient. On peut ainsi se contenter de construire l'arbre de poids minimal à l'aide des arcs entre villes de même zone ou de zones voisines (voir figure 23.9). La figure 23.10 montre un exemple d'arbre de poids optimal obtenu par l'algorithme de Kruskal.

### 23.2.5 Circuit eulérien

Le graphe obtenu par l'algorithme 23.14 est dans notre cas non orienté. Il est possible de passer d'une ville à une autre puis d'en revenir. Ceci signifie donc que chaque sommet est connecté à un nombre pair

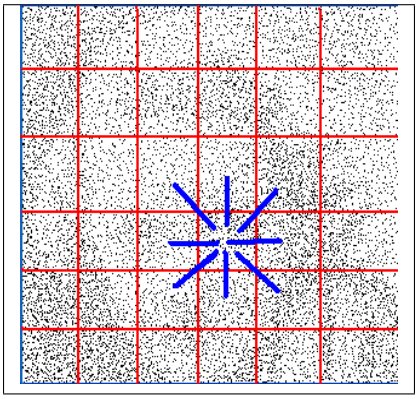


**Figure 23.9 :** *30000 villes sont réparties dans ce carré. Evaluer chaque arc est alors beaucoup trop coûteux. Une solution simple consiste à diviser le plan en zone puis à ne considérer les arcs entre deux sommets de la même zone ou de deux zones voisines. L'ensemble des arcs pris en compte dans l'algorithme de Kruskal (23.13) n'est plus* $\frac{n(n-1)}{2}$ *mais un multiple de* $n$ *d'autant plus grand que les zones sont grandes. On peut par exemple déterminer le nombre de zones* $z$ *en essayant de faire en sorte qu'elles contiennent un nombre constant* $\alpha$ *de villes. Le nombre d'arcs envisagés est majoré par :* $9\alpha n$.

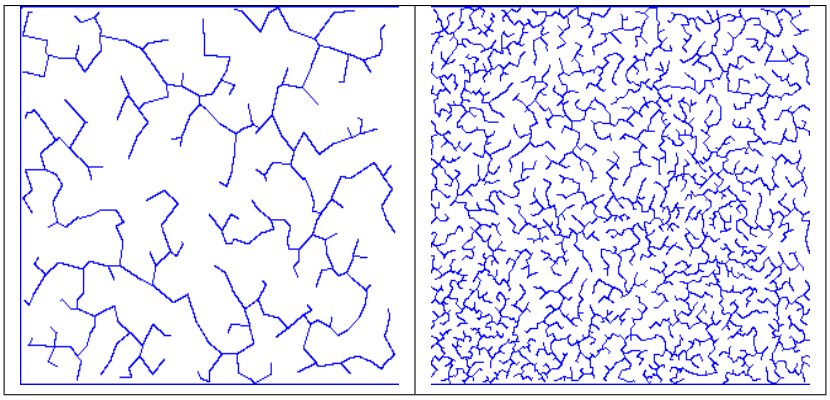


**Figure 23.10 :** *Exemples d'arbres obtenus par l'algorithme de Kruskal (23.13). Le premier pour une centaine de villes, le second pour environ 3000 villes.*

d'arcs. Par conséquent, il est possible de construire un chemin qui passe une seule fois par tous les arcs du graphe (voir figure 23.3, page 518).

La définition d'un circuit eulérien ne fait pas intervenir de considérations géométriques. Toutefois, pour déterminer un circuit eulérien, nous allons utiliser les coordonnées des villes qui composent les sommets du graphe afin de parcourir cet arbre selon le sens trigonométrique inverse (voir figure 23.11).

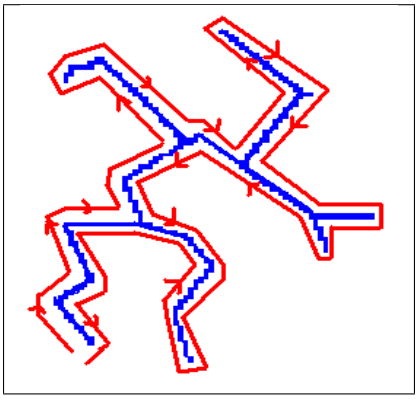


**Figure 23.11 :** *Parcours de l'arbre de poids minimal de façon à former un circuit eulérien. On part d'une extrémité puis on parcourt le graphe dans le sens trigonométrique inverse jusqu'à revenir au point de départ.*

**Algorithme 23.14 : construction d'un circuit eulérien**
On suppose que le graphe dont il faut obtenir un circuit eulérien est un arbre non-orienté de poids minimal comme celui retourné par l'algorithme 23.13. On suppose également qu'à chaque sommet $x$ sont associés des coordonnées $p(x)$ et que deux sommets ne sont jamais confondus. L'arbre contient $n$ sommets et $2n$ arcs.

**Etape A : initialisation**

On choisit un nœud $x$ connecté à un seul autre sommet. $ch \longleftarrow (x)$ et $t \longleftarrow 1$.

Pour chaque arc $e$, $u(e) = \begin{cases} 1 & \text{si l'arc a été parcouru} \\ 0 & \text{sinon} \end{cases}$

**Etape B : itération**

**tant que** $(t < 2n)$ **faire**

$x$ désigne le dernier sommet visité, $x^-$ désigne le sommet précédent dans le chemin $ch$. On choisit le sommet suivant $x^+$ de telle sorte que :

1. L'arc $e = (x \longrightarrow x^+)$ existe et vérifie $c(e) = 0$.
2. Parmi tous les arcs vérifiant la première condition, on choisit celui qui maximise l'ange $\left(\overrightarrow{p(x^-)p(x)}, \overrightarrow{p(x)p(x^+)}\right)$.

$$\begin{array}{lcl} t & \longleftarrow & t+1 \\ ch & \longleftarrow & ch \cup (x^+) \\ c(x \rightarrow x^+) & \longleftarrow & 1 \end{array}$$

**fin tant que**

$ch$ est le chemin eulérien cherché.

Le coût de cet algorithme est en $O(n)$.

### 23.2.6 Circuit hamiltonien

A partir d'un circuit eulérien, on construit un circuit hamiltonien en évitant simplement les nœuds déjà parcourus. Cette construction est possible puisque le graphe est entièrement connecté. Il est donc possible de passer d'un nœud quelconque à un autre nœud quelconque lui aussi. Il est possible que cet arc

n'appartienne pas au graphe.

**Algorithme 23.15 : construction d'un circuit hamiltonien**
On suppose que le graphe $G = (X, E)$ est entièrement connecté. *ch* désigne un chemin eulérien.
**Etape A : initialisation**
Pour les sommets $x \in X$, on pose $c(x) = 0$. $H \longleftarrow \emptyset$, $H$ est le chemin hamiltonien cherché.
**Etape B : parcours**
On parcourt le chemin eulérien *ch* dans l'ordre. Pour chaque sommet $x$ du chemin, si $x$ n'a pas encore été visité ($c(x) = 0$) alors $H \longleftarrow H \cup (x)$ et $c(x) = 1$. On poursuit avec les sommets suivants.

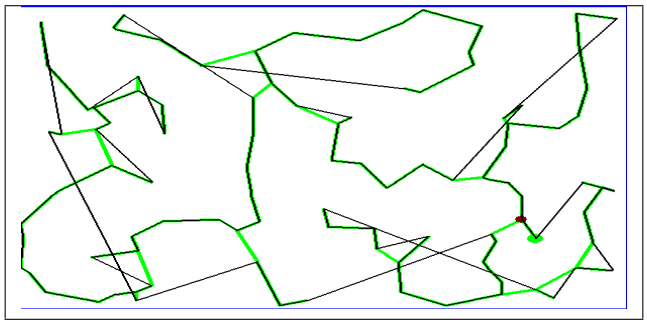


**Figure 23.12 :** *Résultat obtenu pour un ensemble de 300 villes, la solution retournée est obtenue rapidement mais présente des erreurs évidentes qu'il est possible de corriger en échangeant la position de nœuds dans le chemin (voir figure 23.13). Les arcs verts font partie de l'arbre de poids minimal, les arcs noirs font partie du circuit hamiltonien.*

La figure 23.12 montre un chemin hamiltonien obtenu pour 300 villes. Le coût de cet algorithme est en $O(n)$.

### 23.2.7 Simplification du circuit hamiltonien

La figure 23.12 montre des imperfections évidentes qui pourraient être corrigées simplement en échangeant la position de nœuds dans le chemin hamiltonien obtenu après l'exécution des trois premières étapes de l'algorithme 23.10. Un échange de nœuds est pertinent s'il fait décroître la longueur du chemin. Une autre solution consiste à déplacer un morceau du chemin pour l'insérer entre deux villes. L'ensemble des possibilités envisagées sont inspirées de [Helsgaun2000] et sont illustrées par la figure 23.13.

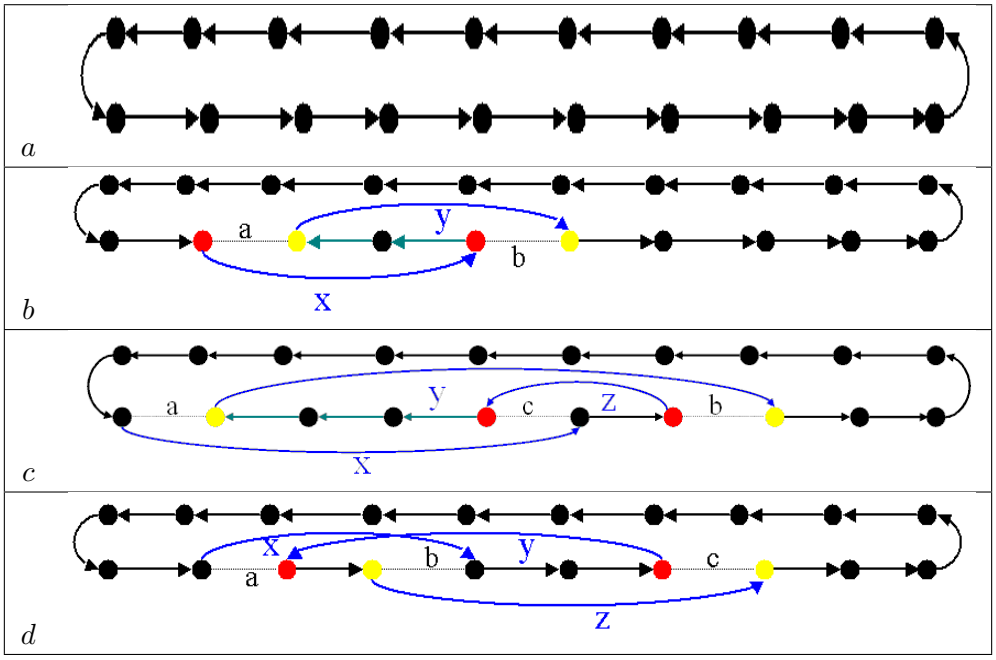



**Figure 23.13 :** *Trois Schémas simplificateurs envisagés pour supprimer les croisements entre segments d'un chemin. La première image représente un chemin. La seconde image représente le schéma d'un retournement d'une portion du chemin. Ceci permet d'éviter que le chemin décrive une figure en forme de $\infty$. La troisième figure présente le déplacement d'une portion du chemin entre deux autres nœuds. La dernière image présente le déplacement et le retournement d'une portion du chemin entre deux villes.*

L'algorithme qui suit reprend le schéma développé par Lin-Kernighan (voir [Helsgaun2000]).

**Algorithme 23.16 : simplification du circuit hamiltonien**
Soit un circuit hamiltonien $v = (v_1, \ldots, v_n)$ passant par les $n$ nœuds - ou villes - d'un graphe. Pour tout $i \notin \{1, \ldots, n\}$, on définit la ville $v_i$ par $v_i = v_{i \equiv n}$. Il est possible d'associer à ce chemin un coût égal à la somme des poids associés aux arêtes $c = \sum_{i=1}^{n} c(v_i, v_{i+1})$. Cet algorithme consiste à opérer des modifications simples sur le chemin $v$ tant que son coût $c$ décroît. Les opérations proposées sont :

1. Le retournement (image $b$, figure 23.13) : il consiste à retourner une sous-partie du chemin. Si on retourne le sous-chemin entre les villes $i$ et $j$, le chemin complet devient $(v_1, ..., v_{i-1}, v_j, v_{j-1}, ..., v_i, v_{j+1}, ..., v_n)$. Le retournement dépend de deux paramètres.
2. Le déplacement (image $d$, figure 23.13) : il consiste à déplacer une sous-partie du chemin. Si on déplace le sous-chemin entre les villes $i$ et $j$ entre les villes $k$ et $k+1$, le chemin complet devient $(v_1, ..., v_{i-1}, v_{j+1}, ..., v_k, v_i, v_{i+1}, ..., v_j, v_{k+1}, v_n)$. Le déplacement dépend de trois paramètres.
3. Le déplacement retourné (image $d$, figure 23.13) : il allie les deux procédés précédents. Si on déplace et on retourne le sous-chemin entre les villes $i$ et $j$ entre les villes $k$ et $k+1$, le chemin complet devient $(v_1, ..., v_{i-1}, v_{j+1}, ..., v_k, v_j, v_{j-1}, ..., v_i, v_{k+1}, v_n)$. Le déplacement retourné dépend aussi de trois paramètres.

Ces deux opérations (retournement, déplacement) dépendent d'au plus trois paramètres. Le coût de cet algorithme est donc en $O(n^3)$, ce qui est très coûteux lorsque le nombre de villes dépasse quelques milliers [4] C'est pourquoi toutes les combinaisons possibles pour les deux paramètres d'un retournement ou les trois paramètres d'un déplacement ne seront pas envisagées. Peu d'entre elles sont susceptibles d'avoir un résultat positif et pour un grand nombre de villes, le temps d'exécution devient très long. Les contraintes

4. Le coût des algorithmes 23.13, 23.14, 23.15 qui précèdent celui-ci est en $O(n)$.

choisies sur les paramètres déterminent la vitesse de convergence et affecte les performances.

Concernant les contraintes, la première idée (voir [Helsgaun2000]) est de se resteindre au retournement ou au déplacement de sous-chemin d'au plus quelques villes - pas plus d'une dizaine -. La seconde idée consiste à se concentrer sur des zones où il paraît possible de diminuer la longueur du chemin. La figure 23.14 montre quelques schémas récurrents que les retournements ou déplacements cherchent à résorber ainsi que l'utilisation de zones pour repérer le lieu probable de ces schémas. L'inconvénient d'une telle méthode est qu'elle n'est applicable que si les nœuds du graphe ont des coordonnées.

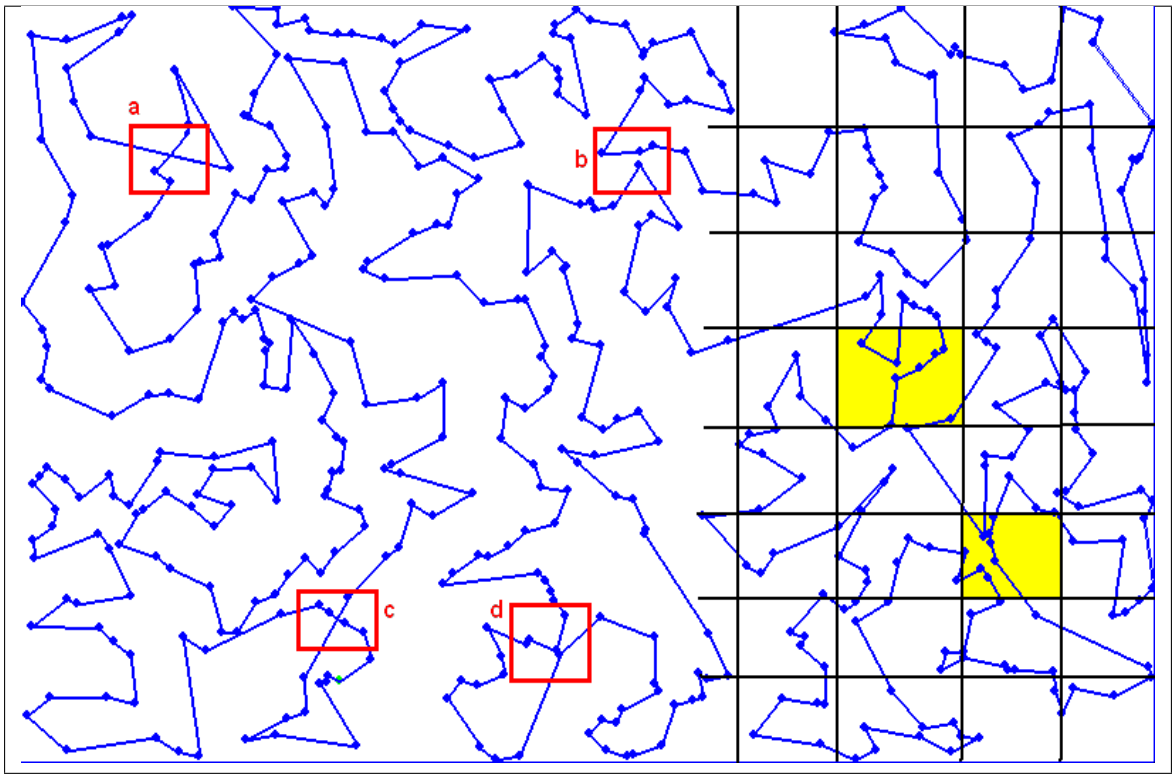



**Figure 23.14 :** *Cette figure montre un exemple de chemin hamiltonien de 500 villes avant l'utilisation de l'algorithme 23.16. Les zones a et c représentent une intersection de segments. Les zones b et d contiennent un ville dont le déplacement dans un segment proche diminuerait la longueur du chemin. Afin de repérer plus rapidement les lieux probables où il est possible de raccourcir le chemin, on quadrille l'image puis on recense pour chaque case l'ensemble des arêtes la traversant, puis l'ensemble des villes aux extrémites de ces arêtes. Ceci permet d'extraire une liste de villes pour lesquelles il est intéressant de tester des hypothèses de retournements ou de déplacements. Par exemple, pour les zones colorées à droite de l'image, le chemin peut être raccourci de manière évidente mais il n'est pas nécessaire de tester des hypothèses de retournements ou de déplacements associant des villes situées dans cette zone avec des villes situées à l'extérieur.*

**Remarque 23.17 : retournement, déplacement, ...**
Il est possible d'imaginer d'autres transformations que les retournements ou déplacements. Il y a par exemple les permutations où deux sous-parties qui peuvent être de longueurs différentes sont permutées. Chaque transformation dépend d'un certain nombre de paramètres ou degrés de liberté, plus ils sont nombreux, plus l'optimisation du chemin a de chance d'aboutir au chemin optimal, et plus cette optimisation sera longue. En règle générale, plus le chemin à optimiser est long, moins les transformations choisies seront complexes, d'abord parce que cela prendrait trop de temps, ensuite parce que le gain qu'on peut en attendre est moins important sur de grands problèmes. En effet, pour un circuit optimal avec peu d'étapes, changer une arête augmente beaucoup sa longueur. Sur un circuit optimal avec beaucoup d'étape, changer une arête a généralement peu d'impact comparé à la longueur totale du circuit.

**Remarque 23.18 : utilité de l'arbre de poids minimum**
L'algorithme 23.16 peut tout-à-fait être utilisé seul à partir d'un circuit hamiltonien initial déterminé de manière aléatoire. Pour des problèmes de petites tailles (quelques centaines de villes), la détermination

d'un premier circuit hamiltonien à partir d'un arbre de poids minimum n'est pas nécessaire. En revanche, pour des problèmes de plus grandes tailles, cette première étape dont le coût est en $O(n)$ permet d'accélérer la convergence de l'algorithme 23.16.

**Remarque 23.19 : convergence de l'algorithme 23.16**
Etape après étape, l'algorithme 23.16 fait décroître la longueur du chemin. Il n'est pas toujours nécessaire de terminer l'algorithme lorsque cette longueur ne décroît plus mais lorsqu'elle ne décroît plus suffisamment. La figure 23.15 reprend un exemple d'évolution de la longueur du chemin étape après étape. La courbe décroît rapidement au cours des premières itérations puis décroît lentement ensuite.

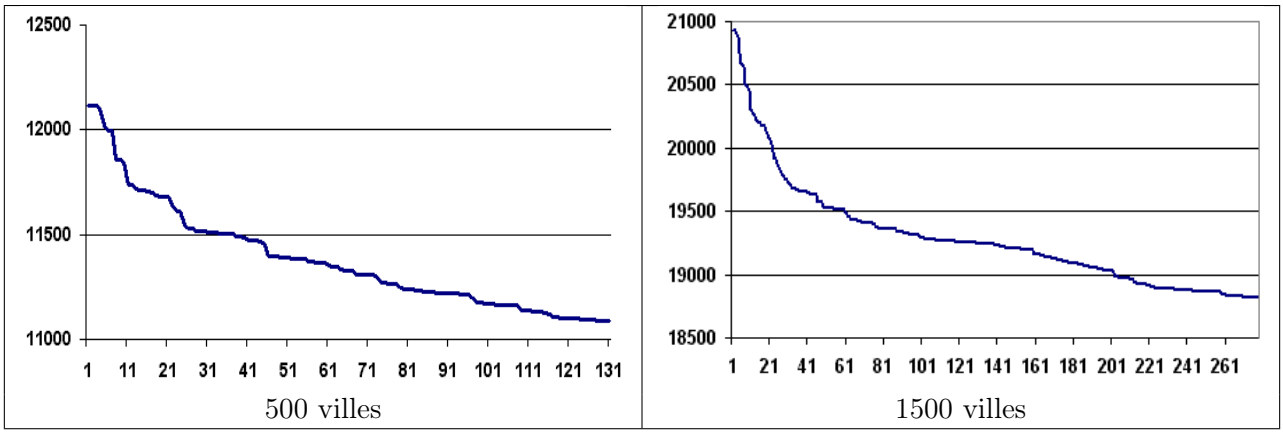



**Figure 23.15 :** *Exemples de décroissance de la longueur du chemin obtenue avec l'algorithme 23.16. L'essentiel des améliorations est faite dans la première moitié des itérations. Pour ces deux expériences, 500 villes, puis 1500 villes, ont été réparties aléatoirement dans un rectangle* $800 \times 500$.

### 23.2.8 Correction

Les paragraphes qui suivent ne reviennent pas sur les algorithmes présentés dans l'énoncé. Ils incluent deux programmes, le premier implémente la recherche du plus court chemin selon l'algorithme des cartes de Kohonen, le second implémente les méthodes utilisés par l'algorithme 23.10.

### 23.2.9 Carte de Kohonen

La figure 23.16 donne un exemple de résultat retourné par le programme qui suit et implémentant les cartes de Kohonen. Ce programme inclut des fonctions d'affichage (`display_ville`, `display_neurone`, `attendre_clic`), des fonctions qui effectuent des opérations simples sur un T-uple représentant les deux coordonnées d'un point ou d'un vecteur (`distance_euclidienne_carree`, `ajoute_vecteur`, `soustrait_vecteur`, `multiplie_vecteur`, `poids_attirance`, `vecteur_norme`).

Les trois fonctions importantes sont tout d'abord la paire `iteration` et `deplace_neurone` qui déplacent les neurones vers les villes ; puis la fonction `modifie_structure` qui ajoute ou supprime des neurones lorsque ceux-ci sont respectivement attirés par plusieurs villes à la fois ou attirés par aucune ville. Le programme principal suit l'instruction `if __name__ == "__main__"` : qui permet d'utiliser ce programme en tant que module si nécessaire par la suite.

```
# -*- coding: cp1252 -*-
import random                 # pour tirer aléatoirement des nombres
import math                   # fonctions cos, sin
import pygame                 # pour les affichages

def construit_ville(n, x =1000, y = 700):
    """tire aléatoirement n villes dans un carrée x * y, on choisit
```

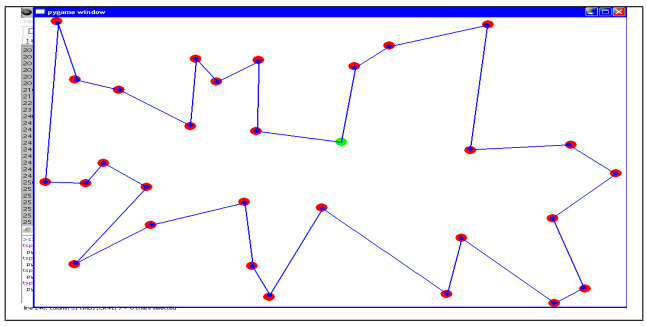


**Figure 23.16 :** *Résultat obtenu pour un ensemble de 30 villes, l'algorithme de Kohonen a convergé vers une solution.*

```
    ces villes de sortent qu'elles ne soient pas trop proches"""
    # deux villes ne pourront pas être plus proches que mind
    mind = math.sqrt (x*x+y*y) / (n * 0.75)
    # liste vide
    l = []
    while n > 0:
        # on tire aléatoirement les coordonnées d'une ville
        xx = x * random.random ()
        yy = y * random.random ()
        # on vérifie qu'elle n'est pas trop proche d'aucune autre ville
        ajout = True
        for t in l :
            d1 = t [0] - xx
            d2 = t [1] - yy
            d  = math.sqrt (d1*d1+d2*d2)
            if d < mind :
                ajout = False  # ville trop proche
        # si la ville n'est pas trop proche des autres, on l'ajoute à la liste
        if ajout:
            l.append ((xx,yy))
            n -= 1  # une ville en moins à choisir
    return l

def display_ville(villes,screen,bv):
    """dessine les villes à l'écran"""
    color  = 255,0,0
    color2 = 0,255,0
    for v in villes:
        pygame.draw.circle (screen, color, v, 10)
    pygame.draw.circle (screen, color2, villes [bv], 10)

def construit_liste_neurones (villes, nb = 0):
    """place les neurones sur l'écran,
    il y a autant de neurones que de villes,
    le paramètre villes est la liste des villes"""

    if nb == 0 : nb = len (villes)

    # coordonnées maximale
    maxx,maxy = 0,0
    for v in villes:
        if v [0] > maxx : maxx = v [0]
        if v [1] > maxy : maxy = v [1]

    maxx /= 2
    maxy /= 2

    if nb > 1 :
        # dispose les neurones en ellipse
```

```
        n = []
        for i in range (0, nb):
            x = maxx + maxx * math.cos (math.pi * 2 * float (i) / nb) / 4
            y = maxy + maxy * math.sin (math.pi * 2 * float (i) / nb) / 4
            n.append ((x,y))
        return n
    else :
        n = [ (maxx, maxy) ]
        return n

def distance_euclidienne_carree (p1,p2):
    """calcule la distance euclidienne entre deux points"""
    x = p1[0] - p2[0]
    y = p1[1] - p2[1]
    return x*x+y*y

def ajoute_vecteur (v,n):
    """ajoute deux vecteurs entre eux"""
    return ( v [0] + n [0], v [1] + n [1])

def soustrait_vecteur (v,n):
    """soustrait deux vecteurs"""
    return ( v [0] - n [0], v [1] - n [1])

def multiplie_vecteur (v,f):
    """multiplie un vecteur par un scalaire"""
    return ( v [0] * f, v [1] * f)

def poids_attirance(p,dist):
    """calcul le poids d'attraction d'une neurone vers une ville"""
    d  = p [0] * p [0] + p [1] * p [1]
    d  = math.sqrt (d)
    d  = dist / (d + dist)
    return d

def vecteur_norme(p):
    """calcul la norme d'un vecteur"""
    return math.sqrt (p [0] * p [0] + p [1] * p [1])

def deplace_neurone (n,villes,neurones, dist_w, forces, compte):
    """déplace le neurone de plus proche de la ville n, déplace ses voisins
        @param    villes        liste des villes
        @param    neurones      liste des neurones
        @param    dist          distance d'attirance
        @param    forces        force de déplacement des voisins du neurones
        @param    compte        incrémente compte [n] où n est l'indice du neurone choisi
        @return                 indice du neurone le plus proche"""

    # recherche du neurone le plus proche
    v       = villes [n]
    proche  = 0
    dist    = distance_euclidienne_carree(v, neurones [0])
    for i in xrange(1,len(neurones)):
        d = distance_euclidienne_carree (v, neurones [i])
        if d < dist :
            dist    = d
            proche  = i

    # vecteur de déplacement
    i             = proche
    compte [i]   += 1
    n             = neurones [i]
    vec           = soustrait_vecteur (v,n)
    poids         = poids_attirance (vec, dist_w)
    vec           = multiplie_vecteur (vec, poids)
```

```
    n             = ajoute_vecteur (n, vec)
    neurones [i]  = n

    # déplacement des voisins
    for k in xrange(0,len (forces)):
        i1  = (i + k + 1)   % len (neurones)
        i2  = (i - k - 1 + len (neurones)) % len (neurones)
        n1  = neurones [i1]
        n2  = neurones [i2]

        vec   = soustrait_vecteur (n, n1)
        poids = poids_attirance (vec, dist_w)
        vec   = multiplie_vecteur (vec, poids)
        vec   = multiplie_vecteur (vec, forces [k])
        n1    = ajoute_vecteur (n1, vec)

        vec   = soustrait_vecteur (n, n2)
        poids = poids_attirance (vec, dist_w)
        vec   = multiplie_vecteur (vec, poids)
        vec   = multiplie_vecteur (vec, forces [k])
        n2    = ajoute_vecteur (n2, vec)

        neurones [i1] = n1
        neurones [i2] = n2

    return proche

def iteration (villes,neurones, dist, forces, compte_v, compte_n):
    """choisit une ville aléatoirement et attire le neurones le plus proche,
    choisit cette ville parmi les villes les moins fréquemment choisies
        @param    villes     liste des villes
        @param    neurones   liste des neurones
        @param    dist       distance d'attirance
        @param    forces     force de déplacement des voisins du neurones
        @param    compte_v   incrémente compte_v [n] où n est l'indice de la ville choisie
        @param    compte_n   incrémente compte_n [n] où n est l'indice du neurone choisi
        @return              indices de la ville et du neurone le plus proche"""

    m             = min (compte_v)
    ind           = [i for i in xrange (0,len (villes)) if compte_v [i] == m]
    n             = random.randint (0,len(ind)-1)
    n             = ind [n]
    compte_v [n] += 1
    return n, deplace_neurone (n,villes,neurones,dist, forces, compte_n)

def display_neurone(neurones,screen,bn):
    """dessine les neurones à l'écran"""
    color  = 0,0,255
    color2 = 0,255,0
    for n in neurones :
        pygame.draw.circle (screen, color, n, 5)
    pygame.draw.circle (screen, color2, neurones [bn], 5)
    if len (neurones) > 1:
        pygame.draw.lines (screen, color, True, neurones, 2)

def modifie_structure(neurones,compte, nb_sel):
    """modifie la structure des neurones, supprime les neurones jamais
    déplacés, et ajoute des neurones lorsque certains sont trop sollicités"""

    def cmp_add (i,j):
        return cmp (i [0], j [0])

    if nb_sel > 0 :
        # supprime les neurones les moins sollicités
        sup = [i for i in xrange (0, len(neurones)) if compte [i] == 0 ]
```

```
        if len(sup) > 0:
            sup.sort ()
            sup.reverse ()
            for i in sup:
                del compte [i]
                del neurones [i]

        # on ajoute un neurone lorsque max (compte) >= 2 * min (compte)
        add = []
        for i in xrange (0,len(compte)):
            if compte [i] > nb_sel:
                d1     = math.sqrt ( distance_euclidienne_carree (neurones [i], \
                                neurones [(i+1) % len (neurones)] ))
                d2     = math.sqrt ( distance_euclidienne_carree (neurones [i], \
                                neurones [(i-1 + len (neurones)) % len (neurones)] ))
                if d1 > d2 : d1 = d2
                p      = neurones [i]
                p      = (p[0] + random.randint (0, int (d1 / 2)) , \
                                p [1] + random.randint (0, int (d1 / 2)) )
                add.append ((i,p,0))

        add.sort (cmp_add)
        add.reverse ()
        for a in add:
            neurones.insert (a [0], a [1])
            compte.insert (a [0], a [2])

    # on remet les compteurs à zéros
    for i in xrange (0, len (compte)) : compte [i] = 0

def moyenne_proximite (villes):
    """retourne la distance moyenne entre deux villes les plus proches"""
    c = 0
    m = 0
    for v in villes:
        mn = 100000000
        for vv in villes:
            if v == vv : continue
            d = distance_euclidienne_carree (v,vv)
            if d < mn : mn = d
        c += 1
        m += math.sqrt (mn)
    m /= float (c)
    return m

def attendre_clic (screen,x,y):
    """dessine une croix sur l'écran et attend la pression d'un clic de souris"""
    color   = 0,0,0
    pygame.draw.line (screen, color, (0,0), (x-1,y-1))
    pygame.draw.line (screen, color, (x-1,0), (0,y-1))
    pygame.display.flip ()
    reste = True
    while reste:
        for event in pygame.event.get():
            if event.type == pygame.MOUSEBUTTONUP :
                reste = False
                break

################################################################################

if __name__ == "__main__" :
    pygame.init ()
    size      = width, height = x,y = 800, 500
    black     = 0, 0, 0
    white     = 255,255,255
```

```
    screen    = pygame.display.set_mode(size)
    villes    = construit_ville (30, x,y)
    neurones  = construit_liste_neurones (villes, 3)
    compte_n  = [0 for i in neurones]
    compte_v  = [0 for i in villes]
    tour      = 2
    maj       = tour * len (villes)

    dist      = moyenne_proximite (villes) * 4
    fs        = [ 1.5, 1, 0.75, 0.5, 0.25 ]

    iter      = 0
    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT: sys.exit()
            if event.type == pygame.MOUSEBUTTONUP:
                attendre_clic (screen,x,y)

        iter += 1

        if iter % maj == 0 :
            modifie_structure (neurones,compte_n, tour)
            dist *= 0.99
            f2    = [ w * 0.90 for w in fs]
            fs    = f2

        bv, bn = iteration (villes,neurones, dist, fs, compte_v, compte_n)

        screen.fill (white)
        display_ville (villes, screen, bv)
        display_neurone (neurones, screen, bn)

        if iter == 1: attendre_clic (screen,x,y)

        pygame.display.flip ()
```

### 23.2.10 Construction à partir d'un arbre de poids minimum

Le programme suivant reprend les mêmes fonctionnalités que celles du programme précédent liées à l'affichage (`display_ville`, `display_neurone`, `attendre_clic`). Il y a toujours quelques manipulations de vecteurs (`vecteur_points`, `vecteur_norme`, `vecteur_cosinus`, `vecteur_sinus`, `oppose_vecteur`, `equation_droite`, `intersection_segment`).

Le programme utilise le module `bresenham_ligne` (voir paragraphe 27.3.2, page 708). Ce module permet de transcrire une droite reliant deux points par un ensemble de pixels formant une droite. Il est utilisé pour savoir quelles zones traverse un segment entre deux villes (voir figure 23.14). Les autres fonctions implémentent l'algorithme 23.10.

Les fonctions `repartition_zone`, `voisinage_zone`, `dessin_arete_zone`, `voisinage_zone_xy` manipulent des zones. Afin d'accélérer les algorithmes, l'image est quadrillée (voir figure 23.14). A chaque case du quadrillage correspond une zone qui recense toutes les villes incluses dans cette partie de l'image. On considère que deux villes sont voisines si elles appartiennent à la même zone et des zones voisines. Cet intermède permet ainsi de déterminer rapidement quels sont les voisins d'une ville en explorant seulement un sous-ensemble des villes. Implémenté à l'aide de zones, l'algorithme de recherche de l'arbre de poids minimum a un coût en $O(n)$ au lieu de $O(n^2)$ avec $n$ égal au nombre de villes.

La fonction `arbre_poids_minimal` détermine l'arbre de poids minimum (algorithme 23.13). La fonction `circuit_eulerien` implémente l'algorithme 23.14. La fonction `circuit_hamiltonien` implémente l'algorithme 23.15.

Les autres fonctions optimisent la solution obtenue après la construction du circuit hamiltonien. Les deux

fonctions `retournement_essai` et `echange_position_essai` implémentent un retournement et un déplacement, retourné ou non. Les trois fonctions `retournement_essai`, `echange_position`, `supprime_croisement`, testent plusieurs hypothèses de retournements ou déplacements. Enfin, la fonction `amelioration_chemin` utilise les trois précédentes pour implémenter l'algorithme 23.16.

Il est possible d'utiliser le module `psyco` pour aller plus vite. Les deux instructions `import psyco` et `pysco.full()` suffisent à multiplier la vitesse du programme par trois ou quatre.

```
# -*- coding: cp1252 -*-
import random                # pour tirer aléatoirement des nombres
import math                  # fonctions cos, sin
import pygame                # pour les affichages
import copy                   # pour copier des résultats intermédiaires
import bresenham_ligne as brl # pour dessiner les arêtes dans l'écran des zones
#import psyco                  # pour accélérer le programme

def attendre_clic (screen,x,y):
    """dessine une croix sur l'écran et attend la pression d'un clic de souris"""
    color   = 0,0,0
    pygame.draw.line (screen, color, (0,0), (x-1,y-1))
    pygame.draw.line (screen, color, (x-1,0), (0,y-1))
    pygame.display.flip ()
    reste = True
    while reste:
        for event in pygame.event.get():
            if event.type == pygame.MOUSEBUTTONUP :
                reste = False
                break


def construit_ville(n, x =1000, y = 700):
    """tire aléatoirement n villes dans un carrée x * y, on choisit
    ces villes de sortent qu'elles ne soient pas trop proches"""
    # deux villes ne pourront pas être plus proches que mind
    mind = math.sqrt (x*x+y*y) / (n * 0.75)
    # liste vide
    l = []
    while n > 0:
        # on tire aléatoirement les coordonnées d'une ville
        xx = x * random.random ()
        yy = y * random.random ()
        # on vérifie qu'elle n'est pas trop proche d'aucune autre ville
        ajout = True
        for t in l :
            d1 = t [0] - xx
            d2 = t [1] - yy
            d  = math.sqrt (d1*d1+d2*d2)
            if d < mind :
                ajout = False  # ville trop proche
        # si la ville n'est pas trop proche des autres, on l'ajoute à la liste
        if ajout:
            l.append ((xx,yy))
            n -= 1  # une ville en moins à choisir
    return l

def display_ville(villes,screen,bv):
    """dessine les villes à l'écran"""
    color  = 255,0,0
    color2 = 0,255,0
    for v in villes:
        pygame.draw.circle (screen, color, v, 3)
    pygame.draw.circle (screen, color2, villes [bv], 3)

def display_neurone(neurones,screen,bn):
    """dessine les neurones à l'écran"""
```

```
    color  = 0,0,255
    color2 = 0,255,0
    for n in neurones :
        pygame.draw.circle (screen, color, n, 3)
    pygame.draw.circle (screen, color2, neurones [bn], 3)
    if len (neurones) > 1:
        pygame.draw.lines (screen, color, True, neurones, 2)

def distance (p1,p2):
    """calcule la distance entre deux villes"""
    x = p1 [0] - p2[0]
    y = p1 [1] - p2[1]
    return math.sqrt (x*x + y*y)

def repartition_zone (villes, zone_taille, ask_zone = False) :
    """répartit les villes en zones, retourne les villes rangées par zones,
    chaque éléments zones [z][k] contient :
        - les coordonnées de la ville
        - ses coordonnées en zone, (zx, zy)
        - son indice dans la liste villes
    la fonction retourne également le nombre de zones
    selon l'axe des abscisses et l'axe des ordonnées,
    retourne aussi le nombre de zones, si ask_zone est True,
    retourne un paramètre supplémentaire : zone"""
    print "gestion des zones"
    X,Y = 0,0
    for v in villes:
        X = max (v [0] // zone_taille, X)
        Y = max (v [1] // zone_taille, Y)
    X += 1
    Y += 1

    # attribution des zones
    zone    = []
    nb      = len (villes)
    Zmax    = 0
    for i in xrange (0,len (villes)):
        v       = villes [i]
        x       = int (v [0] // zone_taille)
        y       = int (v [1] // zone_taille)
        z       = int (y * X + x)
        Zmax    = max (z,Zmax)
        zone.append ((z, v, (x,y), i))

    # rangement par zone
    Zmax   += 1
    zones = [ [] for i in xrange (0,Zmax) ]
    for z in zone:
        zones [ z [0] ].append ((z [1], z [2], z [3]))

    if ask_zone :
        return zones,X,Y,Zmax,zone
    else :
        return zones,X,Y,Zmax

def voisinage_zone (z,Zmax,X,Y):
    """retourne la liste des voisins d'une zone z
    sachant qu'il y a X zones sur l'axe des abscisses et Y zones sur l'axe des ordonnées,
    Zmax est le nombre de zones,
    inclus z dans cette liste"""
    x       = z % X
    y       = z // X
    voisin_ = [z]
    if x > 0: voisin_.append (z-1)
    if x < X: voisin_.append (z+1)
```

```
    if y > 0: voisin_.append (z-X)
    if y < Y: voisin_.append (z+X)
    if x > 0 and y > 0 : voisin_.append (z-1-X)
    if x > 0 and y < Y : voisin_.append (z-1+X)
    if x < X and y > 0 : voisin_.append (z+1-X)
    if x < X and y < Y : voisin_.append (z+1+X)
    voisin = [ int (i) for i in voisin_ if i >= 0 and i < Zmax ]
    return voisin

def arbre_poids_minimal(villes, zone_taille):
    """construit l'arbre de poids minimal, retourne une liste de
    listes, chaque sous-liste associée à une ville contient la liste des ses voisins,
    zone_taille permet de découper l'image en zones,
    les distances ne seront calculées que si
    deux éléments sont dans la même zone ou dans une zone voisine"""

    def tri_distance (u,v):
        if u [2] < v [2] : return -1
        elif u [2] > v [2] : return 1
        else : return 0

    zones,X,Y,Zmax = repartition_zone (villes, zone_taille)

    # calcul des distances
    print "calcul des distances"
    li      = []

    for z in xrange (0,len (zones)):
        if z % 1000 == 0 : print "zone ", z, len(zones)
        voisin  = voisinage_zone (z,Zmax,X,Y)
        for v in zones [z]:
            for zz in voisin:
                for u in zones [zz]:
                    d = distance (v [0], u [0])
                    li.append ((v [2], u [2], d))

    print "tri ", len(li)
    li.sort (tri_distance)

    # composantes connexes
    print "construction de l'arbre"

    # nombre de composantes connexes
    nb_comp     = len (villes)

    # indice de la composante d'une ville
    num_comp    = [ i for i in xrange(0,len(villes)) ]

    # liste des voisins pour chaque ville
    arbre       = [ [] for i in xrange(0,len(villes)) ]

    # liste des villes par composante connexe
    list_comp   = [ [i] for i in xrange(0,len(villes)) ]

    iii = 0
    for c in li:
        if iii % 10000 == 0 : print "arête ", iii, len(li)
        iii += 1
        i,j = c [0], c [1]
        if num_comp [i] != num_comp [j]:
            # on relie les villes i et j car elles appartiennent
            # à des composantes connexes différentes
            arbre [i].append (j)        # i est voisine de j
            arbre [j].append (i)        # j est voisine de i
            cl = num_comp [i]           # composante connexe restante
```

```
            ki = num_comp [j]           # composante connexe à aggréger à la précédente
            for k in list_comp [ki]:
                num_comp [k] = cl
                list_comp [cl].append (k)
            list_comp [ki] = []
            nb_comp -= 1                # une composante connexe en moins

        if nb_comp == 0:
            break  # il n'y a plus qu'une seule composante connexe, inutile de continuer

    return arbre

def display_arbre (villes, arbre, mult = 1):
    """dessine le graphe de poids minimal défini par arbre"""
    if mult == 2 :
        color = 0,255,0
        l = 4
    else :
        l = 1
        color = 0,0,255

    for i in xrange (0, len (villes)):
        for j in arbre [i]:
            v  = (villes [i][0] * mult, villes [i][1] * mult)
            vv = (villes [j][0] * mult, villes [j][1] * mult)
            pygame.draw.line (screen, color, v, vv, l)

def vecteur_points (p1,p2):
    """retourne le vecteur entre les points p1 et p2"""
    return ( p2 [0] - p1 [0], p2 [1] - p1 [1])

def vecteur_norme (vec):
    """retourne la norme d'un vecteur"""
    return math.sqrt (vec [0] * vec [0] + vec [1] * vec [1])

def vecteur_cosinus (vec1, vec2):
    """retourne le cosinus entre deux vecteurs, utilise le produit scalaire"""
    norm1 = vecteur_norme (vec1)
    norm2 = vecteur_norme (vec2)
    if norm1 == 0: return 1
    if norm2 == 0: return 1
    scal  = vec1 [0] * vec2 [0] + vec1 [1] * vec2 [1]
    return scal / (norm1 * norm2)

def vecteur_sinus (vec1, vec2):
    """retourne le sinus entre deux vecteurs, utilise le produit vectoriel"""
    norm1 = vecteur_norme (vec1)
    norm2 = vecteur_norme (vec2)
    if norm1 == 0:  return 0
    if norm2 == 0:  return 0
    scal  = vec1 [0] * vec2 [1] - vec1 [1] * vec2 [0]
    return scal / (norm1 * norm2)

def oppose_vecteur (vec):
    """retourne le vecteur opposé"""
    return (- vec [0], - vec [1])

def circuit_eulerien (villes, arbre):
    """définit un circuit eulérien, villes contient la liste des villes,
    tandis que arbre est une liste de listes, arbre [i] est la liste des villes
    connectées à la ville i,
    on suppose que arbre est un graphe de poids minimal non orienté,
    l'algorithme ne marche pas s'il existe des villes confondues"""

    # on choisit une ville qui est une extrémité et parmi celle-là on la choisit au hasard
```

```
    has = []
    for i in xrange (0,len (villes)):
        n = len (arbre [i])
        if n == 1: has.append (i)

    bm = random.randint (0, len (has) -1)
    bm = has [bm]

    # vecteur, le circuit eulérien contient
    # nécessairement 2 * len (villes) noeuds puisque c'est
    # le graphe eulérien d'un arbre de poids minimal non orienté
    vec     = (1,1)
    chemin  = [bm]
    while len (chemin) < 2 * len (villes)-1 :

        v = villes [bm]

        ma      = - math.pi - 1
        bvec    = vec
        opvec   = oppose_vecteur (vec)
        for k in xrange (0, len (arbre [bm])) :
            l     = arbre [bm][k]
            vec2  = vecteur_points (v, villes [l])
            if opvec == vec2 :
                angle = -math.pi
            else :
                cos   = vecteur_cosinus (vec, vec2)
                sin   = vecteur_sinus   (vec, vec2)
                angle = math.atan2 (sin, cos)
            if angle > ma :
                ma      = angle
                bl      = k
                bvec    = vec2

        b   = arbre [bm][bl]
        chemin.append (b)
        del arbre [bm][bl]  # on supprime l'arc pour ne plus l'utiliser
        bm  = b
        vec = bvec

    return chemin

def circuit_hamiltonien (chemin):
    """extrait un circuit hamiltonien depuis un circuit eurlérien"""
    nb      = max (chemin) + 1
    res     = []
    coche   = [ False for i in xrange (0, nb) ]
    for c in chemin :
        if coche [c] : continue
        res.append (c)
        coche [c] = True

    return res

def equation_droite (p1,p2) :
    """retourne l'équation d'une droite passant par p1 et p2,
    ax + by + c = 0, retourne les coefficients a,b,c"""
    vec = vecteur_points (p1,p2)
    a = vec [1]
    b = - vec [0]
    c = - a * p1 [0] - b * p1 [1]
    return a,b,c

def evaluation_droite (a,b,c,p):
    """l'équation d'une droite est : ax + by + c, retourne la valeur
```

```
    de cette expression au point p"""
    return a * p [0] + b * p [1] + c

def intersection_segment (p1,p2,p3,p4):
    """dit si les segments [p1 p2] et [p3 p4] ont une intersection,
    ne retourne pas l'intersection"""
    # équation de la droite (p1 p2)
    a1,b1,c1 = equation_droite (p1,p2)
    a2,b2,c2 = equation_droite (p3,p4)
    s1 = evaluation_droite (a2,b2,c2,p1)
    s2 = evaluation_droite (a2,b2,c2,p2)
    s3 = evaluation_droite (a1,b1,c1,p3)
    s4 = evaluation_droite (a1,b1,c1,p4)
    if s1 * s2 <= 0 and s3 * s4 <= 0 : return True
    else : return False

def longueur_chemin (chemin):
    """retourne la longueur d'un chemin"""
    s   = 0
    nb  = len (chemin)
    for i in xrange (0,nb) :
        s += distance ( chemin [i], chemin [ (i+1) % nb ])
    return s

def retournement_essai (chemin, i,j, negligeable = 1e-5):
    """dit s'il est judicieux de parcourir le chemin entre les sommets i et j
    en sens inverse, si c'est judicieux, change le chemin et retourne True,
    sinon, retourne False,
    si j < i, alors la partie à inverser est
    i, i+1, i+2, ..., len(chemin) -1, 0, 1, ..., j"""

    nb = len (chemin)
    if i == j : return False
    if j - i == -1 : return False
    if j - i - nb == -1 : return False

    ia = (i - 1 + nb) % nb
    ja = (j + 1 ) % nb
    # arcs enlevés
    d_ia_i = distance (chemin [ia], chemin [i])
    d_j_ja = distance (chemin [j],  chemin [ja])
    # arcs ajoutés
    d_ia_j = distance (chemin [ia], chemin [j])
    d_i_ja = distance (chemin [i],  chemin [ja])
    # amélioration ?
    d = d_ia_j + d_i_ja - d_j_ja - d_ia_i
    if d >= - negligeable : return False

    # si amélioration, il faut retourner le chemin entre les indices i et j
    jp = j
    if jp < i : jp = j + nb
    ip = i

    while ip < jp :
        i           = ip % nb
        j           = jp % nb
        ech         = chemin [i]
        chemin [i]  = chemin [j]
        chemin [j]  = ech
        ip = ip+1
        jp = jp-1

    return True

def retournement (chemin, taille) :
```

```
    """amélioration du chemin par un algorithme simple,
    utilise des retournements de taille au plus <taille>,
    retourne le nombre de modifications"""

    # traitement des petits retournements
    nb = len (chemin)
    nb_change = 1
    nbtout    = 0
    retour      = { }
    while nb_change > 0 :
        nb_change   = 0
        for t in xrange (1,taille+1):
            retour [t] = 0
            for i in xrange (0, nb) :
                j = (i + t) % nb
                b = retournement_essai (chemin, i, j)
                if b :
                    retour [t] += 1
                    nb_change  += 1
        nbtout += nb_change
    print "nombre de retournements %d longueur : \t %10.0f détail \t" \
                % (nbtout, longueur_chemin (chemin)), " détail : ", retour
    return nbtout

def echange_position_essai (chemin, a,b, x, inversion, negligeable = 1e-5):
    """échange la place des villes ka à kb pour les placer entre les villes i et i+1,
    si inversion est True, on inverse également le chemin inséré, si inversion est False,
    on ne l'inverse pas,
    si cela améliore, déplace les villes et retourne True, sinon, retourne False"""

    nb = len(chemin)
    xa = (x + 1) % nb
    ka = (a - 1 + nb) % nb
    kb = (b + 1 ) % nb

    if not inversion :

        if x == ka : return False
        if x == kb : return False
        if xa == ka : return False
        if b < a :
            if a <= x <= b + nb : return False
        elif a <= x <= b : return False
        if b < a :
            if a <= x + nb <= b + nb : return False
        elif a <= x + nb <= b : return False

        # arcs enlevés
        d_x_xa = distance (chemin [x],  chemin [xa])
        d_ka_a = distance (chemin [ka], chemin [a])
        d_b_kb = distance (chemin [b],  chemin [kb])
        # arcs ajoutés
        d_ka_kb = distance (chemin [ka], chemin [kb])
        d_x_a   = distance (chemin [x],  chemin [a])
        d_b_xa  = distance (chemin [b],  chemin [xa])

        d = d_ka_kb + d_x_a + d_b_xa - d_x_xa - d_ka_a - d_b_kb
        if d >= -negligeable : return False

        # villes à déplacer
        ech = []
        bp = b
        if bp < a : bp = b + nb
        for i in xrange (a,bp+1):
            ech.append (chemin [i % nb] )
```

```
        diff = bp - a + 1

        xp = x
        if xp < b : xp += nb

        for l in xrange (b+1,xp+1) :
            ll              = l % nb
            bp              = (a + l - b-1) % nb
            chemin [bp]     = chemin [ll]

        for l in xrange (0,len(ech)) :
            chemin [ (x+l-diff+1 + nb) % nb ] = ech [l]

        return True

    else :

        if x == ka : return False
        if x == kb : return False
        if xa == ka : return False
        if b < a :
            if a <= x <= b + nb : return False
        elif a <= x <= b : return False
        if b < a :
            if a <= x + nb <= b + nb : return False
        elif a <= x + nb <= b : return False


        # arcs enlevés
        d_x_xa = distance (chemin [x],  chemin [xa])
        d_ka_a = distance (chemin [ka], chemin [a])
        d_b_kb = distance (chemin [b],  chemin [kb])
        # arcs ajoutés
        d_ka_kb = distance (chemin [ka], chemin [kb])
        d_x_b   = distance (chemin [x],  chemin [b])
        d_a_xa  = distance (chemin [a],  chemin [xa])

        d = d_ka_kb + d_x_b + d_a_xa - d_x_xa - d_ka_a - d_b_kb
        if d >= -negligeable  : return False

        # villes à déplacer
        ech = []
        bp = b
        if bp < a : bp = b + nb
        for i in xrange (a,bp+1):
            ech.append (chemin [i % nb] )
        ech.reverse ()
        diff = bp - a + 1

        cop = copy.copy (chemin)

        xp = x
        if xp < b : xp += nb

        for l in xrange (b+1,xp+1) :
            ll              = l % nb
            bp              = (a + l - b-1) % nb
            chemin [bp]     = chemin [ll]

        for l in xrange (0,len(ech)) :
            chemin [ (x+l-diff+1 + nb) % nb ] = ech [l]

        return True

def dessin_arete_zone (chemin, taille_zone, X,Y):
```

```
    """retourne une liste de listes de listes,
    res [i][j] est une liste des arêtes passant près de la zone (x,y) = [i][j],
    si k in res [i][j], alors l'arête k,k+1 est dans la zone (i,j),
    X est le nombre de zones horizontalement, Y est le nombre de zones verticalement,
    taille_zone est la longueur du côté du carré d'une zone"""
    res = [ [ [] for j in xrange (0,Y+1) ]  for i in xrange (0,X+1) ]
    nb = len (chemin)
    for i in xrange (0,nb):
        a       = chemin [i]
        b       = chemin [(i+1) % nb]
        x1,x2   = int (a [0] // taille_zone), int (b [0] // taille_zone)
        y1,y2   = int (a [1] // taille_zone), int (b [1] // taille_zone)
        line = brl.trace_ligne (x1,y1,x2,y2)
        for x,y in line:
            res [x][y].append (i)
    return res

def voisinage_zone_xy (x,y,X,Y):
    """retourne la liste des voisins d'une zone (x,y)
    sachant qu'il y a X zones sur l'axe des abscisses et Y zones sur l'axe des ordonnées,
    inclus z dans cette liste"""
    voisin = [(x,y)]
    if x > 0   : voisin.append ((x-1,y))
    if x < X-1 : voisin.append ((x+1,y))
    if y > 0   : voisin.append ((x,y-1))
    if y < Y-1 : voisin.append ((x,y+1))
    if x > 0   and y > 0   : voisin.append ((x-1,y-1))
    if x > 0   and y < Y-1 : voisin.append ((x-1,y+1))
    if x < X-1 and y > 0   : voisin.append ((x+1,y-1))
    if x < X-1 and y < Y-1 : voisin.append ((x+1,y+1))
    return voisin

def echange_position (chemin, taille, taille_zone, X,Y, grande = 0.5) :
    """regarde si on ne peut pas déplacer un segment de longueur taille
    pour supprimer les arêtes les plus longues,
    au maximum <grande> longues arêtes,
    retourne le nombre de changement effectués,
    X est le nombre de zones horizontalement,
    Y est le nombre de zones verticalement,
    taille_zone est la longueur d'un côté du carré d'une zone"""

    nb = len(chemin)

    def tri_arete (x,y):
        """pour trier la liste l par ordre décroissant"""
        if x [2] < y [2] : return 1
        elif x [2] > y [2] : return -1
        else : return 0

    # list des arêtes triés par ordre décroissant
    la = []
    for i in xrange (0,nb) :
        im = (i+1) % nb
        la.append ( (i, im, distance (chemin [i], chemin [im]) ) )
    la.sort (tri_arete)

    # zone associé à chaque arête
    zone = dessin_arete_zone (chemin, taille_zone, X, Y)

    dseuil    = la [ int (nb * grande) ] [ 2 ]
    nbtout    = 0
    nb_change = 0
    iarete    = 0
    retour    = { }
    for t in xrange (1, taille+1): retour [t] = 0
```

```
while iarete < nb :
    nb_change   = 0
    arete       = la [iarete]
    iarete     += 1
    x           = arete [0]
    xm          = arete [1]
    a           = chemin [x]
    b           = chemin [xm]
    d           = distance ( a,b )
    if d < dseuil : break    # arête trop petite

    # zone traversée par la ligne
    x1,x2   = int (a [0] // taille_zone), int (b [0] // taille_zone)
    y1,y2   = int (a [1] // taille_zone), int (b [1] // taille_zone)
    ens     = brl.trace_ligne (x1,y1,x2,y2)
    ville   = []
    for k,l in ens:
        voisin = voisinage_zone_xy (k,l,X,Y)
        for u,v in voisin :
            ville.extend (zone [u][v])

    # on supprime les doubles
    ville.sort ()
    if len (ville) == 0 : continue
    sup = []
    mx = -1
    for v in ville:
        if v == mx : sup.append (v)
        mx = v
    for s in sup:
        ville.remove (s)

    # on étudie les possibilités de casser l'arête (x,xm) aux alentours des villes
    # comprise dans l'ensemble ville
    for t in xrange (1, taille+1):

        for i in ville:

            # on essaye d'insérer le sous-chemin (x- t + 1 + nb)  --> x
            # au milieu de l'arête i,i+1
            b = echange_position_essai (chemin, (x- t + 1 + nb) % nb,x, i, False)
            if b :
                nb_change  += 1
                retour [t] += 1
                continue

            # on essaye d'insérer le sous-chemin (xm+ t - 1)  --> xm
            # au milieu de l'arête i,i+1
            b = echange_position_essai (chemin, (xm + t - 1) % nb, xm, i, False)
            if b :
                nb_change  += 1
                retour [t] += 1
                continue

            # on essaye de casser l'arête x,xm en insérant
            # le sous-chemin i --> (i+t) % nb
            b = echange_position_essai (chemin, i, (i+t) % nb, x, False)
            if b :
                nb_change  += 1
                retour [t] += 1
                continue
            # idem
            b = echange_position_essai (chemin, i, (i+t) % nb, x, True)
            if b :
```

```
                    retour [t] += 1
                    nb_change  += 1
                    continue
                # idem
                b = echange_position_essai (chemin, (i-t+nb) % nb, i, x, False)
                if b :
                    nb_change  += 1
                    retour [t] += 1
                    continue
                # idem
                b = echange_position_essai (chemin, (i-t + nb) % nb, i, x, True)
                if b :
                    retour [t] += 1
                    nb_change  += 1
                    continue

        nbtout += nb_change

    print "nombre de déplacements %d longueur :   \t %10.0f détail \t" \
                    % (nbtout, longueur_chemin (chemin)), " détail : ", retour
    return nbtout

def supprime_croisement (chemin, taille_zone, X,Y) :
    """supprime les croisements d'arêtes,
    retourne le nombre de changement effectués,
    X est le nombre de zones horizontalement,
    Y est le nombre de zones verticalement,
    taille_zone est la longueur d'un côté du carré d'une zone"""

    nb = len(chemin)

    # zone associé à chaque arête
    zone    = dessin_arete_zone (chemin, taille_zone, X, Y)
    nbtout  = 0

    for i in xrange (0,nb) :
        im      = (i+1) % nb
        a       = chemin [i]
        b       = chemin [im]

        # zone traversée par la ligne
        x1,x2   = int (a [0] // taille_zone), int (b [0] // taille_zone)
        y1,y2   = int (a [1] // taille_zone), int (b [1] // taille_zone)
        ens     = brl.trace_ligne (x1,y1,x2,y2)
        ville   = []
        for k,l in ens:
            voisin = voisinage_zone_xy (k,l,X,Y)
            for u,v in voisin :
                ville.extend (zone [u][v])

        # on supprime les doubles
        ville.sort ()
        if len (ville) == 0 : continue
        sup = []
        mx = -1
        for v in ville:
            if v == mx : sup.append (v)
            mx = v
        for s in sup:
            ville.remove (s)

        nb_change = 0
        for v in ville :
            b = retournement_essai (chemin, i,v)
            if b :
```

```
                nb_change += 1
                continue
            b = retournement_essai (chemin, im,v)
            if b :
                nb_change += 1
                continue

        nbtout += nb_change

    print "nombre de croisements %d longueur :      \t %10.0f détail \t" \
                  % (nbtout, longueur_chemin (chemin))
    return nbtout

def amelioration_chemin (chemin, taille_zone, X, Y, taille = 10, screen = None):
    """amélioration du chemin par un algorithme simple,
    utilise des retournements de taille au plus taille,
    traite les arcs qui se croisent,
    traite les grands arcs, utilise un quadrillage de taille window,
    X est le nombre de zones horizontalement,
    Y est le nombre de zones verticalement,
    taille_zone est la longueur d'un côté du carré d'une zone"""

    white = 255,255,255

    #première étape rapide
    nb = 1
    while nb > 0:
        nb  = retournement (chemin, taille)
        if screen != None:
            screen.fill (white)
            display_neurone(chemin, screen, 0)
            pygame.display.flip ()

    # amélioration
    nb = 1
    while nb > 0:
        nb  = retournement (chemin, taille)
        if screen != None:
            screen.fill (white)
            display_neurone(chemin, screen, 0)
            pygame.display.flip ()
        nb += echange_position (chemin, taille / 2, taille_zone, X, Y)
        if screen != None:
            screen.fill (white)
            display_neurone(chemin, screen, 0)
            pygame.display.flip ()
        nb += supprime_croisement (chemin, taille_zone,X,Y)
        if screen != None:
            screen.fill (white)
            display_neurone(chemin, screen, 0)
            pygame.display.flip ()

        if screen != None :
            for event in pygame.event.get():
                if event.type == pygame.MOUSEBUTTONUP :
                    print "attendre......................................................"
                    attendre_clic (screen,0,0)
                    print "reprise"
                    break

################################################################################

if __name__ == "__main__" :
    pygame.init ()
    # pysco.full ()   # pour accélérer le programme, va trois à quatre fois plus vite
```

```
nb_ville  = 500
partie    = 10

size        = width, height = x,y = 800, 500
dec         = size [0]
black       = 0, 0, 0
white       = 255,255,255
screen      = pygame.display.set_mode(size)
villes      = construit_ville (nb_ville, x, y)
taille_zone = 20
X           = int (x // taille_zone)
Y           = int (y // taille_zone)

print "taille de l'image : ", size
print "nombre de villes : ", len(villes)

screen.fill (white)
display_ville (villes, screen, 0)
pygame.display.flip ()
attendre_clic(screen,0,0)

print "calcul"
window  = int (math.sqrt (x*y / len(villes)) * 2)
arbre   = arbre_poids_minimal (villes, window)
arbre2  = copy.deepcopy (arbre)

print "dessin" ,window
display_arbre (villes, arbre)
attendre_clic(screen,0,0)

print "circuit eulérien"
chemin = circuit_eulerien (villes ,arbre)

print "circuit hamiltonien"
neurone = circuit_hamiltonien (chemin)

print "dessin"
neurones = [ villes [i] for i in neurone ]
display_neurone(neurones, screen, 0)
attendre_clic(screen,0,0)

print "amelioration_chemin et dessin"
amelioration_chemin (neurones, taille_zone, X, Y, partie, screen)
screen.fill (white)
display_ville (villes, screen, 0)
display_neurone(neurones, screen, 0)
attendre_clic(screen,0,0)

# taille double
print "taille double"
size      = (size [0], size [1])
screen    = pygame.display.set_mode(size)
screen.fill (white)
#display_arbre (villes, arbre2, 2)
display_neurone (neurones, screen, 2)
attendre_clic(screen,0,0)
```

## 23.3 Classification à l'aide de plus proches voisins

### 23.3.1 Enoncé

La figure 23.17 représente un problème de classification classique. On dispose d'un nuage de points réparti en deux classes. Un nouveau point semblable aux précédents se présente, sa classe est inconnue. L'objectif est de lui attribuer une classe en utilisant le fait qu'on connaît la classe d'appartenance des autres points.

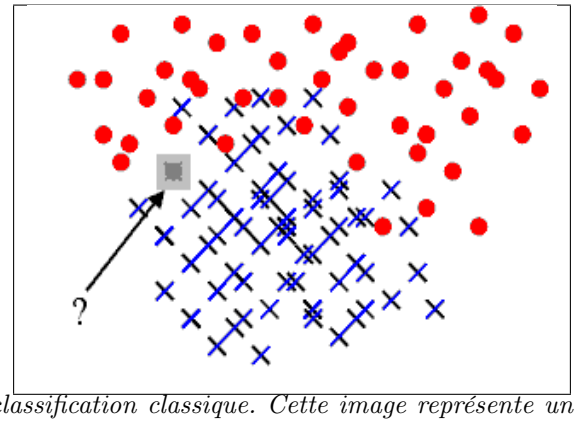



**Figure 23.17 :** *Problème de classification classique. Cette image représente un nuage de points, chacun d'entre eux appartient à l'une des deux classes, ici représentées par un cercle ou une croix. A partir d'un nuage de points pour lesquels la classe d'appartenance est connue, comment classer un nouveau point pour lequel cette classe est inconnue ?*

Une méthode simple consiste à attribuer à ce nouveau point la même classe que le plus proche des points appartenant au nuage initial. C'est la méthode des plus proches voisins (ou *nearest neighbours*). Elle est facile à implémenter mais peu utilisée car souvent très gourmande en temps de calcul lorsque le nuage de points est conséquent. Le premier paragraphe décrit cette méthode, les suivants cherchent à accélérer l'algorithme selon que le nuage de points appartient à un espace vectoriel ou non. La dernière partie présente l'algorithme LAESA pour le cas où le nuage de points appartient à un espace métrique quelconque.

### 23.3.2 Principe

Cette méthode est la plus simple puisqu'elle consiste à associer à $x$, l'élément à classer, le label $c(x_{i^*})$ de l'élément le plus proche $x_{i^*}$ dans l'ensemble $(x_1, \ldots, x_N)$. Ceci mène à l'algorithme de classification suivant :

> **Algorithme 23.20 : 1-PPV ou plus proche voisin**
> Soit $X = (x_1, \ldots, x_N) \subset E$ un ensemble d'éléments d'un espace métrique quelconque, soit $(c(x_1), \ldots, c(x_N))$ les classes associées à chacun des éléments de $X$. On note $d$ la distance définie sur l'espace métrique $E$. Soit $x$ un élément à classer, on cherche à déterminer la classe $\hat{c}(x)$ associée à $x$. On définit $x_{i^*}$ comme étant :
>
> $$x_{i^*} = \underset{i \in \{1, \cdots, N\}}{\arg\min} \; d(x_i, x)$$
>
> Alors : $\hat{c}(x) = c(x_i^*)$

Cet algorithme est souvent appelé *1-PPV* (ou *1-NN* pour Nearest Neighbors). Il existe une version amé-

liorée *k-PPV* qui consiste à attribuer à $x$ la classe la plus représentée parmi ses $k$ plus proches voisins.

**Algorithme 23.21 : k-PPV ou k-plus proches voisins**
Soit $X = (x_1, \ldots, x_N) \subset E$ un ensemble d'éléments d'un espace métrique quelconque, soit $(c(x_1), \ldots, c(x_N))$ les classes associées à chacun des éléments de $X$. On note $d$ la distance définie sur l'espace métrique $E$. $\omega(x, y)$ est une fonction strictement positive mesurant la ressemblance entre $x$ et $y$. Soit $x$ un élément à classer, on cherche à déterminer la classe $c(x)$ associée à $x$. On définit l'ensemble $S_k^*$ incluant les $k$-plus proches voisins de $x$, cet ensemble vérifie :

$$card(S_k^*) = 0 \text{ et } \max_{y \in S_k^*} d(y, x) \leqslant \min_{y \in X - S_k^*} d(y, x)$$

On calcule les occurrences $f(i)$ de chaque classe $i$ dans l'ensemble $S_k^*$ :

$$f(i) = \sum_{y \in S_k^*} \omega(x, y) \ \mathbb{1}_{\{c(y)=i\}} \tag{23.10}$$

On assigne alors à $x$ la classe $c(x)$ choisie dans l'ensemble :

$$\hat{c}(x) \in \arg\max_{i \in \mathbb{N}} f(i)$$

Dans sa version la plus simple, la fonction $\omega(x, y)$ utilisée lors du calcul de la contribution $f$ (23.10) est constante. Mais il est possible de lui affecter une valeur tenant compte de la proximité entre $x$ et $y$. La table 23.1 donne quelques exemples de contributions possibles.

| | |
|---|---|
| fonction constante | $\omega(x, y) = 1$ |
| distance inverse | $\omega(x, y) = \frac{1}{1+d(x,y)}$ |
| noyau | $\omega(x, y) = \exp\left(-d^2(x, y)\right)$ |

**Table 23.1 :** *Exemple de contribution $w(x, y)$ pour l'algorithme 23.21 des k-PPV. Ces fonctions sont toutes décroissantes (strictement ou non) par rapport à la distance $d$.*

L'inconvénient majeur de la méthode des plus proches voisins est sa longueur puisqu'elle implique le calcul des distances entre $x$ et chacun des éléments de l'ensemble $(x_1, \ldots, x_N)$. C'est pourquoi de nombreuses méthodes d'optimisation ont été développées afin d'accélérer ce processus. Les deux premiers paragraphes traitent le cas où les points $x_i$ appartiennent à un espace vectoriel et ont donc des coordonnées. Les suivant traitent le cas où les points $x_i$ n'ont pas de coordonnées et appartiennent à un espace métrique quelconque.

### 23.3.3 B+ tree

Ce premier algorithme s'applique dans le cas réel afin d'ordonner des nombres dans un arbre de sorte que chaque nœud ait un père et pas plus de $n$ fils (voir figure 23.18).

**Définition 23.22 : B+ tree**
Soit $B_n$ un B+ tree, soit $N$ un nœud de $B_n$, il contient un vecteur $V(N) = (x_1, \ldots, x_t)$ avec $0 \leqslant t \leqslant n$ et $x_1 < ... < x_t$. Ce nœud contient aussi exactement $t-1$ nœuds fils notés $(N_1, \ldots, N_{t-1})$. On désigne par $D(N_t)$ l'ensemble des descendants du nœud $N_t$ et $G(N_t) = \{V(M) \mid M \in D(N_t)\}$. Le nœud $N$ vérifie :

$$\forall x \in G(N_t),\ x_t \leqslant x < x_{t+1}$$
$$\text{avec par convention } x_0 = -\infty \text{ et } x_{t+1} = +\infty$$

Cet arbre permet de trier une liste de nombres, c'est une généralisation du tri "quicksort" pour lequel

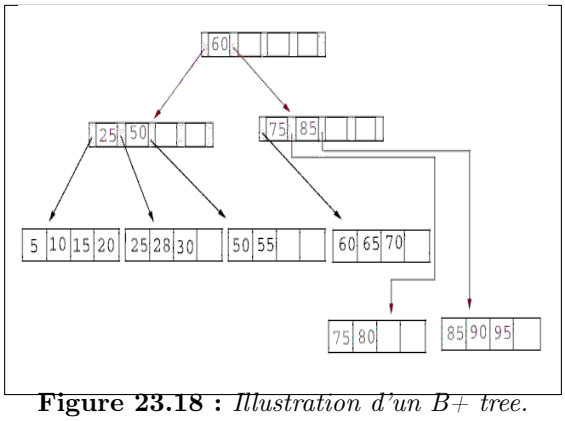



**Figure 23.18 :** *Illustration d'un B+ tree.*

$n = 2$. Comme pour le tri quicksort, l'arbre est construit à partir d'une série d'insertions et de cet ordre dépend la rapidité du tri. L'espérance du coût (moyenne sur tous les permutations possibles de $k$ éléments), le coût de l'algorithme est en $O(k \log_n k)$.

### 23.3.4 R-tree ou Rectangular Tree

L'arbre R-tree est l'adaptation du mécanisme du B+ tree au cas multidimensionnel (voir [Guttman1984]). La construction de cet arbre peut se faire de manière globale - construction de l'arbre sachant l'ensemble de points à classer - ou de manière progressive - insertion des points dans l'arbre les uns à la suite des autres -. Toutefois, ces méthodes sont resteintes à des espaces vectoriels.

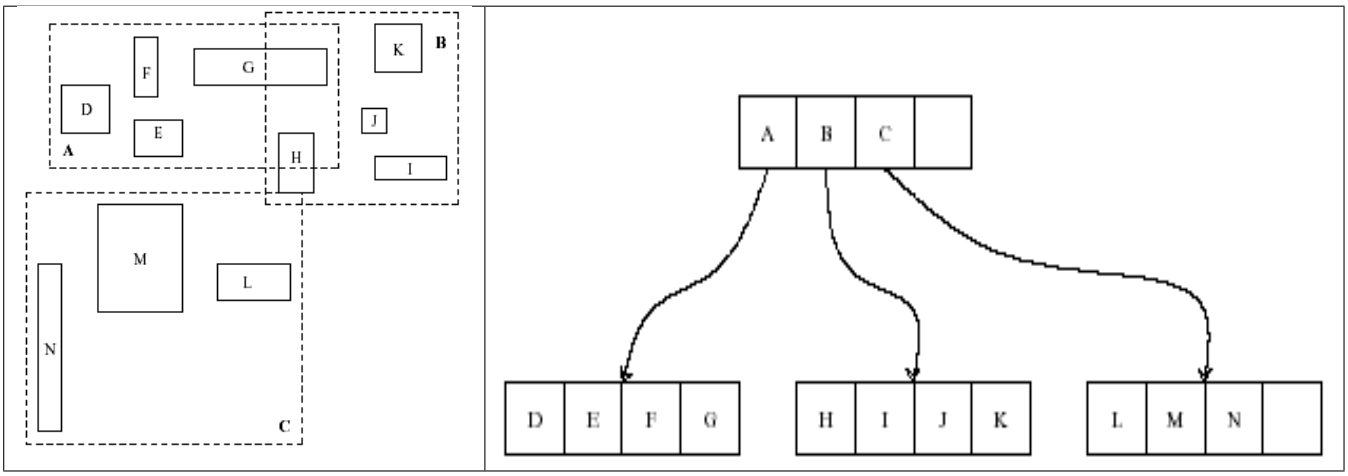



**Figure 23.19 :** *Illustration d'un R-tree en deux dimensions, figure extraite de [Sellis1987], la première image montre des rectangles pointillés englobant d'autres rectangles en trait plein. Chaque style de trait correspond à un niveau dans le graphe de la seconde image.*

Il n'existe pas une seule manière de construire un R-tree, les nœuds de ces arbres suivent toujours la contrainte des B+ Tree qui est d'avoir un père et au plus $n$ fils. Les R-Tree ont la même structure que les B+ Tree ôtée de leurs contraintes d'ordonnancement des fils. De plus, ces arbres organisent spatialement des rectangles ou boîtes en plusieurs dimensions comme le suggère la figure 23.19. Les boîtes à organiser seront nommés les objets, ces objets sont ensuite regroupés dans des boîtes englobantes. Un nœud $n$ d'un R-tree est donc soit une feuille, auquel cas la boîte qu'il désigne est un objet, dans ce cas, il n'a aucun fils, soit le nœud désigne une boîte englobante $B(n)$. On désigne par $\mathcal{B}$ l'ensemble des boîtes d'un espace vectoriel quelconque et $v(b)$ désigne son volume. Pour un nœud $n$ non feuille, $A(n)$ désigne l'ensemble des descendants de ce nœud. $B(n)$ est défini par :

$$B(n) = \arg\min \left\{ v(b) \mid b \in \mathcal{B} \text{ et } \forall n' \in A\left(n'\right),\ B\left(n'\right) \subset B(n) \right\}$$

La recherche dans un R-tree consiste à trouver tous les objets ayant une intersection avec une autre boîte ou fenêtre $W$, soit l'ensemble $L$ :

$$L = \{B(n) \mid B(n) \text{ est un objet et } B(n) \cap W \neq \emptyset\}$$

Cet ensemble est construit grâce à l'algorithme suivant :

**Algorithme 23.23 : recherche dans un R-tree**
Les notations sont celles utilisées dans ce paragraphe. On désigne par $r$ le nœud racine d'un R-tree. Soit $n$ un nœud, on désigne par $F(n)$ l'ensemble des fils de ce nœud.

**Etape A : initialisation**

$L \longleftarrow 0$
$N \longleftarrow \{r\}$

**Etape B : itération**

**tant que** $(N \neq \emptyset)$ **faire**
  **pour chaque** $n \in N$ **faire**
    **si** $W \cap B(n) \neq \emptyset$ **alors**
      $N \longleftarrow N \cup F(n)$
      **si** $B(n)$ **est un objet alors**
        $L \longleftarrow B(n)$
      **fin si**
    **fin si**
  **fin pour**
**fin tant que**

$L$ est l'ensemble cherché.

Il reste à construire le R-tree, opération effectuée par la répétition successive de l'algorithme 23.24 per-

mettant d'insérer un objet dans un R-tree.

**Algorithme 23.24 : insertion d'un objet dans un R-tree**
Les notations utilisées sont les mêmes que celles de l'algorithme 23.23. On cherche à insérer l'object $E$ désigné par son nœud feuille $e$. On suppose que l'arbre contient au moins un nœud, sa racine $r$. On désigne également par $p(n)$ le père du nœud $n$. Chaque nœud ne peut contenir plus de $s$ fils. On désigne par $v^*(G) = \min\left\{P \mid P \in \mathcal{B} \text{ et } \bigcup_{g\in G} B(g) \subset P\right\}$.

**Etape A : sélection du nœud d'insertion**
$n^* \longleftarrow r$
**tant que ($n^*$ n'est pas un nœud feuille) faire**
On choisit le fils $f$ de $n^*$ qui minimise l'accroissement $v_f - v(B(f))$ du volume avec $v_f$ défini par :

$$v_f = \min\{v(P) \mid P \in \mathcal{B} \text{ et } B(f) \cup B(e) \subset P\} \tag{23.11}$$

$n^* \longleftarrow f$
**fin tant que**

**Etape B : ajout du nœud**
Si $p(n^*)$ a moins de $s$ fils, alors le nœud $e$ devient le fils de $p(n^*)$ et $B(p(n^*))$ est mis à jour d'après l'expression (23.11). L'insertion est terminée. Dans le cas contraire, on sépare découpe le nœud $p(n^*)$ en deux grâce à l'étape suivante.

**Etape C : découpage des nœuds**
L'objectif est de diviser le groupe $G$ composé de $s+1$ nœuds en deux groupes $G_1$ et $G_1$. Tout d'abord, on cherche le couple $(n_1, n_2)$ qui minimise le critère

$$d = v^*(\{n_1, n_2\}) - v(B(n_1)) - v(B(n_2))$$

Alors : $G_1 \longleftarrow n_1$, $G_2 \longleftarrow n_2$ et $G \longleftarrow G - G_1 \cup G_2$
**tant que ($G \neq \emptyset$) faire**
On choisit un nœud $n \in G$, on détermine $i^*$ tel que $v(\{n\} \cup G_i) - v(G_i)$ soit minimal.
$G \longleftarrow G - \{n\}$
$G_{i^*} \longleftarrow G_{i^*} \cup \{n\}$
**fin tant que**

Si la recherche est identique quel que soit l'arbre construit, chaque variante de la construction de l'arbre tente de minimiser les intersections des boîtes et leur couverture. Plus précisément, l'étape C qui permet de découper les nœuds est conçue de manière à obtenir des boîtes englobantes de volume minimale et/ou d'intersection minimale avec d'autres boîtes englobantes. L'algorithme R+ Tree (voir [Sellis1987]) essaye de minimiser les intersections entre boîtes et les objets à organiser sont supposés n'avoir aucune intersection commune. La variante R* Tree (voir [Beckmann1990]) effectue un compromis entre l'intersection et la couverture des boîtes englobantes. L'algorithme X-Tree (voir [Berchtold1996]) conserve l'historique de la construction de l'arbre ce qui lui permet de mieux éviter les intersections communes entre boîtes. Ces techniques appartiennent à une classe plus larges d'algorithmes de type *Branch and Bound*.

### 23.3.5 LAESA

Cet algorithme permet de chercher les plus proches voisins dans un ensemble inclus dans un espace métrique quelconque. Il s'appuie sur l'inégalité triangulaire. L'algorithme LAESA (Linear Approximating Eliminating Search Algorithm, voir [Rico-Juan2003]) consiste à éviter un trop grand nombre de calculs de distances en se servant de distances déjà calculées entre les éléments de $E$ et un sous-ensemble $B$ inclus dans $E$ contenant des "pivots". La sélection des pivots peut être aléatoire ou plus élaborée comme celle

effectuée par l'algorithme qui suit, décrit dans l'article [Moreno2003].

**Algorithme 23.25 : LAESA : sélection des pivots**
Soit $E = \{y_1, \dots, y_N\}$ un ensemble de points, on cherche à déterminer l'ensemble $B = \{p_1, \dots, p_P\} \subset E$ utilisé par l'algorithme **??**.
**Etape A : initialisation**
$B \longleftarrow y \in E$ choisi arbitrairement.

**Etape B : calcul de la fonction** $g$
**pour chaque** $y \in E - B$ **faire**
$g(y) \longleftarrow 0$
**pour chaque** $p \in B$ **faire**
$g(y) \longleftarrow g(y) + d(y, p)$
**fin pour**
**fin pour**

**Etape C : mise à jour de** $B$
Trouver $p^* \in \arg\max \{g(p) \mid p \in E - B\}$
$B \longleftarrow B \cup \{p^*\}$
Si $card(B) < P$, retour à l'étape B sinon fin.

L'algorithme LAESA utilise les pivots pour diminuer le nombre de calculs en utilisant l'inégalité triangulaire. Par exemple, soit $x$ un élément à classer, $p_j$ un pivot, $y_i$ un point du nuage. On suppose qu'on connaît $d(x, p_j)$, $d(p_j, y_i)$ et $d^*$ la distance du point $x$ à un autre point du nuage. L'inégalité triangulaire permet d'affirmer que si : $d(x, y_i) \geqslant |d(x, p_j) - d(p_j, y_i)| > d^*$, alors il n'est pas nécessaire de calculer la distance $d(x, y_i)$ pour affirmer que $d(x, y_i) > d^*$. L'élément $y_i$ ne peut être l'élément le plus proche.

**Algorithme 23.26 : LAESA**
Soit $E = \{y_1, \dots, y_N\}$ un ensemble de points, $B = \{p_1, \dots, p_P\} \subset E$ un ensemble de pivots inclus dans $E$. On cherche à déterminer le voisinage $V(x)$ de $x$ inclus dans $E$ vérifiant :

$$\forall y \in V(x),\ d(x, y) \leqslant \rho$$

On suppose que la matrice $M = (m_{ij})_{\substack{\leqslant i \leqslant P \\ 1 \leqslant j \leqslant N}}$ a été calculée préalablement comme suit :

$$\forall (i, j),\ m_{ij} = d(p_i, y_j)$$

**Etape A : initialisation**
**pour** $i = 1$ **à** $P$ **faire**
$d_i \longleftarrow d(x, p_i)$
**fin pour**
$d^* \longleftarrow \min \{d_i \mid 1 \leqslant i \leqslant P\}$.
$d^*$ est la distance du point $x$ au pivot le plus proche.

**Etape B : recherche du plus proche élément**
$S \longleftarrow \emptyset$
**pour** $i = 1$ **à** $N$ **faire**
$d' \longleftarrow \max \{|d_j - m_{ji}|\}$
**si** $d' < d^*$ **alors**
$d \longleftarrow d(x, y_i)$
**si** $d' \leqslant d^*$ **alors**
$d^* \longleftarrow d'$
$S \longleftarrow \{y_i\}$
**fin si**
**fin si**
**fin pour**

### 23.3.6 Résultats théoriques

L'article [Faragó1993] démontre également qu'il existe une majoration du nombre moyen de calcul de distances pour peu que la mesure de l'espace contenant l'ensemble $E$ et l'élément $x$ soit connue et que l'ensemble $B = \{p_1, \ldots, p_P\}$ des pivots vérifie :

$$\begin{aligned} &\exists\,(\alpha,\beta) \in \mathbb{R}_*^+ \text{ tels que} \\ &\quad \forall\,(x,y) \in E^2,\ \forall i \quad \alpha\, d\,(x,y) \geqslant |d\,(x,p_i) - d\,(p_i,y)| \qquad (23.12) \\ &\quad \forall\,(x,y) \in E^2,\ \max_i\ |d\,(x,p_i) - d\,(p_i,y)| \geqslant \beta\, d\,(x,y) \qquad (23.13) \end{aligned}$$

L'algorithme développé dans [Faragó1993] permet de trouver le point de plus proche d'un élément $x$ dans

un ensemble $E = \{x_1, \ldots, x_N\}$ selon l'algorithme suivant :

**Algorithme 23.27 : plus proche voisin d'après [Faragó1993]**
Soit $E = \{x_1, \ldots, x_N\}$ et $B = \{p_1, \ldots, p_P\} \subset E \subset X$. Soit $x \in X$ un élément quelconque. On suppose que les valeurs $m_{ij} = d(x_i, p_j)$ ont été préalablement calculées.
**Etape A : initialisation**
On calcule préalablement les coefficients $\gamma(x_i)$ :

$$\forall i \in \{1, \ldots, N\},\ \gamma(x_i) \longleftarrow \max_{j \in \{1,\ldots,P\}} |m_{ij} - d(x, p_j)|$$

**Etape B : élaguage**
On définit $t_0 \longleftarrow \min_i \gamma(x_i)$.
Puis on construit l'ensemble $F(x) = \{x_i \in E \mid \gamma(x_i)\} \leqslant \frac{\alpha}{\beta} t_0$.

**Etape C : plus proche voisin**
Le plus proche $x^*$ voisin est défini par : $x^* \in \arg\min \{d(x, y) \mid y \in F(x)\}$.

**Théorème 23.28 : [Faragó1993]**[1]
Les notations sont celles de l'algorithme 23.27. L'algorithme 23.27 retourne le plus proche voisin $x^*$ de $x$ inclus dans $E$. Autrement dit, $\forall x \in X,\ x^* \in F(x)$.

**Théorème 23.29 : [Faragó1993]**[2]
Les notations sont celles de l'algorithme 23.27. On définit une mesure sur l'ensemble $X$, $B(x, r)$ désigne la boule de centre $x$ et de rayon $r$, $Z \in X$ une variable aléatoire, de plus :

$$p(x, r) = P_X(B(x, r)) = \mathbb{P}(Z \in B(x, r))$$

On suppose qu'il existe $d > 0$ et une fonction $f : X \longrightarrow \mathbb{R}$ tels que :

$$\lim_{r \to 0} \frac{p(x, r)}{r^d} = f(x) > 0$$

La convergence doit être uniforme et presque sûre. On note également $F_N$ le nombre de calculs de dissimilarité effectués par l'algorithme 23.27 où $N$ est le nombre d'élément de $E$, $P$ désigne toujours le nombre de pivots, alors :

$$\limsup_{n \to \infty} \mathbb{E}(F_N) \leqslant k + \left(\frac{\alpha}{\beta}\right)^{2d}$$

### 23.3.7 Correction

### 23.3.8 Programmes annexes

Ces deux programmes permettent d'obtenir un jeu de données permettant d'utiliser la méthode des plus proches voisins afin de reconnaître de chiffres manuscrits. Le premier programme `html_file.py` permet d'écrire un fichier au format HTML, il est utilisé afin de générer une page internet permettant de prendre connaissance des résultats de classification (ou de reconnaissance).

```
# -*- coding: cp1252 -*-
import os
import os.path
import PIL.Image

class html_file (object) :
    """cette classe propose quelques méthodes pour écrire un fichier HTML simplement"""

    header = """<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
<HEAD>
</HEAD>
<BODY >"""

    foot = """</BODY>
</HTML>
    """

    def __init__ (self, file) :
        """construction d'un fichier HTML"""
        self.file = file

    def open (self) :
        """ouverture du fichier"""
        self.f = open (self.file, "w")
        self.f.write (self.header)

    def close (self) :
        """fermeture du fichier"""
        self.f.close ()

    def text (self, s) :
        """écrit du texte dans un fichier HTML"""
        s = s.replace ("\n", "<BR>")
        self.f.write (s)

    def text_line (self) :
        """passe une ligne dans un fichier HTML"""
        self.f.write ("<BR>")

    def table_begin (self, nc) :
        """commence une table, nc est le nombre de colonnes"""
        self.f.write ("""<TABLE cellSpaceing="1" """)
        self.f.write ("""cellPadding="1" border="1">\n  <TR>\n    <TD>""")

    def table_end (self) :
        """termine une table"""
        self.f.write ("""    </TD>\n  </TR>\n</TABLE>\n""")

    def table_next (self) :
        """passe à la colonne suivante"""
        self.f.write ("""    </TD>\n    <TD>""")

    def table_next_line (self) :
        """passe à la ligne suivante"""
        self.f.write ("""    </TD>\n  </TR>\n  <TR>\n    <TD>""")

    def relative_path (self, url) :
        """trouve le chemin relatif par rapport au fichier"""
        pr = os.path.commonprefix ( [ self.file, url ] )
        if pr != None :
            d = os.path.dirname (self.file)
            if pr.count (d) == 1 :
                pr = d + "\\"
```

```
            url = url [ len (pr) : len (url) ]
        return url

    def clean_url (self, url) :
        """enlève les espaces et redresse les barres"""
        url = url.replace (" ", "\%20")
        url = url.replace ("\\", "/")
        return url

    def add_link (self, url, caption, isfile) :
        """ajoute un lien dans le fichier html,
        url est l'adresse, caption est l'intitulé du lien,
        si isfile vaut True, cherche un chemin relatif par rapport à self.file"""
        if isfile : url = self.relative_path (url)
        url = self.clean_url (url)
        s   = """<A HREF="%s">%s</A>""" % (url, caption)
        self.f.write (s)

    def add_image (self, image, size = None, zoom = None) :
        """ajoute une image dans le fichier html,
        celle-ci est enregistrée avec un numéro, offre la possibilité de zoomer
        ou de modifier la taille"""
        if not self.__dict__.has_key ("num_image") : self.num_image = 0
        name, ext = os.path.splitext (self.file)
        name      = name + str (self.num_image) + ".png"
        image.save (name)
        self.add_image_link (name, size, zoom)
        self.num_image += 1

    def add_image_link (self, url, size = None, zoom = None) :
        """ajoute une image dans le fichier html,
        cette image n'est pas recopiée, il est possible de spécifier une taille,
        ou de multiplier cette taille"""

        if size == None :
            im      = PIL.Image.open (url)
            size    = im.size
        if zoom != None :
            size = (int (size [0] * zoom), int (size [1] * zoom))

        url = self.relative_path (url)
        url = self.clean_url (url)

        s = """<IMG width=%d height=%d border=0 SRC="%s" alt="%s">""" \
                        % (size [0], size [1], url, url)
        self.f.write (s)


if __name__ == "__main__" :

    print "écriture du fichier ",file
    file = "c:\\temp\\essai.html"

    html = html_file (file)
    html.open ()
    html.text ("""première ligne d'un fichier HTML,
    seconde ligne""")
    html.table_begin (2)
    html.text ("1")
    html.table_next ()
    html.text ("2")
    html.table_next_line ()
    html.text("3")
    html.table_next ()
    html.text ("4")
```

```
    html.table_end ()
    html.add_link ("c:\\temp\\cours.py", "cours.py", True)
    html.text_line ()
    im = PIL.Image.new ("RGB", (100,100), color = (100,100,100))
    html.add_image (im)
    html.add_image (im, zoom = 2)
    html.close ()

    print "fin"
```

Le second fichier permet de lire la base de données (MNIST) regroupant les images de caractères. Celle-ci est disponible à l'adresse *http ://yann.lecun.com/exdb/mnist/*. La base MNIST est scindée en deux sous-bases d'apprentissage et de tests. La première sert à estimer les paramètres d'un système de classification, la seconde sert à évaluer les performances de ce système sur des données non apprises. Ce site internet référence des performances obtenues avec des méthodes classiques de classification telles que les réseaux de neurones, les Support Vector Machines (SVM), les plus proches voisins... Le programme `mnist.py` permet d'extraire les images de chiffres des fichiers présents sur ce site.

```
# -*- coding: cp1252 -*-
"""décomposition de la base d'image MNIST"""

import os
import struct
import PIL  # ligne à supprimer si on n'a pas besoin des images
import PIL.Image as PIM  # ligne à supprimer si on n'a pas besoin des images

def transcription_entier (str) :
    """convertit un entier lu en format binaire en entier"""
    t = struct.unpack ("BBBB", str)
    i = t [3] + t [2] * 256 + t [1] * 256 * 256 + t [0] * 256 * 256 * 256
    return i

def lire_image (fim, nl, nc) :
    """lit une image dans le fichier fim, cette image a nl lignes et nc colonnes,
    retourne l'image sous la forme d'un t-uple"""
    nb  = nl * nc
    str = fim.read (nb)
    t   = struct.unpack ("B" * nb, str)
    return t

def lire_image (fim, nl, nc) :
    """lit une image dans le fichier fim, cette image a nl lignes et nc colonnes,
    retourne l'image sous la forme d'un t-uple"""
    nb  = nl * nc
    str = fim.read (nb)
    t   = struct.unpack ("B" * nb, str)
    return t

def lire_label (fla) :
    """lit un label (un chiffre) dans le fichier fla, retourne un entier"""
    str = fla.read (1)
    t   = struct.unpack ("B", str)
    return t [0]

def binarisation (image, binar) :
    """binarisation d'une image"""

    res = []
    for i in image :
        if i < 255 - binar : res.append (255)
        else : res.append (0)
```

```
    return res

def write_image_matlab (fmat, image) :
    """écrit une image au format matlab, chaque nombre indique
    la position du prochain pixel noir,
    la suite est aussi longue qu'il y a de pixels dans l'image
    et se termine par une suite de 0 qui indique qu'il n'y a plus de pixels noirs"""
    nb = 0
    for i in xrange (0, len(image)) :
        if image [i] == 0 :
            fmat.write ("," + str (i+1))
            nb += 1
    for i in xrange (0, len (image) - nb) :
        fmat.write (",0")

def decompose_mnist (file_image, file_label, file_matlab, binar, \
                        nbcl = -1, dir_image1 = None, dir_image2 = None) :
    """décompose la base MNIST en un format utilisable sous MATLAB,
    commence par [, termine par ], chaque ligne est composée comme suit :
    premier nombre = classe, voir fonction write_image_matlab ;,
    nbcl est le nombre d'images à extraire par classe,
    si nbcl == -1, traite toute la base,
    binar est le seuil de binarisation, voir fonction binarisation,
    si dir_image1 != None, les images des nombres sont écrites,
    si dir_image2 != None, on continue d'explorer le fichier des images
    mais les images des nombres sont écrites dans le répertoire dir_image2"""

    fim = open (file_image, "rb")
    fla = open (file_label, "rb")

    s           = fla.read (4) # , (Hal_byte*) &umagic_an) ;
    magic_an    = transcription_entier (s)
    s           = fla.read (4) # , (Hal_byte*) &unb_an) ;
    nb_an       = transcription_entier (s)

    s           = fim.read (4) # , (Hal_byte*) &umagic_im) ;
    magic_im    = transcription_entier (s)
    s           = fim.read (4) # , (Hal_byte*) &unb_im) ;
    nb_im       = transcription_entier (s)
    s           = fim.read (4) #, (Hal_byte*) &unl_im) ;
    nl_im       = transcription_entier (s)
    s           = fim.read (4) #, (Hal_byte*) &unc_im) ;
    nc_im       = transcription_entier (s)

    if nbcl == -1 : nbcl = nb_im

    print "nombre de labels ", nb_an
    print "nombre magique label , vérification (2049) ", magic_an
    print "nombre d'images ", nb_im
    print "nombre magique image, vérification (2051) ", magic_im
    print "nombre de lignes ", nl_im
    print "nombre de colonnes ", nc_im

    if file_matlab != None :
        fmat = open (file_matlab, "wt")
        fmat.write ("[")

    label_count = { }

    nb      = 0
    cent    = int (nb_im / 100)
    while nb < nb_im :

        if nb % cent == 0 : print "avancement ", nb / cent, " %"
```

```
        nb += 1

        image = lire_image (fim, nl_im, nc_im)
        label = lire_label (fla)

        image = binarisation (image, binar)

        if not label_count.has_key (label) : label_count [label] = 0
        if 0 <= label_count [label] < nbcl :
            dir_image = dir_image1
        elif nbcl <= label_count [label] < nbcl * 2 :
            dir_image = dir_image2
        else :
            dir_image = None
            continue


        if file_matlab != None :
            fmat.write (str (label))
            write_image_matlab (fmat, image)
            fmat.write (";\n")

        if dir_image != None :
            nom     = "image_%d_%5d" % (label, label_count [label] % nbcl)
            nom     = dir_image + "\\" + nom + ".tif"
            surf    = PIM.new ("RGB", (nc_im, nl_im) )
            for i in xrange (0, len (image)) :
                x,y = i % nc_im, i // nc_im
                c   = image [i]
                surf.putpixel ((x,y), (c,c,c))

            surf.save (nom)

        label_count [label] += 1

    if file_matlab != None :
        fmat.write ("]")
        fmat.close ()

    fim.close ()
    fla.close ()

    key = label_count.keys ()
    key.sort ()
    for i in key :
        print "label ", i , " \t : ", label_count [i]

    return label_count


if __name__ == "__main__" :
        file_image  = "C:\\Downloads\\data_image\\mnist\\t10k-images.idx3-ubyte"
        file_label  = "C:\\Downloads\\data_image\\mnist\\t10k-labels.idx1-ubyte"
        file_matlab = "c:\\temp\\mnist2\\matlab.txt"
        dir_image   = None # "c:\\temp\\mnist2"
        nb          = 20
        binar       = 190

        decompose_mnist (file_image, file_label, file_matlab, \
                         binar, nb, dir_image1, dir_image2)
```

### 23.3.9 Programme de classification

Les deux premiers fichiers implémentent la méthode de classification des plus proches voisins dans le cas d'une reconnaissance de caractères. Aucune optimisation n'est effectuée. Le premier fichier `kppv.py` concrétise l'algorithme de classification, le second fichier `kppv_image.py` s'occupent de charger les images de caractères et d'utiliser le fichier précédent pour les classer, ce qui équivaut à les reconnaître.

Le premier fichier contient la classe `nuage_points`. La méthode `ppv` recherche le plus proches voisins dans le nuage `nuage`. La méthode `ppv_nuage` l'utilise pour classer tous les éléments d'un nuage qu'elle reçoit en paramètre. Pour utiliser, il faut créer une classe qui dérive de celle-ci afin de surcharger les méthodes `distance` et `label` dont le code dépend des éléments à classer.

```
# -*- coding: cp1252 -*-
import math
import random


class nuage_points (object) :
    """définit une classe de nuage de points"""

    def __init__ (self) :
        """constructeur"""
        self.nb = 0         # aucun élément
        self.nuage = []     # aucun élément

    def __iter__(self) :
        """retourne un itérateur sur l'ensemble de points"""
        return self.nuage.__iter__ ()

    def __len__ (self) :
        """retourne le nombre d'élément"""
        return len (self.nuage)

    def distance (self, obj1, obj2) :
        """retourne une distance entre deux éléments obj1 et obj2"""
        return 0

    def label (self, obj) :
        """retourne le label d'un objet"""
        return None

    def ppv (self, obj) :
        """retourne l'élément le plus proche de obj et sa distance avec obj"""
        mn = None
        dm = None
        for l in self.nuage :
            d = self.distance (l, obj)
            if dm == None or d < dm :
                dm = d
                mn = l
        return mn,dm

    def ppv_nuage (self, nuage, bienclasse = None, erreur = None) :
        """calcule un taux de classification pour un nuage,
        utilise la fonction ppv pour chaque élément de nuage,
        retourne le nombre d'élément bien classés, et mal classés,
        si bienclasse est une liste vide (!= None), cette liste contiendra
        la liste des couples d'éléments (x,y) in (nuage, self) pour lesquels
        la classification est bonne,
        si erreur est une liste vide (!= None), cette liste contiendra
        la liste des couples d'éléments (x,y) in (nuage,self)
        pour lesquels la classification est mauvaise"""
        disp    = len (nuage) / 10 ;
        good    = 0
```

```
        bad     = 0
        n       = 0
        for x in nuage :
            obj,d = self.ppv (x)
            if self.label (obj) == self.label (x) :
                good += 1
                if bienclasse != None : bienclasse.append ((obj,x))
            else :
                bad += 1
                if erreur != None : erreur.append ((obj, x))
            if n % disp == 0 : print "ppv_nuage ", n * 100 / len (nuage), "%"
            n += 1
        return good, bad
```

Le second fichier contient la classe `nuage_point_distance_label` qui dérive de `nuage_points` pour redéfinir les méthodes `distance` et pour des images de taille fixe 32x32. La distance entre deux images est le nombre de pixels qui différent entre elles. La seconde classe se contente de charger les images aux travers des méthodes `image` et `__init__`. La méthode `html_couple` construit un fichier HTML permettant de visionner les erreurs de reconnaissance.

Enfin, la méthode `ppv_nuage` appelle la classification d'un ensemble d'images test à l'aide d'un ensemble d'image apprentissage pour lesquelles la classe est supposée connue.

```
# -*- coding: cp1252 -*-
import os
import os.path
import mnist
import string
import PIL
import PIL.Image
import html_file as html
import psyco
import random
import kppv
import time

# définit une classe héritant de kppv.nuage_points qui surcharge les
# fonctions distance et label
class nuage_point_distance_label (kppv.nuage_points) :
    """hérite de kppv.nuage_point et surcharge les méthodes :
            - distance
            - label
    """

    def distance (self, obj1, obj2) :
        """surcharge de la fonction distance, comme les images sont toutes
        de dimensions identiques, on peut compter les pixels de couleurs différentes,
        le résultat est la distance entre deux images"""
        if len (obj1) != len (obj2) :
            print "erreur, len (obj1) != len (obj2)"
        d = 0
        for i in xrange (2, len (obj1)) :
            if obj1 [i] != obj2 [i] : d += 1
        return d

    def label (self, obj) :
        """retourne le label d'un objet"""
        return obj [0]


class nuage_image (object) :
    """nuage de points, chaque élément est une image,
```

```
    les images sont en noir et blanc, liste de 0 ou 1"""

    def __init__ (self, nuage_ppv, rep, nb, extension = "tif") :
        """initialise le nuage d'images avec un répertoire
        qui est l'emplacement des images pour ce nuage,
        le nom des images contient une chaîne de caractères suivi
        d'un label séparés par un blanc soulignés,
        extension est le type de images à charger,
        on ne considère que les nb premières images,
        nuage est une instance de la classe nuage_point ou nuage_point_laesa"""

        def binarise (p) :
            if p [0] == 0 : return 1
            else : return 0

        self.nb     = 0
        self.nuage  = []

        file = os.listdir (rep)
        nb   = min (nb, len (file))
        step = nb / 10
        n    = 0
        for f in file :
            if n >= nb : break
            if n % step == 0 : print "nuage_image, avancement ", n * 100 / nb, "%"
            n += 1
            ext = os.path.splitext (f)
            ext = ext [len (ext) - 1].lower ()
            ext = ext [1 : len (ext)]
            if ext != "tif" : continue
            s       = f.split ("_")
            label   = s [1]
            im      = PIL.Image.open (rep + "\\" + f)
            size    = im.size
            data    = [ binarise (im.getpixel ((x,y))) for y in xrange (0, size [1]) \
                                                       for x in xrange (0, size [0]) ]
            data.insert (0, size)
            data.insert (0, label)
            self.nuage.append ( tuple (data))

        self.ppv          = nuage_ppv
        self.ppv.nb       = self.nb
        self.ppv.nuage    = self.nuage


    def image (self, obj) :
        """reconstruit une image à partir d'un élément"""
        size    = obj [1]
        im      = PIL.Image.new ("RGB", size)
        nb      = len (obj) - 2
        for i in xrange (0, nb) :
            if obj [i] == 0 : im.putpixel ( (i % size [0], i // size [1]), (255,255,255))
            else : im.putpixel ((i % size [0], i // size [1]), (0,0,0))
        return im

    def html_couple (self, fichier, l, zoom = None) :
        """écrit un fichier html contenant toutes les images mal classées,
        à partir de la liste erreur construite par la méthode nuage_points.ppv_nuage"""

        # nombre de colonnes maximales
        maxc = 0
        for x in l : maxc = max (maxc, len (x))

        f = html.html_file (fichier)
        f.open ()
```

```
        f.table_begin (maxc*2+1)

        f.text ("indice")
        f.table_next ()
        f.text ("label")
        f.table_next ()
        f.text ("à classer")
        f.table_next ()
        for n in xrange (1, maxc) :
            f.text ("label")
            f.table_next ()
            f.text ("voisin " + str (n))
            f.table_next ()
        f.table_next_line ()

        n = 0
        for x in l :
            f.text (str (n))
            f.table_next ()
            n += 1
            for el in x :
                im = self.image (el)
                f.text (self.ppv.label (el))
                f.table_next ()
                f.add_image (im, zoom = zoom)
                f.table_next ()
            f.table_next_line ()

        f.table_end ()
        f.close ()

    def ppv_nuage (self, nuage, bienclasse = None, erreur = None) :
        """appelle self.nuage.ppv_nuage"""
        return self.ppv.ppv_nuage (nuage, bienclasse, erreur)

    def __len__ (self) :
        """retourne len (self.nuage)"""
        return len (self.ppv)

    def __iter__(self) :
        """retourne iter (self.nuage)"""
        return iter (self.ppv)



if __name__ == "__main__" :

    psyco.full ()

    rep_app     = "c:\\temp\\mnist\\app"
    rep_test    = "c:\\temp\\mnist\\test"
    nb_app      = 400
    nb_test     = 400
    html_file1   = "c:\\temp\\mnist\\bienclasse.html"
    html_file2   = "c:\\temp\\mnist\\erreur.html"

    # si les images n'existent pas
    if not os.path.exists (rep_app) or not os.path.exists (rep_test) :
        file_image  = "C:\\Downloads\\data_image\\mnist\\t10k-images.idx3-ubyte"
        file_label  = "C:\\Downloads\\data_image\\mnist\\t10k-labels.idx1-ubyte"
        os.makedirs (rep_app)
        os.makedirs (rep_test)
        binar = 220
        nbcl  = 40
        mnist.decompose_mnist (file_image, file_label, None, binar, nbcl, \
```

```
                              rep_app, rep_test)

    ppv1 = nuage_point_distance_label ()
    ppv2 = nuage_point_distance_label ()

    print "construction du nuage d'apprentissage"
    nuage_app = nuage_image (ppv1, rep_app, nb_app)
    print "nombre de points : ", len (nuage_app)

    print "construction du nuage de test"
    nuage_test = nuage_image (ppv2, rep_test, nb_test)
    print "nombre de points : ", len (nuage_test)

    print "résultat de la classification"
    erreur      = []
    bienclasse  = []
    temps1      = time.clock ()
    good,bad = nuage_app.ppv_nuage (nuage_test, bienclasse, erreur)
    temps2      = time.clock ()
    print "--------------------------------------------------------------------------------"
    print "temps de traitement en secondes ", temps2 - temps1
    print "good, bad = ", good, bad
    print "taux de classification  : ", float (good) / (good + bad)
    print "écriture du fichier ", html_file1
    nuage_app.html_couple (html_file1, bienclasse,   zoom = 4)
    print "écriture du fichier ", html_file2
    nuage_app.html_couple (html_file2, erreur,       zoom = 4)
```

### 23.3.10 Optimisation

Encore deux fichiers pour réaliser la méthode des plus proches voisins à l'aide de l'algorithme LAESA. Ils sont construits selon le même schéma que les deux fichiers précédents. Le premier contient la classe `nuage_point_laesa` qui hérite de `nuage_points`. Elle redéfinit la méthode `ppv` pour déterminer plus rapidement le plus proche voisins. La méthode `selection_pivots` sélectionne aléatoirement les pivots.

```
# -*- coding: cp1252 -*-
import math
import random
import kppv


class nuage_point_laesa (kppv.nuage_points) :
    """implémente l'algorithme des plus proches voisins,
    version LAESA"""

    def __init__ (self) :
        """construit la classe"""
        kppv.nuage_points.__init__ (self)

    def selection_pivots (self,nb) :
        """sélectionne nb pivots aléatoirements"""
        nb = min (nb, len (self.nuage))
        self.pivots = []
        while len (self.pivots) < nb :
            i = random.randint (0,len (self.nuage))
            if not self.nuage [i] in self.pivots :
                self.pivots.append (self.nuage [i])

        # on calcule aussi la distance de chaque éléments au pivots
        self.dist = []
```

```
        for el in self.nuage :
            dl = []
            for p in self.pivots :
                d = self.distance (el, p)
                dl.append (d)
            self.dist.append (dl)

    def ppv (self, obj) :
        """retourne l'élément le plus proche de obj et sa distance avec obj,
        utilise la sélection à l'aide pivots"""

        # initialisation
        dp  = [ self.distance (obj, p) for p in self.pivots ]

        # pivots le plus proche
        dm  = dp [0]
        im  = self.pivots [0]
        for i in xrange (0, len(self.pivots)) :
            d = dp [i]
            if d < dm :
                dm = d
                im = self.pivots [i]

        # améliorations
        for i in xrange (0, len (self.nuage)) :

            # on regarde si un pivots permet d'éliminer l'élément i
            calcul = True
            for j in xrange (0, len (self.pivots)) :
                d = abs (dp [j] - self.dist[i][j])
                if d > dm :
                    calcul = False
                    break


            # dans le cas contraire on calcule la distance
            if calcul :
                d = self.distance (obj, self.nuage [i])
                if d < dm :
                    dm = d
                    im = self.nuage [i]

        return im,dm
```

Le dernier fichier est construit exactement selon le même principe que `kppv_image.py`. La classe `nuage_image_leasa`. Elle redéfinit son constructeur pour appeler la sélection ds pivots.

```
# -*- coding: cp1252 -*-
import os
import os.path
import mnist
import string
import PIL
import PIL.Image
import html_file as html
import psyco
import random
import kppv_laesa
import time
import kppv_image

# définit une classe héritant de kppv.nuage_points qui surcharge les
```

```
# fonctions distance et label
class nuage_point_leasa_distance_label (kppv_laesa.nuage_point_laesa) :
    """hérite de kppv.nuage_point et surcharge les méthodes :
            - distance
            - label
    """

    def distance (self, obj1, obj2) :
        """surcharge de la fonction distance, comme les images sont toutes
        de dimensions identiques, on peut compter les pixels de couleurs différentes,
        le résultat est la distance entre deux images"""
        if len (obj1) != len (obj2) :
            print "erreur, len (obj1) != len (obj2)"
        d = 0
        for i in xrange (2, len (obj1)) :
            if obj1 [i] != obj2 [i] : d += 1
        return d

    def label (self, obj) :
        """retourne le label d'un objet"""
        return obj [0]


class nuage_image_leasa (kppv_image.nuage_image) :
    """nuage de points, chaque élément est une image,
    les images sont en noir et blanc, liste de 0 ou 1"""

    def __init__ (self, nuage_ppv, rep, nb, extension = "tif") :
        """initialise le nuage d'images avec un répertoire
        qui est l'emplacement des images pour ce nuage,
        le nom des images contient une chaîne de caractères suivi
        d'un label séparés par un blanc soulignés,
        extension est le type de images à charger,
        on ne considère que les nb premières images,
        nuage est une instance de la classe nuage_point ou nuage_point_laesa"""

        kppv_image.nuage_image.__init__(self, nuage_ppv, rep, nb, extension)
        self.ppv.selection_pivots (20)




if __name__ == "__main__" :

    psyco.full ()

    rep_app     = "c:\\temp\\mnist\\app"
    rep_test    = "c:\\temp\\mnist\\test"
    nb_app      = 400
    nb_test     = 400
    html_file1   = "c:\\temp\\mnist\\bienclasse.html"
    html_file2   = "c:\\temp\\mnist\\erreur.html"

    # si les images n'existent pas
    if not os.path.exists (rep_app) or not os.path.exists (rep_test) :
        file_image  = "C:\\Downloads\\data_image\\mnist\\t10k-images.idx3-ubyte"
        file_label  = "C:\\Downloads\\data_image\\mnist\\t10k-labels.idx1-ubyte"
        os.makedirs (rep_app)
        os.makedirs (rep_test)
        binar = 220
        nbcl  = 40
        mnist.decompose_mnist (file_image, file_label, None, binar, nbcl, \
                               rep_app, rep_test)
```

```
ppv1 = nuage_point_leasa_distance_label ()
ppv2 = nuage_point_leasa_distance_label ()

print "construction du nuage d'apprentissage"
nuage_app = nuage_image_leasa (ppv1, rep_app, nb_app)
print "nombre de points : ", len (nuage_app)

print "construction du nuage de test"
nuage_test = nuage_image_leasa (ppv2, rep_test, nb_test)
print "nombre de points : ", len (nuage_test)

print "résultat de la classification"
erreur      = []
bienclasse  = []
temps1      = time.clock ()
good,bad = nuage_app.ppv_nuage (nuage_test, bienclasse, erreur)
temps2      = time.clock ()
print "------------------------------------------------------------------------"
print "temps de traitement en secondes ", temps2 - temps1
print "good, bad = ", good, bad
print "taux de classification  : ", float (good) / (good + bad)
print "écriture du fichier ", html_file1
nuage_app.html_couple (html_file1, bienclasse,   zoom = 4)
print "écriture du fichier ", html_file2
nuage_app.html_couple (html_file2, erreur,       zoom = 4)
```

## 23.4 Optimisation numérique du second ordre

### 23.4.1 Enoncé

Lorsqu'on doit trouver le minimum d'une fonction définie sur un espace vectoriel, le premier réflexe consiste à dériver la fonction puis à l'annuler de manière à obtenir un système d'équation. Il reste ensuite à déterminer si les zéros de la dérivée correspondent à un minimum ou à un maximum. Cependant, il n'est pas toujours possible de résoudre le système d'équation obtenu. Par exemple, la fonction $f(x) = \cos(x)+e^x$ a pour dérivée $f'(x) = -\sin(x) = e^x$. Résoudre l'équation $f'(x) = 0$ est impossible.

C'est pourquoi il existe des méthodes de résolution approchées qui déterminent numériquement le minimum de la fonction. Il existe des algorithmes permettant de trouver une solution approchée à condition toutefois que la fonction à maximiser ou minimiser soit dérivable. Plusieurs variantes sont proposées regroupées sous le terme de *descente de gradient* ou *méthode de Newton*. Par la suite, on cherchera à minimiser une fonction $g$. Maximiser cette même fonction revient à minimiser la fonction $-g$.

### 23.4.2 Algorithme et convergence

Soit $g : \mathbb{R}^d \longrightarrow \mathbb{R}$ une fonction dérivable dont il faut trouver $\overset{*}{x} = \underset{x\in\mathbb{R}}{\arg\min}\ g(x)$. L'algorithme suivant propose une méthode permettant à partir d'un $x_0 \in \mathbb{R}^d$ quelconque de se rapprocher petit à petit du

minimum $x^*$ comme le montre le schéma 23.20 dans le cas où $g(x) = x^2$.

**Algorithme 23.30 : descente de gradient de Newton**
Soit $g : \mathbb{R}^d \longrightarrow \mathbb{R}$ une fonction dérivable et minorée. On cherche à déterminer le minimum de la fonction $g$. Soit $x_0 \in \mathbb{R}^d$. Soit $(\epsilon_t)_{t \geqslant 0}$ une suite réelle positive vérifiant :

$$\sum_{t=0}^{\infty} \epsilon_t = \infty \text{ et } \sum_{t=0}^{\infty} \epsilon_t^2 < \infty$$

**Etape A : initialisation**
$t \longleftarrow 0$

**Etape B : mise à jour**
$t \longleftarrow t+1$
$x_t \longleftarrow x_{t-1} - \epsilon_t \frac{\partial g}{\partial x}(x_t)$

**Etape C : condition d'arrêt**
Si $f(x_t) \approx f(x_{t-1})$, l'algorithme s'arrête, sinon on retourne à l'étape B.

On note $x_t$ l'abscisse à l'itération $t$.
On note $\dfrac{\partial g(x_t)}{\partial x}$ le gradient de $g(x) = x^2$.
L'abscisse à l'itération $t+1$ sera :
$x_{t+1} = x_t - \varepsilon_t \left[ \dfrac{\partial g(x_t)}{\partial x} \right]$
$\varepsilon_t$ est le pas de gradient à l'itération $t$.

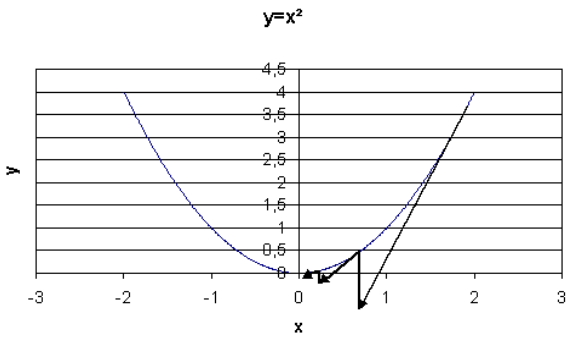



**Figure 23.20 :** *Minimisation par descente de gradient. A chaque itération, on se déplace dans le sens opposé à celui du gradient, direction le minimum est susceptible de se trouver.*

L'étape C détermine si l'algorithme doit continuer à chercher le minimum de la fonction $g$ ou si la valeur approchée est satisfaisante puisque, aux alentours de ce minimum, le gradient est presque nul et la suite $(x_t)$ presque constante. La condition $f(x_t) \approx f(x_{t-1})$ peut par exemple être interprétée comme une différence relative d'une itération à la suivante : $\left| \frac{f(x_t) - f(x_{t-1})}{f(x_{t-1})} \right| < A$ où $A$ est une constante positive petite. Plus elle est petite, plus la précision sera grande. Il est aussi possible d'arrêter l'algorithme dès que $f(x_t) > f(x_{t-1})$ mais la suite $(f(x_t))_t$ n'est pas toujours décroissante, c'est pourquoi la condition d'arrêt précédente est préférable.
La suite $(\epsilon_t)_t$ doit vérifier quelques contraintes comme la suite $\epsilon_t = \frac{\epsilon_0}{1+t}$. Ces contraintes assurent la

convergence de l'algorithme vers un minimum de la fonction comme le montre le théorème suivant.

**Théorème 23.31 : convergence de la méthode de Newton (Bottou1991)**
Soit une fonction continue $g : W \in \mathbb{R}^M \longrightarrow \mathbb{R}$, de classe $C^1$. On suppose les hypothèses suivantes vérifiées :

***H1*** $\underset{W\in\mathbb{R}^q}{\arg\min}\, g(W) = \{W^*\}$ est un singleton

***H2*** $\forall \varepsilon > 0, \inf_{|W-W^*|>\varepsilon} \left[(W - W^*)'.\nabla g(W)\right] > 0$

***H3*** $\exists (A, B) \in \mathbb{R}^2$ tels que $\forall W \in \mathbb{R}^p,\ \|\nabla g(W)\|^2 \leqslant A^2 + B^2 \|W - W^*\|^2$

***H4*** la suite $(\varepsilon_t)_{t\geqslant 0}$ vérifie, $\forall t > 0, \quad \varepsilon_t \in \mathbb{R}_+^*$ et $\sum_{t\geqslant 0} \varepsilon_t = +\infty, \quad \sum_{t\geqslant 0} \varepsilon_t^2 < +\infty$

Alors la suite $(W_t)_{t\geqslant 0}$ construite de la manière suivante :

$$W_0 \in \mathbb{R}^M \text{ et } \forall t \geqslant 0,\ W_{t+1} = W_t - \varepsilon_t \nabla g(W_t)$$

vérifie $\lim_{t\longrightarrow+\infty} W_t = W^*$

L'hypothèse *H1* implique que le minimum de la fonction $g$ est unique et l'hypothèse *H2* implique que le demi-espace défini par l'opposé du gradient contienne toujours le minimum de la fonction $g$.

**Démonstration (théorème 23.31) :**

**Partie A (démonstration de 23.31)**

Soit la suite $u_t = \ln\left(1 + \varepsilon_t^2 x^2\right)$ avec $x \in \mathbb{R}$, comme $\sum_{t\geqslant 0} \varepsilon_t^2 < +\infty,\ u_t \sim \varepsilon_t^2 x^2$, on a $\sum_{t\geqslant 0} u_t < +\infty$

Par conséquent, si $v_t = e^{u_t}$ alors $\prod_{t=1}^{T} v_t \overset{T\to\infty}{\longrightarrow} D \in \mathbb{R}$

**Partie B (démonstration de 23.31)**

On pose $h_t = \|W_t - W^*\|^2$
Donc :

$$h_{t+1} - h_t = \|W_t - \varepsilon_t \nabla g(W_t) - W^*\|^2 - \|W_t - W^*\|^2 \tag{23.14}$$

Par conséquent :

$$h_{t+1} - h_t = -2\varepsilon_t \underbrace{(W_t - W^*)' \nabla g(W_t)}_{>0} + \varepsilon_t^2 \|\nabla C(W_t)\|^2 \leqslant \varepsilon_t^2 \|\nabla g(W_t)\|^2 \leqslant \varepsilon_t^2 \left(A^2 + B^2 h_t\right)$$

D'où :

$$h_{t+1} - h_t\left(1 + \varepsilon_t^2 B^2\right) \leqslant \varepsilon_t^2 A^2$$

On pose $\pi_t = \prod_{k=1}^{t} \left(1 + \varepsilon_k^2 B^2\right)^{-1}$ alors, en multipliant des deux côtés par $\pi_{t+1}$, on obtient :

$$\pi_{t+1}h_{t+1} - \pi_t h_t \leqslant \varepsilon_t^2 A^2 \pi_{t+1}$$
$$\text{d'où } \pi_{q+1}h_{q+1} - \pi_p h_p \leqslant \sum_{t=p}^{q} \varepsilon_t^2 A^2 \pi_{t+1} \leqslant \sum_{t=p}^{q} \varepsilon_t^2 A^2 \Pi \leqslant \sum_{t=p}^{q} \varepsilon_t^2 A^2 \Pi \underset{t \longrightarrow \infty}{\longrightarrow} 0$$

Comme la série $\sum_t (\pi_{t+1}h_{t+1} - \pi_t h_t)$ vérifie le critère de Cauchy, elle est convergente. Par conséquent :

$$\lim_{q \to \infty} \pi_{q+1} h_{q+1} = 0 = \lim_{q \to \infty} \Pi h_{q+1}$$

D'où :

$$\lim_{q \to \infty} h_q = 0 \tag{23.15}$$

**Partie C (démonstration de 23.31)**

La série $\sum_t (h_{t+1} - h_t)$ est convergente car $\Pi h_t \sim \pi_t h_t$.

$\sum_{t \geqslant 0} \varepsilon_t^2 \| \nabla g(W_t) \|^2$ l'est aussi (d'après H3).

D'après (23.14), la série $\sum_{t \geqslant 0} \varepsilon_t (W_t - W^*)' \nabla g(W_t)$ est donc convergente. Or d'après les hypothèses (H2, H4), elle ne peut l'être que si :

$$\lim_{t \to \infty} W_t = W^* \tag{23.16}$$

**(23.31)** □

Ce théorème peut être étendu dans le cas où la fonction $g$ n'a plus un seul minimum global mais plusieurs minima locaux (voir [Bottou1991]), dans ce cas, la suite $(W_t)$ converge vers un mimimum local. Une généralisation de ce théorème est présentée dans [Driancourt1996].

Si ce théorème prouve la convergence de la méthode de Newton, il ne précise pas à quelle vitesse cette convergence s'effectue et celle-ci peut parfois être très lente. Plusieurs variantes ont été développées regroupées sous le terme de méthodes de quasi-Newton, ou méthodes du second ordre, dans le but d'améliorer la vitesse de convergence.

### 23.4.3 Méthode du second ordre

L'algorithme 23.30 fournit le canevas des méthodes d'optimisation du second ordre. Seule la mise à jour des coefficients (étape B) est différente : elle prend en compte les dernières valeurs des coefficients ainsi que les derniers gradients calculés. Ce passé va être utilisé pour estimer une direction de recherche pour le minimum différente de celle du gradient, cette direction est appelée *gradient conjugué* (voir [Moré1977]).

La figure 23.21 est couramment employée pour illustrer l'intérêt des méthodes d'optimisation du second ordre ou méthode de gradient conjugué. Le problème consiste à trouver le minimum d'une fonction quadratique, par exemple, $G(x, y) = 3x^2 + y^2$. Tandis que le gradient est orthogonal aux lignes de niveaux de la fonction $G$, le gradient conjugué se dirige plus sûrement vers le minimum global.

Ces techniques sont basées sur une approximation du second degré de la fonction à minimiser. On note toujours $g : \mathbb{R}^d \longrightarrow \mathbb{R}$ la fonction à minimiser. Au voisinage de $x_0$, un développement limité donne :

$$g(x) = g(x_0) + \frac{\partial g(x_0)}{\partial x}(x - x_0) + (x - x_0)' \frac{\partial^2 g(x_0)}{\partial x^2}(x - x_0) + o\|x - x_0\|^2$$

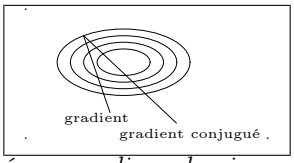



**Figure 23.21 :** *Gradient et gradient conjugué sur une ligne de niveau de la fonction $G(x,y) = 3x^2 + y^2$, le gradient est orthogonal aux lignes de niveaux de la fonction G, mais cette direction est rarement orientée vers le minimum à moins que le point $(x,y)$ se situe sur un des axes des ellipses, le gradient conjugué agrège les derniers déplacements et propose une direction de recherche plus plausible pour le minimum de la fonction.*

Par conséquent, sur un voisinage de $x_0$, la fonction $g(x)$ admet un minimum local si $\frac{\partial^2 g(x_0)}{\partial x^2}$ est définie positive strictement [5]. Une matrice symétrique définie strictement positive est inversible, et le minimum est atteint pour la valeur :

$$x_{\min} = x_0 + \frac{1}{2}\left[\frac{\partial^2 g(x_0)}{\partial x^2}\right]^{-1}\left[\frac{\partial g(x_0)}{\partial x}\right] \tag{23.17}$$

Néanmoins, pour un réseau de neurones, le calcul de la dérivée seconde est coûteux, son inversion également. C'est pourquoi les dernières valeurs des coefficients et du gradient sont utilisées afin d'approcher cette dérivée seconde ou directement son inverse. Une méthode couramment employée est l'algorithme BFGS (Broyden-Fletcher-Goldfarb-Shano, voir [Broyden1967], [Fletcher1993]).

Ces méthodes proposent une estimation de la dérivée seconde (ou de son inverse) utilisée en (23.17). Dans les méthodes du premier ordre, une itération permet de calculer les poids $x_{t+1}$ à partir des poids $x_t$ et du gradient $g_t$. Si ce gradient est petit, on peut supposer que $g_{t+1}$ est presque égal au produit de la dérivée seconde par $g_t$. Cette relation est mise à profit pour construire une estimation de la dérivée seconde. Cette matrice notée $B_t$ dans l'algorithme 23.32 est d'abord supposée égale à l'identité puis actualisée à chaque itération en tenant de l'information apportée par chaque déplacement.

**Algorithme 23.32 : algorithme BFGS**
Soit $g : \mathbb{R}^d \longrightarrow \mathbb{R}$ une fonction dérivable et minorée. On cherche à déterminer le minimum de la fonction $g$. Soit $x_0 \in \mathbb{R}^d$. Le nombre de paramètres de la fonction $f$ est $d$. $n$ est un entier strictement positif.

**Etape A : initialisation**
Le premier jeu de coefficients $x_0$ du réseau de neurones est choisi aléatoirement.
$t \longleftarrow 0$
$E_0 \longleftarrow f(x_0)$
$B_0 \longleftarrow I_d$
$i \longleftarrow 0$

**Etape B : calcul du gradient**
$g_t \longleftarrow \frac{\partial f}{\partial x}(x_t)$
$c_t \longleftarrow B_t g_t$

5. **Rappel :** $\frac{\partial^2 g(x_0)}{\partial x^2}$ est définie positive strictement $\Longleftrightarrow \forall Z \in \mathbb{R}^N,\ Z \neq 0 \Longrightarrow Z' \frac{\partial^2 g(x_0)}{\partial x^2} Z > 0$

**Etape C : recherche de $\epsilon^*$**
$\epsilon^* \longleftarrow \epsilon_0$
**faire**
$\epsilon^* \longleftarrow \frac{\epsilon^*}{2}$
$x_{t+1} \longleftarrow x_t - \epsilon^* c_t$
**tant que** ($f_{x_{t+1}} \geqslant E_t$ **et** $\epsilon^* \gg 0$)
**si** $\epsilon_* \approx 0$ **et** $B_t \neq I_d$ **alors**
$B_t \longleftarrow I_d$
$i \longleftarrow t$
retour à l'étape B.
**fin si**

**Etape D : mise à jour des coefficients**
$x_{t+1} \longleftarrow x_t - \epsilon^* c_t$
$E_{t+1} \longleftarrow f(x_{t+1})$
$t \longleftarrow t+1$

**Etape E : mise à jour de la matrice $B_t$**
**si** $t - i \geqslant nd$ **ou** $g'_{t-1} B_{t-1} g_{t-1} \leqslant 0$ **ou** $g'_{t-1} B_{t-1} (g_t - g_{t-1}) \leqslant 0$ **alors**
$B_t \longleftarrow I_d$
$i \longleftarrow t$
**sinon**
$s_t \longleftarrow x_t - x_{t-1}$
$d_t \longleftarrow g_t - g_{t-1}$
$$B_t \longleftarrow B_{t-1} + \left(1 + \frac{d'_t B_{t-1} d_t}{d'_t s_t}\right) \frac{s_t s'_t}{s'_t d_t} - \frac{s_t d'_t B_{t-1} + B_{t-1} d_t s'_t}{d'_t s_t}$$
**fin si**

**Etape F : terminaison**
si $\frac{E_t}{E_{t-1}} \approx 1$ alors l'apprentissage a convergé sinon retour à l'étape B.

Lorsque la matrice $B_t$ est égale à l'identité ($I_d$), le gradient conjugué est égal au gradient. Au fur et à mesure des itérations, cette matrice toujours symétrique évolue en améliorant la convergence de l'optimisation. Néanmoins, la matrice $B_t$ doit être "nettoyée" (égale à l'identité) fréquemment afin d'éviter qu'elle n'agrège un passé trop lointain. Elle est aussi nettoyée lorsque le gradient conjugué semble trop s'éloigner du véritable gradient et devient plus proche d'une direction perpendiculaire.

L'algorithme DFP (Davidon-Fletcher-Powell, voir [Davidon1959], [Fletcher1963]) est aussi un algorithme de gradient conjugué qui propose une approximation différente de l'inverse de la dérivée seconde. Pour appliquer cette méthode, il suffit de remplacer la mise à jour de la matrice $B_t$ dans l'étape E de l'algorithme 23.32 par la suivante :

$$B_t \longleftarrow B_{t-1} + \frac{d_t d'_t}{d'_t s_t} - \frac{B_{t-1} s_t s'_t B_{t-1}}{s'_t B_{t-1} s_t} \tag{23.18}$$

### 23.4.4 Minimisation avec gradient stochastique

La méthode du gradient stochastique s'applique lorsque la fonction à minimiser est une somme de fonctions dérivables et minorées.

$$g(x) = \sum_{i=1}^{N} e_i(x) \tag{23.19}$$

Compte tenu des courbes d'erreurs très "accidentées" (figure 23.22) dessinées par ces fonctions, il existe une multitude de minima locaux. De ce fait, l'apprentissage global converge rarement vers le minimum global de la fonction d'erreur lorsqu'on applique les algorithmes basés sur le gradient global. L'optimisation avec gradient stochastique est une solution permettant de mieux explorer ces courbes d'erreurs.

De plus, les méthodes de gradient conjugué nécessite le stockage d'une matrice trop grande parfois pour des fonctions ayant quelques milliers de paramètres ($g : \mathbb{R}^{1000} \longrightarrow \mathbb{R}$). C'est pourquoi l'apprentissage avec gradient stochastique est souvent préféré à des méthodes du second ordre trop coûteuses en calcul et en espace mémoire. En contrepartie, la convergence est plus lente. La démonstration de cette convergence nécessite l'utilisation de quasi-martingales et est une convergence presque sûre (voir [Bottou1991]).

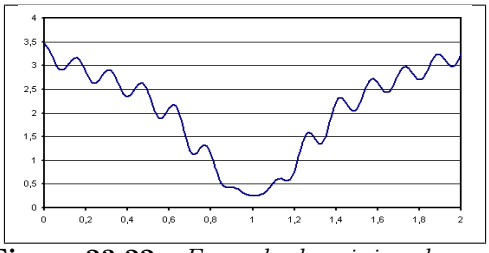



**Figure 23.22 :** *Exemple de minima locaux.*

**Algorithme 23.33 : optimisation stochastique**

Soit $g = \sum_{i=1}^{N} e_i(x)$, $\forall i,\ e_i : \mathbb{R}^d \longrightarrow \mathbb{R}$ une fonction dérivable et minorée. On cherche à déterminer le minimum de la fonction $g$. Soit $x_0 \in \mathbb{R}^d$. Soit $(\epsilon_t)_{t \geqslant 0}$ une suite réelle positive vérifiant :

$$\sum_{t=0}^{\infty} \epsilon_t = \infty \text{ et } \sum_{t=0}^{\infty} \epsilon_t^2 < \infty$$

**Etape A : initialisation**

$t \longleftarrow 0$

$E_0 \longleftarrow \sum_{i=1}^{N} e_i(x_0)$

**Etape B : récurrence**

$x_{t,0} \longleftarrow W_0$

**pour** $t' = 0$ **à** $N-1$ **faire**

$i \longleftarrow$ nombre aléatoire dans $\{1, \ldots, N\}$

$g \longleftarrow \frac{\partial e_i}{\partial x}(x_{t,t'})$

$x_{t,t'+1} \longleftarrow x_{t,t'} - \epsilon_t g$

**fin pour**

$x_{t+1} \longleftarrow x_{t,N}$

$E_{t+1} \longleftarrow \sum_{i=1}^{N} e_i(x_{t+1})$

$t \longleftarrow t+1$

**Etape C : terminaison**

si $\frac{E_t}{E_{t-1}} \approx 1$ alors l'apprentissage a convergé sinon retour à l'étape B.

En pratique, il est utile de converser le meilleur jeu de coefficients : $x^* = \arg\min_{u \geqslant 0} E_u$ car la suite $(E_u)_{u \geqslant 0}$ n'est pas une suite constamment décroissante.

### 23.4.5 Premier exemple : optimisation d'une fonction quadratique

La figure 23.21 illustre un exemple pour lequel l'utilisation d'un algorithme d'optimisation du second degré est pertinente et pour le vérifier concrètement, on choisit la fonction suivante :

$$f(x,y) = x^4 + x^3 + \frac{y^2+y}{100} \tag{23.20}$$

Le miminum est atteint pour $(x,y) = \left(-\frac{3}{4}, -\frac{1}{2}\right)$. Toutefois, du fait du coefficient très faible devant les termes en $y^2$ et $y$, l'utilisation d'un algorithme de descente de gradient du premier ordre doit converger très lentement contrairement à une méthode du second ordre, ce que le programme décrit au paragraphe 23.4.9 montrera numériquement.

### 23.4.6 Court rappel sur l'estimateur du maximum de vraisemblance

La méthode de l'estimateur du maximum de vraisemblance est une méthode permettant d'estimer les paramètres d'une densité. Par exemple, on mesure les tailles d'une trentaine de personnes pour obtenir un échantillon $(X_1, \ldots, X_n)$. On calcule pour cet échantillon l'espérance (ou la moyenne) et l'écart-type :

$$\mu = \mathbb{E}(X) = \frac{1}{n}\sum_{i=1}^{n} X_i \text{ et } \sigma = \sqrt{\mathbb{V}(X)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - \mathbb{E}(X))^2} \tag{23.21}$$

Ces valeurs apparaissent souvent en statistiques, elles proviennent d'une hypothèse faite sur la loi que suivent les variables de l'échantillon $(X_1, \ldots, X_n)$ : une loi normale de moyenne $\mu$ et d'écart-type $\sigma$. On suppose alors que toutes les variables $X_i$ sont indépendantes et suivent la même loi normale de moyenne $\mu$ et d'écart-type $\sigma$ pour l'instant inconnus. Cela signifie aussi que la densité de la variable $X_i$ a pour expression :

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{23.22}$$

Cette expression permet par exemple de calculer la probabilité d'un événement comme le fait que les tailles des $n$ personnes soient inférieures à 1,80 mètres et supérieures à 1,60 mètres. Notons $A$ cet événement.

$$\mathbb{P}(A) = \mathbb{P}(1,6 \leqslant X_1 \leqslant 1,8 \text{ et } ... \text{ et } 1,6 \leqslant X_n \leqslant 1,8) \tag{23.23}$$

Etant donné que les variables de l'échantillon sont toutes indépendantes. Il est possible d'écrire que :

$$\mathbb{P}(A) = \mathbb{P}(1,6 \leqslant X_1 \leqslant 1,8) \times ... \times \mathbb{P}(1,6 \leqslant X_n \leqslant 1,8) = \prod_{i=1}^{n} \mathbb{P}(1,6 \leqslant X_i \leqslant 1,8) \tag{23.24}$$

On utilise maintenant le fait que $X_i$ a pour densité $f$ :

$$\mathbb{P}(A) = \prod_{i=1}^{n}\left[\int_{1,6}^{1,8} f(x)dx\right] = \prod_{i=1}^{n}\left[\int_{1,6}^{1,8} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)dx\right] \tag{23.25}$$

$$= \int_{1,6}^{1,8} ... \int_{1,6}^{1,8} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x_1-\mu)^2}{2\sigma^2}\right) ... \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x_n-\mu)^2}{2\sigma^2}\right) dx_1 ... dx_n \tag{23.26}$$

Cette dernière écriture fait référence à la densité d'un vecteur de variables aléatoires et indépendantes comme par exemple le vecteur $(X_1, \ldots, X_n)$. Puisque la densité d'une variable $X_i$ est $f(x) = \frac{1}{\sqrt{2\pi}\sigma}\ exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$, la densité de l'échantillon $(X_1, \ldots, X_n)$ est :

$$f(x_1, ..., x_n) = \prod_{i=1}^{n} f(x_i) \tag{23.27}$$

$$= \left[\frac{1}{\sqrt{2\pi}\sigma}\right]^n \prod_{i=1}^{n} \exp\left(-\frac{(x_n-\mu)^2}{2\sigma^2}\right) \tag{23.28}$$

On utilise pour cela le fait que la densité d'un ensemble de variables indépendantes est le produit des densités. En fait, cela revient à construire la probabilité infinitésimale $dP$ correspondant au fait que chaque variables $X_i$ appartient à l'intervalle $[x_i,\ x_i + dx_i]$. A partir de cette densité, on construit la vraisemblance [6] $L$ égale à la valeur de la densité $f$ obtenue en remplaçant les $x_i$ par les valeurs observées des variables $X_i$.

$$L(X_1, ..., X_n) = \left[\frac{1}{\sqrt{2\pi}\sigma}\right]^n \prod_{i=1}^{n} \exp\left(-\frac{(X_n-\mu)^2}{2\sigma^2}\right) \tag{23.29}$$

Plutôt que d'avoir un produit, on préfère la log-vraisemblance (ou log-likelihood) :

$$\log L(X_1, ..., X_n) = n \log\left[\frac{1}{\sqrt{2\pi}\sigma}\right] + \sum_{i=1}^{n}\left(-\frac{(X_n-\mu)^2}{2\sigma^2}\right) \tag{23.30}$$

$$= n \log\left[\frac{1}{\sqrt{2\pi}\sigma}\right] - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(X_n-\mu)^2$$

$$= n \log\frac{1}{\sqrt{2\pi}} - n\log\sigma - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(X_n-\mu)^2 \tag{23.31}$$

On cherche maintenant à maximiser cette fonction par rapport à $\mu$ et $\sigma$. En dérivant et en cherchant à annuler les dérivées partielles, on obtient le système d'équations suivant :

$$\begin{cases} \dfrac{\partial L}{\partial \mu} &= \dfrac{1}{\sigma^2}\sum\limits_{i=1}^{n}(X_i-\mu) &= 0 \\ \dfrac{\partial L}{\partial \sigma} &= -\dfrac{n}{\sigma} + \dfrac{1}{\sigma^3}\sum\limits_{i=1}^{n}(X_i-\mu)^2 &= 0 \end{cases} \tag{23.32}$$

On résoud d'abord la première équation puis la seconde pour obtenir :

$$\mu^* = \frac{1}{n}\sum_{i=1}^{n} X_i \text{ et } (\sigma^*)^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i-\mu)^2 \tag{23.33}$$

On retrouve la moyenne et la variance de l'échantillon $(X_1, \ldots, X_n)$. La loi normale la plus appropriée pour modéliser cet échantillon a pour paramètre $\mu^*$ et $\sigma^*$. Et la loi la plus appropriée est celle qui maximise l'expression (23.29). La figure 23.23 suggère que la courbe qui maximise cette expression est celle qui

6. La vraisemblance est souvent noté $L$ pour *likelihood* en anglais.

épouse au mieux la forme de l'histogramme de l'échantillon [7]. Si cet histogramme est l'ensemble des couples $(t_j, h_j)_{1 \leqslant j \leqslant k}$, on peut presque écrire que :

$$\prod_{i=1}^{n} f(X_i) \approx \prod_{j=1}^{k} [f(t_j)]^{h_j} \tag{23.34}$$

Cette dernière expression montre que la fonction $f$ doit éviter d'être nulle en $t_j$ lorsque $h_j$ n'est pas nulle. C'est pour cela que la meilleure courbe $f$ est celle qui épouse au mieux la forme de l'histogramme. Lorsqu'on cherche les meilleurs paramètres d'une densité à l'aide de la méthode de l'estimateur de maximum de vraisemblance, cela revient à choisir une forme de densité [8] et à chercher les paramètres qui font que cette courbe épouse au mieux la vraie distribution des variables observées. L'histogramme est une approximation de cette distribution.

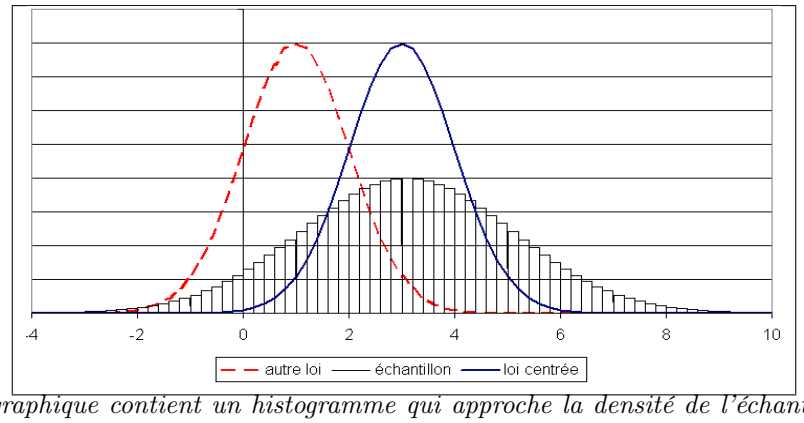



**Figure 23.23 :** *Ce graphique contient un histogramme qui approche la densité de l'échantillon $(X_1, \ldots, X_n)$. La première courbe en trait d'union est celle de la densité d'une loi normale de paramètres tirés au hasard. De manière évidente, l'expression (23.29) calculée pour la courbe de ce graphique est inférieure au résultat obtenu pour une même courbe mais centrée sur l'histogramme. Et le calcul du paragraphe 23.4.6 montre que la meilleure courbe épouse la forme de l'histogramme.*

L'estimateur du maximum de vraisemblance est ce qu'on appelle une méthode paramétrique. Il existe des méthodes non paramétriques comme la méthode des noyaux qui ne fait pas d'hypothèse sur la forme de la densité. D'autres méthodes encore ne s'intéressent qu'à une partie de la densité, plus précisément les queues de distributions.

Par exemple, lors de la construction d'une plateforme pétrolière, il est important de connaître la hauteur de la plus grande vague pour placer la plateforme juste au-dessus. Le coût de construction est plus élevé lorsque la plateforme est surélevée mais elle est hors de portée des vagues qui ne peuvent plus l'emporter. A partir de séries de mesures de hauteurs de vagues, les statisticiens s'intéressent non plus à la distribution de cette hauteur mais à la distribution de la hauteur maximale, c'est-à-dire la probabilité d'événements très rares.

La méthode de l'estimateur du maximum de vraisemblance n'est plus applicable lorsqu'on s'intéresse aux queues de distributions. La densité calculée par ce moyen est proche de la véritable distribution à proximité de la moyenne, là où il y a beaucoup de données. Mais elle est peu précise lorsqu'il y a peu de données car, même si on fait des erreurs, elles sont peu nombreuses.

7. Un histogramme de l'échantillon $(X_1, \ldots, X_n)$ est une liste de couples de valeurs $(t_j, h_j)_{1 \leqslant j \leqslant k}$ où $(t_j)_j$ est une suite croissante de $\mathbb{R}$ et $h_j$ est un nombre entier positif égal au nombre de variables $X_i$ appartenant à l'intervalle $[t_j, t_{j+1}[$. Autrement dit :

$$h_j = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{X_i \in [t_j,\, t_{j+1}[\}}$$

8. Une forme de densité ou plus exactement un modèle, loi normale, loi de Poisson, ... une loi dépendant d'un ou plusieurs paramètres.

Les événements rares forment une branche à part dans la théorie statistique, elle prend de plus en plus d'importance en assurance par exemple car on cherche de plus en plus à estimer la probabilité d'événements rares comme les catastrophes naturels.

**Pour aller plus loin :**

L'expression (23.34) équivaut à la suivante :

$$(23.34) \Longleftrightarrow \sum_{i=1}^{n} \log f\left(X_i\right) \approx \sum_{j=1}^{k} h_i \log f\left(t_j\right) \tag{23.35}$$

Et si on note $g$ la véritable densité de l'échantillon $(X_1, \ldots, X_n)$, l'expression (23.35) converge vers une limite lorsque $n \longrightarrow \infty$ :

$$\frac{1}{n}\sum_{i=1}^{n} \log f\left(X_i\right) \underset{n\to\infty}{\longrightarrow} \int_{\mathbb{R}} g(x) \log f(x) dx = \mathbb{E}_g\left(\log f\right) \tag{23.36}$$

Or l'expression $-\int_{\mathbb{R}} g(x) \log f(x) dx$ est aussi appelée *distance de Kullback-Leiber* entre deux densités. La densité $f$ cherchée est celle qui minimise cette distance :

**Problème 23.34 : distance de Kullback-Leiber**
Soit l'ensemble $F$ des fonctions définies sur l'ensemble $D \subset \mathbb{R}$. Toute fonction $f \in F$ vérifie $\forall x \in D,\ f(x) > 0$ et $\int_D f = 1$. Soit $g \in F$. On cherche la fonction $f^*$ solution du problème :

$$f^* = \arg\min_{f \in F} - \int_D g(x) \log f(x) dx$$

**Théorème 23.35 : distance de Kullback-Leiber**
La solution du problème d'optimisation 23.34 est $f^* = g$.

On peut démontrer ce théorème lorsque l'ensemble $D$ est un sous-ensemble fini de $\mathbb{N}$. La fonction $g$ est définie par une liste de couple $(j, g_j)_{j\in D}$ et $\sum_{j\in D} g_j = 1$. C'est un histogramme. La fonction $f$ est définie de même : $(j, f_j)_{j\in D}$ et $\sum_{j\in D} f_j = 1$. La distance de Kullback-Leiber entre $f$ et $g$ s'écrit :

$$\sum_{j\in D} - g_j \log f_j \tag{23.37}$$

On cherche à démontrer que quelque soit la fonction $f$ choisie, elle vérifie :

$$\sum_{j\in D} g_j \log f_j \leqslant \sum_{j\in D} g_j \log g_j \tag{23.38}$$

$$\Longleftrightarrow \sum_{j\in D} g_j \left(\log f_j - \log g_j\right) \leqslant 0 \tag{23.39}$$

$$\Longleftrightarrow \sum_{j\in D} g_j \log \frac{f_j}{g_j} \leqslant 0 \tag{23.40}$$

Comme la fonction $x \longrightarrow \log x$ est concave et que $\sum_{j \in D} g_j = 1$ :

$$\sum_{j \in D} g_j \log \frac{f_j}{g_j} \leqslant \sum_{j \in D} \log \left( g_j \frac{f_j}{g_j} \right) = \sum_{j \in D} \log f_j \leqslant 0 \tag{23.41}$$

Cette dernière inégalité suffit à démontrer que la disbribution qui minimise la distance de Kullback-Leiber à une autre distribution est elle-même. Cette démonstration convient pour un ensemble $D$ fini. Il reste à l'adapter pour un ensemble dénombrable et plus généralement pour une densité définie sur $\mathbb{R}$, ce qui sort du cadre de ce document [9].

### 23.4.7 Second exemple : proportion de mâles chez les poissons

Un biologiste cherche à étudier la population d'une certaine espèce de poissons. Plus exactement, il voudrait savoir quelle est la proportion de mâles et de femelles. Il a réussi à récupérer une base de données contenant les tailles des poissons de cette espèce capturés par les pêcheurs sur une année entière. Malheureusement, les pêcheurs n'ont pas distingué les mâles des femelles. Pour estimer cette proportion, le biologiste dispose de deux informations :

1. La taille moyenne des mâles est différente de la taille moyenne des femelles, elle est même supérieure.
2. La taille des mâles est distribuée selon une loi normale, il en est de même pour les femelles.

A partir de ces deux informations (ou hypothèses), le biologiste suppose que la distribution des tailles des poissons suit un mélange de loi normales :

$$f(x) = \frac{\alpha_m}{\sqrt{2\pi}\sigma_m} \, exp\left( -\frac{(x-\mu_m)^2}{2\sigma_m^2} \right) + \frac{\alpha_f}{\sqrt{2\pi}\sigma_f} \, exp\left( -\frac{(x-\mu_f)^2}{2\sigma_f^2} \right) \tag{23.42}$$

On sait d'après les hypothèses de modélisation que $\alpha_m + \alpha_f = 1$, $\alpha_m > 0$, $\alpha_f > 0$ et $\mu_m > \mu_f$. Il reste à estimer ces paramètres. La méthode choisie est celle de l'estimateur du maximum de vraisemblance. On note les tailles mesurées par les pêcheurs $(X_1, \ldots, X_N)$. On cherche donc à calculer :

$$\begin{aligned} \left(\alpha^*, \mu_m^*, \sigma_m^*, \mu_f^*, \sigma_f^*\right) = \underset{\alpha,\mu_m,\sigma_m,\mu_f,\sigma_f}{\arg\max} \prod_{i=1}^{N} \Bigg[ & \frac{\alpha}{\sqrt{2\pi}\sigma_m} \, exp\left( -\frac{(X_i-\mu_m)^2}{2\sigma_m^2} \right) + \\ & \frac{1-\alpha}{\sqrt{2\pi}\sigma_f} \, exp\left( -\frac{(X_i-\mu_f)^2}{2\sigma_f^2} \right) \Bigg] \end{aligned} \tag{23.43}$$

On passe au logarithme, il faut donc maximiser la fonction $g$ définie par :

$$g\left(\alpha, \mu_m, \sigma_m, \mu_f, \sigma_f\right) = \sum_{i=1}^{N} \log \left[ \frac{\alpha}{\sqrt{2\pi}\sigma_m} \, exp\left( -\frac{(X_i-\mu_m)^2}{2\sigma_m^2} \right) + \frac{1-\alpha}{\sqrt{2\pi}\sigma_f} \, exp\left( -\frac{(X_i-\mu_f)^2}{2\sigma_f^2} \right) \right] \tag{23.44}$$

On note :

$$h\left(X_i, \alpha, \mu_m, \sigma_m, \mu_f, \sigma_f\right) = \log \left[ \frac{\alpha}{\sqrt{2\pi}\sigma_m} \, exp\left( -\frac{(X_i-\mu_m)^2}{2\sigma_m^2} \right) + \frac{1-\alpha}{\sqrt{2\pi}\sigma_f} \, exp\left( -\frac{(X_i-\mu_f)^2}{2\sigma_f^2} \right) \right]$$

9. Cette démonstration utilise l'inégalité de Jensen. Si $f$ est une fonction convexe sur $\mathbb{R}$ et $X$ une variable aléatoire réelle, alors : $\mathbb{E}\left(f(X)\right) \leqslant f\left(\mathbb{E}\left(X\right)\right)$.

Afin de trouver le maximum de la fonction $h$, on calcule les dérivées de $h$ par rapport à $\alpha, \mu_m, \sigma_m, \mu_f, \sigma_f$ :

$$\frac{\partial g}{\partial \alpha}(\alpha,\mu_m,\sigma_m,\mu_f,\sigma_f) = \sum_{i=1}^{N} \frac{\frac{1}{\sqrt{2\pi}\sigma_m}\ exp\left(-\frac{(X_i-\mu_m)^2}{2\sigma_m^2}\right) - \frac{1}{\sqrt{2\pi}\sigma_f}\ exp\left(-\frac{\left(X_i-\mu_f\right)^2}{2\sigma_f^2}\right)}{h\left(X_i,\alpha,\mu_m,\sigma_m,\mu_f,\sigma_f\right)} \tag{23.45}$$

$$\frac{\partial g}{\partial \mu_m}(\alpha,\mu_m,\sigma_m,\mu_f,\sigma_f) = \sum_{i=1}^{N} \frac{\frac{X_i-\mu_m}{\sigma_m^2}\ \frac{\alpha}{\sqrt{2\pi}\sigma_m}\ exp\left(-\frac{(X_i-\mu_m)^2}{2\sigma_m^2}\right)}{h\left(X_i,\alpha,\mu_m,\sigma_m,\mu_f,\sigma_f\right)} \tag{23.46}$$

$$\frac{\partial g}{\partial \mu_f}(\alpha,\mu_m,\sigma_m,\mu_f,\sigma_f) = \sum_{i=1}^{N} \frac{\frac{X_i-\mu_f}{\sigma_f^2}\ \frac{1-\alpha}{\sqrt{2\pi}\sigma_f}\ exp\left(-\frac{\left(X_i-\mu_f\right)^2}{2\sigma_f^2}\right)}{h\left(X_i,\alpha,\mu_m,\sigma_m,\mu_f,\sigma_f\right)} \tag{23.47}$$

$$\frac{\partial g}{\partial \sigma_m}(\alpha,\mu_m,\sigma_m,\mu_f,\sigma_f) = \sum_{i=1}^{N} \frac{\frac{(X_i-\mu_m)^2}{\sigma_m^3}\ \frac{\alpha}{\sqrt{2\pi}\sigma_m}\ exp\left(-\frac{(X_i-\mu_m)^2}{2\sigma_m^2}\right) - \frac{\alpha}{\sqrt{2\pi}\sigma_m^2}\ exp\left(-\frac{(X_i-\mu_m)^2}{2\sigma_m^2}\right)}{h\left(X_i,\alpha,\mu_m,\sigma_m,\mu_f,\sigma_f\right)} \tag{23.48}$$

$$\frac{\partial g}{\partial \sigma_f}(\alpha,\mu_m,\sigma_m,\mu_f,\sigma_f) = \sum_{i=1}^{N} \frac{\frac{\left(X_i-\mu_f\right)^2}{\sigma_f^3}\ \frac{1-\alpha}{\sqrt{2\pi}\sigma_f}\ exp\left(-\frac{\left(X_i-\mu_f\right)^2}{2\sigma_f^2}\right) - \frac{1-\alpha}{\sqrt{2\pi}\sigma_f^2}\ exp\left(-\frac{\left(X_i-\mu_f\right)^2}{2\sigma_f^2}\right)}{h\left(X_i,\alpha,\mu_m,\sigma_m,\mu_f,\sigma_f\right)} \tag{23.49}$$

Annuler ces dérivées est impossible tout comme obtenir la solution exacte correspondant au maximum de la fonction $f$. Il faut utiliser un algorithme approché, par exemple, une méthode d'optimisation de Newton. Dans cet exemple, le calcul de la proportion des mâles et de femelles reposent sur le choix d'un modèle pour la densité et d'une méthode numérique pour estimer ces paramètres.

Tout d'abord, on vérifie que la méthode numérique permettant d'estimer les paramètres du modèle (23.42) permet effectivement de les retrouver dans une situation où ils sont connus. On simule donc un échantillon de tailles de poissons générées aléatoirement selon la densité (23.42) avec des paramètres $\alpha$, $\mu_m$, $\sigma_m$, $\mu_f$, $\sigma_f$ puis on vérifie que l'algorithme 23.30 ou 23.32 convergent vers ces paramètres.

Cette première étape, si elle est réussie, valide en quelque sorte la méthodologie proposée. Il est alors envisageable de considérer que les résultats obtenus avec des données réelles seront fiables.

**Remarque 23.36 : Algorithme Expectation-Maximisation**
Il existe un autre algorithme permettant d'estimer les paramètres d'une densité faite d'un mélange de lois normales, c'est l'algorithme Expectation-Maximisation (ou EM) établi par Dempster (voir [Dempster1977]). Cet algorithme s'applique pour tout modèle contenant des informations cachées - pour cet exemple, c'est la proportion de mâles et de femelles. Il est souvent préféré à un algorithme d'optimisation classique comme ceux présentés dans ce chapitre : le paragraphe 23.4.12 montrera que le paramètre $\alpha$ liée à l'information cachée fait de la résistance à l'algorithme 23.30.

### 23.4.8 Correction

### 23.4.9 Premier exemple, fonction quadratique

On cherche à tester l'algorithme BFGS pour la fonction $f(x,y) = x^4 + x^3 + \frac{y^2+y}{100}$. Pour cela, on crée la classe `optimisation_bfgs`. Son constructeur prend entre autres comme paramètres `f`, `derivee`, `args`.

- `f(x,...)` est la fonction à minimiser, cette fonction retourne la valeur de $f$ au point `x`.
- `derivee(x,...)` retourne un vecteur égal au gradient de la fonction `f` au point `x`.
- `args` est une liste d'arguments supplémentaires hormis `x` dont les fonctions `f` et `derivee` ont besoin.

Les deux fonctions `f` et `derivee` sont appelées par les méthodes `erreur` et `gradient`, elles-mêmes appelées par la méthode `optimise` qui implémente l'algorithme BFGS 23.32. Toutes ces méthodes utilisent l'objet

`array` du module `Numeric` qui permet de stocker des vecteurs et des matrices. Le module `Numeric` définit également toutes les opérations matricielles dont l'algorithme BFGS a besoin.

Pour appliquer l'algorithme BFGS sur la fonction $f(x,y) = x^4 + x^3 + \frac{y^2+y}{100}$, on définit les fonctions suivantes :

```
    def fonction (x, a,b,c) :
        return x [0]**4 + a * x[0]**3 + b * x[1]**2 + c * x [1]
    def derivee (x, a,b,c) :
        return Num.array ( [4 * x[0]**3 + 3 * a * x[0]**2, b * 2 * x [1] + c] )
```

Puis on applique l'algorithme BFGS :

```
    opt = optimisation_bfgs (fonction, derivee, (1,0.01,0.01))
    x0  = Num.array ( [ 3, 4] )
    x   = opt.optimise (x0)
```

Ces extraits ainsi que l'algorithme BFGS sont présents dans le programme qui suit.

```
# -*- coding: cp1252 -*-
import numpy as Num           # pour les tableaux
import psyco

class optimisation_bfgs (object) :
    """optimisation à l'aide de l'algorithme BFGS"""

    def __init__ (self, f, derivee, args = None, \
                  epsilon = 0.1, erreur = 1e-5, maxiter = 100) :
        """initialisation
        @param      f           fonction à minimiser --> f (x, args)
                                    où x est un vecteur (Numeric.array)
        @param      derivee     dérivée de f  --> derivee (x, args)
        @param      args        autres paramètres de la fonction
        @param      epsilon     coefficient devant le gradient
        @param      erreur      on s'arrête lorsque l'erreur ne décroît plus
        @param      maxiter     nombre d'itérations maximum
        """

        self._f          = f
        self._derivee    = derivee
        self._epsilon    = epsilon
        self._erreur     = erreur
        self._maxiter    = maxiter
        self._args       = args

    def erreur (self, x) :
        """calcule l'erreur"""
        return self._f (x, *self._args)

    def gradient (self, x) :
        """calcule le gradient"""
        return self._derivee (x, *self._args)

    def identite (self, dim) :
        """retourne la matrice identité"""
        b   = Num.array ( [ [ 0 for i in xrange (0, dim) ] for j in xrange (0,dim) ] )
        for i in xrange (0, dim) :
            b [(i,i)] = 1
        return b

    def optimise (self, x0, classic = False, pr = True) :
```

```
"""optimisation selon l'algorithme BFGS,
x0 est le vecteur initiale, retourne la meilleure solution,
si pr == True, affiche des résultats intermédiaires,
classic == True, la matrice bt est mise à jour à chaque itération,
équivaut à une descente de gradient du premier degré"""

dim     = len (self.gradient (x0))
stop    = 1e-12
err     = self.erreur (x0)
bt      = self.identite (dim)
id      = True
t       = 0
iter    = 0
old     = err * (1.0 + self._erreur * 2) * 2
x       = x0
gt_     = self.gradient (x0)
xt_     = x0
nbt     = 0

while   (abs ((old - err) / err) > self._erreur or t == 0) and \
         iter < self._maxiter :

    if pr :
        print "itération ", iter, " (", \
                    t,nbt, ") \t erreur : ", err, " (", old, ")"
    iter    = iter + 1
    gt      = self.gradient (x)
    ct      = Num.dot (bt, gt)

    e   = abs (err) * 2 + 1
    eps = self._epsilon * 2
    while e > err and eps > stop :
        eps = eps / 2
        xt  = x - ct * eps
        e   = self.erreur (xt)

    if eps < 1e-12 and t > 0 :
        bt = self.identite (dim)
        if t == 0 : nbt += 1
        else : nbt = 0
        t  = 0
    else :
        old     = err
        x       = xt
        err     = e
        if not classic :
            gt      = self.gradient (x)
            dt      = gt - gt_
            gbtg    = Num.dot (Num.transpose (gt), Num.dot (bt, gt))
            gbtd    = Num.dot (Num.transpose (gt), Num.dot (bt, dt))
            if (gbtg <= 0 or gbtd <= 0) and nbt < 2 :
                if t == 0 : nbt += 1
                else : nbt = 0
                bt = self.identite (dim)
                t  = 0
            else :
                t       = t + 1
                st      = xt - xt_
                dtbtdt  = Num.dot (Num.transpose (dt), Num.dot (bt, dt))
                dtst    = Num.dot (Num.transpose (dt), st)
                stdt    = Num.dot (Num.transpose (st), dt)
                stst    = Num.outer (st, st)
                stdtbt  = Num.dot (Num.outer (st, dt), bt)
                btdtst  = Num.dot (bt, Num.outer (dt, st))
                c       = 1.0 + dtbtdt / dtst
```

```
                        b       = stst * (c / stdt)
                        b      -= (stdtbt + btdtst) / dtst
                        bt      = bt + b

                gt_ = gt
                xt_ = x

        return x

if __name__ == "__main__" :
    psyco.full ()

    def fonction (x, a,b,c) :
        return x [0]**4 + a * x[0]**3 + b * x[1]**2 + c * x [1]
    def derivee (x, a,b,c) :
        return Num.array ( [4 * x[0]**3 + 3 * a * x[0]**2, b * 2 * x [1] + c] )

    opt = optimisation_bfgs (fonction, derivee, (1,0.01,0.01), \
                                    maxiter = 2000, epsilon = 0.2, erreur = 1e-10)
    x0  = Num.array ( [ 3, 4] )
    print "solution initiale : ", x0
    x   = opt.optimise (x0, classic = False)
    print "solution finale : ", x
```

Après exécution, on vérifie que la solution trouvée correspond bien à $\left(-\frac{3}{4}, -\frac{1}{2}\right)$. Pour comparer avec l'algorithme d'optimisation du premier degré, on remplace la ligne suivante :

```
    x   = opt.optimise (x0, classic = False)
```

Par :

```
    x   = opt.optimise (x0, classic = True)
```

Alors qu'il faut moins d'une centaine d'itérations à l'algorithme BFGS pour converger la solution, il en faut plus d'un millier à l'algorithme du premier ordre.

### 23.4.10 Second exemple, proportion de mâles et de femelles chez les poissons

Pour vérifier la pertinence d'une méthode numérique, on cherche d'abord à vérifier qu'elle marche bien sur un échantillon $(X_i)_i$ dont on sait qu'il vérifie les hypothèses du problème à résoudre - la distribution des $X_i$ est un mélange de deux lois normales. Une fois cette première étape validée, il est alors utilisé sur des données réelles ce qui ne sera pas fait ici.

### 23.4.11 Simulation d'une série $(X_i)_i$

Tout d'abord, on cherche à simuler une série $(X_i)_{1 \leqslant i \leqslant N}$ où chaque $X_i$ suit une loi de densité (voir figure 23.24) :

$$f(x) = \frac{\alpha}{\sqrt{2\pi}\sigma_m} exp\left(-\frac{(x-\mu_m)^2}{2\sigma_m^2}\right) + \frac{1-\alpha}{\sqrt{2\pi}\sigma_f} exp\left(-\frac{(x-\mu_f)^2}{2\sigma_f^2}\right) \tag{23.50}$$

On choisi par exemple $N = 10000$, $\alpha = 0,55$, $\mu_m = 0,40$, $\mu_f = 0,35$, $\sigma_m = 0,020$, $\sigma = 0,015$. Les mâles ont une taille moyenne de 40 cm et un écart-type de $2cm$. Ceci signifie que 95% des mâles ont une taille comprises entre 36cm et 44cm. Les femelles ont une taille moyenne de 40 cm et un écart-type de $1,5$ cm.

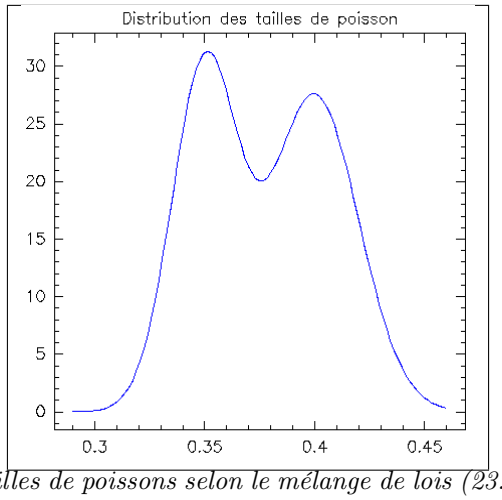



**Figure 23.24 :** *Distribution des tailles de poissons selon le mélange de lois (23.50) avec pour paramètres* $\alpha = 0,55$, $\mu_m = 0,40$, $\mu_f = 0,35$, $\sigma_m = 0,020$, $\sigma = 0,015$.

Ceci signifie que 95% des femelles ont une taille comprise entre 32cm et 38cm. Pour cela, on applique l'alogirthme suivant :

**Algorithme 23.37 : simulation d'une variable selon un mélange de deux lois normales**
On cherche à simuler une variable aléatoire $X$ selon un mélange de deux lois normales :

$$X \sim \frac{\alpha}{\sqrt{2\pi}\sigma_m}\, exp\left(-\frac{(x-\mu_m)^2}{2\sigma_m^2}\right) + \frac{1-\alpha}{\sqrt{2\pi}\sigma_f}\, exp\left(-\frac{(x-\mu_f)^2}{2\sigma_f^2}\right)$$

où $\alpha \in [0,1]$, $\mu_m,\ \mu_f \in \mathbb{R}$, $\sigma_m,\ \sigma_f > 0$.

**Etape A : choix de la composante normale**
Soit $U \sim U[0,1]$, $U$ est une variable aléatoire uniforme sur $[0,1]$.

**Etape B : simulation loi normale**
On simule $V$ selon une loi normale de moyenne nulle et de variance un. Autrement dit, soit $A$, $B$ deux variables générées selon une loi uniforme sur $[0,1]$ (voir [Robert1996]) :

$$V = \sqrt{-2\log A}\,\cos(2\pi B)$$

**Etape C : construction de $X$**
On calcule :

$$X = \mathbb{1}_{\{U\leqslant\alpha\}}(\sigma_m V + \mu_m) + \mathbb{1}_{\{U>\alpha\}}(\sigma_f V + \mu_f)$$

$X$ est la valeur cherchée.

Pour obtenir la série $(X_i)_{1\leqslant i\leqslant N}$, il suffit de calculer $N$ valeurs avec l'algorithme 23.37.

### 23.4.12 Estimation des paramètres avec une méthode du premier degré

On implémente le premier algorithme 23.30 afin de retrouver les paramètres qui ont permis de simuler l'échantillon $(X_i)_{1\leqslant i\leqslant N}$. On cherche donc à maximiser la fonction $g$ définie par l'expression (23.44) dont le gradient est définie par les expressions (23.45) à (23.49).

Il n'est pas toujours possible de choisir des valeurs initiales pour l'algorithme d'optimisation. Toutefois, dans le cas présent, les paramètres de la fonction $g$ ont un sens presque physique. On peut donc choisir

des valeurs sensées pour la solution initiale de manière à accélérer la convergence de l'algorithme :

$$\alpha = \frac{1}{2} \quad \mu_m = \mathbb{E}(X_i) + \sqrt{\mathbb{V}(X_i)} \quad \mu_f = \mathbb{E}(X_i) - \sqrt{\mathbb{V}(X_i)} \quad \sigma_m = \sigma_f = \sqrt{\frac{\mathbb{V}(X_i)}{2}} \tag{23.51}$$

L'algorithme converge alors en une dizaine d'itérations vers le résultat de la figure 23.25. La log-vraisemblance est très proche de celle obtenue avec le modèle initial et on remarque que pour cet exemple, elle est très peu sensible au paramètre $\alpha$ et beaucoup plus aux autres paramètres puisque le paramètre $\alpha$ n'a pas varié de sa valeur initiale [10].

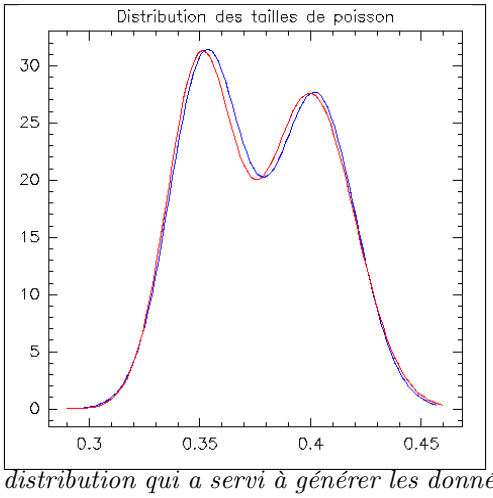



**Figure 23.25 :** *Comparaison de la distribution qui a servi à générer les données ($\alpha = 0,55$, $\mu_m = 0,4$, $\mu_f = 0,35$, $\sigma_m = 0,020$, $\sigma = 0,15$) avec la distribution qui a été estimée ($\alpha = 0,50$, $\mu_m = 0,40$, $\mu_f = 0,35$, $\sigma_m = 0,019$, $\sigma = 0,016$).*

Pour implémenter cet algorithme, on crée la classe `poisson_loi`. Cette classe permet de simuler un échantillon de taille (méthodes `simulation`, méthodes `generate`), de calculer la log-vraisemblance pour un échantillon (méthodes `densite`, `log_densite`, `log_densite_list`), de calculer le gradient pour un point $x$ (méthode `gradient`) ou pour un échantillon méthodes `gradient_total`). Enfin, cette classe implémtente l'algorithme d'optimisation 23.30 par le biais de la méthode `optimisation`.

La partie principale du programme consiste tout d'abord à créer une instance de cette classe afin de générer une série de 10000 tailles de poisson. Une seconde instance est ensuite créée qui estime ses coefficients à l'aide de la méthode `optimisation`. Le résultat est affiché puis les densités sont comparées graphiquement (méthode `trace_densite`).

```
# -*- coding: cp1252 -*-
import random
import math

class poisson_loi (object) :
    """loi de distribution de la taille des poissons"""

    def __init__(self, alpha = 0.5, mm = 1, sm = 1, mf = 1, sf = 1) :
        """initialisation des paramètres de la loi"""
        self.alpha  = alpha
        self.mm     = mm
        self.sm     = sm
        self.mf     = mf
        self.sf     = sf
```

10. Le résultat est le même si la valeur initiale pour $\alpha$ est différente : lors de l'optimisation, ce paramètre évolue très peu.

```
    def init (self, l) :
        """initialisation de la classe avec une liste de variables"""
        m = 0
        s = 0
        for x in l :
            m += x
            s += x*x
        m /= len (l)
        s /= len (l)
        s -= m*m
        self.alpha  = 0.5
        self.sm     = s / 2
        self.sf     = s / 2
        self.sm     = math.sqrt (self.sm)
        self.sf     = math.sqrt (self.sf)
        self.mm     = m + self.sm * 2
        self.mf     = m - self.sf * 2

    def __str__(self) :
        """affichage"""
        s = "classe poisson_loi\n"
        s += "alpha = " + str (self.alpha) + "\n"
        s += "moyenne male = " + str (self.mm) + "\n"
        s += "sigma male = " + str (self.sm) + "\n"
        s += "moyenne femelle = " + str (self.mf) + "\n"
        s += "sigma femelle = " + str (self.sf) + "\n"
        return s

    def simulation (self) :
        """simule une taille de poisson"""
        u = random.uniform (0,1)
        v = random.gauss (0,1)
        if u <= self.alpha :
            return v * self.sm + self.mm
        else :
            return v * self.sf + self.mf

    def generate (self,N) :
        """simule N tailles de poisson"""
        return [ self.simulation () for i in xrange (0,N) ]

    def trace_densite (self, autre = None) :
        """trace la courbe de la densité,
        si autre != None, en trace une autre"""
        xa = min (self.mm, self.mf) - 3.0 * max (self.sm, self.sf)
        xb = max (self.mm, self.mf) + 3.0 * max (self.sm, self.sf)
        n  = 100
        x  = [ xa + (xb - xa) * i / n for i in xrange (0,n+1) ]
        y  = [ self.densite (xa + (xb - xa) * i / n) for i in xrange (0,n+1) ]
        import pylab as pl
        pl.plot ( x, y, "b" )
        if autre != None :
            xa = min (autre.mm, autre.mf) - 3.0 * max (autre.sm, autre.sf)
            xb = max (autre.mm, autre.mf) + 3.0 * max (autre.sm, autre.sf)
            n  = 100
            x  = [ xa + (xb - xa) * i / n for i in xrange (0,n+1) ]
            y  = [ autre.densite (xa + (xb - xa) * i / n) for i in xrange (0,n+1) ]
            pl.plot (x, y, "r")

        pl.title ("Distribution des tailles de poisson")
        pl.xlabel ("x")
        pl.ylabel ("y")

        pl.show()
```

```
    def densite (self, x) :
        """calcule de la densité au point x"""
        dpi = 1.0 / math.sqrt (math.pi * 2)
        xm = (x - self.mm) ** 2 / (2 * self.sm * self.sm)
        xm = math.exp (- xm)
        xf = (x - self.mf) ** 2 / (2 * self.sf * self.sf)
        xf = math.exp (- xf)
        return self.alpha * xm / self.sm + (1.0 - self.alpha) * xf / self.sf

    def log_densite (self, x) :
        """calcule de la log-densité au point x"""
        try :
            return math.log (self.densite (x))
        except :
            print "---------- erreur ------------------------------------------"
            print x
            return 0

    def log_densite_list (self, l) :
        """calcule la somme des log-densités au point x de l"""
        s = 0
        for x in l :
            s += self.log_densite (x)
        s /= len (l)
        return s

    def gradient (self, x) :
        """calcul le gradient au point x, retourne un t-uple"""
        unpim   = 1.0 / (math.sqrt (2 * math.pi) * self.sm)
        unpif   = 1.0 / (math.sqrt (2 * math.pi) * self.sf)
        expm    = math.exp ( - ( x - self.mm)**2 / (2 * (self.sm**2)))
        expf    = math.exp ( - ( x - self.mf)**2 / (2 * (self.sf**2)))
        grada   = unpim * expm - unpif * expf
        gradmm  = (x - self.mm) / (self.sm** 2) * self.alpha * unpim * expm
        gradmf  = (x - self.mf) / (self.sf** 2) * (1.0 - self.alpha) * unpif * expf
        gradsm  = self.alpha * unpim * (x - self.mm) ** 2 / (self.sm ** 3) - \
                  self.alpha * unpim / self.sm
        gradsm *= expm
        gradsf  = (1.0 - self.alpha) * unpif * (x - self.mf) ** 2 / (self.sf ** 3) - \
                  self.alpha * unpif / self.sf
        gradsf *= expf
        f       = self.densite (x)
        return (grada / f, gradmm / f, gradsm / f, gradmf / f, gradsf / f)

    def gradient_total (self, l) :
        """calcul la somme des gradients pour tous les x de l"""
        g = [ 0.0 for i in range (0,5) ]
        for x in l :
            (a,b,c,d,e) = self.gradient (x)
            g [0] += a
            g [1] += b
            g [2] += c
            g [3] += d
            g [4] += e
        for i in range (0,len (g)) :
            g [i] /= len (l)
        return ( g [0], g [1], g [2], g [3], g [4] )

    def optimisation (self, l, epsilon = 0.001, erreur = 1e-5) :
        """recherche du maximum de la fonction f"""
        self.init (l)
        de   = self.log_densite_list (l)
        print "première densité : ", de
        dold = de / 2
        nb   = 0
```

```
        while abs (dold - de) / de > erreur :
            (a,b,c,d,e) = self.gradient_total (l)
            self.alpha      += epsilon * a
            self.mm         += epsilon * b
            self.sm         += epsilon * c
            self.mf         += epsilon * d
            self.sf         += epsilon * e
            dold = de
            de   = self.log_densite_list (l)
            nb  += 1
            print "itération ", nb, " densité ", de, " (", dold, ")  ", \
                    abs (dold - de) / de, " epsilon = ", epsilon


if __name__ == "__main__" :
    # création d'une instance de la classe
    cl = poisson_loi (0.55, 0.40, 0.020, 0.35, 0.015)
    print cl                # affichage
    #cl.trace_densite ()     # courbe densité

    l = cl.generate (10000)
    cl2 = poisson_loi ()
    print "-------------------------------------------------------------"
    print "log densité maximale ", cl.log_densite_list (l)
    (a,b,c,d,e) = cl.gradient_total (l)
    print "gradient idéal ", (a,b,c,d,e)
    print "-------------------------------------------------------------"

    (a,b,c,d,e) = cl2.gradient_total (l)
    print "gradient avant", (a,b,c,d,e)
    cl2.optimisation (l)
    (a,b,c,d,e) = cl2.gradient_total (l)
    print "gradient après ", (a,b,c,d,e)
    print "-------------------------------------------------------------"
    print cl2
    cl2.trace_densite (cl)
    print "log vraisemblance  : ", cl2.log_densite_list (l)
```

### 23.4.13 Estimation des paramètres avec une méthode du second degré

Le second programme utilise l'algorithme BFGS pour déterminer le maximum de vraisemblance (voir figure 23.26). Cette méthode du second degré permet de corriger le fait que le gradient par rapport au paramètre $\alpha$ est plus faible que les dérivées partielles par rapport aux autres coefficients. On trouve cette fois-ci des valeurs proches du modèle qui a permis de simuler les observations.

Ces méthodes sont très calculatoires et pour accélérer la vitesse de convergence de l'algorithme, il est possible d'utiliser le module `pysco` en utilisant ces deux lignes.

```
import psyco
psyco.full ()
```

La vitesse du programme est multipliée par trois ou quatre lorsque celui-ci passe la plus grande partie de son temps à faire des calculs.

```
# -*- coding: cp1252 -*-
import poisson
#import scipy.optimize.lbfgsb as Opt    # optimisation
import bfgs                        # optimisation
```

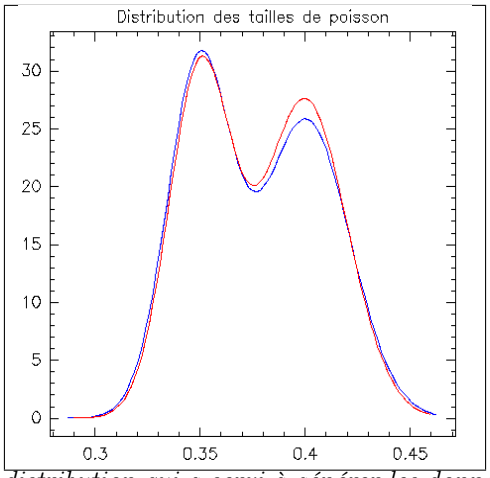



**Figure 23.26 :** *Comparaison de la distribution qui a servi à générer les données ($\alpha = 0,55$, $\mu_m = 0,4$, $\mu_f = 0,35$, $\sigma_m = 0,020$, $\sigma = 0,15$) avec la distribution qui a été estimée ($\alpha = 0,54$, $\mu_m = 0,40$, $\mu_f = 0,35$, $\sigma_m = 0,020$, $\sigma = 0,015$).*

```
import numpy as Num           # pour les tableaux
import psyco

def fonction_bfgs (x, cl, l) :
    """fonction retournant l'opposé de la log-vraisemblance
    de l'instance cl, fonction à minimiser"""
    cl.set (x)
    f = - cl.log_densite_list (l)
    #print f, "\t", x
    return f

def fonction_derivee_bfgs (x, cl, l) :
    """fonction retournant l'opposé de la log-vraisemblance
    de l'instance cl, fonction à minimiser"""
    cl.set (x)
    f           = cl.gradient_total (l)
    return - Num.array (f)

class poisson_loi_bfgs (poisson.poisson_loi) :
    """loi de distribution de la taille des poissons,
    optimisation à l'aide de l'algorithme BFGS"""

    def __init__(self, alpha = 0.5, mm = 1, sm = 1, mf = 1, sf = 1) :
        """initialisation des paramètres de la loi"""
        poisson.poisson_loi.__init__(self, alpha, mm, sm, mf, sf)

    def set (self, x) :
        """initialisation avec un tableau"""
        self.alpha    = x [0]
        self.mm       = x [1]
        self.sm       = abs (x [2])
        self.mf       = x [3]
        self.sf       = abs (x [4])

    def get (self) :
        """retourne un tableau contenant les paramètres"""
        return Num.array ( (self.alpha, self.mm, self.sm, self.mf, self.sf) )

    def __str__(self) :
        """affichage"""
        s = "classe poisson_loi BFGS\n"
        s += "alpha = " + str (self.alpha) + "\n"
```

```
        s += "moyenne male = " + str (self.mm) + "\n"
        s += "sigma male = " + str (self.sm) + "\n"
        s += "moyenne femelle = " + str (self.mf) + "\n"
        s += "sigma femelle = " + str (self.sf) + "\n"
        return s

    def optimisation (self, l, epsilon = 0.001) :
        """recherche du maximum de la fonction f"""
        self.init (l)
        x0     = self.get ()
        print x0
        opt    = bfgs.optimisation_bfgs (fonction_bfgs, \
                                         fonction_derivee_bfgs, \
                                         args = (self, l))
        x = opt.optimise (x0)
        #x      = bfgs.Opt.fmin_l_bfgs_b ( fonction_bfgs, \
        #                                  x0, \
        #                                  fonction_derivee_bfgs, \
        #                                  args = (self, l))
        self.set (x)
        print "optimisation terminée"
        print "valeur maximale : ", self.log_densite_list (l)
        print "message ", d


if __name__ == "__main__" :
    psyco.full ()
    # création d'une instance de la classe
    cl = poisson.poisson_loi (0.55, 0.40, 0.020, 0.35, 0.015)
    print cl                # affichage
    #cl.trace_densite ()     # courbe densité

    l = cl.generate (10000)
    cl2 = poisson_loi_bfgs ()
    print "------------------------------------------------------------"
    print "log densité maximale ", cl.log_densite_list (l)
    (a,b,c,d,e) = cl.gradient_total (l)
    print "gradient idéal ", (a,b,c,d,e)
    print "------------------------------------------------------------"

    (a,b,c,d,e) = cl2.gradient_total (l)
    print "gradient avant", (a,b,c,d,e)
    cl2.optimisation (l)
    (a,b,c,d,e) = cl2.gradient_total (l)
    print "gradient après ", (a,b,c,d,e)
    print "------------------------------------------------------------"
    print cl2
    cl2.trace_densite (cl)
    print "log vraisemblance  : ", cl2.log_densite_list (l)
```

## 23.5 Optimiser de coûts en environnement aléatoire

### 23.5.1 Un problème simple

Un supermarché pourrait vendre en moyenne 80 poulets par semaine s'il pouvait savoir à l'avance combien de poulets à acheter pour satisfaire la demainde. En réalité, le magasin se réapprovisionne une fois par semaine et lorsque la fin de la semaine arrive, tous les poulets invendus sont soldés et supposés vendus. Le gérant du supermarché voudrait savoir quel est le nombre optimal de poulets à commander chaque semaine. On suppose que le prix d'un poulet à l'achat est $p$, son prix à la vente est $q > p$, son prix soldé est $s$. Admettons que le supermarché achète $X$ poulets, en vende au mieux $N$ non soldés et $X - N$ soldés s'il en reste. Pour calculer son bénéfice $B$, il faut tenir compte de deux cas et du fait que le supermarché

ne peut pas vendre plus de poulets qu'il n'en a acheté :

$$\begin{array}{ll} B = X(q-p) & \text{si } N \geqslant X \\ B = N(q-p) + (X-N)(s-p) = X(s-p) + N(q-s) & \text{si } N < X \end{array} \tag{23.52}$$

On peut réduire ces deux expressions à une seule en utilisant la fonction indicatrice :

$$B = f(N, X, p, q, s) = X(q-p)\mathbb{1}_{\{N \geqslant X\}} + [X(s-p) + N(q-s)]\,\mathbb{1}_{\{N<X\}} \tag{23.53}$$

Si $N$ était connu avec certitude, il suffirait de choisir $X = N$, ce serait la réponse optimale mais le nombre de poulets $N$ vendus est inconnu car il varie chaque semaine. Pour avoir une idée plus précise, le gérant du supermarché a délibérément acheté trop de poulets pendant plusieurs semaines. Il s'est aperçu que la variable aléatoire $N$ suit une loi de Poisson de paramètre $\lambda = 80$. On connaît seulement la probabilité que $N$ soit égale à une valeur fixée. La figure 23.27 montre l'allure de cette distribution.

$$\mathbb{P}(X = i) = e^{-\lambda}\frac{\lambda^i}{i!} \tag{23.54}$$

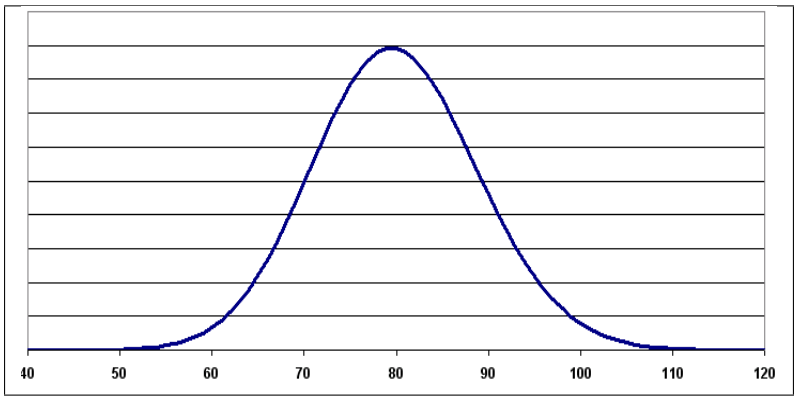



**Figure 23.27 :** *Ce graphe répresente la fonction de densité d'une loi de Poisson de paramètre 80. On observe que le pic est obtenu pour une valeur proche de 80, c'est la valeur la plus probable. Ceci signifie que le nombre de poulets achetés le plus probable est 80.*

Comme le nombre de poulets achetés varie d'une semaine à l'autre, le bénéfice du supermarché varie aussi d'une semaine à l'autre. Ce que le gérant veut optimiser, c'est la somme de ses profits sur une année ce qui est équivalent à maximiser la moyenne de ses profits chaque semaine. Il faut donc chercher à maximiser l'espérence de la variable aléatoire $B$ à $p, q, s$ constant puis à obtenir la valeur $X$ ayant mené à ce maximum.

$$\max_X \mathbb{E}(B) = \max_X \mathbb{E}(f(N, X, p, q, s)) = \max_X \left\{ \sum_{i=0}^{\infty} f(N, X, p, q, s)\mathbb{P}(N = i) \right\} \tag{23.55}$$

Etant donné la forme de la fonction $f$, il n'est pas évident de construire une expression exacte de $X^*$ défini par $\max_X \mathbb{E}(f(N, X, p, q, s)) = f(N, X^*, p, q, s)$. Comme $l = 80$, d'après la figure 23.27, on cherche $X^*$ dans l'ensemble $\{0, ..., 2l = 180\}$, aller au delà de 180 est inutile tant la probabilité est faible. Il suffit de calculer $f$ pour chacune de ces valeurs et de prendre celle qui permet d'obtenir le maximum. Ces calculs longs et répétitifs vont être effectués par un programme informatique qui sera découpé en fonctions comme ceci :

| **fonction** | **objectif** |
|---|---|
| `factorielle(x)` | calcule $x!$ |
| `profit(N,X,p,q,s)` | calcule la fonction $f$ |
| `proba_poisson(l,i)` | calcule la probabilité de Poisson connaissant $\lambda$ et $i$ |
| `esperance(X,p,q,s,l)` | calcule l'espérance (sa moyenne) de la fonction $f$ |
| `maximum(p,q,s,l)` | construit une liste de toutes les valeurs de $f$ |
| `find_maximum(res)` | cherche le maxim dans la liste retournée par la fonction `maximum` |

Le programme obtenu ressemble à celui qui suit, les dernières lignes servent à tracer la courbe présentée par la figure 23.28.

```
import math

def factorielle(x):
     if x == 0: return 1
     else: return x * factorielle(x-1)

def profit (N,X,p,q,s) :
    if X <= N : return X * (q-p)
    else : return X * (s-p) + N * (q-s)

def proba_poisson (l, i) :
    return math.exp (-l) * (l ** i) / factorielle (i)

def esperance (X,p,q,s,l) :
    res = 0.0
    for i in range (0,l*2) :
        res += profit (float (i),X,p,q,s) * proba_poisson (l,i)
    return res

def maximum (p,q,s,l) :
    res = []
    for X in range (0, 2 * l) :
        r = esperance (X,p,q,s,l)
        res.append ((X,r))
    return res

def find_maximum (res) :
    m = (0,0)
    for r in res :
        if r [1] > m [1] :
            m = r
    return m

if __name__ == "__main__" :
    res = maximum (2,5,1,80)
    m = find_maximum (res)
    print m
    # dessin d'une courbe
    from pylab import *
    plot ( [ r [1] for r in res] )
    show ()
```

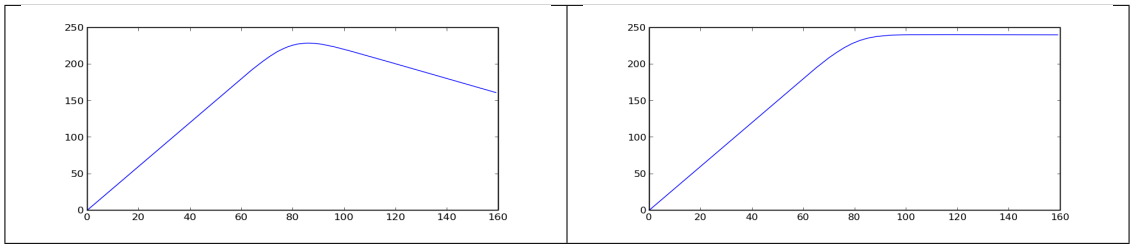



**Figure 23.28 :** *Cette courbe est celle de l'évolution des profits en fonction du nombre de poulets commandés. On suppose que le nombre de poulets achetés suit une loi de Poisson de paramètre 80, que les poulets sont achetés 2€, revendu 5€ et soldés 1€. Le maximum de 228€ est obtenu pour 86 poulets. La seconde courbe montre le résultat dans le cas où les poulets soldés sont vendus 2€ égal au prix des poulets achetés. Le modèle montre ses limites dans ce cas car il suppose que tous les poulets soldés seront achetés et que les contraintes de stockage sont négligeables.*

### 23.5.2 Modélisation de la demande

La représentation de la demande est essentielle, c'est elle qui détermine le résultat. Il est possible de l'affiner comme par exemple supposer que certaines personnes achètent deux ou trois poulets et que la somme des poulets achetés peut être décomposée comme $N = N_1 + 2N_2 + 3N_3$ où $N_i$ est le nombre de personnes achetant $i$ poulets. Dans ce cas, ce n'est plus $N$ qui suit une loi de Poisson mais $N_1$, $N_2$, $N_3$ qui suivent chacune des lois de Poisson de paramètres différents dont il faudra estimer les paramètres.

Cette modification implique l'écriture d'une fonction `proba_poisson_melange` au lieu de `proba_poisson`. La demande n'est plus une loi connue mais un mélange de lois connues dont la densité n'a pas d'expression connue : il faut la tabuler. Pour cela, on utilise deux propriétés sur les lois exponentielles.

**Théorème 23.38 : simulation d'une loi quelconque**
Soit $F = \int f$ une fonction de répartition de densité $f$ vérifiant $f > 0$, soit $U$ une variable aléatoire uniformément distribuée sur $[0, 1]$ alors $F^{-1}(U)$ est variable aléatoire de densité $f$.

**Démonstration (théorème 23.38) :**
Soit $X$ une variable aléatoire de densité $f$, par définition, $\mathbb{P}(X \leqslant x) = F(x)$. Soit $U$ une variable aléatoire uniformément distribué sur $[0, 1]$, alors :

$$\forall u \in [0,1],\ \mathbb{P}(U \leqslant u) = u \tag{23.56}$$
$$\iff \mathbb{P}\left(F^{-1}(U) \leqslant F^{-1}(u)\right) = u \tag{23.57}$$
$$\iff \mathbb{P}\left(F^{-1}(U) \leqslant F^{-1}(F(t))\right) = F(t) \tag{23.58}$$
$$\iff \mathbb{P}\left(F^{-1}(U) \leqslant t\right) = F(t) \tag{23.59}$$

Si la fonction $F$ n'est pas strictement croissante, on pourra prendre $F^{-1}(t) = \inf\{u \mid F(u) \geqslant t\}$. **(23.38)** □

Ce théorème sera appliqué à une loi exponentielle de paramètre $\lambda$. La densité d'une telle loi est $f(x) = \lambda \exp -\lambda x$, $F(x) = \int_0^x f(t)dt = 1 - \exp^{-\lambda x}$. On en déduit que $F^{-1}(t) = -\frac{\ln(1-t)}{\lambda}$, par conséquent : $-\frac{\ln(1-U)}{\lambda}$ suit une loi exponentielle de paramètre $\lambda$ si $U$ est une loi uniforme sur $[0, 1]$.

**Théorème 23.39 : simulation d'une loi de Poisson**
On définit une suite infinie $(X_i)_i > 0$ de loi exponentielle de paramètre $\lambda$. On définit ensuite la série de variables aléatoires $S_i = \sum_{k=1}^{i} X_k$ et enfin $N(t) = \inf\{i \mid S_i > t\}$. Alors la variable aléatoire $N(t)$ suit une loi de Poisson de paramètre $\lambda t$.

La loi exponentielle est souvent utilisée pour modéliser le temps d'attente d'un événement comme le temps d'attente d'un métro une fois sur le quai. On l'utilise aussi pour modéliser la durée de vie d'un outil, d'une ampoule par exemple. La loi de Poisson peut par exemple modéliser le nombre d'ampoules nécessaire pour éclairer une pièce sur une certaine durée. L'exemple 27.1.1 [11] fait également référence au processus de Poisson.

Avant de démontrer le théorème 23.39, il faut définir d'abord la loi Gamma. On pose au préalable $\Gamma(\alpha) = \int_0^\infty u^{\alpha-1}e^{-u}du$. Une variable aléatoire de loi Gamma de paramètres $(\alpha, \lambda)$ a pour densité : $f(x) = \frac{\lambda^\alpha}{\Gamma(\alpha)}t^{\alpha-1}e^{-\lambda t}$. La fonction $\Gamma$ vérifie une propriété utile par la suite : $\forall n \in \mathbb{N}^*$, $\Gamma(n) = (n-1)!$.

**Théorème 23.40 : somme de loi exponentielle iid**
Soit $X_1, ..., X_n$ $n$ variables aléatoires indépendantes et identiquement distribuées de loi $Exp(\lambda)$ alors la somme $\sum_{k=1}^{n} X_k$ suit une loi $Gamma(n, \lambda)$.

11. page 692

**Démonstration (théorème 23.40) :**
La démonstration utilise l'unicité de la fonction caractéristique $\mathbb{E}\left(e^{iX}\right)$. Il suffit de démonstrer que la fonction caractéristique de la somme est celle d'une loi Gamma. On suppose que $X_1, ..., X_n$ suivent des lois exponentielles de paramètre $\lambda$ et $Y$ suit une loi $Gamma(n, \lambda)$.

$$\begin{aligned}
\mathbb{E}\left(\exp\left(i\sum_{k=1}^{n} X_k\right)\right) &= \prod_{k=1}^{n} \mathbb{E}\left(e^{iX_k}\right) \\
&= \left[\int_0^\infty \lambda e^{ix} e^{-\lambda x} dx\right]^n = \lambda^n \left[\int_0^\infty e^{(i-\lambda)x} dx\right]^n \\
&= \lambda^n \left[-\frac{1}{(i-\lambda)}\right]^n = \left[\frac{\lambda}{\lambda - i}\right]^n \qquad (23.60) \\
\mathbb{E}\left(e^{iY}\right) &= \int_0^\infty \frac{\lambda^n}{\Gamma(n)} t^{n-1} e^{-\lambda t} e^{it} dt = \int_0^\infty \frac{\lambda^n}{\Gamma(n)} t^{n-1} e^{(i-\lambda)t} dt \\
&= \frac{\lambda^n}{\Gamma(n)} \frac{\Gamma(n)}{(i-\lambda)^n} = \left[\frac{\lambda}{\lambda - i}\right]^n \qquad (23.61)
\end{aligned}$$

Les lignes (23.60) et (23.61) démontrent le théorème. **(23.40)** □

**Démonstration (théorème 23.39) :**
La démonstration repose sur le fait que $\mathbb{P}(N(t) \geqslant n) \Longleftrightarrow \mathbb{P}(S_n \leqslant t)$. On en déduit que :

$$\mathbb{P}(N(t) = n) = \mathbb{P}(N(t) \geqslant n) - \mathbb{P}(N(t) \geqslant n+1) = \mathbb{P}(S_n \leqslant t) - \mathbb{P}(S_{n+1} \leqslant t) \qquad (23.62)$$

Or d'après le théorème 23.40, $S_n$ suit une loi $Gamma(n, \lambda)$.

$$\begin{aligned}
\mathbb{P}(N(t) = n) &= \int_0^t \frac{\lambda^n}{\Gamma(n)} u^{n-1} e^{-\lambda u} du - \int_0^t \frac{\lambda^{n+1}}{\Gamma(n+1)} u^n e^{-\lambda u} du \qquad (23.63) \\
&= \int_0^t \left[\frac{\lambda^n}{(n-1)!} u^{n-1} e^{-\lambda u} - \frac{\lambda^{n+1}}{n!} u^n e^{-\lambda u}\right] du \qquad (23.64) \\
&= \left[\frac{\lambda^n}{n!} u^n e^{-\lambda u}\right]_0^t = e^{-\lambda t} \frac{(\lambda t)^n}{n!} \qquad (23.65)
\end{aligned}$$

**(23.39)** □

Il suffit d'utiliser ce théorème pour simuler une loi de Poisson de paramètre $\lambda$, ce que fait la fonction `Poisson` suivante :

```
def exponentielle (l) :
    u = random.random ()
    return - 1.0 / l * math.log (1.0 - u)

def poisson (l) :
    s = 0
    i = 0
    while s <= 1 :
        s += exponentielle (l)
        i += 1
    return i-1
```

On vérifie que cette méthode de simulation permet de retrouver les résultats théoriques. Pour cela, on effectue 1000 tirages d'une variable suivant une loi de Poisson(10) puis on compte le nombre de fois qu'on obtient chaque entier compris entre 0 et 40. La figure 23.29 permet de comparer les résultats obtenus.

```
def histogramme_poisson (l, n = 1000) :
    h = [ 0.0 for i in range (0,4*l+1) ]
```

```
    for i in range (0, n) :
        x = poisson (l)
        if x < len (h) :
            h [x] += 1
    s = sum (h)
    for i in range (0, len (h)) :
        h [i] = float (h [i]) / s
    return h

h = histogramme_poisson (10)
from pylab import *
plot (h, "r")
plot ([ proba_poisson (10, i) for i in range (0, len (h))])
legend ( ["histogramme", u"densité" ] )
show ()
```

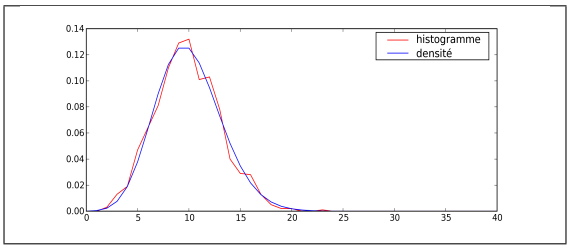



**Figure 23.29 :** *Comparaison entre une fonction de densité estimée empiriquement pour la loi de Poisson de paramètre* $\lambda = 10$ *et sa densité théorique* $f(i) = e^{-\lambda}\frac{\lambda^i}{i!}$.

On cherche maintenant à calculer les probabilités $\mathbb{P}(N = i)$ sachant que $N = N_1 + 2N_2 + 3N_3$ et $N_1 \sim \mathcal{P}(48)$, $N_2 \sim \mathcal{P}(10)$, $N_3 \sim \mathcal{P}(4)$. L'addition de deux lois de Poisson indépendantes est une loi de Poisson. En revanche, si $N_1$ suit une loi de Poisson, $2N_1$ ne suit pas une loi de Poisson [12]. Il n'existe pas d'expression évidente pour la densité du mélange $N$, il faut donc simuler cette variable. C'est l'objectif de la fonction suivante.

```
def poisson_melange (params, coef) :
    s = 0
    for i in range (0, len (params)) :
        p = poisson (params [i])
        s += p * coef [i]
    return s
```

De la même manière, on estime l'histogramme du mélange avec cette fois-ci un plus grand nombre de tirages (10000). Le résultat est celui de la figure 23.30.

```
def histogramme_poisson_melange (params, coef, n = 1000) :
    h = [ 0.0 for i in range (0, 4*sum(params)+1) ]
    for i in range (0, n) :
        x = poisson_melange (params, coef)
        if x < len (h) :
            h [x] += 1
    s = sum (h)
    for i in range (0, len (h)) :
        h [i] = float (h [i]) / s
    return h

h = histogramme_poisson_melange ([48,10,4], [1,2,3])
from pylab import *
```

12. $2N_1$ est une variable paire, c'est une propriété qui n'est jamais vérifié par une loi de Poisson.

```
plot (h, "r")
plot ([ proba_poisson (80, i) for i in range (0, len (h))])
legend ( [u"N1 + 2 N2 + 3 N3", u"Poisson (80)" ], "right")
show ()
```

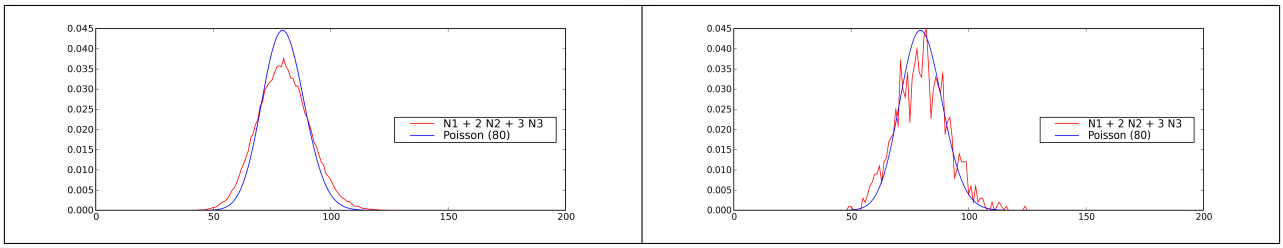



**Figure 23.30 :** *Comparaison entre une fonction de densité estimée empiriquement pour un mélange de loi Poissons $N = N_1 + 2N_2 + 3N_3$ vérifiant $N_1 \sim \mathcal{P}(48)$, $N_2 \sim \mathcal{P}(10)$, $N_3 \sim \mathcal{P}(4)$ avec la densité de la loi de Poisson de paramètre $\lambda = 80 = 48 + 2 * 10 + 3 * 4$. Il apparaît que ce sont deux densités différentes, celle du mélange étant plus applatie. La seconde image montre ce qu'on obtient lorsque le nombre de tirages n'est pas assez important.*

On utilise ces éléments pour modéliser la demande de poulets selon ce mélange de lois Poisson. Le premier programme est modifié pour aboutir au suivant. La figure 23.31 montre le résultat obtenu.

```
# -*- coding: cp1252 -*-
import math
import random
import psyco
psyco.full ()

proba_poisson_melange_tableau = []

def exponentielle (l) :
    u = random.random ()
    return - 1.0 / l * math.log (1.0 - u)

def poisson (l) :
    s = 0
    i = 0
    while s <= 1 :
        s += exponentielle (l)
        i += 1
    return i-1

def poisson_melange (params, coef) :
    s = 0
    for i in range (0, len (params)) :
        p = poisson (params [i])
        s += p * coef [i]
    return s

def histogramme_poisson_melange (params, coef, n = 100000) :
    h = [ 0.0 for i in range (0, 4*max(params)) ]
    for i in range (0, n) :
        x = poisson_melange (params, coef)
        if x < len (h) :
            h [x] += 1
    s = sum (h)
    for i in range (0, len (h)) :
        h [i] = float (h [i]) / s
    return h

def profit (N,X,p,q,s) :
    if X <= N : return X * (q-p)
```

```
    else : return X * (s-p) + N * (q-s)

def proba_poisson_melange (params, coef, i) :
    global proba_poisson_melange_tableau
    if len (proba_poisson_melange_tableau) == 0 :
        proba_poisson_melange_tableau = histogramme_poisson_melange (params, coef)
    if i >= len (proba_poisson_melange_tableau) : return 0.0
    else : return proba_poisson_melange_tableau [i]

def esperance (X,p,q,s,params, coef) :
    res = 0.0
    for i in range (0,4*sum(params)) :
        res += profit (float (i),X,p,q,s) * proba_poisson_melange (params, coef,i)
    return res

def maximum (p,q,s,params, coef) :
    res = []
    for X in range (0, 4*sum (params)) :
        r = esperance (X,p,q,s,params, coef)
        res.append ((X,r))
    return res

def find_maximum (res) :
    m = (0,0)
    for r in res :
        if r [1] > m [1] :
            m = r
    return m

if __name__ == "__main__" :
    res = maximum (2,5,1,[48,10,4], [1,2,3])
    m = find_maximum (res)
    print m
    # dessin d'une courbe
    from pylab import *
    plot ( [ r [1] for r in res] )
    show ()
```

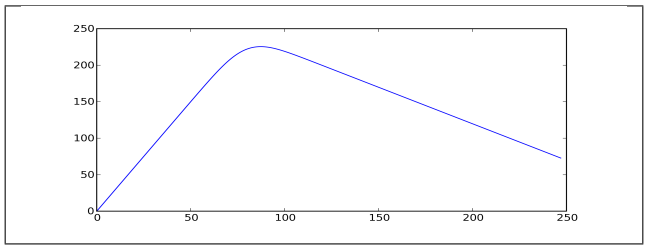



**Figure 23.31 :** *Dans le cas du mélange de lois Poisson décrit par la figure 23.30, le maximum est cette-fois ci obtenu pour 87 poulets et est de 225€. Ces résultats sont légèrement différents de ceux obtenus par une simple loi Poisson (80).*

### 23.5.3 Variations saisonnières

Les paragraphes précédents supposent que la demande est constante et ne dépend pas des saisons. Cette affirmation est peut-être vraie en ce concerne les poulets mais ce n'est certainement pas le cas des huîtres qui sont traditionnellement consommées en décembre. Appliqué à l'exemple des poulets décrits dans cet énoncé, la loi de Poisson appliquée à la consommation dépend maintenant de la semaine.

Tenir compte de la saisonnalité n'est pas forcément un problème de modélisation mais plutôt d'estimation. Au lieu d'avoir une seule consommation moyenne, il y a en aura maintenant 52. Ceci implique d'avoir des

données en nombre suffisant pour estimer les paramètres du modèle : la précision des résultats dépend de celle de l'estimation. Il est possible d'estimer séparément les variations saisonnières et la demande elle-même mais la présentation de ces techniques dépassent le cadre de ce livre, il est préférable de se reporter à [Gouriéroux1983] ou [Saporta2006].

### 23.5.4 Prolongations

Les poulets soldés ne sont pas plus mauvais que les poulets non soldés bien que la date de péremption soit certainement plus rapprochée de la date d'achat. On suppose qu'un gérant concurrent de ce supermarché a eu vent de la technique d'optimisation du magasin, il connaît également le prix du poulet et son prix soldé. Il a également accès au prix d'achat puisqu'il se fournit chez les mêmes agriculteurs. Il lui reste à connaître le nombre de poulets commandés et une estimation de la demande pour savoir si les poulets de son concurrents se vendent mieux que les siens. Il se rend dans le supermarché concurrent tous les jours où les poulets sont soldés et les comptent. Il voudrait pouvoir en déduire le nombre de poulets vendus.

## 23.6 Stratégie automatique de trading en finance

Une stratégie automatique de trading désigne dans le meilleur des cas un algorithme qui prend des décisions de trading et exécute seul les ordres résultants sur les marchés financiers. Cela suppose que cet algorithme reçoive en temps réel les cours financiers et soit également capable de transmettre automatiquement ses ordres à un broker. Le rôle du trader consiste alors à surveiller son algorithme.

L'informatique rend de plus en plus vraisemblable un tel scénario en particulier au sein de certains *Hedge Funds* dont c'est la spécialité [13]. Il va de soi qu'un tel système de trading automatique doit être réalisé avec le plus grand soin car les erreurs de conception ou les bugs ont un impact immédiat en terme financier. Cet automatisme extrême est impératif dans le cas de stratégie de trading intraday, qui réagissent en temps réel à la moindre variation des cours. Il n'est pas indispensable pour des stratégies à plus long terme, dans ce cas, les algorithmes ne sont pas directement reliés aux marchés, ils génèrent des listes d'ordres qui sont effectués manuellement par un trader ou au pire servent d'outils d'aide à la décision.

Néanmoins, ces méthodes se développent de plus en plus, leur efficacité est basée sur l'hypothèse que le futur est similaire au passé. En pratique, l'efficacité d'une stratégie est souvent limitée, il faut sans cesse améliorer les stratégies existantes et en imaginer d'autres. Les paragraphes qui suivent donne une première idée de ce en quoi consiste la construction d'un tel algorithme. Avant de d'être utilisés pour trader directement, ces algorithmes servent d'abord d'aide à la décision. Ils proposent des transactions que le trader est libre ou non d'exécuter.

Il existe un grand nombre de livres décrivant le monde de la finance, cette partie succinte ne prétend pas en donner un reflet exhaustif, elle s'intéresse à un aspect en particulier qui est celui du trading automatique. Un livre est considéré comme la bible en finance, il s'agit de *Options Futures et autres actifs dérivés* de John Hull ([Hull2007]). Il existe certains raccourcis comme le livre *La finance quantitative en 50 questions* de Paul Wilmott ([Wilmott2008]) qui aborde cinquante thèmes ou questions récurrentes en finance. Ce second livre se conclut par des conseils concernant la rédaction d'un CV ou des questions types utilisées lors des entretiens pour tester la vivacité d'esprit des candidats. Un autre livre autour des Hedge Funds intitulé *Hedge funds, private equity, marchés financiers les frères ennemis ?* dirigé par Bertrand Jacquillat ([Jacquillat2008]), c'est un recueil d'articles autour de l'utilité des Hedge Funds au sein du système économique et financier mondial, l'un d'eux se penche sur des moyens de mieux réguler leur activité pour l'instant peu contrainte. Un tiers des Hedge Fund en 2007 était établi dans les Iles Caymans.

Je citerai également deux autres livres concernant les Hedge Funds. Le premier *Hedge Funds, tome 1, Histoire de la gestion alternative et de ses techniques* de Rocchi et Christiaens ([RocchiChristiaens2007])

13. C'est le cas du Hedge Fund *Renaissance*.

décrit les différents style de gestion des Hedge Funds, il revient également sur les personnalités qui ont marqué l'histoire des Hedge Funds. Le second *Hedge Funds* de Gérard-Marie Henry ([Henry2008]) aborde les mêmes thèmes mais est mieux structuré.

### 23.6.1 Domaines financiers

L'idée d'utiliser des algorithmes pour faire du trading convient bien lorsqu'on cherche à concevoir une stratégie qui nécessite de nombreux calculs parfois complexes, lorsqu'on cherche à dépasser les capacités de calculs du cerveau humain. Toutefois la complexité n'est pas souvent la bienvenue, il impératif de comprendre un algorithme lorsque celui-ci prend la décision d'acheter ou de vendre pour plusieurs millions d'euros afin de s'assurer de la pertinence de ses ordres. On préfèrera coupler deux stratégies simples mais de styles différents plutôt que de les combinés en une seule dont la compréhension sera difficile.

#### 23.6.1.1 Marchés liquides

Les algorithmes de trading sont de préférence utilisés sur des marchers financiers liquides et sur des produits standard tels que les actions, les Futures [14], les devises [15] ou encore les taux d'intéret [16]. Des produits simples et liquides limitent considérablement les contraintes associées à leur trading. L'algorithme en sera d'autant moins compliqué à concevoir et à surveiller.

La liquidité est une caractéristique essentielle, elle détermine la facilité avec laquelle on peut vendre et acheter un produit financier. On fait souvent la distinction entre *large cap*, *mid cap*, *small cap*, c'est-à-dire les grandes capitalisations boursières, les moyennes et les petites. Sur le secteur des *small cap*, il n'est pas toujours possible de trouver un acheteur ou un vendeur pour une action donnée. Cela ne veut pas dire que le prix de cette action est momentanément nul, cela signifie qu'il n'existe momentanément personne sur le marché pour échanger.

La liquidité n'est pas une caractéristique constante, il peut arriver qu'un marché soit liquide la plupart du temps et qu'il ne le soit plus en temps de crise. La crise des subprimes liée au marché du crédit en est un bon exemple. Certains produits financiers auparavant très demandés n'ont plus trouvé preneur.

#### 23.6.1.2 Description d'une série financière

Une série financière ne se résume pas à une série de prix. Pour une série quotidienne, on dispose d'autres informations comme le premier prix de la journée (Open), le dernier prix (Close), le plus haut prix (High) et le plus bas (Low). Une autre information importante est le volume de transactions. La figure 23.32 représente toutes ces informations sur un seul graphique pour une vingtaine de jours.

Cette représentation est valable quelque soit la période de la série, que ce soit un jour, cinq minutes, il existe toujours quatre prix, Open High Low Close. Ces mêmes données sur des périodes de cinq minutes peuvent être utilisées pour faire du trading intraday : le trader quitte toutes ses positions chaque soir et ne s'intéresse qu'aux variations des prix au sein d'une même journée. La figure 23.33 représente deux jours d'une telle série. Ce graphe montre que le volume de transactions n'est pas constant, il est souvent élevé lorsque marchés européens et américains sont ouverts ensemble, il est également très élevé lorsque les acteurs du marchés attendent une information financière comme l'annonce d'une baisse des taux ou l'indice de satisfaction des ménages américains [17].

14. Les Futures sont des produits financiers reliés à un sous-jacent qui ne sera échangé entre le vendeur et l'acheteur qu'à une date ultérieure qu'on appelle *maturité*. Lorsqu'on achète un produit Future sur du pétrole par exemple, on s'engage en fait à acheter ce dit pétrole à une date fixée par le contrat. Au moment de la transaction, il n'est pas nécessaire d'avoir effectivement l'argent, on pourra toujours clore ce contract avant d'avoir à acheter réellement le pétrole. A cause de la spéculation financière, il y a bien plus de contrats ouverts que de pétrole disponible.

15. Parier sur une devise, c'est parier son évolution par rapport à une autre devise, par exemple, le cours Euro/Dollars.

16. Les taux d'intérêt ne sont pas les mêmes d'un pays à l'autre, ces taux ne sont pas non plus constants dans le temps.

17. Le site http://www.abcbourse.com/marches/stats_eco.aspx répertorie quelques statistiques économiques, principalement américaines, qui sont observées attentivement par les marchés financiers. Une valeur plus haute ou plus basse qu'attendue peut entraîner temporairement de gros volumes d'échanges et de grandes variations de prix sur la plupart des marchés.

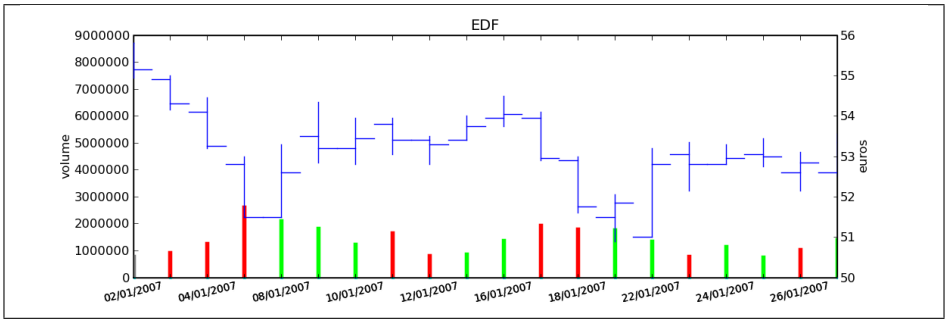



**Figure 23.32 :** *Graphe Open-High-Low-Close-Volume d'une série financière. Les histogrammes représentant les les volumes, vert pour journée positive, rouge pour une journée négative. Chaque barre verticale relie les prix Low et High d'une même journée, les barres horizontales sont les prix Open à gauche et Close à droite.*

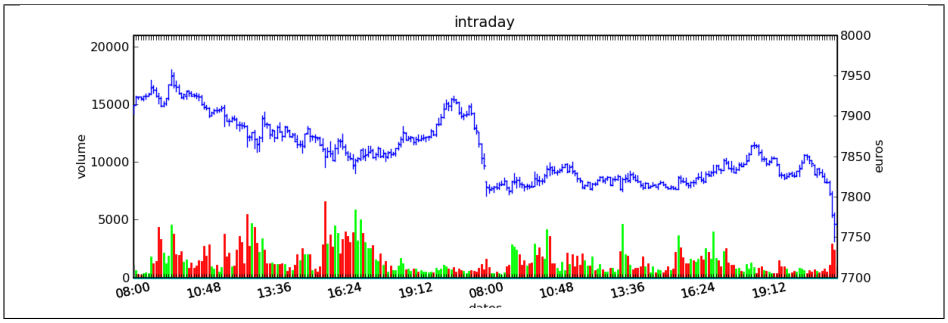



**Figure 23.33 :** *Graphe Open-High-Low-Close-Volume d'une série financière intraday. Les volumes représentés ici sont ceux d'une série européenne, il y a une première vague avant midi, juste avant la pause déjeuner, il y a une seconde vague qui correspond à l'ouverture des marchés américains. Certaines statistiques américaines tombe parfois à 13h30 heure française et ont un fort impact très localisé dans le temps sur les séries financières les plus traitées.*

Certains organismes financiers bâtissent des stratégies qui nécessitent des données encore plus précises qu'on appelle données *tick by tick*. Un tick est un ordre exécuté, il correspond à une quantité et un prix, ce prix est la valeur de l'action jusqu'au prochain tick. Ainsi les quatre prix Open High Low Close sur une période sont calculées à partir des données tick by tick. Ces données sont très volumineuses et nécessitent des systèmes informatiques spécialisés. Elles sont aussi bruitées, il arrive parfois qu'un ordre passé à un temps $t$ ne soit répercuté dans la série que plus tard.

On associe souvent aux données tick by tick les carnets d'ordres : ce sont les intentions de chaque acteur du marché, elles sont classées par prix croissants pour les intentions de vente et décroissants pour les intentions d'achat. Une donnée souvent mesurée est l'écart entre le prix de vente le plus bas et le prix d'achat le plus haut qu'on appelle *bid-offer spread*. Cette différence est positive (voir figure 23.34). Le prix d'une action n'évolue pas de manière continue, il ne peut augmenter ou diminuer que d'une valeur précise appelée *tick*. Fournir des données financières nettoyées est un service qui se vend assez cher.

#### 23.6.1.3 Rendements, Volatilité, Corrélation

Le CAC40 a perdu 0.2% aujourd'hui, -0.2% est le rendement du CAC40 sur cette journée. Si on définit une série ou quotidienne ou *daily* par $(X_t)_t$, le rendement journalier est défini par :

$$r_t = \ln \frac{X_t}{X_{t-1}} \sim \frac{X_t - X_{t-1}}{X_{t-1}} \tag{23.66}$$

Les logarithmes se manipulent assez bien puisque un rendement annuel devient la somme des rendements quotidiens. Ce dernier n'est souvent pas très représentatif, on préfère un rendement annualisé. Comme il

CARNET D'ORDRES — Tout afficher

| Ordres | Qté | Achat | Vente | Qté | Ordres |
|---|---|---|---|---|---|
| **1** | **360** | **41.96** | **41.98** | **775** | **1** |
| 1 | 340 | 41.92 | 42.00 | 33 660 | 4 |
| 1 | 94 | 41.91 | 42.01 | 5 169 | 1 |
| 1 | 197 | 41.90 | 42.03 | 174 | 2 |
| 1 | 360 | 41.88 | 42.04 | 10 565 | 1 |
| 3 | 2 509 | 41.87 | 42.05 | 11 923 | 1 |
| 1 | 110 | 41.86 | 42.06 | 1 963 | 1 |
| 1 | 3 400 | 41.85 | 42.07 | 7 716 | 1 |
| 1 | 11 923 | 41.83 | 42.08 | 5 964 | 2 |
| 1 | 7 933 | 41.82 | 42.10 | 14 459 | 4 |
| **12** | **27 226** | **TOTAL** | **TOTAL** | **92 368** | **18** |

**Figure 23.34 :** *Exemple de carnet d'ordres, tous les ordres ne sont pas représentés.*

y a environ $N = 220$ jours de trading par an, le rendement annualisé devient :

$$R^{year} = NR^{day} \Longleftrightarrow R^{day} = \frac{R^{year}}{N} \tag{23.67}$$

Un rendement annuel de 10% correspondant à un rendement journalier moyen de 0,045%. Un rendement de 10% par an est un excellent rendement mais il est intéressant de savoir si ce résultat a été obtenu de façon graduelle tout au long de l'année ou si c'était plutôt par à coup. C'est ce que tente de mesurer la volatilité d'une série : c'est l'écart-type des rendements.

$$V^{day} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(r_t^{day} - \overline{r_t^{day}}\right)^2} \tag{23.68}$$

$$\text{avec } \overline{r_t^{day}} = \frac{1}{N}\sum_{i=1}^{N} r_t^{day} \tag{23.69}$$

Cette volatilité est quotidienne ou *daily*, là encore, on préfère parler de volatilité annualisée [18] :

$$\sigma^{year} = \sqrt{N}\sigma^{day} \Longleftrightarrow \sigma^{day} = \frac{\sigma^{year}}{\sqrt{N}} \tag{23.70}$$

Un dernier indicateur souvent utilisée est l'indice de corrélation entre deux séries. Lorsqu'on compare deux séries financières issues du même secteur (Société Générale et BNP par exemple), il est fort probable que ces deux séries réagissent de manière similaire à des événements économiques relatifs au secteur bancaire. Pour mesurer la proximité entre ces deux séries, on utilise la corrélation entre rendements :

$$\rho(R_1, R_2) = \frac{1}{N\sigma_1^{day}\sigma_2^{day}}\sum_{i=1}^{N} R_{1i}^{day} R_{2i}^{day} \tag{23.71}$$

Cet indicateur est compris dans l'intervalle $[-1, 1]$. Il est souvent calculé sur la dernière année écoulée, il exprime la proximité de deux séries financières. Investir sur deux actions fortement corrélées revient à investir le double sur une seule des deux puisque les rendements sont sensiblement les mêmes.

Ces indicateurs sont des moyennes, ils peuvent être estimés sur des périodes plus ou moins longues mais de par leur nature, ils sont peu sensibles aux crises courtes. Pendant ces crises, le comportement des séries se ressemblent quelque soit le marché, la corrélation reflètent mal ces rapprochements momentanés.

18. Le rendement annuel est la somme des rendements quotidiens, on suppose que ceux-ci sont tous indépendants les uns des autres et identiquement distribués, par conséquent :

$$\sigma^{year} = \sqrt{\mathbb{E}\left((R^{year})^2\right)} = \sqrt{\mathbb{E}\left(\sum_1^N (R_i^{day})^2\right)} = \sqrt{\mathbb{E}\left(N(R^{day})^2\right)} = \sqrt{N}\sigma^{day}$$

#### 23.6.1.4 Moyenne mobile, bandes de Bollinger

La moyenne mobile est un indicateur couramment utilisé car il permet de mieux visualiser la tendance d'une courbe financière en gommant les variations quotidiennes. Cette moyenne est simplement la moyenne des dernières valeurs passées. On définit l'indicateur $MM(n,t)$ à la date $t$ la moyenne mobile $n$ par [19] :

$$MM(n,t) = \frac{1}{n}\sum_{i=0}^{n-1} X_{t-i} \tag{23.72}$$

Cette moyenne est décentrée car elle ne tient compte que des valeurs passées, on dit souvent que la moyenne est en retard par rapport à la série elle-même. Cet indicateur permet de dégager une tendance mais prévoit toujours avec retard un changement de tendance. La figure 23.35 représente le cours d'une action à côté de sa moyenne mobile. Cet indicateur est couramment accompagnée des bandes de Bollinger qui enserrent la série financière dans une sorte de tube. On définit tout d'abord la distance moyenne entre la série et sa moyenne :

$$dist(n,t) = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1} \left(X_t - MM(n,t)\right)^2} \tag{23.73}$$

Les bandes de Bollinger sont définies par l'intervale $[MM(n,t) - \alpha dist(n,t),\ MM(n,t) + \alpha dist(n,t)]$ où $\alpha$ est un paramètre qui détermine la largeur du couloir. Le fait que la série sorte de ce couloir indique un jour de plus forte variation.

De nombreux autres indicateurs sont utilisés par les traders au cours de leur analyse technique. Le tableau 23.2 en reprend quelques-uns. On note par $(O_t, H_t, L_t, C_t)$ les quatre prix Open High Low Close pour la période $t$.

#### 23.6.1.5 Achats, ventes, levier

Il n'est pas nécessaire de posséder une action pour la vendre. Au sein d'une banque ou d'un Hedge Fund, vendre une action puis la racheter plus tard est tout-à-fait possible. Ce système permet d'"attraper" les tendances baissières. Il suffit de vendre au moment où la baisse commence puis d'acheter lorsque celle-ci s'arrête.

Un terme revient fréquement lorsqu'on parle de finance, il s'agit du *levier*. A priori, avec un fond de 100, il est possible d'acheter pour 100 d'actions. En empruntant 100 de plus, il est alors possible d'acheter pour 200 d'actions. On dit que le levier est de 200% ou que la stratégie est *leveragé* [20] deux fois. Cette pratique est particulièrement intéressante lorsque la performance du Hedge Fund est bien supérieure au taux de l'emprunt. Ces sociétés ont pris beaucoup d'essort entre 2001 et 2007, favorisés par la politique de taux bas (2%) pratiqués par Banque Fédérale Américaine. Le danger apparaît en temps de crise, un fort levier implique des pertes possibles beaucoup plus grandes. Le livre [Jacquillat2008] ouvre une discussion quant à la responsabilité des Hedge Funds durant la crise des subprimes.

#### 23.6.1.6 Hedge Funds

Accusés de biens des maux lors de la crise des subprimes, les Hedge Funds propose ce qu'on appelle une gestion alternative de fonds. Ils proposent des rendements en moyenne de 10% par an avec une part de risque un peu plus importante. Majoritairement basés aux Iles Cayman et aux Etats-Unis (voir table 23.3), ils favorisent des placements à très courts termes (quelques mois) et sont fortement *leveragés*.

La table 23.3 présente également différentes stratégies utilisés par les Hedge Funds. Parmi ces fonds, 31% sont multi-stratégies : ce sont des fonds de fonds, des Hedge Funds qui investissent dans d'autres fonds. La description des autres stratégies s'appuie sur le livre [RocchiChristiaens2007].

19. MA pour Moving Average en anglais.
20. anglicisme couramment utilisé pour désigner un fort levier

| | |
|---|---|
| moyenne mobile exponentielle | C'est une moyenne mobile qui accorde plus de poids aux valeurs récentes. Elle est définie par un paramètre $\alpha$ : $MME_\alpha(t) = \alpha X_t + (1-\alpha)MME_\alpha(t-1)$. Le paramètre $\alpha$ est souvent défini comme étant $\alpha = \frac{2}{d+1}$ où $d$ est un nombre de périodes. MME = EMA pour Exponential Moving Average en anglais. |
| True Range | Il est défini par $TR_t = \max\{C_{t-1}, H_t\} - \min\{L_t, C_{t-1}\}$. Il donne une estimation différente des variations opérées par une série financière durant la période. L'indicateur $TR_t$ permet de prendre en compte la variabilité intra-période (intraday si les périodes sont des jours). Rapporté à la série elle-même, c'est une mesure semblable à la volatilité mais beaucoup plus réactive puisque ce n'est pas une moyenne. |
| MACD | Cet indicateur sert à détecter les changements de tendance en calculant la différence entre deux moyennes mobiles d'horizons différents $n > m$ : $MACD_{m,n}(t) = EMA_m(t) - EMA_n(t)$. Un changement de signe indique un changement de tendance. Pour un trading daily, les horizons sont souvent choisis dans l'ensemble $(n, m) \in \{9, 12, 26\}^2$. En pratique, l'indicateur utilisé n'est pas directement le MACD mais une moyenne mobile de celui-ci. |
| Parabolic SAR (Stop And Reverse) | Cet indicateur cherche à détecter les tendances, il est défini par : $SAR(t) = SAR(t-1) + \alpha\,(EP(t) - SAR(t-1))$. $\alpha$ est un paramètre en général fixé à $0,02$ et qui croît de $0,02$ à chaque changement de tendance jusqu'à la valeur $0,2$. $EP(t)$ désigne le prix extrême observé au cours de la tendance en cours, il correspond à un prix maximal pour un trend haussier et un prix minimal pour un trend baissier. Le $SAR$ détermine si le trend est haussier s'il se situe en-dessous du prix actuel, le trend est baissier s'il se situe au-dessus du prix actuel. |
| RSI (Relative Strength Indicator | Cet indicateur sert à comparer les forces des mouvements baissiers et haussiers. On définit tout d'abord les deux séries $U_t = \max\{0, C_t - C_{t-1}\}$ et $D_t = \max\{0, C_{t-1} - C_t\}$. On définit ensuite $EMU_n(t)$ et $EMD_n(t)$ comme étant les moyennes mobiles exponentielles des séries $(U_t)$ et $(D_t)$. On défint l'indicateur $RS_n(t) = \frac{EMU_n(t)}{EMD_n(t)}$. Enfin, l'indicateur $RSI_n(t) = 100 - \frac{100}{1+RS_n(t)}$. |

**Table 23.2 :** *Indicateurs techniques calculés sur une série financière définie par la donnée de $(O_t, H_t, L_t, C_t)$, les prix Open High Low Close pour la période $t$. Les indicateurs liés aux moyennes mobiles sont écrits avec la syntaxe française MM = moyenne mobile, en anglais, le terme approprié est MA = moving average, EMA = exponential moving average.*

La stratégie **Long / Short Equity** regroupe les stratégies qui prennent des positions à la fois vendeuses et acheteuses sur des actions. Le pair trading (voir paragraphe 23.6.2.3) est d'ailleurs l'une d'entre elles. Les prises de positions peuvent s'étendre sur différents secteurs économiques. Pour éviter une trop grande exposition et réduire les risques de pertes, les gérants font parfois en sorte que la somme des positions acheteuses soit équivalente à celle des positions vendeuses pour chaque secteur. Ce cas particulier s'appelle **Equity Market Neutral**.

La stratégie **Event Driven** se focalise sur les sociétés dont l'actualité est mouvementée avec un fort impact sur le cours de ses actions. Les gérants de ce type de fonds essayent d'anticiper des événements ayant trait à une société particulière comme une fusion ou une acquisition, une offre publique d'achat (OPA). L'annonce des retards de livraison de l'avion A380 rentre dans cette catégorie. La réussite nécessite une bonne connaissance de l'histoire des sociétés dont on souhaite acheter ou vendre les actions, de se pencher sur leur bilan financier.

La stratégie **Commodity Trading Advisor** ou tout simplement **CTA** s'applique à des produits Futures comme les Futures sur les matières premières ou *Commodities*. C'est le domaine de prédilection des fonds systématique qui utilisent des algorithmes de trading automatique (voir paragraphe 23.6.2). Les Futures sont des produits très liquides qui concernent aussi bien le pétrole que le blé ou l'or, les indices comme le CAC40. Un Future est la promesse d'échanger un produit à une date donnée appelée *maturité*. Par

| | |
|---|---|
| Iles Cayman | 34 % |
| USA | 20 % |
| British Virgin Islands | 14 % |
| Bermudes | 5 % |
| Luxembourg | 5 % |
| France | 4 % |
| Irlande | 3 % |
| Bahamas | 3 % |
| Guernsey | 2 % |
| Antilles Néerlandaises | 2 % |

| | |
|---|---|
| Multi-stratégie | 31 % |
| Long / Short Equity | 23 % |
| Event Driven | 13 % |
| Commodity Trading Advisor (CTA) | 6 % |
| Fixed Income Arbitrage | 5 % |
| Emerging Markets | 4 % |
| Global Macro | 4 % |
| Equity Market Neutral | 4 % |

**Table 23.3 :** *Implantation des Hedge Funds de par le monde et répartition selon les différentes stratégies de trading. Source Lipper Mars 2007, extrait de [Jacquillat2008].*

exemple, le *Brent Crude Oil* côté sur le *New-York Merchantile Exchange (NYMEX)* est un produit Future dont il existe une maturité par mois. Le café n'est échangé que tous les trois mois. Lorsqu'un Hedge Fund achète un Future Brent Crude Oil Aug08, il prend l'engagement d'acheter du pétrole à la fin du mois d'août 2008, il ne paiera qu'à cette date [21]. Les Hedge Funds n'achètent bien entendu jamais de pétrole, lorsque la fin du mois d'août arrive, il vende ce produit pour acheter celui correspondant à la maturité suivante (en anglais *roll*). Avec ce système, il n'y a pas besoin d'emprunter, une position vendeuse est aussi facile à prendre qu'une position acheteuse puisque rien n'est échangé avant la date de maturité.

La stratégie **Fixed Income Arbitrage** concerne les taux d'intérêt. Lorsque l'argent est prêté, le taux d'intérêt dépend de la durée. Cette stratégie consiste à jouer avec ces taux, à parier sur leur évolution.

Le terme **Emerging Markets** désigne des stratégies qui prennent positions sur des marchés émergents, des économies émergents comme celle du Brésil ou de la Chine. Ces économies affichent de forts taux de croissance dont les Hedge Funds espèrent profiter.

La stratégie **Global Macro** nécessite d'excellentes connaissances en économie car il s'agit de prendre des paris sur l'évolution à court terme de l'économie mondiale. Anticiper la hausse du prix du pétrole en fait partie, comme parier sur l'évolution des taux d'intérêts américains et européens ou prévoir la croissance de l'ensemble d'un secteur économique.

Prendre des paris, quelque soit la stratégie utilisée, demande plus de solides connaissances en économie que de solides connaissances mathématiques. Le terme *Hedge* signifie se couvrir, se couvrir contre un pari trop risqué comme prendre des positions inversées sur des produits similaires, acheter ou vendre des options [22].

Les investisseurs qui souhaitent investir dans un Hedge Funds regardent son *track record* qui désigne sa performance passée, sa capacité à afficher des rendements positifs chaque année, synonyme d'une bonne gestion. L'investisseur regarde aussi la volatilité de la performance, lorsqu'elle est élevée, l'incertitude sur la performance est plus grande. L'investisseur regarde également la corrélation avec le marché, le Hedge Funds est un placement risqué, une bonne gestion signifie aussi une absence de corrélation avec le marché, ce qui le rend supposément moins sensible aux crises du marché.

21. Cette affirmation n'est pas complètement vraie, pour éviter qu'un investisseur ne fasse défaut, il est tenu de verser une somme forfaitaire (un appel de marge) à la chambre des compensations, il doit compléter cette somme dès que le prix du Future s'écarte par paliers du prix initial.

22. Une option est un produit financier qui permet d'assurer l'acheteur de cette option contre une variation des prix. L'acheteur d'une option achète le droit d'acheter ou de vendre une action à un prix donné et à une date donnée. L'acheteur peut ou non exercer son droit d'acheter ou de vendre. Par exemple, un acteur achète une option qui lui confère le droit d'acheter dans un mois une action à 110 euros sachant qu'elle est à 100 aujourd'hui. Si au bout d'un mois, l'action est à 120, l'acheteur exercera son option, son bénéfice sera de 10 moins le prix de l'option, dans le cas contraire, il n'exercera pas son option, il ne paiera que le prix de l'option. Les termes *call* et *put* sont couramment utilisés pour désigner les options. Un call est le droit d'acheter, un put est le droit de vendre. Estimer le prix d'une option relève du calcul stochastique.

### 23.6.2 Familles de stratégies

Les paragraphes qui suivent présentent différentes stratégies qui cherchent à capter chacune un aspect particulier d'une série financière. Ces stratégies s'appuient principalement sur des informations numériques calculées à partir des données numériques elles-mêmes (Open High Low Close Volume). Une exception pourtant : les stratégies de type *style* s'appliquent aux actions et utilisent d'autres informations relatives à la société émettrice de ses actions comme le chiffre d'affaire et tout autre chiffre extrait de leur bilan financier.

Il n'existe pas de meilleures stratégies, il est rare qu'une stratégie efficace sur tous les secteurs économique ou sur tous les types de produits, action, pétrole, indices, taux, ... Il est rare qu'une stratégie soit tout le temps performante, il est toujours préférable de constuire un système en utilisant plusieurs, la volatilité est moins élevée.

Il n'est pas simple d'intégrer dans des systèmes automatiques des informations quantitatives relatives aux informations économiques comme l'annonce d'un plan de licenciement, des retards dans les livraisons d'avions, une nouvelle dépréciations d'actifs. Outre la complexité qu'entraînerait la prise en compte des telles informations, un argument qui justifie la seule utilisation de l'analyse technique est l'efficience des marchés : les nouvelles économiques sont prises en compte par les prix eux-mêmes qui sont la résultante des ordres passés sur le marché. Les marchés financiers corrigent d'eux-mêmes les prix car ils intègrent toute l'information connue.

Un second argument, sans doute plus probant, est que ces nouvelles économiques ayant un fort impact sont rares. Une stratégie s'intéresse avant tout à un comportement moyen. Une moyenne mobile ne peut pas prendre en compte un jour de trading aberrant, une journée de crise, une stratégie cherche avant tout à profiter d'un comportement récurrent d'une série telle qu'une tendance et doit faire l'impasse sur des comportements erratiques et passagers. Ces derniers ne sont pas assez fréquents pour être étudiés, ils sont à chaque fois différents et leur compréhension dépasse le cadre de l'analyse technique.

#### 23.6.2.1 Trend Following

Le *Trend Following* consiste à suivre une tendance qu'elle soit haussière ou baissière. Une simple stratégie de Trend Following est illustrée par la figure 23.35. Ce type de stratégie parie sur le long terme, le temps nécessaire pour qu'une tendance se forme et dure. On parle parfois d'attraper une tendance : la stratégie prend plusieurs fois de mauvaises décisions et décide de couper sa position [23] peu de temps après, de temps en temps, la décision est bonne et la pose est gardée le plus longtemps possible, jusqu'à ce que la tendance prenne fin.

Cette stratégie prend une position acheteuse ou *long* lorsque la tendance est haussière et vendeuse ou *short* lorsque la tendance est baissière. Ce type de stratégie est averse au changement de tendance qu'il détecte le plus souvent avec retard, surtout si cette stratégie ne s'appuie que sur des moyennes mobiles. Plus généralement, lorsque la volatilité est grande, ce type de stratégie est déconseillée, il est préférable d'utiliser le *Mean Reversing*.

Concevoir un indicateur de tendance n'est pas chose facile. Même si l'œil humain est habitué à analyser des courbes financières, il n'en est pas de même pour un algorithme qui fait face aux effets de seuil. La stratégie décrit par la figure 23.35 prend des décisions lorsque sa courbe touche une de ses bandes de Bollinger. Un expert humain pourra prendre une décision si la distance entre la courbe et la bande est petite visuellement, un ordinateur a besoin de seuils constants pour prendre sa décision qui est binaire. On pourrait modifier la largeur de la bande de Bollinger mais la stratégie est souvent très sensible à cette largeur. Ce point sera évoqué plus loin au paragraphe 23.6.3.

Un autre facteur est la longueur de la tendance. La stratégie toujours décrite par la figure 23.6.3 s'appuie sur une moyenne mobile de 50 jours. Elle détecte bien les tendances dont la longueur se situe autour de

23. On parle de position ou pose pour une quantité négative ou positive d'actions. Couper sa pose consiste à annuler sa position : tout vendre si on possédait des actions ou tout acheter si la position était négative. Après avoir coupé sa position, le portefeuille n'est plus constitué que d'argent.

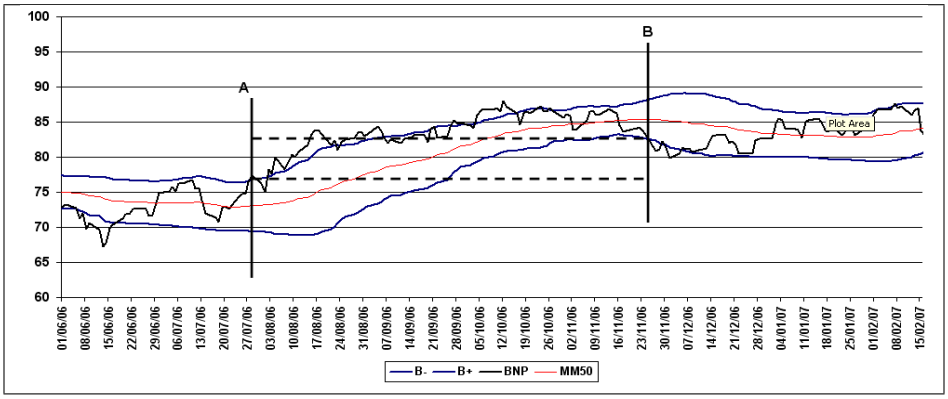



**Figure 23.35 :** *Cours de l'action BNP accompagné par sa moyenne mobile 50 et ses bandes de Bollinger. Une stratégie simple de trend following consiste à acheter lorsque le cours dépasse sa bande supérieure de Bollinger (point A) et à revendre lorsque le cours passe sa bande inférieure (point B). Le gain est alors la différence des cours d'achat et de vente. Lorsque la tendance est baissière, il suffit de vendre d'abord puis d'acheter ensuite.*

cette valeur mais elle est susceptible de prendre de mauvaises valeurs si la tendance est plus courte ou si une trentaine de jours assez volatiles s'immiscent au sein d'une tendance longue de plusieurs mois.

En pratique, la stratégie est munie de quelques mécanismes qui permettent de limiter les pertes. Lorsque la stratégie achète une action à un prix $p$, si le prix descend en dessous d'un seuil égal à $p(1-\epsilon)$ alors la position est coupée. La position est coupée si le prix passe au-dessus du seuil $p(1+\epsilon)$ dans le cas d'une position vendeuse. Ce système permet de limiter les pertes de la stratégie, il est souvent présent dans les stratégies qui suivent avec le même objectif, réduire le risque.

Cette règle peut encore être améliorée lorsque la stratégie est gagnante depuis un certain temps, cette règle propose de stopper la stratégie à un niveau loin de son niveau actuel. Dans ce cas, on rapproche de temps en temps le niveau auquel la position est coupée pour éviter des pertes trop importantes. On rapproche le niveau de stop. Dans le même ordre d'idée, on peut décider de couper sa pose parfois lorsque la courbe s'éloigne beaucoup de sa moyenne mobile et que la position est gagnante : c'est prendre ses profits avant que ceux-ci ne diminuent.

#### 23.6.2.2 Mean Reversing

C'est la stratégie opposé au Trend Following. Elle s'intéresse aux périodes durant lesquels la volatilité est élevée et où aucune tendance ne se dégage. Cela suppose que le cours de l'action va osciller autour d'une position d'équilibre et que la meilleure stratégie à suivre est d'acheter lorsque le cours vient de baisser et de vendre lorsque le cours vient de monter.

En terme de décision, cette stratégie prend beaucoup de poses, les garde peu de temps et celles-ci sont souvent gagnantes. La stratégie commence à perdre lorsque la période de haute volatilité laisse place à une nouvelle tendance. Cette situation est mal gérée par cette stratégie qui se retrouve avec une position inverse à celle qu'elle aurait dû prendre. La figure 23.36 illustre ce principe.

Le terme *Contrarian* apparaît parfois pour désigner cette stratégie. Ce terme désigne une stratégie dont les positions prises sont inverses au consensus suivi par le marché. Toutefois, la stratégie est souvent munie d'un mécanisme limitant les pertes tel que celui décrit au paragraphe 23.6.2.1.

#### 23.6.2.3 Pair Trading

Le pair trading consiste à construire un portefeuille de deux actions. On étudie dans ce cas la série du rapport des prix des deux actions. On choisit le plus souvent deux actions appartenant au même secteur économique (BNP, Société Générale par exemple) de façon à obtenir une série moins sensible aux événements économiques. En cas de crise ou de rebond du secteur, les deux actions sont toutes deux

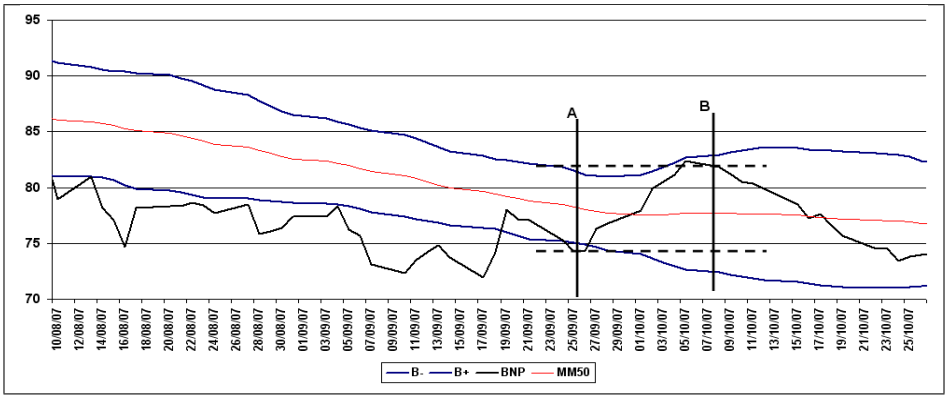



**Figure 23.36 :** *Cours de l'action BNP accompagné par sa moyenne mobile 50 et ses bandes de Bollinger. Une stratégie simple de mean reversing consiste à acheter lorsque le cours dépasse sa bande inférieure de Bollinger (point A) et à revendre lorsque le cours revient vers sa borne supérieure (point B). Le gain est alors la différence des cours d'achat et de vente.*

susceptibles d'être atteintes, le rapport des prix ne dépend plus que des différences des deux sociétés. Ce procédé permet de construire une série moins sensible aux tendances qui s'appliquent à un secteur dans son ensemble (voir figure 23.37).

Les stratégies appliquées à ce rapport de prix sont plutôt de type mean reversing, on s'attend à ce que temporairement le rapport des prix s'écarte de sa moyenne puis y reviennent. La différence intervient lors de la prise de décision, au lieu d'acheter ou de vendre une action, prendre une position consiste à acheter une action et vendre l'autre, quitter la position revient à effectuer la manipulation inverse.

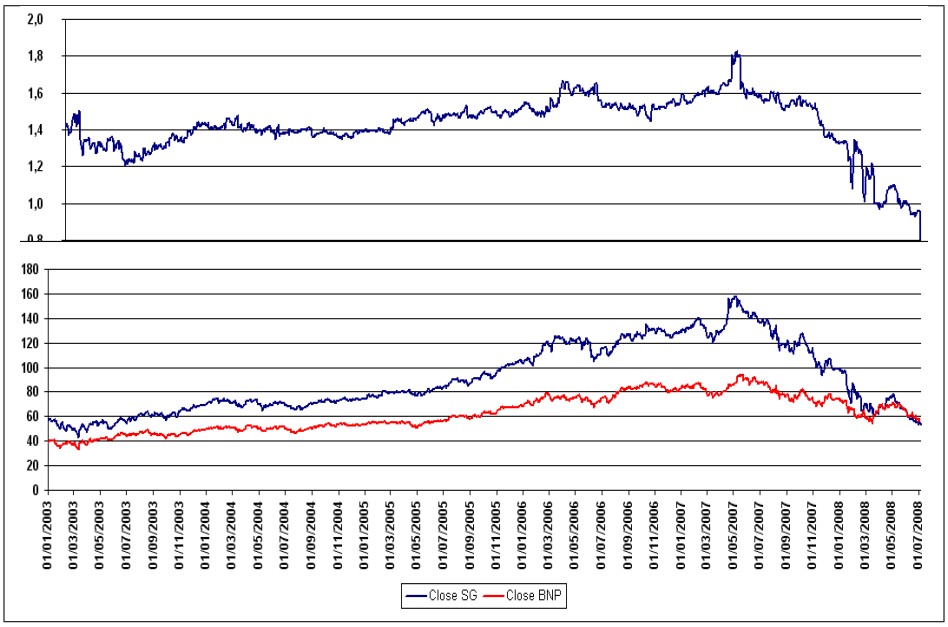



**Figure 23.37 :** *Rapport entre l'action Société Générale et l'action BNP. On observe une croissance supérieure pour la Société Générale jusqu'en août 2007 date du début de la crise des subprimes puis une nette dégradation depuis l'affaire Kerviel en janvier 2008. Auparavant, la série du rapport paraît plus stable et il semble plus judicieux de faire du mean reversing.*

#### 23.6.2.4 Styles, Value, Growth

Le pair trading permet de prendre des paris sur une paires d'actions, une stratégie construite à partir de style propose une façon de jouer simultanément avec beaucoup d'actions. Elle utilise des indicateurs qui décrivent la santé financière d'une entreprise, ils sont généralement calculés à partir des bilans financiers que les sociétés cotées sont obligées de produire régulièrement. On distingue souvent deux classes de stratégies, les *growth*[24] et les *value*[25]. Les indicateurs servent à estimer si pour une compagnie, il est préférable de suivre une stratégie plutôt *growth* ou plutôt *value*.

Une société "*growth*" affiche un fort taux de croissance. Le prix de l'action est élevé mais les perspectives de croissance suggère une hausse. Il est intéressant dans ce cas d'acheter le stock. Une société "*value*" est plutôt estimée à son juste prix et les perspectives de hausse de cours de l'action sont faibles, il dans ce cas préférable d'attendre une baisse du cours avant d'acheter.

Le tableau 23.4 regroupe quelques indicateurs très utilisés pour étudier les sociétés cotées en bourse. Il existe plus d'une centaine d'indicateurs que les acteurs des marchés financiers suivent. Pour chacun d'entre eux, il faut savoir ce qu'est une bonne valeur, une mauvaise, quelle décision (acheter ou vendre) il faut prendre lorsque l'indicateur est élevé.

| | |
|---|---|
| Earnings before Interest, Taxes, Depreciation, and Amortization (EBITDA) | revenus avant Intérêts, impôts (Taxes), Dotations aux Amortissements et provisions |
| Earnings Per Share (EPS) $= \frac{\text{Net Earnings}}{\text{Outstanding Shares}}$ | c'est le bénéfice d'une entreprise rapporté aux nombres de parts ou d'action |
| Price to Sales (P/S) $= \frac{\text{Market Cap}}{\text{Revenues}}$ | c'est la capitalisation boursière rapportée au chiffre d'affaires |
| Dividend Payout Ratio (DPR) $= \frac{\text{Dividends Per Share}}{\text{EPS}}$ | c'est le dividende d'une action divisé par EPS |
| Price to Earnings Ratio (P/E) $= \frac{\text{Stock Price}}{\text{EPS}}$ ou P/E ratio $= \frac{\text{Price per Share}}{\text{Annual Earnings per Share}}$ | c'est le prix d'une action divisé par EPS ou aussi le prix d'une action divisé par le dividende |

**Table 23.4 :** *Indicateurs relatifs aux sociétés.*

Un indice P/E élevé indique un petit dividende comparé au prix de l'action, il est donc préférable de ne pas acheter. Il n'est pas toujours facile de savoir ce qu'est une valeur intéressante pour un indicateur mais on peut supposer que pour un secteur économique donné, il existe au moins une société dont l'indicateur est intéressant. A la date $t$, en classant par ordre croissant tous les indicateurs d'un même secteur économique, on peut supposer que les indicateurs extrêmes correspondent à des sociétés intéressantes.

Par exemple, supposons qu'au début de chaque mois, c'est à dire à la date $t$, on dispose d'une nouvelle valeur de l'indicateur $I_t^i$ pour la société $i$. On les trie par ordre croissant : $I_t^{\sigma(1)} \leqslant I_t^{\sigma(2)} \leqslant ... \leqslant I_t^{\sigma(N)}$. Pour cet indicateur, une petite valeur suggère une position acheteuse. Par conséquent, on va prendre une position acheteuse pour les premiers 10% et une position vendeuse pour les derniers 10%.

24. croissance
25. valeur

| société | position |
|---|---|
| $\sigma(1)$ | acheteuse |
| ... | acheteuse |
| $\sigma(10)$ | acheteuse |
| $\sigma(11)$ | - |
| ... | - |
| $\sigma(N-11)$ | - |
| $\sigma(N-10)$ | vendeuse |
| ... | vendeuse |
| $\sigma(N)$ | vendeuse |

Le mois d'après, le classement a changé, quatre cas sont possibles :

1. La société reçoit un classement équivalent et sa position ne change pas.
2. La société n'apparaît plus dans les extrémités du classement, sa position est coupée.
3. La société apparaît dans les extrémités du classement, on prend une position.
4. La société passe d'une extrémité à une autre, on retourne la position.

Cette étape qui consiste tous les mois à conserver, couper, prendre ou retourner une position est souvent appelée *rebalancing*. Il est préférable de n'utiliser cette méthode que sur des sociétés appartenant au même secteur économique, dans le cas contraire, classer les indicateurs par ordre croissant peut ne pas être pertinent.

Ce type de stratégie suppose le choix d'un indicateur ou d'une combinaison d'indicateurs. Le choix est souvent guidé par des raisons économiques et aussi l'utilisation de backtest [26] pour déterminer un bon indicateur. En ce qui concerne leur combinaison, elle n'est pas toujours facile. Tout d'abord, il faut faire attention au sens de chaque indicateur : un P/E faible suggère une position acheteuse, un DPR élevé suggère aussi une position acheteuse. Il est aussi difficile de combiner linéairement des indicateurs qui ont des ordres de grandeur différents. Une combinaison simple qui contourne ce problème est de combiner le rang des sociétés obtenus en les classant selon chaque indicateur de la combinaison. Par exemple, on classe les sociétés selon $-P/E$ et $DPR$, les rangs obtenus sont additionnés et c'est le rang final qui servira à sélectionner les sociétés.

#### 23.6.2.5 Gestion de portefeuille

Un nom est incontestablement associé à ce thème qu'est la gestion de portefeuille, c'est l'économiste Harry Markowitz qui reçu le prix Nobel d'économie pour ses travaux en 1990. L'idée repose sur la construction d'un portefeuille d'action qui permet d'obtenir un rendement donné avec un risque moindre comparé à celui de chaque action prise séparément.

Chaque action est ici décrit par son rendement moyen $R_i$ et sa volatilité $\sigma_i$. Construire un portefeuille consiste à répartir son argent entre les différents actifs financiers. On affecte un poids $\alpha_i$ à chaque action $i$. La somme des poids vérifie les contraintes suivantes :

$$\sum_{i=1}^{N} \alpha_i = 1 \tag{23.74}$$

$$\forall i \in \{1, \ldots, N\}, \ 0 \leqslant \alpha_i \leqslant 1 \tag{23.75}$$

Le rendement moyen du portefeuille est défini par :

$$R(\alpha_1, \ldots, \alpha_N) = \sum_{i=1}^{N} \alpha_i R_i \tag{23.76}$$

Si on note $\rho_{ij}$ la corrélation entre les deux actions $i$ et $j$, le risque ou la volatilité du portefeuille est définie par :

$$\sigma(\alpha_1, \ldots, \alpha_N) = \sqrt{\sum_{i,j} \alpha_i \alpha_j \sigma_i \sigma_j \rho_{ij}} \tag{23.77}$$

26. voir paragraphe 23.6.3, page 620

La construction du portefeuille optimal passe par l'optimisation sous contrainte d'un des deux problèmes suivant :

$$\left|\begin{array}{ll} & \min\limits_{(\alpha_1,\ldots,\alpha_n)} \sigma\left(\alpha_1,\ldots,\alpha_N\right) \\ \text{avec} & R\left(\alpha_1,\ldots,\alpha_N\right) \geqslant R_{min} \\ & \sum_{i=1}^{N} \alpha_i = 1 \\ & \forall i,\ 0 \leqslant \alpha_i \leqslant 1 \end{array}\right. \quad \left|\begin{array}{ll} & \max\limits_{(\alpha_1,\ldots,\alpha_n)} R\left(\alpha_1,\ldots,\alpha_N\right) \\ \text{avec} & \sigma\left(\alpha_1,\ldots,\alpha_N\right) \leqslant \sigma_{max} \\ & \sum_{i=1}^{N} \alpha_i = 1 \\ & \forall i,\ 0 \leqslant \alpha_i \leqslant 1 \end{array}\right.$$

Lorsque les exigences sur le rendement minimal $R_{min}$ ou la volatilité maximale $\sigma_{max}$ ne sont pas trop fortes, la solution mène à des poids situés dans l'intervalle $]0,1[$. Il se peut qu'il n'y ait pas de solution, dans ce cas, il faudra assouplir la contrainte sur le rendement minimal ou la volatilité maximale. Dans tous les cas, ces problèmes se résolvent grâce à la méthode des multiplicateurs de Lagrange [27].

**Remarque 23.41 : portefeuille d'actions, position vendeuse**
Si les positions vendeuses sont acceptées alors la contrainte sur les poids devient $\forall i,\ -1 \leqslant \alpha_i \leqslant 1$. Pour une stratégie *Equity Market Neutral*, on ajoute une contrainte supplémentaire qui correspond à l'exigence d'avoir autant de positions vendeuses qu'acheteuses : $\sum_{i=1}^{N} \alpha_i p_i = 0$ où $p_i$ est le prix de chaque actif.

**Remarque 23.42 : portefeuille d'actions, de stratégies**
On a supposé que le portefeuille était un portefeuille d'actions mais dans la mesure où ce dernier est défini par un ensemble de poids affectés à des objets décrits par leur rendement et leur variance, on pourrait tout à fait imaginer que ces objets sont des stratégies. La méthode de Markowitz reviendrait à répartir intelligemment son argent entre les différentes stratégies de trading.

#### 23.6.2.6 Horizon de trading, intraday, daily

Il ne paraît pas plus compliqué d'appliquer ces stratégies au trading intraday qu'au trading daily. Dans le second cas, les positions sont tenues plusieurs jours, plusieurs mois et on s'intéresse peu aux variations dans une même journée. Dans le premier cas, on s'intéresse plus particulièrement aux variations qui ont lieu dans une même journée et qui peuvent être importantes. Le prix de clôture peut être équivalent au prix d'ouverture alors que les prix ont montré des variations de 1% ou 2% au cours de la journée.

Les séries financières, les indicateurs s'expriment de la même façon que la période soit de un jour ou de cinq minutes. En pratique, certaines contraintes font que le trading intraday est plus complexe à mettre en œuvre. Le premier obstacle est informatique : stocker des prix toutes les cinq minutes est autrement plus coûteux que de stocker un prix par jour. Les temps de calcul sont également plus longs. Pour des données tick by tick, il vaut mieux être épaulé par un informaticien chevronné. Le second obstacle est statistique : il faut contrôler les données en trading intraday. Il n'est pas rare de manquer de données pendant 15 minutes puis d'obtenir des volumes de transactions quatre fois plus important pour les cinq minutes suivantes. Ceci est illustré par l'exemple suivant :

```
date time         Open    High    Low     Close   Volume
07/11/2007 19:42  7822    7823    7818    7819.5  1130
07/11/2007 19:48  7819.5  7830    7819.5  7822.5  1543
07/11/2007 19:54  7823    7827.5  7819.5  7824    1244
07/11/2007 20:00  7824    7825.5  7822.5  7824.5   216      # 20.00
07/11/2007 20:24  7828    7833    7825.5  7830     640      # 20.24, il manque 3 périodes
07/11/2007 20:30  7829.5  7831.5  7827    7829.5   478
07/11/2007 20:36  7830    7830.5  7821    7829     716
07/11/2007 20:42  7829.5  7834.5  7826    7828     681
```

27. Ce n'est pas la seule méthode, ce problème est un problème d'optimisation non linéaire avec des contraintes non linéaires, il existe d'autres méthodes de résolution comme la programmation séquentielle quadratique ou l'algorithme du gradient projeté avec contraintes d'inéglité. Le document http://www.dma.utc.fr/polytex/cours.pdf propose un cadre théorique.

A l'inverse, il peut arriver qu'aucun ordre n'ait été passé pendant cinq minutes, au tout début de l'ouverture d'un marché, ou durant la nuit sur un marché ouvert 24h/24. Dans ce cas, le volume sera nul et les prix immobiles.

Les périodes d'ouverture et de fermeture des marchés ne sont pas les mêmes d'une année sur l'autre, les bourses ont tendance à rester ouverte de plus en plus tard au fur et à mesure qu'elles s'informatisent. Il faut en tenir compte dans les historiques de données.

### 23.6.3 Analyse d'une stratégie

Cette partie décrit les principales étapes de la mise au point d'une stratégie automatique. Elle sera illustrée avec une stratégie appliquée sur le cours de la BNP (voir figure 23.38) de début 2003 à mi-2008. C'est une stratégie Trend Following fonctionnant avec trois paramètres [28] :

1. La longueur $d$ de la moyenne mobile, par défaut $d = 200$ jours.
2. La largeur $\alpha$ des bande de Bollinger, par défaut $\alpha = 1$ * la distance moyenne sur 200 jours du prix Close à sa moyenne mobile.
3. Lorsque la stratégie a pris une position acheteuse au prix $p$, si le prix redescend en deça de $p(1-\beta)$, la position est coupée. A l'opposé, si la stratégie prend une position vendeuse, si le prix monte au-dessus de $p(1+\beta)$, la position est également coupée. Par défaut, $\beta = 0.05$.

La stratégie est définie par trois règles :

1. La première règle définit l'ouverture d'une position lorsque la position est nulle. Si le cours d'une action dépasse la bande de Bollinger supérieure, la stratégie entre en position acheteuse, si le cours dépasse la bande de Bollinger inférieure, la stratégie entre en position vendeuse.
2. La seconde règle est appliquée lorsque le cours franchit à nouveau ses bandes de Bollinger alors que la position n'est plus nulle. Si le cours franchit la bande de Bollinger supérieure et que la position est vendeuse, celle-ci est coupée. Si le cours franchit la bande de Bollinger inférieure et que la position est acheteuse, la position est aussi coupée.
3. La troisième règle est destinée à limiter les pertes, si la stratégie a ouvert une position acheteuse au prix $p$ et que le cours redescend à un niveau $p(1-\beta)$, la position est coupée. A l'inverse, si la stratégie a ouvert une position vendeuse au prix $p$ et que le cours remonte à un niveau $p(1+\beta)$, la position est aussi coupée.

Cette stratégie très simple est très inefficace sur de courtes périodes très volatiles alors que le cours de l'action sort des bandes de Bollinger puis revient très rapidement à un niveau où la position est coupé par la seconde règle. Une stratégie réellement utilisée par un Hedge Funds s'appuie sur plus d'une dizaine de règles et autant de paramètres.

#### 23.6.3.1 Backtest

La validation d'une stratégie passe par son évaluation sur le passé de l'action sur laquelle on souhaite l'appliquer (voir la figure 23.39). Ce passé est appelé *backtest*. Ce passé doit être suffisamment grand : il n'est pas difficile de concevoir une stratégie gagnante sur six mois, sur dix ans, c'est moins facile. Toutefois, cette validation a deux inconvénients :

1. Il est impossible de valider la stratégie sur des situations probables dans le futur mais absentes du backtest. Ceci signifie que la stratégie est susceptible de mal se comporter pour toute situation imprévue.

28. Cette stratégie a été implémentée en *Python* et le code est disponible aux adresses suivantes :
http://www.xavierdupre.fr/hal_python/hal_python_help_html/sample_test_python_timeseries_ohlc_correlation.html
http://www.xavierdupre.fr/hal_python/hal_python_help_html/sample_test_python_strategy.html
http://www.xavierdupre.fr/hal_python/hal_python_help_html/sample_test_python_strategy_analyse.html
http://www.xavierdupre.fr/hal_python/hal_python_help_html/index.html
La dernière adresse est celle du module avec lequel les trois programmes qui précèdent fonctionnent.

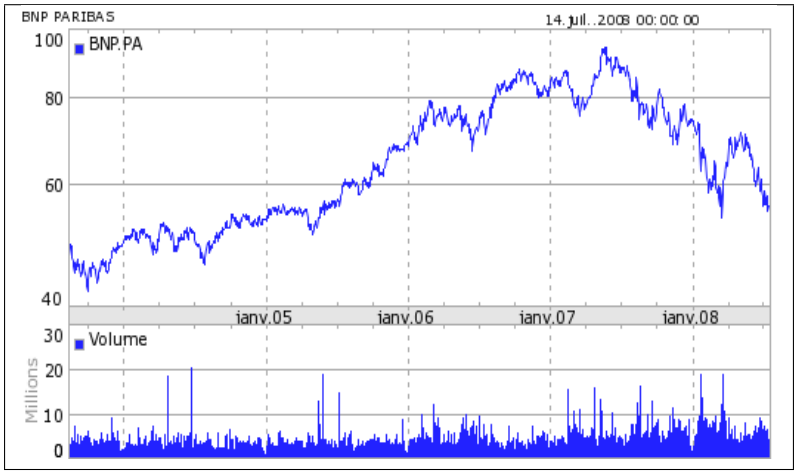



**Figure 23.38 :** *Le cours de la BNP entre début 2003 et mi 2008.*

2. Les situations de crises ne sont pas fréquentes : elles sont statistiquement peu significatives. Par conséquent, la stratégie n'est validée que sur des jours de trading "normaux" et c'est dans ces périodes qu'elle fait l'essentiel de ses gains. En temps de crise, son comportement peut décevoir.

Le backtest n'est pas toujours suffisant pour valider une stratégie. Sur un historique de cinq ans, il n'y figure que quelques crises et certainement pas le scénario de la prochaine. C'est pourquoi il faut être vigilant lors de l'utilisation de telles stratégies ou alors lui faire confiance et supposer que l'algorithme se remettra à gagner une fois la crise passée. Il peut être intéressant de valider la stratégie sur d'autres backtest provenant d'actions plus ou moins corrélées à la première. C'est une autre façon d'améliorer la qualité du backtest.

En règle générale, les stratégies sont éprouvés avec un levier de 100% : la position maximale [29] ne doit pas dépasser la somme initiale. Ce principe permet de comparer les stratégies entre elles. D'autres part, utiliser un levier plus ou plus grand modifie certes les rendements et la volatilité mais ne modifie pas leur rapport. Une bonne stratégie est de préférence une stratégie peu volatile, un levier adéquat permettra d'en augmenter le rendement.

Au final, la validation d'une stratégie sur un backtest aboutit à la courbe des gains ou Net Asset Value (NAV), ce sont les gains qui ne tiennent pas compte des frais de gestion, des charges de la société financière. Il est fréquent aussi qu'on parle de NAV non réinvestie, ceci signifie que la position est constante tout le temps du backtest même si en réalité, les gains sont réinvestis.

On observe souvent que la recherche de la meilleure stratégie sur un backtest donne des résultats nettement supérieure à ceux que la stratégie obtient lorsqu'elle est vraiment utilisée. Ceci signifie aussi que la stratégie est parfaite sur le passé : elle est trop bien ajustée. Cee écueil est quasiment inévitable, il est très difficile de savoir si une stratégie est trop performante sur le passé et risque de ne plus l'être après.

**Remarque 23.43 : plusieurs ordres passés dans la même journée**
Lors de la simulation d'une stratégie sur un backtest, il peut arriver que plusieurs ordres soient passés durant la même journée. Il est impossible de savoir dans quel ordre ceux-ci doivent être passés car seules quatre prix sont connus au cours de cette période. Il n'est pas possible de savoir si la valeur maximale a été atteinte avant la valeur minimale par exemple. On suppose malgré tout que ce genre de situation a peu d'impact sur le résultat final. S'il survient de façon trop fréquente, alors il serait sans doute avisé d'en tenir compte lors de l'attribution de valeurs aux coûts de transactions et au slippage définis ci-dessous.

29. La position maximale est la somme des positions acheteuses et vendeuses qui ne soustraient pas : on fait la somme des valeurs absolues des deux positions.

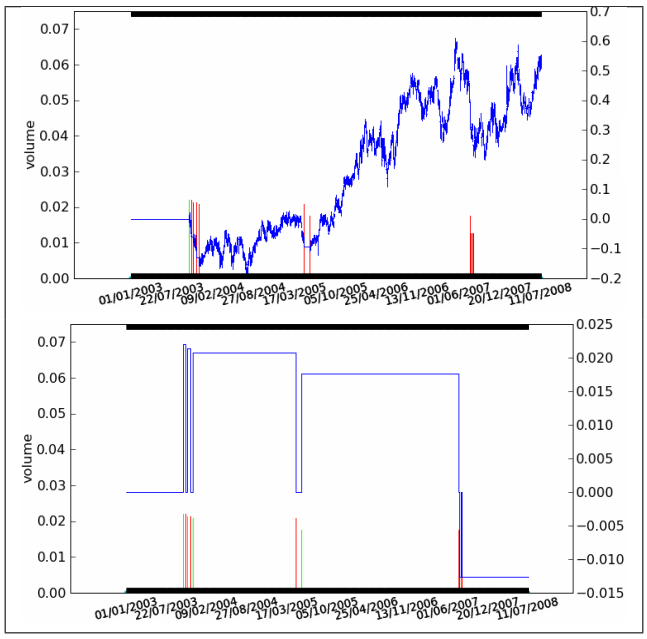



**Figure 23.39 :** *NAV de la stratégie Trend Following décrite au paragraphe 23.6.3. Le second graphe représente la position de la stratégie, une position positive signifie une position acheteuse (long), une position négivative signife une position vendeuse (short). C'est un graphe qui ne représente pas la position mais la quantité d'actions achetées ou vendues pour une position proche de un euros. Ceci explique que cette seconde courbe présente des paliers de hauteurs différentes, il s'agit de l'inverse du prix observé lors de l'ouverture d'une position.*

#### 23.6.3.2 Modélisation

Pour améliorer la validation des stratégies sur un backtest, on intègre dans le modèle deux défauts qui surviennent lors des passages d'ordres. Même utilisée sur des marchés liquides, si la stratégie impose un achat d'action à un prix donné $p$, il y a toujours un décalage entre le temps où le prix dépasse ce niveau $p$ et celui où l'ordre est passé. Ce décalage ou *slippage* peut être dans un sens ou dans l'autre mais par principe, ce décalage sera toujours supposé être en défaveur de la stratégie.

Le slippage est souvent exprimé en nombre de *tick*. En effet, le prix de tout produit côté n'est pas continu, il évolue de tick en tick. Le tick est une fraction de l'action et dépend de chaque action. Une action de 5 euros aura un tick faible, le tick d'une action de 1000 euros sera plus élevé. Un bon ordre de grandeur pour le slippage est de quelques tick.

Passer des ordres a un coût, de quelques pourcents du prix de l'action pour un particulier, de quelques dizième de pourcents pour un Hedge Fund. Une stratégie performante mais qui beaucoup d'ordres gagnera moins d'argent. Ce sont des paramètres qu'il ne faut pas négliger pour des stratégies Mean Reversing qui cherchent à profiter d'une forte volatilité grâce à de fréquents passages d'ordres. Il faut prendre en compte ce qu'on appelle les *coûts de transaction*.

Ces deux défauts peuvent être mesurés une fois que la stratégie est mise en place. Néanmoins, il est préférable de les surestimer pour tenir compte du fait qu'une fois validée sur backtest, la stratégie sera toujours utilisée sur des données nouvelles. Certains Hedge Funds donnent à ces paramètres non pas les valeurs qu'ils observent en pratique mais des valeurs plus fortes qui leur permettent de de faire décroître les performances des backtests jusqu'aux performances réellement observées.

Le slippage est ici modélisé comme une constante mais il serait sans doute plus judicieux de l'ajuster en fonction d'une variabilité locale (par rapport à la différence High - Low) qui pourrait pénaliser davantage la stratégie en temps de crise.

#### 23.6.3.3 Résultats standards

Même si le rendement d'une stratégie est le résultat important, il faut aussi regarder comment il est obtenu. C'est pour cela qu'on regarde d'autres indicateurs comme l'Information Ratio ou le ratio de Sharpe. La première étape consiste à annualiser la performance et la volatilité obtenus sur le backtest (voir paragraphe 23.6.1.3). On cherche ensuite à construire le tableau suivant qui n'est pas exhaustif :

| | |
|---|---|
| Information Ratio | C'est le rendement annualisé rapporté sur la volatilité $\frac{R}{\sigma}$. $R$ mesure la performance, $\sigma$ le risque pour l'obtenir. Si ce ratio est inférieur à 1, cela signifie que le risque est plus élevé que la performance qui est sujette à caution même si elle est importante. |
| ratio de Sharpe | Les Hedge Funds partiquent souvent un levier supérieur à 200%. Cela veut dire qu'ils empruntent pour placer le double ou le triple de l'argent qu'ils gèrent. Cet emprunt n'est pas gratuit, c'est pourquoi on retranche à la performance obtenu par le Hedge Fund le taux sans risque $r$ qui correspond au taux de l'emprunt : $\frac{R-r}{\sigma}$. |
| drawdown | C'est la perte maximale de la stratégie. Obtenir 10% en fin d'année ne veut pas dire que le système n'est pas passé par -5% en cours d'année. La perte maximale n'est pas le niveau le plus bas depuis le début de l'année, c'est le plus grand écart entre un gain maximal et une perte maximale qui lui succède. |
| rendement roulant | Lorsqu'on construit une stratégie à long terme, il peut être intéressant de construire la courbe des rendements roulant qui est par exemple pour une date $t$, la performance obtenue entre $t - 6mois$ et $t$ par exemple. Pour une stratégie à long terme, il devrait exister très peu de rendements roulant à un an négatifs. |
| corrélation | Lorsqu'un investisseur cherche à investir son argent dans un placement alternatif, il regarde si ce placement lui offre des rendements qui ne sont pas corrélés au marché. Dans le cas d'une stratégie appliquée à une action, il s'agit de calculer la corrélation entre le cours de l'action et la NAV de la stratégie. Une corrélation de 1 ou -1 signifie que la stratégie a été d'acheter ou de vendre une action puis de conserver cette position. Il est intéressant d'avoir une corrélation faible, d'avoir une stratégie qui ne reproduisent pas les pertes et les gains d'une action. C'est aussi avoir de la valeur ajoutée : la stratégie atteint son objectif, elle propose un placement alternatif. |

Il existe de nombreuses manières de mesurer la performance d'une stratégie. D'autres critères peuvent être mesuré comme le nombre d'ordres passés, la VAR (Value At Risk) qui mesure les pertes maximales quotidiennes [30]. Cela dépend de la stratégie et des écueils qu'on la souhaite la voir éviter.

Il est également intéressant de se pencher sur les plus mauvais jours comme les meilleurs jours de la stratégie obtenus sur le backtest. Leur étude fournit en général des informations importantes sur son comportement en temps de crise. Le tableau 23.5 présente les résultats pour la stratégie décrite au paragraphe 23.6.3.

#### 23.6.3.4 Variation de paramètres

Une stratégie dépend de paramètres. Un trend following simple dépend de la longueur de la moyenne mobile (20, 200 jours) et de l'écart entre les bandes de Bollinger. On peut se poser la question de savoir si la stratégie est sensible ou non à la modification d'un paramètre. Etant donné que les stratégies sont testées sur un backtest, les paramètres sont ajustés en fonction de ce backtest, une stratégie sensible à un paramètre signifie qu'un backtest un peu différent aura vraisemblement des répercussions importantes sur la performance de la stratégie.

30. Pour calculer la VAR, on considère les rendements de l'année écoulée qu'on trie par ordre croissant. On considère que les plus grandes pertes représentent ce que la stratégie peut perdre au pire le lendemain. Ce calcul est appelé VAR historique.

| | |
|---|---|
| rendement annualisé | 7,7 % |
| volatilité annualisée | 8,5 % |
| Information Ratio | 0.88 |
| ratio de Sharpe | 0.42 |
| corrélation | 82.6 % |
| drawdown | 24.4 % |

**Table 23.5 :** *Statistiques classiques permettant de mesurer la performance de la stratégie décrite au paragraphe 23.6.3. La stratégie est ici peu efficace. La volatilité est élevée, le drawdown maximal est très élevé.*

Dans le cas de notre stratégie, on a fait varier les trois paramètres et mesurer le rendement et la volatilité pour chaque expérience. Ceci aboutit aux graphes de la figure 23.40.

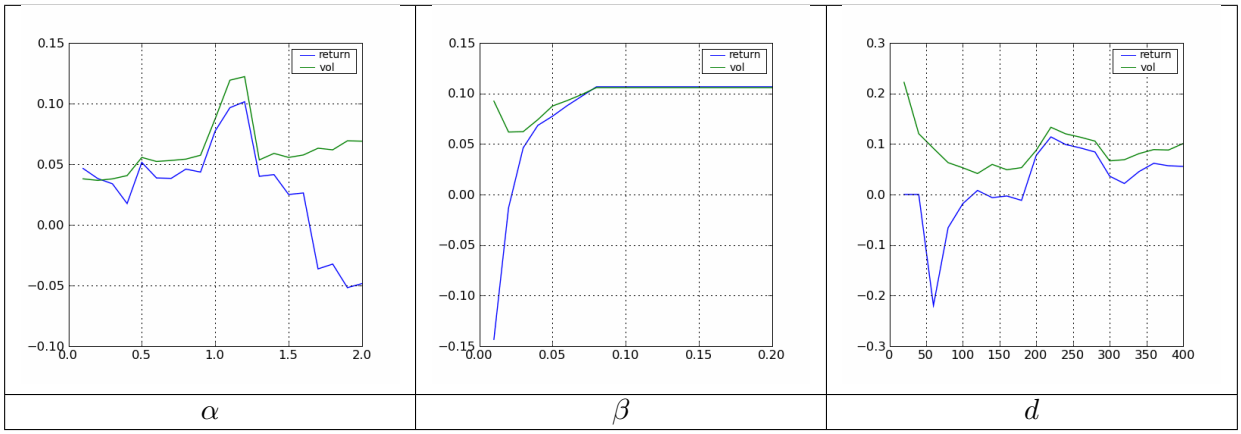



**Figure 23.40 :** *Variations selon les trois paramètres $\alpha$, $\beta$, $d$. Le pic obtenu pour la valeur $\alpha \sim 1.2$ est très localisé autour de cette valeur, il faut s'attendre à ce que la stratégie obtienne des résultats très différents sur des séries similaires. Les rendements passent du simple au double en fonction de la largeur des bandes de Bollinger. Ces graphes ne montrent que l'évolution des rendements et de la volatilité, ils pourraient aussi montrer l'évolution des drawdowns ou du ratio de Sharpe.*

#### 23.6.3.5 Autres séries financières de même nature

Il existe nécessairement d'autres séries financières corrélées avec celles du backtest. Si le backtest est le cours de l'action d'une banque, il faut tester la stratégie sur l'action d'une autre banque pour vérifier qu'elle a le même comportement sur une série fortement corrélée. Ceci permet de confirmer la robustesse de la stratégie.

A l'inverse, il est également souhaitable de tester la stratégie sur des séries financières décorrélées, issues de secteurs économiques différents. Deux conclusions sont possibles, soit la stratégie est encore positive auquel cas elle est très robuste, soit la stratégie présente des résults négatifs qui permettront peut-être d'appréhender les limites de l'algorithme et de pouvoir identifier des scénari dans lesquels la stratégie ne produira pas de bons résultats.

La figure 23.41 montre les cours de trois banques, la BNP, la Société Générale, le Crédit Agricole et une série issue d'un autre secteur Danone. La table 23.6 contient la matrice des autocorrélations. Enfin, la figure 23.42 montre les résultats de la stratégie sur chacune des quatre séries.

Il est très peu probable qu'une stratégie soit efficace sur chaque action mais il est souhaitable qu'elle soit positive sur des séries corrélées et qu'elle limite les pertes sur les autres séries financières.

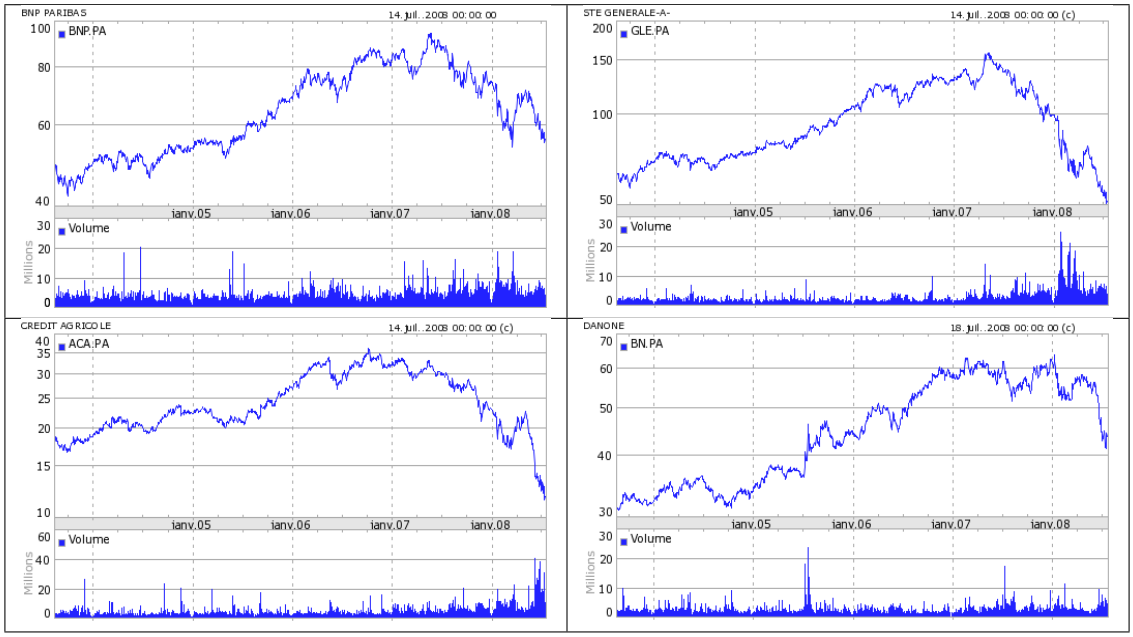



**Figure 23.41 :** *Quatre actions, trois banquaires (BNP, Société Générale, Crédit Agricole) et une société issue de l'alimentaire Danone.*

| | BNP | SG | CA | Danone |
|---|---|---|---|---|
| **BNP** | 1,00 | 0,75 | 0,67 | 0,44 |
| **SG** | 0,75 | 1,00 | 0,63 | 0,42 |
| **CA** | 0,67 | 0,63 | 1,00 | 0,35 |
| **Danone** | 0,44 | 0,42 | 0,35 | 1,00 |

**Table 23.6 :** *Matrice des autocorrélations pour les rendements quotidiens des quatre actions BNP, Société Générale, Crédit Agricole et Danone. Les corrélations sont moins fortes entre Danone et une banque qu'entre banques. Bien que les données quotidiennes soient beaucoup plus propres que des données intraday, il faut quand même vérifier que les séries sont homogènes et contiennent les mêmes dates avant de calculer les corrélations sur les rendements.*

#### 23.6.3.6 Décomposition en deals

Ce procédé permet parfois de découvrir le style d'une stratégie ou de mesurer la pertinence de l'algorithme lorsqu'il coupe sa pose. Un *deal* est le fait d'ouvrir puis de fermer une position comme le décrit la figure 23.43. Un deal est donc défini par :

1. une date d'ouverture de la position $t_1$
2. une date de fermeture de la position $t_2$
3. la quantité $q$ (positive si on a acheté, négative si on a vendu)
4. le prix d'ouverture $p_1$
5. le prix de fermeture $p_2$

Ces informations permettent de calculer le gain associé au deal : $q(p_2 - p_1)$. Si on note un deal comme un 5-uplet $d_i = \left(t_1^i, t_2^i, q^i, p_1^i, p_2^i\right)$, le gain de la stratégie sur l'ensemble du backtest devient : $\sum_i q^i(p_2^i - p_1^i)$. Cette décomposition s'inspire de l'article [Potters2005] qui étudie la répartition des gains d'une stratégie Trend Following, elle présente des caractéristiques qui la différencie d'autres stratégies. Un Trend Following se reconnaît car il aboutit à un grand nombre de petits deals négatifs et quelques gros deals positifs comme le montre la figure 23.44.

On peut s'interroger sur le cas d'une stratégie exclusivement Trend Following dont la distribution des deals sur backtest est différente de ce profil décrit par la figure 23.44. Ses paramètres pourraient avoir été

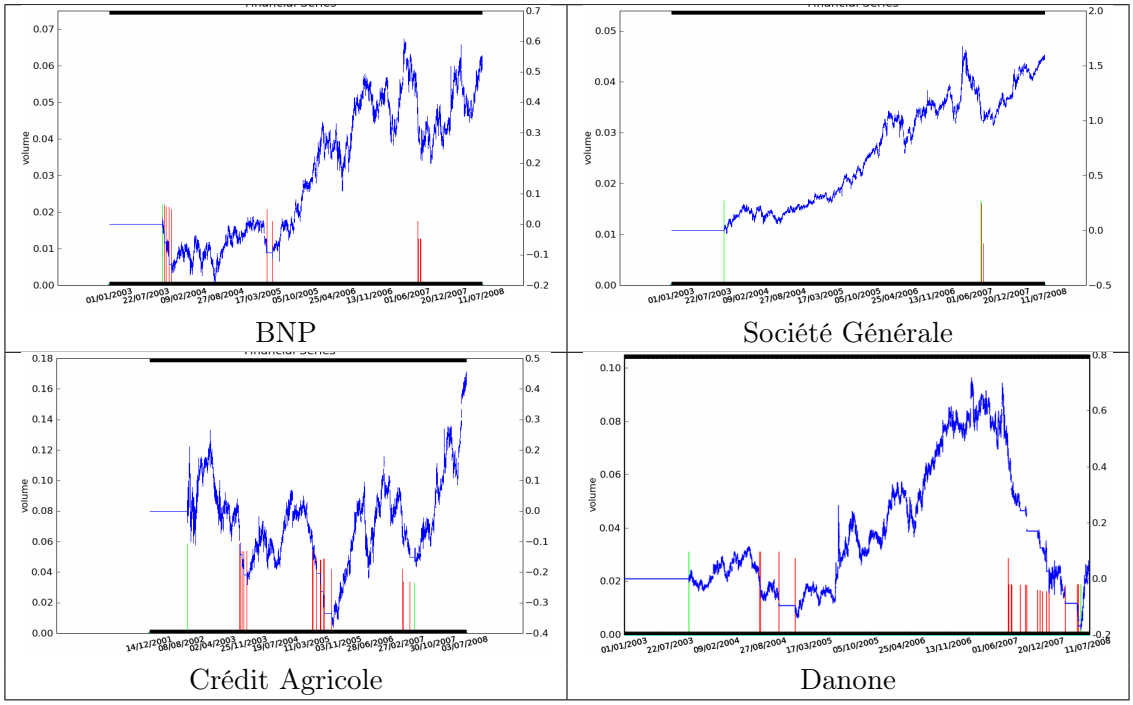



**Figure 23.42 :** *La stratégie Trend Following est appliquée aux quatre séries avec des résultats mitigés. Elle réagit bien sur la BNP et la Société Générale, elle donne de mauvais résultats sur le Crédit Agricole. Sur Danone, la stratégie conserve une position acheteuse puis perd tout ce qu'elle avait gagné par la suite alors que la série n'a plus de trend aussi évident et que sa volatilité est élevée. La série du Crédit Agricole est plus longue de deux ans mais cela n'explique pas ses mauvais résultats, la stratégie prend une série de mauvaises décisions ce qui tend à montrer qu'elle n'est pas suffisamment robuste.*

trop bien calculés pour s'ajuster au backtest, ceci implique que cette stratégie aurait sans doute plus de mal à reproduire des rendements équivalents sur des données futures. On peut envisager cette distribution comme un test statistique.

Le second intérêt de la décomposition en deals est le calcul de la perte et du gain maximale que la stratégie aurait pu observer en coupant plus tôt sa position. On définit $H(t_1, t_2)$ le prix maximal observé dans la période $[t_1, t_2]$ et $L(t_1, t_2)$ le prix minimal observé dans la même période. Pour chaque deal long (position positive), ces deux prix vérifient l'inégalité :

$$q\left(L(t_1, t_2) - p_1\right) \leqslant q(p_2 - p_1) \leqslant q\left(H(t_1, t_2) - p_1\right) \tag{23.78}$$

Le graphe $(q(p_2 - p_1),\ q\left(H(t_1, t_2) - p_1\right))$ permet de représenter l'écart entre le gain et le gain maximal qu'on aurait pu obtenir sur chaque deal long. Un deal short (position négative), on s'intéresse à l'ensemble des points $(q(p_2 - p_1),\ q\left(L(t_1, t_2) - p_1\right))$. Ceci est illustré par le graphe 23.45.

#### 23.6.3.7 Pour aller plus loin

Sur des stratégies Trend Following, le passé d'une action ne suffit pas à tester une stratégie : moins d'une dizaine d'ordres vont être exécutés. La première solution est de tester cette stratégie sur plus de séries similaires. La seconde solution est plus ambitieuse car elle suppose l'altération de la série initiale.

Le premier objectif est de créer une série proche mais suffisamment différente pour tester la robustesse de la stratégie. On peut par exemple construire une seconde série où chaque rendement quotidien sera tiré aléatoirement parmi les cinq derniers rendements quotidiens. Le second objectif est d'ajouter à la série des scénarios de crises. On peut soit s'inspirer des crises déjà présentes ou créer artificiellement des scénarios volontairement exagérés de façon à tester la stratégie dans des cas extrêmes.

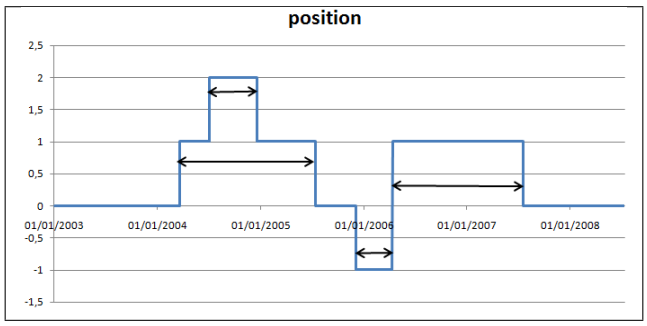



**Figure 23.43 :** *Les deals sont construits à partir de la position de la stratégie. Chaque flèche à double sens désigne un deal Lorsqu'un ordre d'achat est passé alors que la position est déjà positive, on décompose en deal dont les temps de vie se superposent.*

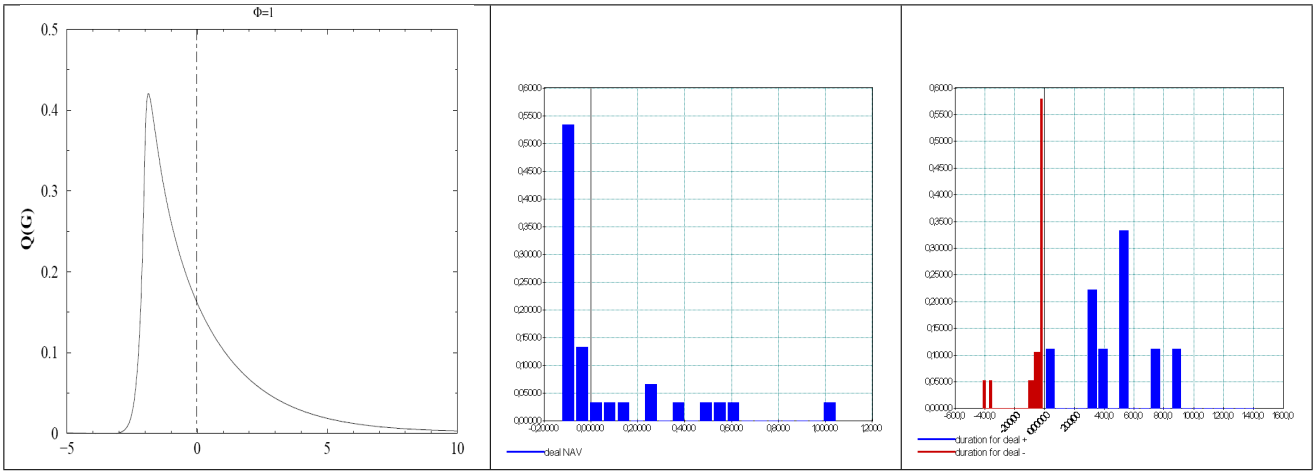



**Figure 23.44 :** *La première image est celle de la distribution théorique des gains calculée par Bouchaud et Potters dans leur article [Potters2005]. La distribution empirique n'est pas toujours facile à construire sur des historiques qui ne sont pas assez longs : dans le cas de notre stratégie Trend Following, il n'existe que cinq deals. Il faut donc assembler les deals de la même stratégie sur plusieurs séries. Le résultat correspond assez bien à la distribution théorique. Le troisième graphe représente la distribution des durées des deals exprimées en jours (axe des abscisses). Les deals négatifs sont nombreux et de courtes durées.*

Bruiter les séries financières est un projet ambitieux en terme de conception et de temps de calcul. Certaines directions de recherches visent à modéliser les acteurs des marchés financiers pour reproduire artificiellement le fonctionnement d'une salle des marchés et ses crises. Cette voie est plus proche de l'intelligence artificielle, des sytèmes multi-agents ou de la microéconomie que de la finance elle-même.

### 23.6.4 Diversification

Gagner de l'argent à partir d'une seule stratégie et d'une seule action est beaucoup trop risqué. Il faut constuire un portefeuille afin de réduire les risques. Ce portefeuille n'est plus simplement composé d'actions mais de couples action - stratégie qui sont décrits par un rendement et une volatilité.

L'objectif reste le même que celui présenté au paragraphe 23.6.2.5 et sa résolution est identique également. L'intérêt de combiner des actions de secteurs économiques différents est toujours de composer un portefeuille d'actions qui ne soient pas trop corrélées afin d'éviter que toutes les actions montent ou baissent en même temps.

Une stratégie n'est jamais performante tout le temps, l'intérêt de combiner des stratégies différentes sur la même action est de pouvoir compenser la perte d'un algorithme par le gain d'un autre. Autrement dit, la combinaison de plusieurs stratégies ne sera pas forcément plus rentable mais aura tendance à faire

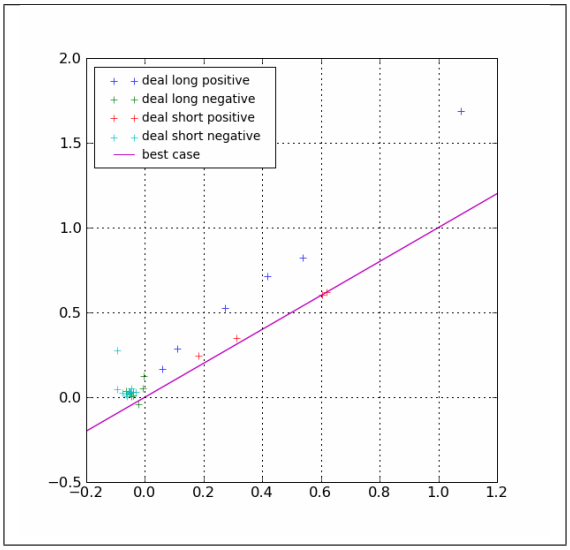



**Figure 23.45 :** *Dans ce graphe, pour un deal positif, plus il est proche de la diagonale, plus la décision de coupure de la position a été proche du maximum envisageable. Dans ce graphe, on voit que la stratégie est meilleure lorsqu'elle coupe une position vendeuse plutôt qu'acheteuse. Il faudrait sans doute pour l'améliorer tenir compte du signe de la position avant de couper même si cette conclusion est osée étant donné le peu de deals short positifs.*

diminuer la volatilité.

### 23.6.5 Conclusion

Ce document n'avait pas pour objectif de présenter une stratégie automatique performante, le Trend Following utilisé comme exemple n'est ni suffisamment robuste ni suffisamment testé. En revanche, cet exposé introduit différents styles de trading applicables à des algorithmes automatiques et aussi évoque différents moyens de s'assurer de la performance de tels algorithmes. Au delà du traditionnel backtest, il existe des indicateurs, des représentations graphiques qui apportent un autre point de vue que la seule donnée du rendement et de la volatilité annualisés.

### 23.6.6 Morceaux informatiques

#### 23.6.6.1 Dessin d'une série financière

La plupart des résultats issus d'une stratégie automatique de trading sont des graphes. Le premier d'entre eux est d'une série financière pour laquelle on connaît les cours *Open High Low Close* de chaque jour. Ces graphes sont spécifiques au domaine de la finance et ils ne sont pas toujours faciles à créer avec des modules comme `MatPlotLib`. Les données sont souvent conservées dans des fichiers texte dont le format ressemble à celui qui suit.

```
Date      Open   High   Low    Close  Volume
20/03/08  56.01  56.75  55.26  55.84  2434000
19/03/08  57.53  57.95  56.40  56.77  2714400
18/03/08  58.85  58.99  57.13  57.61  3265900
14/03/08  60.27  60.78  59.00  59.53  2105500
13/03/08  59.01  60.47  59.00  60.15  2702700
12/03/08  60.84  60.85  59.35  60.02  2982400
11/03/08  61.12  61.56  60.00  60.47  2480900
```

L'objectif est d'obtenir le graphe de la figure 23.46 dans lequel figurent les prix OHLC (*Open High Low Close*), les volumes et aussi les dates qui permettent de relier cette série à des événements économiques

comme la crise des subprimes qui a commencé en été 2007.

```
def GraphOHLC (title, dates, ohlc, file, size = (800,400), t1 = 0, t2 = -1) :
    """dessine une série financière avec matplotlib
    - title est une titre
    - dates est une liste de date
    - ohlc est une liste de dictionnaire avec les clés
                      ("Open", "High", "Low", "Close", "Volume")
    - file est un nom de fichier, le graphe y sera sauvegardé
    - size est la taille désirée pour le graphique
    - t1 est l'indice de la première date du graphique
    - t2 est l'indice de la dernière date du graphique
    """

    # on coupe la série si besoin
    if t2 == -1 : t2 = len (dates)
    dates = dates [t1:t2]
    ohlc = ohlc [t1:t2]

    import pylab

    # on crée les labels pour les abscisses
    # sans afficher toutes les dates car elles deviennent illisibles sinon
    # on en affiche que 10
    ind  = pylab.arange (len (dates))
    lab  = ["" for i in ind]
    for i in range (0,len (dates),len(dates)/10) : lab [i] = dates [i]

    # on crée la figure
    fig  = pylab.figure (figsize=(size [0]/100, size [1]/100))
    ax   = fig.add_subplot(111)

    # la première série à tracer est celle des volumes sous forme d'histogramme
    for i in range (0, len (ohlc)) :
        oh = ohlc [i]

        # l'histogramme est vert si c'est un jour de hausse
        # rouge si c'est un jour de baisse
        if i == 0 : co = (0.5,0.5,0.5)
        elif ohlc [i]["Close"] > ohlc [i-1]["Close"] : co = (0.0,1.0,0.0)
        else : co = (1.0,0.0,0.0)
        if len (dates) > 300 :
            pylab.plot ( [i, i], [0, oh ["Volume"]], "k", color = co)
        elif len (dates) > 100 :
            pylab.plot ( [i, i], [0, oh ["Volume"]], "k", \
                                        linewidth = 2.0, color = co)
        else :
            pylab.plot ( [i, i], [0, oh ["Volume"]], "k", \
                                         linewidth = 4.0, color = co)

    # on écrit la légende du volume
    v = [0.0 for i in ind]
    pylab.plot (ind, v, "c.")
    pylab.ylabel ("volume")

    # on construit un second axe au graphique pour la série des prix
    ymin, ymax = pylab.ylim()
    pylab.ylim (ymin, ymax*3)
    pylab.xticks (ind, lab, fontsize = "small", rotation = 13  )
    ax2 = pylab.twinx()

    # puis un dessine les prix sur le graphique selon le même schéma,
    # la même série de points
    # (i-0.5, Open) (i,Open) (i,Low) (i,High) (i,Close) (i+0.5,Close)
    for i in range (0, len (dates)) :
```

```
        oh = ohlc [i]
        co = (0.0,0.0,1.0)
        pylab.plot ([i-0.5, i], [oh["Open"],  oh["Open"]],  "k", \
                                    linewidth = 1.0, color = co)
        pylab.plot ([i, i],     [oh["Low"],   oh["High"]],  "k", \
                                    linewidth = 1.0, color = co)
        pylab.plot ([i, i+0.5], [oh["Close"], oh["Close"]], "k", \
                                    linewidth = 1.0, color = co)

    # on termine par le titres, la légende du second axe et celles des abscisses
    pylab.title   (title)
    pylab.ylabel  ("euros")
    pylab.xticks  (ind, lab, fontsize = "small", rotation = 13)
    pylab.xlabel  ("dates")
    # il ne reste plus qu'à sauver la figure
    pylab.savefig (file, orientation='paysage')
```

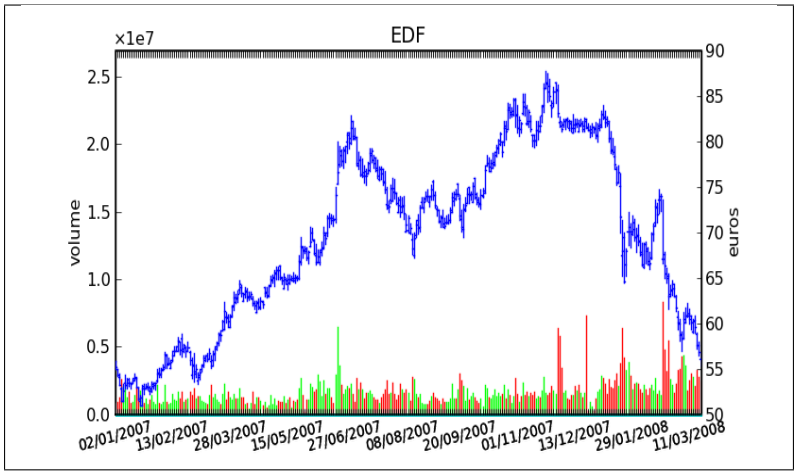



**Figure 23.46 :** *Graphe Open-High-Low-Close-Volume d'une série financière.*

Sixième partie

# EXERCICES VARIÉS

# Chapitre 24

# Sujets informatiques

---

## 24.1 Dictionnaires (1)

### 24.1.1 Enoncé

L'objectif est de compter le nombre de mots commencer par une même lettre.

**1)** Télécharger le fichier *td_ note_ texte.txt*[1].

**2)** Ecrire une fonction qui lit le fichier précédemment téléchargé et qui place le résultat dans une liste

**3)** Ecrire une fonction qui compte le nombre de mots commençant par la lettre "A" (définir l'entrée et la sortie)

**4)** Ecrire une fonction qui compte le nombre de mots commençant par une lettre donnée (définir les deux entrées et la sortie)

**5)** Ecrire une fonction qui compte le nombre de mots commençant par la lettre "A", puis la lettre "B", ... jusqu'à la lettre "Z" (définir l'entrée et la sortie)

**6)** La fonction précédente parcourt 26 fois la liste de mots. On cherche à ne le faire qu'une fois à l'aide d'un dictionnaire.

**7)** Comparer le coût des deux algorithmes (sans ou avec dictionnaire).

**8)** Peut-on faire encore plus rapidement si on tient compte du fait que la liste de mots est triée ? Quel est le coût de cet algorithme ?

### 24.1.2 Correction

```
# question 2
def lit_fichier (file) :
    f = open (file, "r")  # ouverture du fichier en mode lecture "r"
    li = f.readlines ()     # lecture des lignes du fichier texte
    f.close ()               # fermeture du fichier, l'acces est libre pour d'autres applications

    # on enleve les separateurs de fin de lignes,
    # caracteres invisibles qui permettent aux editeurs de texte (Scite, Notepad)
    # de se retrouver dans les lignes
    res = []
    for l in li :
        s = l.replace ("\n", "")
        res.append (s)
    return res

# question 3
def compte_A (li) :
    nb = 0
    for l in li :
        if l [0] == "A" : nb += 1
```

1. *http ://www.xavierdupre.fr/html/programmation/langage_ py_ thon/python_ td_ simple/td_ note_ texte.txt*

```
    return nb

# question 4
def compte (li, lettre) :
    nb = 0
    for l in li :
        if l [0] == lettre : nb += 1
    return nb

# question 5
def compte_toute (li) :
    alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    res = []
    for a in alpha :
        n = compte (li, a)
        res.append (n)
    return res

# question 6
def compte_dico (li) :
    res = { }
    for l in li :
        a = l [0]
        if a in res : res [a] += 1
        else : res [a] = 1
    return res


lis = lit_fichier ("td_note_texte.txt")

a = compte_A (lis)
print a  # donne 65

a = compte (lis, "A")
print a  # donne 65

all = compte_toute (lis)
print len (all), all # affiche 26 [65, 16, 69, 46, 53, 39, ...

res = compte_dico (lis)
print res  # {'A': 65, 'C': 69, 'B': 16, 'E': 53,  ...

# question 7
"""
cout fonction compte_toute : 26 * cout fonction compte = 26 * longueur de la liste
cout fonction compte_dico :    longueur de la liste * cout recherche dichotomique
                                    = longueur de la liste * ln(26)/len(2)
"""

# question 8
def compte_8 (li) :
    res = [ ]
    nb = 1
    le = li [0][0]
    for i in range (1, len (li)) :
        if li [i][0] != li [i-1][0] :
            res.append ( (le, nb) )
            nb = 1
            le = li [i][0]
        else :
            nb += 1
    res.append ( (le, nb) )
    return res

res = compte_8 (lis)
```

```
print res  # [('A', 65), ('B', 16), ('C', 69), ('D', 46), ('...
```

**fin correction TD 24.1.1** □

## 24.2 Dictionnaires (2)

### 24.2.1 Enoncé

Le fichier *star.txt*[2] contient des prénoms que portent des enfants de stars. Dans la première colonne, les prénoms classés par ordre alphabétique, dans la seconde, le ou les parents.

**1)** Télécharger le fichier *star.txt*.

**2)** Ecrire une fonction qui lit le fichier précédemment téléchargé et qui place le résultat dans une liste.

**3)** Quel est le prénom le plus fréquent dans cette liste (on ne tient pas compte des prénoms des parents).

**4)** Ecrire une fonction qui rassemble les prénoms par famille.

**5)** Calculer le nombre moyen et le nombre médian d'enfants par famille.

### 24.2.2 Correction

```
# question 1,2
#
# extrait du fichier une matrice de n lignes et deux colonnes
# separateur de ligne \n
# separateur de colonne \t
#
def lit_fichier (file) :
    f = open (file, "r")
    li = f.readlines ()   # separation en lignes
    f.close ()

    mat = []
    for l in li :
        s = l.replace ("\n", "")
        s = s.split ("\t")  # separation en colonne
        if len (s) == 2 :
            # ce test est ajoute pour verifier qu'on ajoute pas
            # des lignes vides dans la matrice
            # les lignes vides ne contiennent pas de separateur \t
            # et ne contiennent qu'une colonne
            mat.append (s)
    return mat

# question 3
def prenom_frequent (mat) :
    # les prenoms sont en colonne 0
    res = { }
    for i in range (0, len (mat)) :
        p = mat [i][0]
        if p in res : res [p] += 1
        else : res [p] = 1
```

2. http ://www.xavierdupre.fr/html/programmation/langage_py_thon/python_td_simple/star.txt

```
    # on recherche le maximum : maximum sur toutes les valeurs
    mx = max ( res.values () )

    # on recherche tous les prenoms qui sont aussi frequent que mx
    freq = []
    for p in res :
        if res [p] == mx : freq.append (p)

    return freq,mx

# question 4
def family_group (mat) :
    # on choisit de representer les familles sont la forme d'un dictionnaire de listes
    # cle : parents (colonne 1 de mat)
    # valeur : liste de prenoms (colonne 0 de mat)
    res = { }
    for i in range (0, len (mat)) :
        pr = mat [i][0]
        pa = mat [i][1]
        if pa in res : res [pa].append (pr) # on ajoute le prenom a la liste
        else : res [pa] = [ pr ]   # liste d'un seul prenom
    return res

# question 5
def moyenne_mediane (dico) :
    # on repart du resultat de la fonction family_group
    # et on construit la liste des tailles de familles
    taille = []
    for pa in dico :
        taille.append ( len (dico [pa] ) )

    moyenne = float (sum (taille)) / len (taille)
    taille.sort ()
    mediane = taille [ len (taille) / 2 ]
    return moyenne, mediane

# test
mat = lit_fichier ("star.txt")
freq,mx = prenom_frequent (mat)
print "prenom frequents ", freq
print "present ", mx, " fois"
fam = family_group (mat)
moyenne, mediane = moyenne_mediane (fam)
print "moyenne ", moyenne, " mediane ", mediane

for pa in fam :
    print pa
    for pr in fam [pa] :
        print "    ", pr
```

**fin correction TD 24.2.1** □

## 24.3 Popularité selon différents moteurs de recherce

### 24.3.1 Enoncé

Internet peut se révéler un moyen original de mesurer la popularité d'une personne ou d'un personnage de roman. La recherche d'un acteur avec un moteur de recherche quelconque aboutit à une multitude de

pages qu'il est souvent impossible de parcourir jusqu'à la dernière.

Le programme présent à l'adresse *http ://perso.wanadoo.fr/x.dupre/programmation/ langage_python/ python_td_simple_correction/recherche_voila_popularite.html* permet de compter le nombre de pages dénichées par les moteurs de recherche Voila, Lycos, MSN, eBay.

L'objectif du TD est, en réutilisant certaines parties de ce programme, de comparer la popularité de certaines populations d'acteurs.

**1)** La première étape consiste à créer une classe regroupant toutes les informations concernant une personne ou un personnage : nom, prénom, âge, langue parlée, sexe, réalité... ainsi que le nombre de pages associé selon les différentes moteurs de recherche.

**2)** Une fois cette classe définie, il ne reste plus qu'à créer autant d'instances que nécessaires et à les renseigner.

**3)** On cherche dans un premier temps à compter le nombre de pages moyen pour une population donnée (vérifiant certains critères comme actrices francophones par exemple) et un moteur de recherche donné.

**4)** Est-ce que tous les moteurs de recherche donnent des résultats similaires lorsqu'on cherche à comparer la popularité des actrices anglophones et francophones ?

### 24.3.2 Correction

```
#!/usr/bin/python
# -*- coding: iso-8859-15 -*-
import urllib


def combien_voila (requete) :
    requete = requete.replace (" ", "%20")
    requete = requete.replace ("\"", "%22")
    #url = "http://search.ke.voila.fr/S/voila?dt=*&image.x=0&bhv=web_mondial&image.y=0&kw=" + requete
    url = "http://search.ke.voila.fr/S/voila?rtype=kw&rdata=" + requete
    url += "&profil=voila&bhv=web_fr"
    whole = ""
    f       = urllib.urlopen (url)
    try :
        d       = f.read ()
        whole  += d
    except Exception, exc:
        print "erreur"
    f.close ()
    str = "r&eacute;ponses"
    pos = whole.find (str)
    if pos == -1 : return 0

    tex = whole [pos-30: pos + 5]

    pos = tex.find ("sur")
    if pos >= 0 : tex = tex [pos+3:]

    res = ""
    for t in tex :
        if "0" <= t <= "9" : res += t
    tex = res

    try :
        nb = int (tex)
        return nb
    except :
        return 0
```

```
def combien_lycos (requete) :
    requete = requete.replace (" ", "%20")
    requete = requete.replace ("\"", "%22")
    #url = "http://vachercher.lycos.fr/cgi-bin/pursuit?query=" + requete
    #url += "&x=0&y=0&cat=lycos&tld=com&family=off"
    url = "http://vachercher.lycos.fr/cgi-bin/pursuit?query=" + requete
    url += "&x=0&y=0&cat=web&enc=utf-8&="
    whole = ""
    f       = urllib.urlopen (url)
    try :
        d       = f.read ()
        whole  += d
    except Exception, exc:
        print "erreur"
    f.close ()
    str = "Monde  sur"
    pos = whole.find (str)

    if pos == -1 :
        return 0

    tex = whole [pos-5: pos + 30]

    pos = tex.find ("sur")
    if pos >= 0 : tex = tex [pos+3:]

    res = ""
    for t in tex :
        if "0" <= t <= "9" : res += t
    tex = res

    try :
        nb = int (tex)
        return nb
    except :
        return 0

def combien_msn (requete) :
    requete = requete.replace (" ", "%20")
    requete = requete.replace ("\"", "%22")
    #url = "http://search.msn.fr/results.aspx?FORM=MSNH&CP=1252&q=" + requete
    url = "http://search.live.com/results.aspx?q=" + requete + "&go=Rechercher&mkt=fr-fr&scope=&FORM=LIVSOP"

    whole = ""
    f       = urllib.urlopen (url)
    try :
        d       = f.read ()
        whole  += d
    except Exception, exc:
        print "erreur"
    f.close ()
    str = "sur "
    pos = whole.find (str)

    if pos == -1 :
        return 0

    tex = whole [pos-5: pos + 30]

    pos = tex.find ("sur")
    if pos >= 0 : tex = tex [pos+3:]

    res = ""
    for t in tex :
        if "0" <= t <= "9" : res += t
```

```
    tex = res


    try :
        nb = int (tex)
        return nb
    except :
        return 0

def combien_yahoo (requete) :
    requete = requete.replace (" ", "+")
    requete = requete.replace ("\"", "%22")
    #url = "http://search.msn.fr/results.aspx?FORM=MSNH&CP=1252&q=" + requete
    url = "http://fr.search.yahoo.com/search?p=" + requete + "&fr=yfp-t-501&ei=UTF-8&rd=r1"

    whole = ""
    f       = urllib.urlopen (url)
    try :
        d       = f.read ()
        whole  += d
    except Exception, exc:
        print "erreur"
    f.close ()
    str = "sur environ"
    pos = whole.find (str)

    if pos == -1 :
        return 0

    tex = whole [pos-5: pos + 30]

    pos = tex.find ("environ")
    if pos >= 0 : tex = tex [pos+3:]

    res = ""
    for t in tex :
        if "0" <= t <= "9" : res += t
    tex = res

    try :
        nb = int (tex)
        return nb
    except :
        return 0

def combien_ebay (requete) :
    requete = requete.replace (" ", "%20")
    requete = requete.replace ("\"", "%22")
    url = "http://shop.ebay.fr/items/?_nkw=" + requete.replace (" ", "+") + "&_fromfsb=0&_trksid=m270.l1313"
    whole = ""
    f       = urllib.urlopen (url)
    try :
        d       = f.read ()
        whole  += d
    except Exception, exc:
        print "erreur"
    f.close ()
    str = "sultats trouv"
    pos = whole.find (str)

    if pos == -1 : return 0
    tex = whole [pos - 50 : pos-10]
    pos = tex.find ("\">")
    if pos >= 0 : tex = tex [pos+1:]
    tex = tex.lower ().strip ("abcdefghijklmnopqrstuvwxyz<>/' =")
```

```
    try :
        nb = int (tex)
        return nb
    except :
        return 0

def combien_acteur (acteur) :
    r1 = combien_voila ("\"" + acteur + "\"")
    r2 = combien_msn ("\"" + acteur + "\"")
    r3 = combien_lycos ("\"" + acteur + "\"")
    r4 = combien_ebay ("\"" + acteur + "\"")
    r5 = combien_yahoo ("\"" + acteur + "\"")
    return r1,r2,r3,r4,r5


#l = ["harry potter"]
#l += ["orlando bloom", "bruce willis", "harrison ford", "johnny depp", "sean connery", "daniel auteuil", "gerard depa
#l += ["jodie foster", "michel blanc", "nathalie portman", "ewan mcgregor", "scarlett johansson", "mick jagger", "tim
#l += ["susan sarandon", "zooey deschanel", "evangeline lilly", "robert downey jr", "katie holmes", "tom cruise", "nic
#l += ["anne heche", "pierce brosnan", "calista flockhart", "portia de rossi", "lucy liu", "matt damon", "ben affleck"
#l += ["geoffrey rush", "hugh jackman", "robert redford", "clint eastwood", "woody allen", "kenneth branagh", "emma th
#l += ["maggie smith", "kristen dunst", "chloe sevigny", "jennifer beals", "mia kirschner", "karina lombard", "pam gri
#l += ["katherine moenning"]
#l += ["patrick timsit", "patrick bruel", "richard berry", "thierry lhermite", "sandrine kiberlain", "pascal gregory"]
#l += ["cecile de france", "julia roberts", "marion cotillard", "audrey tautou", "romain duris"]

#for i in l :
#    print i, "\t", combien_acteur (i)

class Acteur :
    def __init__ (self, nom, prenom, age, nationalite, sexe) :
        self.nom            = nom
        self.prenom         = prenom
        self.age            = age
        self.nationalite    = nationalite
        self.sexe           = sexe

        s = "\"" + prenom + " " + nom + "\""
        self.voila          = combien_voila (s)
        self.msn            = combien_msn (s)
        self.lycos          = combien_lycos (s)
        self.ebay           = combien_ebay (s)
        self.yahoo          = combien_yahoo (s)

    def liste_compteur (self) :
        return [ self.voila, self.msn, self.lycos, self.ebay, self.yahoo ]

print "commencement"
act = []
act.append ( Acteur ("foster", "jodie", 40, "us", "f") )
act.append ( Acteur ("depardieu", "gerard", 55, "fr", "h") )
act.append ( Acteur ("deneuve", "catherine", 60, "fr", "f") )
act.append ( Acteur ("harrison", "ford", 60, "us", "h") )
if False :  # False pour corriger le programme plus rapidement, True sinon
    act.append ( Acteur ("redford", "robert", 60, "us", "h") )
    act.append ( Acteur ("depp", "johnny", 40, "us", "h") )
    act.append ( Acteur ("eastwoord", "clint", 75, "us", "h") )
    act.append ( Acteur ("sarandon", "susan", 60, "us", "f") )
    act.append ( Acteur ("dunst", "kirsten", 20, "us", "f") )
    act.append ( Acteur ("portman", "nathalie", 25, "us", "h") )
    act.append ( Acteur ("roberts", "julia", 40, "us", "f") )
    act.append ( Acteur ("tautou", "audrey", 30, "fr", "f") )
    act.append ( Acteur ("cotillard", "marion", 30, "fr", "f") )
    act.append ( Acteur ("binoche", "juliette", 40, "fr", "f") )
    act.append ( Acteur ("berry", "richard", 50, "fr", "h") )
```

```
    act.append ( Acteur ("bruel", "patrick", 45, "fr", "h") )

def ajoute_liste (l1, l2) :
    for i in range (0, len (l1)) :
        l1 [i] += l2 [i]

def divise_liste (l1, s) :
    for i in range (0, len (l1)) :
        l1 [i] /= s


# france contre etats - unis
fr = [ 0,0,0,0,0 ]
us = [ 0,0,0,0,0 ]
nb_fr = 0
nb_us = 0
for a in act :
    print "cherche ", a.nom,
    l = a.liste_compteur ()
    print l
    if a.nationalite == "fr" :
        nb_fr += 1
        ajoute_liste (fr, l)
    if a.nationalite == "us" :
        nb_us += 1
        ajoute_liste (us, l)

divise_liste (us, nb_us)
divise_liste (fr, nb_fr)

print "------------------------------"
print "compteur nb_us = ", nb_us, " nb_fr : ", nb_fr
for i in range (0, len (us)) :
    print "us : ", us [i], " \t fr : ", fr [i]

# homme contre femme
f = [ 0,0,0,0,0 ]
h = [ 0,0,0,0,0 ]
nb_f = 0
nb_h = 0
for a in act :
    print "cherche ", a.nom,
    l = a.liste_compteur ()
    print l
    if a.sexe == "f" :
        nb_f += 1
        ajoute_liste (f, l)
    if a.sexe == "h" :
        nb_h += 1
        ajoute_liste (h, l)

divise_liste (h, nb_h)
divise_liste (f, nb_f)

print "------------------------------"
print "compteur nb_h = ", nb_h, " nb_f : ", nb_f
for i in range (0, len (us)) :
    print "h : ", h [i], " \t f : ", f [i]
```

**fin correction TD 24.3.1** □

## 24.4 Tri simple et tri fusion

### 24.4.1 Enoncé

On cherche dans cette séance à programmer deux méthodes différentes de tri.

**1)** La première s'inspire de celle vue lors de la première séance de cours. Celle-ci déterminait d'abord le maximum d'une liste puis le plaçait à la fin d'une seconde liste. Ces deux étapes répétées autant de fois que d'éléments dans la liste permettait de construire une seconde liste triée. Il s'agit ici d'adapter cette méthode pour ne plus créer une seconde liste triée mais de modifier la première de telle sorte que celle-ci soit triée.

**2)** On s'intéresse ici à la méthode de tri par fusion. Le tri par fusion consiste à morceler la liste initiale en petites sous-listes triées puis à les regrouper de sortes qu'elles restent triées. La première étape consiste à écrire une fonction permettant de fusionner deux listes *triées* de sorte que la liste fusionnée reste *triée*.

**3)** Le tri par fusion peut être conçu comme une méthode récursive : si une liste n'est pas triée alors on la coupe en deux, on tri chacune des deux sous-listes puis on fusionne chacune des deux listes triées en une seule triée elle aussi.

**4)** Quel est le coût du tri par fusion ?

### 24.4.2 Correction

**1)**

```
def tri(l):
    """effectue le tri classique :
    on recherche le minimum, on le place en premiere
    position et on recommence avec les elements restant"""
    for k in range (0,len (l)-1) :
        # recherche du minimum
        p = k
        for n in range (k+1, len (l)) :
            if l [n] < l [p] :
                p = n
        # echange
        ech     = l [k]
        l [k]     = l [p]
        l [p]     = ech

k=[2,8,9,5,15,4,56,78,85,15,45,3]
print "non triee ", k
tri(k)
print "triee ",k
```

**2) 3)**

```
def fusion(a,b):
    """fonction fusionnant les deux listes a,b supposees triees,
    retourne une liste triee.

    a chaque passage dans la boucle, on ajoute un element
    provenant de l'une des deux listes, on conserve pour ces
    deux listes l'indice de l'element suivant a inserer
    dans la liste resultat : k pour la liste a, n pour la liste b
```

```
    a chaque passage, on choisit le plus petit,
       si c'est celui de la liste a, on incremente k,
       si c'est celui de la liste b, on incremente n

       il faut faire attention lorsqu'on a ajouter les elements d'une liste,
       parce que l'indice k ou n ne designe aucun element, de plus,
       cela signifie qu'on doit ajouter les derniers elements
       de l'autre liste.
    """
    c=[]
    l=len(a)+len(b)
    k=0
    n=0
    while len(c)<l:
        if k==len(a):     # liste a terminee
            c.extend (b[n:])
        elif n==len(b):   # liste b terminee
            c.extend(a[k:])
        elif a[k]<b[n]:   # element de la liste plus petit
            c.append(a[k])
            k=k+1
        else :            # element de la liste b plus petit
            c.append(b[n])
            n=n+1
    return c

def tri(l):
    """effectue le tri par fusion, coupe la liste en deux,
    appelle recursivement le tri par fusion sur chacun des
    deux morceaux puis assemble les deux listes triees

    le cout de cette fonction depend de n :
        - fusionner de liste a un cout en O(n)
        - on appelle la fonction fusion sur :
                - le tableau initial : n elements
                - sur deux moities : 2 * n/2
                - sur quarts quarts : 4 * n/4
                - ...
                - sur n/2 sous-liste de 2 elements : n/2 * 2
    le cout du tri par fusion est n * autant de fois
    qu'on peut divisier n par 2 : O(n log (n))"""
    if len(l)>1:
        n=int(len(l)/2)
        a=l[n:len(l)]  # premiere moitie
        b=l[0:n]       # seconde moitie
        tri(a)         # tri de la premier moitie
        tri(b)         # tri de la seconde moitie
        t=fusion(a,b)  # fusion des deux listes triees
        for i in range(0,len(l)):  # on recopie la liste triee dans la
            l[i]=t[i]              # liste initiale
        # on ne peut pas ecrire que :
        #   l = t
        # car dans ce cas, on sert du nom l pour stocker autre chose
        # que la liste l, cette ligne attribuerait le nom l a une autre liste
        # qui serait perdue a la fin de la fonction

k=[2,8,9,5,15,4,56,78,85,15,45,3]
print "non triee ", k
tri(k)
print "triee ",k
```

**fin correction TD 24.4.1** □

## 24.5 Equation différentielle

### 24.5.1 Enoncé

L'étude du mouvement du pendule suspendu aboutit, après une simplification, á une équation différentielle linéaire. On cherche ici á déterminer á partir de quelle condition initiale la solution approchée s'écarte de la véritable solution.

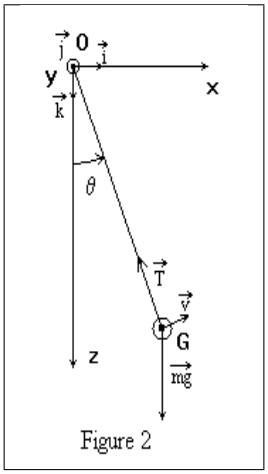



**Figure 24.1 :** *Force agissant au niveau d'un pendule suspendu, le pendule est suspendu á une distance R constante du point O.*

Pour obtenir l'équation différentielle qui régit ce système dynamique, on utilise le fait que la somme des forces est égale á la masse mulitpliée par l'accélération. Le pendule est matérialisé par le point $M$ de coordonnées $(x = R\sin\theta(t),\, y = R\cos\theta(t))$. Son accélération est $\overrightarrow{a} = (x''(t),\, y''(t))$. Ceci donne :

$$m\overrightarrow{a} = \overrightarrow{T} + m\overrightarrow{g} \tag{24.1}$$

On projette sur les deux axes du repère carthésien :

$$\left\{ \begin{array}{lcl} mx'' & = & -T\sin\theta \\ my'' & = & -T\cos\theta + mg \end{array} \right. \tag{24.2}$$

Lorsque les oscillations sont de faible ampleur, on suppose que $y'' \sim 0$ et $R\sin\theta \sim R\theta \sim x$. Le système précédent aboutit á l'équation différentielle linéaire suivante :

$$(24.2) \Longrightarrow mR\theta'' + mg\theta = 0 \Longrightarrow \theta'' + \frac{g}{R}\theta = 0 \tag{24.3}$$

La solution de cette żquation diffżrentielle est :

$$(24.3) \Longrightarrow \theta = A\cos\left(\sqrt{\frac{g}{R}}\,t + \phi\right) \tag{24.4}$$

$A$ et $\phi$ sont des constantes dépendant des conditions initiales. En revanche, lorsque les oscillations sont plus grandes, ces approximations ne sont plus valables, il faut calculer les dérivées secondes $x''$ et $y''$ en fonction de $\theta$ pour obtenir :

$$\left\{\begin{array}{rcl} mx'' &=& mR\left(\theta''\cos\theta - (\theta')^2\sin\theta\right) = -T\sin\theta \\ my'' &=& -mR\left(\theta''\sin\theta + (\theta')^2\cos\theta\right) = -T\cos\theta + mg \end{array}\right. \tag{24.5}$$

On utilise la première équation pour obtenir la valeur de $T$ et la remplacer dans la seconde équation et simplifier par $m$ :

$$\theta''\sin\theta + (\theta')^2\cos\theta = -\left(\theta''\cos\theta - (\theta')^2\sin\theta\right)\frac{\cos\theta}{\sin\theta} - \frac{g}{R} \tag{24.6}$$

On obtient finalement :

$$\begin{aligned} \theta''\left[\sin\theta + \frac{\cos^2\theta}{\sin\theta}\right] + \frac{g}{R} &= 0 \\ \Longrightarrow \theta''\left[\cos^2\theta + \sin^2\theta\right] + \frac{g}{R}\sin\theta &= 0 \\ \Longrightarrow \theta'' + \frac{g}{R}\sin\theta &= 0 \end{aligned} \tag{24.7}$$

Lorsque $\theta$ est petit, $\sin\theta \sim \theta$, on retrouve le résultat (24.3).

**1)** En s'inspirant du programme développé lors de la seconde séance de cours, on désire tracer la solution de l'équation différentielle (24.7).

**2)** La solution est-elle cyclique ? Qu'elle est sa période ?

**3)** On suppose que le pendule part toujours avec une vitesse nulle mais un angle initial $\theta_0$ différent á chaque expérience, on veut tracer la courbe de la période en fonction de l'angle initial.

**4)** A partir de quel angle initial $\theta_0$, les solutions des équations (24.3) et (24.7) sont-elles sensiblement différentes ?

### 24.5.2 Correction

```
ïż£#!/usr/bin/env pythonw2.3

# lignes utilisees pour matplotlib
import matplotlib
matplotlib.use("WXAgg")
import matplotlib.pylab as pylab
import Numeric as N
import math


################################################################################
# debut du programme
################################################################################

def pendule_lineaire (tt, vt, at, gR, dt) :
        """calcule t (t+1), v(t+1), a(t+1)
        t'' + gR t = 0"""
        at = - gR * tt
        vt += at * dt
        tt += vt * dt
        return tt,vt,at
```

```
def pendule_non_lineaire (tt, vt, at, gR, dt) :
        """calcule t (t+1), v(t+1), a(t+1)
        t'' + gR t = 0"""
        at = - gR * math.sin (tt)
        vt += at * dt
        tt += vt * dt
        return tt,vt,at

def periode (courbe,dt) :
        """estime la pÃľriode d'une courbe dont on sait qu'elle est cyclique"""
        lp = []
        p = 0
        for i in range (1, len (courbe)-1) :
                if courbe [i-1] <= courbe [i] and courbe [i] >= courbe [i+1] :
                        # maximum
                        lp.append (i)
        # moyenne entre les durees entre deux maximas
        s = 0
        for l in range (1,len (lp)) :
                s += lp [l] - lp [l-1]
        return float (s) / (len (lp)-1) * dt


# initialisation : un angle t0 != 0, pas de vitesse, pas d'accÃľlÃľration
t0                        = 1
tt,vt,at         = t0,0,0
stt,svt,sat = t0,0,0
dt                         = 0.01
gR                         = 0.5
t                         = 0

T                         = []
th                         = []
sth                 = []
cth                 = []

# boucle
print "simultation"
for i in range (0,4000) :
        T.append (t)
        th.append (tt)
        sth.append (stt)
        ctt                  = t0 * math.cos (math.sqrt (gR) * t)
        cth.append (ctt)
        t                           = t + dt
        tt,vt,at         = pendule_lineaire (tt, vt, at, gR, dt)
        stt,svt,sat = pendule_non_lineaire (stt, svt, sat, gR, dt)

print "periode theorique : ", math.pi * 2 / math.sqrt (gR)
print "periode courbe theorique ", periode (cth, dt)
print "periode courbe lineaire ", periode (th, dt)
print "periode courbe non lineaire ", periode (sth, dt)

# pour calculer l'angle a partir duquel la solution lineaire
# n'est plus viable, il faut calculer la difference entre les periodes
# et choisir un seuil au dela duquel la difference est trop grande

# affichage des premieres valeurs
print "premieres valeurs "
print "angle initial : ", t0 * 180 / math.pi
for i in range (0,10) :
        print T[i], " : ", th[i], "\t", cth [i], "\t", sth [i]


# matplotlib n'accepte que des donnees stockees dans un type de tableau
```

```
# definies dans le package Numeric
aT = N.array (T)
ath = N.array (th)
acth = N.array (cth)
asth = N.array (sth)

# dessin de la courbe
print "dessin de la courbe"
pylab.title("Oscillation d'un pendule")
pylab.xlabel("t")
pylab.ylabel("theta")
pylab.grid (True)
pylab.plot (aT, ath, linewidth=1.0, color = "blue")
pylab.plot (aT, acth, linewidth=1.0, color = "red")
pylab.plot (aT, asth, linewidth=1.0, color = "green")
pylab.show ()
```

**fin correction TD 24.5.1** □

# Chapitre 25

# Sujets autour des interfaces graphiques

---

## 25.1 Interface graphique

### 25.1.1 Enoncé

A l'aide du module `Tkinter`, on désire créer un programme avec une interface graphique permettant de tracer le graphe d'une fonction réelle.

**1)** Le premier objectif est de dessiner l'interface graphique. On s'appuie pour cela sur l'exemple suivant (téléchargeable depuis Internet) :

```
import Tkinter as T
root = T.Tk ()

# moitie gauche de la fenetre
f1 = T.Frame ()
f1.grid (column = 0, row = 0)

# moitie droite de la fenetre
f2 = T.Frame ()
f2.grid (column = 1, row = 0)

# creation d'une zone de texte dans la moitie gauche (f1)
l1 = T.Label (f1, text = "f(x) = ")
l1.grid (column = 0, row = 0)

# creation d'une zone de texte dans la moitie droite (f2)
lf2 = T.Label (f2, text = "dessin de la fonction")
lf2.grid (column = 0, row = 0)

# creation d'une zone de saisie
fonction = T.Entry (f1)
fonction.grid (column = 1, row = 0)
fonction.insert (0, "x**2 + x * math.cos (x)")

# creation d'une zone de dessin
dessin = T.Canvas (f2, width = 400, height = 400)
dessin.grid (column = 0, row = 1)
dessin.create_line (0,200,400,200, fill = "blue", width = 2)
dessin.create_text (10,210, text = "-5", fill = "black", font = "arial")

# creation d'un bouton quitter
bouton = T.Button (f1, text = "quitter")
bouton.grid (column = 0, row = 6)
bouton.config (command = root.destroy)

# creation d'un autre bouton
bouton = T.Button (f1, text = "dessin")
bouton.grid (column = 0, row = 2)

# on associe le bouton a une fonction
import random
def fonction_bouton_dessin () :
```

```
    dessin.create_line (0,0, random.randint (0,400), random.randint (0,400))

bouton.config (command = fonction_bouton_dessin)

root.mainloop ()
```

Chaque élément qui apparaît à l'écran est en fait un objet :

| | |
|---|---|
| `Label` | une zone de texte |
| `Entry` | une zone de saisie |
| `Canvas` | une zone de dessin |
| `Button` | un bouton |
| `Frame` | une zone ou une sous-fenêtre |

L'exemple précédent divise la fenêtre principale en deux zones `f1` et `f2`. La première zone (moitié gauche) permet de récupérer la fonction, la seconde zone (moitié droite) est la zone de dessin. On construit ensuite chacune des deux zones en plaçant chaque objet dans une grille (méthode `grid`). Tout ce qui est affiché est une zone de texte `Label`, tout ce qui est saisi et modifiable est une zone de saisie `Entry`.

**2)**

La fenêtre permet seulement pour le moment de saisir des informations. Il lui manque deux boutons, un pour quitter le programme, un autre pour demander au programme de dessiner le graphe de la fonction. Il faut donc dans un premier temps créer et placer ces boutons dans l'interface graphique. Dans un second temps, il faut dire au bouton d'exécuter une fonction lorsqu'il est pressé. On commence par le plus facile, c'est-à-dire le bouton *quitter*.

**3)**

Dans ce type de programme, il n'existe plus vraiment un début ou une fin, tout dépend du bouton pressé. Le bouton *quitter* a été configuré à la précédente question, il reste le bouton *dessiner*.

Le dernier objectif est de faire en sorte que lorsque le bouton *dessiner* est pressé, le programme dessine la fonction à l'écran. Cela revient à créer une fonction et à l'associer au bouton *dessiner*. Deux informations supplémentaires sont nécessaires :

1. Pour obtenir le contenu d'une zone de saisie, il faut lui appliquer la méthode `get`. Le contenu est une chaîne de caractères.
2. La fonction `eval` évalue des expressions sous forme de chaînes de caractères. Par exemple :

```
import math
x = 0
print eval ("math.cos (x) + math.exp (x)")  # affiche 2
```

### 25.1.2 Correction

```
# on importe le module Tkinter avec le pseudonyme T
import Tkinter as T
import math

# on cree la fenetre principale sans l'afficher
# la fenetre est vide, on lui ajoute des elements par la suite
root = T.Tk ()

# taille de l'ecran (carre)
ecran = 400
```

```
# moitie gauche de la fenetre
f1 = T.Frame ()
f1.grid (column = 0, row = 0)

# moitie droite de la fenetre
f2 = T.Frame ()
f2.grid (column = 1, row = 0)

# creation d'une zone de texte dans la moitie gauche (f1)
l1 = T.Label (f1, text = "f(x) = ")
l1.grid (column = 0, row = 0)

# creation d'une zone de texte dans la moitie droite (f2)
lf2 = T.Label (f2, text = "dessin de la fonction")
lf2.grid (column = 0, row = 0)

# creation d'une zone de saisie pour la fonction
fonction = T.Entry (f1)
fonction.grid (column = 1, row = 0)
fonction.insert (0, "x**2 + x * math.cos (x)")

# creation d'une zone de dessin
dessin = T.Canvas (f2, width = ecran, height = ecran)
dessin.grid (column = 0, row = 1)

# creation des quatre zones de texte et de saisie
# pour les limitesdu graphiques
lxmin = T.Label (f1, text = "xmin")
lxmax = T.Label (f1, text = "xmax")
lymin = T.Label (f1, text = "ymin")
lymax = T.Label (f1, text = "ymax")
xmin = T.Entry (f1)
xmax = T.Entry (f1)
ymin = T.Entry (f1)
ymax = T.Entry (f1)
lxmin.grid (column = 0, row = 2)
lxmax.grid (column = 0, row = 3)
lymin.grid (column = 0, row = 4)
lymax.grid (column = 0, row = 5)
xmin.grid (column = 1, row = 2)
xmax.grid (column = 1, row = 3)
ymin.grid (column = 1, row = 4)
ymax.grid (column = 1, row = 5)
xmin.insert (0, "-5")
xmax.insert (0, "5")
ymin.insert (0, "-5")
ymax.insert (0, "5")

# creation d'une zone de texte dans la moitie droite (f2)
lcouleur = T.Label (f1, text = "couleur")
lcouleur.grid (column = 0, row = 6)

# creation d'une zone de saisie pour la couleur
couleur = T.Entry (f1)
couleur.grid (column = 1, row = 6)
couleur.insert (0, "blue")

# creation d'un bouton quitter
bouton = T.Button (f1, text = "quitter")
bouton.grid (column = 0, row = 7)
bouton.config (command = root.destroy)

# creation d'un bouton dessiner
bouton = T.Button (f1, text = "dessin")
```

```
bouton.grid (column = 1, row = 7)

# dessin de la fonction
def fonction_bouton_dessin () :

    # on recupere les bornes du graphique converties en reels
    rxmin = float (xmin.get ())
    rxmax = float (xmax.get ())
    rymin = float (ymin.get ())
    rymax = float (ymax.get ())

    # on recupere la fonction
    f    = fonction.get ()

    # on recupere la couleur
    c    = couleur.get ()

    # tout le probleme consiste a convertir des coordonnees
    # reelles en pixels, c'est un changement de repere
    t    = (rxmax - rxmin) / ecran  # un pixel correspond a t

    # on va relier les points ( rxmin + n t, f ( rxmin + n t )
    # pour tous les n compris entre 0 et ecran
    x    = rxmin

    # tant que x de depasse pas rxmax
    while x <= rxmax :

        # on evalue la fonction f
        y  = eval (f)

        # on convertit le point (x, f(x)) en pixels (ix, iy)
        ix = int ((x - rxmin) / (rxmax - rxmin) * ecran)
        iy = ecran - int ((y - rymin) / (rymax - rymin) * ecran)

        if x == rxmin :
            # si c'est le premier point du graphe,
            dessin.create_line (ix,iy,ix,iy, fill = c, width = 1)
        else :
            # s'il y a deja eu d'autres points traces (ix_, iy_),
            # on les relie a ix, iy
            dessin.create_line (ix_,iy_,ix,iy, fill = c, width = 1)

        # on memorise ix,iy dans ix_,iy_
        ix_,iy_ = ix,iy

        # on passe au pixel suivant
        x += t

# on associe le bouton dessin a la fonction fonction_bouton_dessin
bouton.config (command = fonction_bouton_dessin)

# on lance la fenetre principale
root.mainloop ()
```

**fin correction TD 25.1.1** □

### 25.1.3 Enoncé

A l'aide du module `Tkinter`, on désire écrire le jeu du pendu avec une interface graphique

**1)** Le premier objectif est de dessiner l'interface graphique. On s'appuie pour cela sur l'exemple suivant :

```
#-*- coding:latin-1 -*-
import Tkinter                          # import du module des interfaces graphiques

# fenêtre principale
root = Tkinter.Tk ()

# on place un label
l = Tkinter.Label (text = "label")      # on crée l'objet
l.pack ()                               # on l'insère dans la fenêtre principale

# on place une zone de saisie
e = Tkinter.Entry ()                    # on crée l'objet
e.pack ()                               # on l'insère dans la fenêtre principale

# on place un bouton
b = Tkinter.Button (text = "bouton")    # on crée l'objet
b.pack ()                               # on l'insère dans la fenêtre principale

# on affiche la fenêtre
root.mainloop ()
```

Chaque élément qui apparaît à l'écran est en fait un objet :

| | |
|---|---|
| `Label` | une zone de texte |
| `Entry` | une zone de saisie |
| `Button` | un bouton |

A partir de cet exemple, constuire une fenêtre qui contiennent tous les éléments pour jouer au pendu.

**2)** Le programme ne fait pour le moment rien à part afficher une fenêtre. Lorsqu'on clique sur le bouton, il ne se passe rien. Pour changer cela, il faut attacher une fonction au bouton :

```
def fonction () :
    # faire quelque chose ici
b.config (command = fonction)
```

De cette manière, la fonction est attachée au bouton. Les deux instructions suivantes permettent de modifier l'intitulé de l'objet `Label` et de récupérer le contenu de la zone de saisie `Entry`.

```
l.config (text = "nouvel intitulé")  # changer l'intitulé
contenu = e.get ()                   # récupérer le contenu de la zone de saisie
```

Par exemple, le programme suivant attache une fonction au bouton qui récupère le contenu de la zone de saisie et qui le place dans l'objet `Label`.

```
#-*- coding:latin-1 -*-
import Tkinter                          #
root = Tkinter.Tk ()                    #
l = Tkinter.Label (text = "label")      # même programme que précédemment
l.pack ()                               #
e = Tkinter.Entry ()                    #
e.pack ()                               #
b = Tkinter.Button (text = "bouton")    #
b.pack ()                               #

def fonction () :                       #
```

```
    contenu = e.get ()                  #
    l.config (text = contenu)           # lignes insérées
b.config (command = fonction)           #

root.mainloop ()                        # même programme que précédemment
```

Dans cette question, il faut utiliser l'exemple précédent pour compter le nombre de fois qu'on clique sur un bouton et modifier l'intitulé d'un objet `Label`.

**Remarque 25.1 : problème d'exécution**
Certaines fois, il paraît impossible d'exécuter le programme qu'on vient de modifier. Il faut d'abord vérifier que le programme précédent est bien arrêté et l'arrêter de force le cas échéant. En effet, lorsqu'une erreur se produit dans le programme, celui déclenche une exception mais laisse l'interface graphique apparente qu'il faut malgré tout fermer avant de pouvoir recommencer avec un programme corrigé.

**Remarque 25.2 : variables globales**
Les fonctions attachées aux boutons ne prennent pas de paramètres. Pour contourner ce problème, on peut soit utiliser des classes [1], soit utiliser des variables globales. Au sein d'une fonction, on précise qu'une variable grâce au mot-clé `global` suivi du nom de la variable.

**3)** On revient au jeu du pendu, il faut écrire une fonction qui prend comme arguments un mot et une liste de lettres saisies par l'utilisateur, elle retourne un autre mot qui contient des blancs soulignés pour les lettres non trouvées.

**4)** Il ne reste plus qu'à terminer le jeu.

### 25.1.4 Correction

**1)** voir denière question.

**2)**

```
#-*- coding:latin-1 -*-
import Tkinter
root = Tkinter.Tk ()
l = Tkinter.Label (text = "label")
l.pack ()
e = Tkinter.Entry ()
e.pack ()
b = Tkinter.Button (text = "bouton")
b.pack ()

coup = 0

def fonction () :
    global coup
    coup += 1
    contenu = e.get ()
    message = "coup " + str (coup) + " saisie " + contenu
    l.config (text = message)
b.config (command = fonction)

root.mainloop ()
```

**3) 4)**

1. voir remarque 9.4 page 189 ou encore paragraphe 9.6.2, page 197

```
#-*- coding:latin-1 -*-
import Tkinter

mot_a_trouve = "pendu"
nb_coup      = 0

# dessin de la fenêtre
root = Tkinter.Tk ()

coup = Tkinter.Label ()
coup.pack ()

jeu = Tkinter.Label ()
jeu.pack ()

lettre = Tkinter.Entry ()
lettre.pack ()

b = Tkinter.Button (text = "jouer")
b.pack ()

# fonction pour le pendu
def blanc_souligne (mot, lettres) :
    res = ""
    for m in mot :
        if m in lettres : res += m
        else : res += "_"
    return res

def fonction_jouer () :
    global mot_a_trouve
    global nb_coup
    global coup
    global jeu
    global lettre

    nb_coup += 1
    m = blanc_souligne (mot_a_trouve, lettre.get ())

    if "_" in m :
        coup.config (text = "nombre de coups " + str (nb_coup))
    else :
        coup.config (text = "gagné")

    jeu.config (text = m)

b.config (command = fonction_jouer)

# root
root.mainloop ()
```

**fin correction TD 25.1.3** □

## 25.2 Jeu avec *PyGame*

### 25.2.1 Enoncé

Deux girafes sont en danger car un avion ne peut les éviter (voir figure 25.1). Votre mission est, si vous l'acceptez, de ralentir l'arrivée du bolide en frappant le plus vite possible alternativement les flèches gauche

et droite. Le meilleur joueur est celui qui aura amené l'avion le plus bas possible. L'explosion de l'avion sera annoncée à grands bruits. L'objectif du TD est de découvrir le module PyGame qui permet facilement de programmer un jeu, comment savoir qu'une touche a été pressée, comment déplacer un avion sans altérer le fond de l'écran, comment jouer des sons, ... Voilà quelques questions abordées durant ce TD.

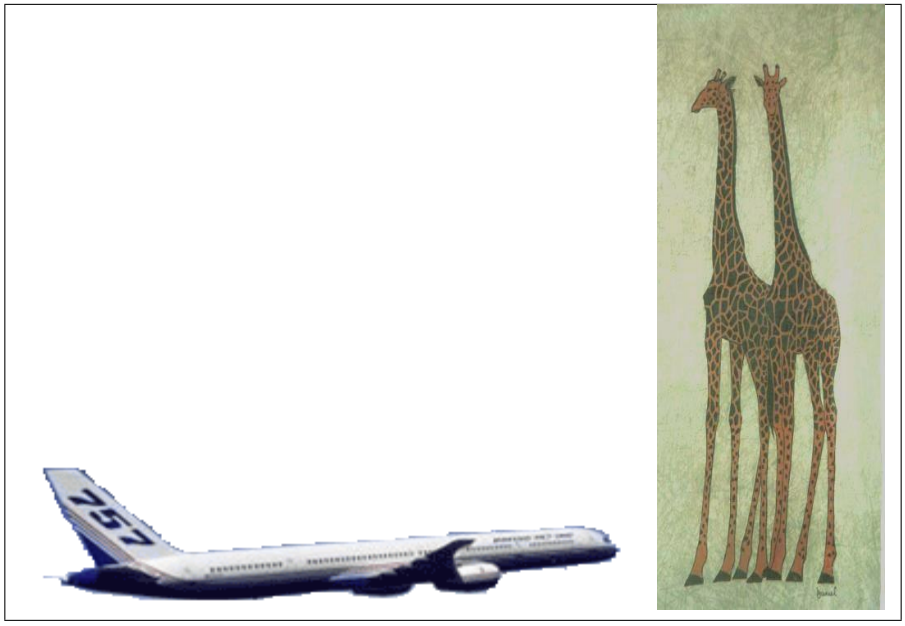



**Figure 25.1 :** *Deux girafes en danger à l'approche d'un avion.*

Le programme final nécessite quelques images et sons qu'il faut télécharger depuis le site *http ://www.xavierdupre.fr/*.

**1)** Le module PyGame facilite la création d'un jeu. En quelques lignes, il est possible de créer une fenêtre et d'y placer une image, ce que font les instructions suivantes.

```
    import pygame
    pygame.init ()
    image = pygame.image.load ("girage.jpg")
    sizeim = image.get_size ()
    size = (sizeim[0]*3, sizeim[1])
    screen = pygame.display.set_mode (size)
    screen.blit (image, (0,0))
    pygame.display.flip ()
```

Plus précisément, l'image `girafe.jpg` est lue et conservée en mémoire, on récupère ses dimensions puis on crée une fenêtre de jeu trois fois plus large que l'image. Toutefois le programme précédent s'exécute si rapidement qu'il est impossible de voir quoique ce soit. Pour cela, il faut appeler la fonction suivante à la fin du programme. D'après son nom, on peut deviner ce qu'elle est censée faire.

```
def attendre_clic (screen,x,y):
    while True:
        for event in pygame.event.get():
            if event.type == pygame.MOUSEBUTTONUP :
                return None
```

La première question du TD, sera de modifier le programme pour placer les deux girafes à l'autre bout de la fenêtre.

**2)** On voudrait maintenant afficher l'image `ciel.jpg` de façon à ce qu'elle apparaisse en dessous des girafes.

Pourquoi ne pas faire un petit tour sur Internet pour découvrir le rôle de l'instruction `pygame.display.flip()`, la recherche des mots-clé *pygame* et *pygame.display.flip* sur *Google* devrait vous y aider.

**3)** Au tour de l'image `avion.PNG` d'apparaître. Le résultat est-il vraiment satisfaisant ? Ce serait sympathique de faire disparaître tout ce blanc, un nouveau petit tour par Internet et l'utilisation de la méthode `set_colorkey` avec comme argument $(255, 255, 255)$.

**4)** On sait maintenant afficher toutes les images, les placer là où on en a envie. Il ne reste plus qu'à les animer. Le code suivant permet de faire patienter le programme pendant 100 millisecondes.

```
    import time
    time.sleep (0.1)
```

On veut maintenant, à l'aide d'une boucle, faire bouger l'avion depuis le coin supérieur gauche jusqu'aux girafes, et lorsque l'avion arrive à l'autre bout de la fenêtre, le programme doit s'arrêter.

**5)** Maintenant, lorsque l'avion atteint les girafes, on veut afficher une explosion, celle de l'image `explosion.jpg`.

**6)** Et pour faire plus réaliste, une explosion sonore serait la bienvenue. Le code suivant permet de jouer un son :

```
son = pygame.mixer.Sound ("bomb.wav")
son.play ()
```

Plusieurs sons sont proposés, `bomb.wav`, `explosion.wav`, `explosion_2.wav`, `missile.wav`, `sheep.wav`, `toilet_flush.wav`, `toilet_flush_2.wav`. A vous de composer votre petite explosion en sélectionnant le son qui vous plaît ou ceux qui vous plaisent tout en sachant que le code suivant produira non pas deux explosions consécutives mais deux explosions simultanées.

```
son = pygame.mixer.Sound ("bomb.wav")
son.play ()
son = pygame.mixer.Sound ("bomb.wav")
son.play ()
```

**7)** Question facultative... le code suivant permet d'afficher du texte à l'écran.

```
    font = pygame.font.Font ("freesansbold.ttf", 15)
    text = font.render ("message en blanc à la position 100,100", True, (255,255,255))
    screen.blit(text, (100,100))
    pygame.display.flip ()
```

Internet ou le chargé de TD vous donneront le sens de chaque instruction. Libre à vous de vous en servir pour décorer votre fenêtre de jeu.

**8)** Tous les éléments graphiques ou sonores vous sont désormais connus, il ne reste plus qu'à faire en sorte que le programme réagisse en fonction des touches pressées par le joueur. La fonction suivante retourne une chaîne de caractères si une des touches suivantes est pressée : flèche haut, flèche bas, flèche droite, flèche gauche, ECHAP.

```
def attendre_touche () :
    for event in pygame.event.get():
        if event.type == pygame.MOUSEBUTTONUP : return "clic"
        elif event.type == pygame.KEYDOWN :
            if event.key == 275 : return "right"
            elif event.key == 276 : return "left"
            elif event.key == 273 : return "up"
            elif event.key == 274 : return "down"
            elif event.key == 27 : return "quit"
```

```
            else :
                pass
                #print "key ", event.key # on imprime quand on ne connaît pas le code des touches
        else :
            #print event
            pass
    return ""
```

Il ne reste plus qu'à insérer cette fonction dans le programme pour permettre au joueur de ralentir l'inexorable course de l'avion vers les deux girafes. A vous de décider quelle touches permettront d'influencer sa course.

### 25.2.2 Correction

```
import pygame
import time

def attendre_touche () :
    """cette fonction retourne différente valeur selon la touche pressée :
       - right : pour la flèche droite
       - gauche : pour la flèche gauche
       - up : pour la flèche haut
       - down : pour la flèche bas
       - quit : pour la touche ESC ou ECHAP
       - '' : chaîne vide pour aucune touche pressée
    """
    for event in pygame.event.get():
        if event.type == pygame.MOUSEBUTTONUP : return "clic"
        elif event.type == pygame.KEYDOWN :
            if event.key == 275 : return "right"
            elif event.key == 276 : return "left"
            elif event.key == 273 : return "up"
            elif event.key == 274 : return "down"
            elif event.key == 27 : return "quit"
            else :
                pass
                #print "key ", event.key
        else :
            #print event
            pass
    return ""


class ImagePosition :
    """chaque image présente sur le disque dur doit être chargée en
    mémoire (self.image), cette image est affichée à l'écran (self.screen),
    à une position donnée (self.pos),
    si cette image est en fait un motif sur fond blanc, on peut préciser
    à l'ordinateur que ce fond blanc doit laisser apparaître le fond de l'écran
    """
    def __init__ (self, image, pos, screen, transparent = False) :
        """initialisation
           - image : nom d'image
           - pos : position ou afficher l'image
           - screen : variable modélisant l'écran
           - transparent : l'image est-elle un motif dont le fond blanc
                           doit être transparent ?
        """
        self.image  = pygame.image.load (image)
        self.x      = pos [0]
```

```
        self.y      = pos [1]
        self.screen = screen
        if transparent :
            # on précise ici que la couleur blanche (255,255,255)
            # doit être transparent
            self.image.set_colorkey ((255,255,255))

    def display (self, transparent = False) :
        """affiche l'image à une position donnée sur l'écran"""
        screen.blit (self.image, (self.x, self.y))

def affiche (images) :
    """affiche une liste d'image dans l'ordre dans lequel elles
    apparaissent dans la liste"""
    white  = (255,255,255)
    images [0].screen.fill (white) # met tout en blanc avant de dessiner les images
    for i in images :
        i.display ()

def deplacer_avion (images, avion, explosion, sounds, limitx) :
    """jeu, sans pression d'aucun touche, l'avion se déplace
    vers les deux girafes et fait tout exploser, on peut ralentir
    l'approche de l'avion en :
        - pressant les touches de directions
        - en frappant sans s'arrêter et alternativement les touches gauche droite
    """
    pair = 0   # pair : il faut presser la flèche gauche, impair, flèche droite
    tour = 0   # compte le nombre de tour d'attente, sert à réduire
               # l'impact des touches pressées par le joueur

    # séquence de trois instructions affichant le jeu
    affiche (images)    # on affiche le ciel et les girafes
    avion.display ()    # on affiche l'avion
    pygame.display.flip ()  # on rafraîchit l'écran

    # on regarde si une touche a été pressée
    t = attendre_touche ()

    # pour afficher le score
    font = pygame.font.Font ("freesansbold.ttf", 15)

    # le jeu s'arrête lorsqu'on presse la touche ECHAP ou si l'avion
    # touche les girafes
    while t != "quit" and avion.x < limitx :

        # on compte le nombre de tours
        tour += 1
        # en fonction de ce nombre de tour, on détermine l'impact d'une
        # séquence gauche droite
        dx    = max ((400 - tour) / 100, 1)

        # si une touche a été pressée, alors :
        if t == "left" : # flèche gauche
            avion.x -= 1
            if pair % 2 == 0 :
                avion.x -= dx
                pair += 1

        elif t == "right" :  # flèche droite
            avion.x += 1
            if pair % 2 == 1 :
                avion.x -= dx
                pair += 1

        elif t == "up" :   # flèche haut
```

```
            avion.y -= 1

        elif t == "down" : # flèche bas
            avion.y += 1


        # quelque soit la touche pressée, on actualise le jeu
        # c'est la même séquence de trois touches qu'au début de la fonction
        affiche (images)
        avion.display ()

        # on affiche le score également à l'écran
        text = font.render ("score " + str (avion.y), True,(255,255,255))

        screen.blit(text, (100,100))

        pygame.display.flip ()

        # le programme attend 50 millisecondes avant de continuer
        time.sleep (0.05)

        # on déplace l'avion de force
        avion.x += 2
        avion.y += 1

        # on regarde si une touche a été pressée
        t = attendre_touche ()

    # on est sorti de la boucle : c'est l'explosion
    # on affiche l'image et on rafraîchit l'écran
    explosion.display ()
    pygame.display.flip ()

    # on joue quelques sons
    for i in range (0, len (sounds)) :
        sounds [i].play ()
        time.sleep (1.0*i)  # on attend un peu avant de passer au son suivant
                            # il est possible de ne pas attendre et de tous les
                            # lancer en même temps

    # on rejoue les sons d'explosion 2 fois séparées de 2 secondes
    for i in range (0, 2) :
        sounds [0].play ()
        sounds [1].play ()
        time.sleep (2.0)

    # on affiche le score : plus l'avion descend, meilleur on est
    print "score ", avion.y

    # on affiche le score également à l'écran
    text = font.render ("score " + str (avion.y), True,(255,255,255))

    screen.blit(text, (100,100))
    pygame.display.flip ()

    # on attend la pression de la touche ECHAP avant de quitter le programme
    while t != "quit" :
        t = attendre_touche ()


if __name__ == "__main__" :

    # initialisation du module pygame
    pygame.init ()
```

```
# on récupère la taille de l'image de la girafe
sizeim = pygame.image.load("girafe.jpg").get_size ()

# on définit la taille de l'écran comme 3 fois plus large que celle de la girafe
size   = ( sizeim [0] * 3, sizeim [1])
screen = pygame.display.set_mode (size)

# on définit l'image de l'avion
avion       = ImagePosition ("avion.PNG", (0,0), screen, True)

# on définit l'image de l'explosion
explosion   = ImagePosition ("explosion.jpg", (0,0), screen)

# on estime la limite au delà de laquelle l'avion explose
limitx      = size [0] - sizeim [0] + sizeim [0] / 3 - avion.image.get_size () [0]

# on définit les sons qui doivent être joués à la fin du jeu
soundname   = ["bomb.wav", "explosion.wav", "explosion2.wav", "missile.wav", "sheep.wav"]
sounds = []
for s in soundname :
    try :
        # exception au cas où le son ne pourrait pas être lu
        sounds.append ( pygame.mixer.Sound (s) )
    except Exception, e :
        print "impossible de charger ", s
        print e

# on construit la liste des images qui doivent être affichées à chaque fois
images = []
images.append ( ImagePosition ("ciel.jpg", (0,0), screen) )
images.append ( ImagePosition ("girafe.jpg", (size [0] - sizeim [0], 0), screen) )

# on joue
deplacer_avion (images, avion, explosion, sounds, limitx)
```

**fin correction TD 25.2.1** □

# Chapitre 26

# Sujets mathématiques, statistiques

---

## 26.1 Casser le code de Vigenère

### 26.1.1 Enoncé

César en son temps utilisait déjà le cryptage pour transmettre ses messages. Il remplaçait chaque lettre par sa suivante. A devenait B, B devenait C, et ainsi de suite. Même sans connaître le décalage, décrypter un tel code est simple puisqu'il suffit de compter l'occurrence de chaque lettre du message. On peut raisonnablement affirmer que la lettre la plus fréquente est la lettre "e" ou, si l'auteur est Georges Pérec, au moins une voyelle. Par déduction, il devient facile de deviner les autres. Certains cryptographes plus subtiles eurent tôt fait d'agrandir l'alphabet de chiffrement et de faire en sorte qu'une lettre aurait d'autant plus de représentations cryptées qu'elle serait fréquente. Toutefois, les cryptanalystes ripostèrent par l'analyse des fréquences non plus des lettres mais des couples de lettres.

Pour résister à ces méthodes de décryptage, Blaise de Vigenère inventa aux alentours de 1586 (date de la publication de son livre *Le traité des chiffres*) un code qui ne se trahissait plus par l'étude des lettres ou des couples de lettres les plus fréquents. Cette méthode de chiffrement résista jusqu'en 1854, année où Babbage élabora une méthode qui permettait de casser ce code. Mais tout d'abord voyons en quoi consiste le code de Vigenère et cela commence par la description du **carré de Vigenère** : un alphabet recopié et décalé d'un cran vers la gauche à chaque ligne.

```
ABCDEFGHIJKLMNOPQRSTUVWXZ
BCDEFGHIJKLMNOPQRSTUVWXZA
CDEFGHIJKLMNOPQRSTUVWXZAB
DEFGHIJKLMNOPQRSTUVWXZABC
EFGHIJKLMNOPQRSTUVWXZABCD
FGHIJKLMNOPQRSTUVWXZABCDE
GHIJKLMNOPQRSTUVWXZABCDEF
HIJKLMNOPQRSTUVWXZABCDEFG
IJKLMNOPQRSTUVWXZABCDEFGH
JKLMNOPQRSTUVWXZABCDEFGHI
KLMNOPQRSTUVWXZABCDEFGHIJ
LMNOPQRSTUVWXZABCDEFGHIJK
MNOPQRSTUVWXZABCDEFGHIJKL
NOPQRSTUVWXZABCDEFGHIJKLM
OPQRSTUVWXZABCDEFGHIJKLMN
PQRSTUVWXZABCDEFGHIJKLMNO
QRSTUVWXZABCDEFGHIJKLMNOP
RSTUVWXZABCDEFGHIJKLMNOPQ
STUVWXZABCDEFGHIJKLMNOPQR
TUVWXZABCDEFGHIJKLMNOPQRS
UVWXZABCDEFGHIJKLMNOPQRST
VWXZABCDEFGHIJKLMNOPQRSTU
WXZABCDEFGHIJKLMNOPQRSTUV
XZABCDEFGHIJKLMNOPQRSTUVW
ZABCDEFGHIJKLMNOPQRSTUVWX
ABCDEFGHIJKLMNOPQRSTUVWXZ
```

Ensuite, nous avons besoin d'une clé : un mot assez court et sans accent, par exemple CODE. Et bien sûr,

il ne manque plus qu'un message à coder : *Les codes secrets ont joué un rôle discret mais important dans l'Histoire*. Tout d'abord, on va juxtaposer le messsage à coder et la clé maintes fois répétées :

```
lescodessecretsontjoueunrolediscretmaisimportantdanslhistoire
CODECODECODECODECODECODECODECODECODECODECODECODECODECODECODEC
```

A chaque lettre du message correspond ainsi une lettre de la clé et cette lettre va déterminer, à l'aide du carré de Vigenère, son équivalent crypté. En effet, la première lettre du message L est associée à la lettre C de la clé, cette première lettre est remplacée par celle se trouvant à l'intersection de la ligne commençant par C et de la colonne commençant par L, cela donne N. On continue comme cela jusqu'à la fin du message.

```
lescodessecretsontjoueunrolediscretmaisimportantdanslhistoire
CODECODECODECODECODECODECODECODECODECODECODECODECODECODECODEC
NSV...
```

En aparté, le petit programme suivant a permis de générer le carré de Vigenère.

```
al = "abcdefghijklmnopqrstuvwxyz"
al = al.upper ()

carre = []
for i in range (0,26) :
        carre.append (al)
        al = al [1:26] + al [0]

for c in carre :
        print c
```

L'objectif dans ce TD est de travailler à deux. Le premier s'occupe de crypter le message, le second de le décrypter.

**1)** Deux personnes, deux programmes, le premier pour chiffrer, le second pour déchiffrer.

**2)** Vérifier que les deux programmes fonctionnent en déchiffrant avec le second programme un message chiffré avec le premier. On évitera de recopier les programmes pour n'en faire qu'un seul pour plutôt les utiliser sous la forme de modules par l'intermédiaire du mot-clé `import`.

Au cas où un groupe de TD serait impair, le tout seul restant pourrait soit envisager de tout faire tout seul soit de s'associer à un groupe et d'implémenter la première étape d'un décryptage sans l'aide de la clé tel qu'il a été proposé par Babbage et dont la description suit.

Babbage s'est dit qu'un groupe de trois lettres consécutives avaient toutes les chances, à chaque fois qu'il apparaissait dans le message chiffré, d'être la conséquence du chiffrement des mêmes lettres du message avec les mêmes lettres de la clé. Pour un groupe de quatre lettres, c'est encore plus probable. Par conséquent, l'espacement entre deux mêmes groupes de lettres chiffrées est un multiple de la longueur de la clé. Par exemple, si la répétition d'un groupe est espacée de 30 lettres, puis celle d'un autre de 25, le plus grand diviseur commun de 25 et 30 est 5. La clé possède donc dans ce cas 5 lettres.

**3)** La tâche de la troisième personne consiste à programmer l'algorithme qui permet de calculer la longueur de la clé. On pourra penser à utiliser un dictionnaire.

**4)** Dernière question subsidiaire, pourquoi connaître la longueur de la clé permet-il, en estimant des occurrences, de terminer le décryptage du message ? S'il vous reste du temps...

### 26.1.2 Correction

```
def CarreVigenere () :
    """creation du carre de Vigenere"""
    al = "abcdefghijklmnopqrstuvwxyz"
    al = al.upper ()

    carre = []
    for i in range (0,26) :
        carre.append (al)
        al = al [1:26] + al [0]

    return carre


def CodeVigenere (message, cle, carre) :
    """code un message selon le code de Vigenere,
    message est le message a coder
    cle est la cle
    carre est le carre de Vigenere"""

    alphabet = carre [0]  # contient l'alphabet (premiere ligne du carre)
    resultat = ""         # contiendra le message code

    # on parcourt le message
    for i in range (0, len (message)) :

        # j est la position de la lettre dans le cle
        # associee a la lettre i dans le message
        # % retourne le reste d'une division entiere
        j = i % len (cle)

        a = alphabet.find (message [i])  # numero de la lettre message [i]
        b = alphabet.find (cle [j])      # numero de la lettre cle [j]

        c = carre [b][a]
        resultat += c

    return resultat

def DecodeVigenere (message, cle, carre) :
    """decode un message selon le code de Vigenere,
    message est le message a decoder
    cle est la cle
    carre est le carre de Vigenere"""

    alphabet = carre [0]  # contient l'alphabet (premiere ligne du carre)
    resultat = ""         # contiendra le message code

    # on parcourt le message
    for i in range (0, len (message)) :

        # j est la position de la lettre dans le cle
        # associee a la lettre i dans le message
        # % retourne le reste d'une division entiere
        j = i % len (cle)

        b = alphabet.find (cle [j])       # numero de la lettre cle [j]
        a = carre [b].find (message [i])  # numero de la lettre message [i]
                                          # dans la ligne du carre correspondant
                                          # au decalage introduit par cle [j]

        c = carre [0][a]
        resultat += c
```

```
    return resultat

message = "ceci est le message non code sans signe de ponctuation"
message = message.replace (" ", "")
message = message.upper ()
cle     = "VIGENERES"
carre   = CarreVigenere ()

code    = CodeVigenere (message, cle, carre)
decode  = DecodeVigenere (code, cle, carre)

print "------------------ cle"
print cle
print "------------------ message"
print message
print "------------------ message code"
print code
print "------------------ message decode"
print decode
print "------------------"
if decode == message : print "message bien retranscrit"
else : print "message mal retranscrit"


################################################################################
################################################################################
################################################################################

print ""
print "########################################################################"
print "seconde partie : decoder sans la cle"
print "cette methode ne marche que sur un message plus long ou de nombreux"
print "message mis bout a bout, on essaye la methode sur le fichier :"
print "hugo_dernier_jour_condamne.txt"
print "########################################################################"

def PGCD (m,n) :
    """determine le PGCD"""
    if m == 1 or n == 1 : return 1
    if m == n : return m
    if m < n : return PGCD (m, n-m)
    return PGCD (n, m-n)


def DecodeVigenereLongueurCle (message, mot = 3) :
    """cette fonction determine la longueur de la cle, elle
    repere les groupes de trois lettres qui se repete dans le message code
    et suppose qu'il y une tres forte probabilite qu'un meme groupe de trois
    soit code avec les memes trois lettres du message et les memes trois
    lettres de la cle

    message  : .....DES...........DES...........DES.........DES....DES
    cle      : ABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCD
    code     : .....EGV.........................EGV.........EGV..........
    distance :      <----------24-------------><----8----->

    la longueur de la cle divise le PGCD de 24 et 8
    """
    al = "abcdefghijklmnopqrstuvwxyz"
    al = al.upper ()


    # parcours du message pour recenser toutes les positions
    dico = {}
    for i in xrange (0, len (message)-2) :
```

```
        t = message [i:i+mot]
        if dico.has_key (t) : dico [t].append (i)
        else : dico [t] = [i]

    # on va garder toutes les distances entre
    # entre deux occurrences du meme mot de n lettres
    dis = []
    for d in dico :
        p = dico [d]
        if len (p) > 1 :
            for i in range (0, len (p)-1) :
                #print d, p [i+1] - p [i], " --- ", float (p [i+1] - p [i]) / 8
                dis.append ( p [i+1] - p [i] )

    # on extrait le PGCD
    if len (dis) == 0 :
        print "impossible de determiner la cle"
        return 0

    if len (dis) == 1 : return dis [0]

    longueur = PGCD (dis [0], dis [1])
    for d in dis :
        longueur = PGCD (longueur, d)

    if longueur > 5 :
        # si la longueur est suffisante, le resultat a des chances d'etre bon
        return longueur
    else :
        # sinon, on relance l'algorithme avec des mots plus grand
        return DecodeVigenereLongueurCle (message, mot+1)


def DecodeVigenereCle (code, l) :
    """determine la cle du message code, connaissant sa longueur,
    on suppose que la lettre E est la lettre la plus frequente
    """
    al  = "abcdefghijklmnopqrstuvwxyz"
    al  = al.upper ()
    cle = ""
    for i in xrange (0, l) :
        nombre = [ 0 for a in al]
        sous   = code [i:len (code):l]  # on extrait toutes les lettres
                                        # i, i+l, i+2l; i+3l, ...

        # on compte les lettres
        for k in sous : nombre [ al.find (k) ] += 1

        # on cherche le maximum
        p = 0
        for k in range (0, len (nombre)) :
            if nombre [k] > nombre [p] : p = k

        # on suppose que al [p] est la lettre E code,
        # il ne reste plus qu'a trouver la lettre de la cle
        # qui a permis de coder E en al [p]
        cle += al [ (p + 26 - al.find ("E")) % 26 ]

    return cle



# on lit Victor Hugo
f = open ("hugo_dernier_jour_condamne.txt")
```

```
message = f.read ()  # lit tout d'une seule traite
f.close ()

# on limite la taille du fichier
message = message [5000:7000]

# on enleve les signes de ponctuation et on met en majuscule
message = message.replace ("\n", "")
message = message.replace ("\r", "")
message = message.replace ("\t", "")
message = message.replace (" ", "")
message = message.replace (",", "")
message = message.replace (";", "")
message = message.replace (":", "")
message = message.replace (".", "")
message = message.replace ("'", "")
message = message.replace ("\"", "")
message = message.replace ("-", "")
message = message.replace ("!", "")
message = message.replace ("?", "")
message = message.replace ("(", "")
message = message.replace (")", "")
message = message.upper ()

# on code
print "on code, longueur du message ", len (message)
code    = CodeVigenere (message, cle, carre)
memoire = cle
cle     = None   # on oublie la cle

# on determine la longueur de la cle
l           = DecodeVigenereLongueurCle (code)
# on determine la cle en suppose que la lettre E est la plus frequente
# ne marche pas pour les textes anglais
cle_code    = DecodeVigenereCle (code, l)
# decode le texte
decode      = DecodeVigenere (code, cle_code, carre)



print "------------------ vraie"
print memoire
print "------------------ cle trouve par Babbage"
print "longueur ", l, " cle : ", cle_code
if memoire == cle_code : print "bonne cle"
else : print "mauvaise cle"
print "------------------ message"
print message [:200]
print "------------------ message code"
print code [:200]
print "------------------ message decode"
print decode [:200]
print "------------------"
if decode == message : print "message bien retranscrit"
else :
    for i in xrange (0, len (decode)) :
        if message[i] != decode [i] :
            print i, message[i], decode[i]
    print "message mal retranscrit"
```

**fin correction TD 26.1.1** □

## 26.2 Disparition des gens aux yeux bleus

### 26.2.1 Enoncé

***L'Express du 14/08/2003, Les blondes, par Marie Cousin, Jean-Pierre Dufreigne***

*Elles nous fascinent, elles nous agacent, elles nous séduisent. Leur chevelure claire en fait rêver plus d'une jusque sur le continent africain et au Japon. Pourtant, sur une planète de 6 milliards d'humains, elles ne sont que 3 millions. Alors pourquoi tant de passion ? Enquête*

*Il n'y a pas un an, en septembre 2002, la nouvelle s'est répandue comme une traînée de poudre d'or de la fée Clochette sur le Pays imaginaire de Peter Pan. Les chaînes de télé américaines ABC et CNN ainsi que plusieurs journaux britanniques (non tabloïds) annonçaient que, dans deux cents ans, les blondes naturelles auraient disparu de la surface de la planète. Le gène récessif de la couleur blonde se perdrait dans les méandres du métissage à l'horizon 2200. Tel était, en substance, le communiqué de l'Organisation mondiale de la santé (OMS) cité par les médias. Alerte à Malibu ! De Californie au cercle polaire arctique, blonds et blondes se déclarèrent prêts à faire don de leurs cheveux aux chercheurs, dans l'espoir que la génétique permettrait d'éviter l'inéluctable.*

*Quelques jours plus tard, l'OMS démentait l'information : l'extinction du gène des cheveux blonds n'a jamais été d'actualité. Le temps où les blondes ne seront plus que des châtains, des brunes ou des rousses décolorées n'est pas encore venu. Il n'empêche. Selon les chiffres de Martin Monestier dans son ouvrage Les Poils (Le Cherche Midi), la blonde est une denrée rare : "A l'échelle de l'humanité, on estime qu'il y a moins de 1 blonde pour 1 000 brunes ou châtains." 1 blonde pour 1 000 nous fait toutefois, sur 6 milliards d'individus, dont 52% de femmes, 3 gros millions de blondes.*

Après cette brève introduction, on cherche dans ce TD à étudier l'évolution d'un gène récessif comme celui des yeux bleus dans une population de quelques milliers d'individus.

**1)** On réduit un individu à la couleur de ses yeux : marrons ou bleus. Cette couleur est déterminée par deux gênes : bleus si les deux gênes sont bleus, marrons pour les trois autres cas. Cet individu sera représenté dans le programme informatique par une classe incluant ces deux gênes ainsi qu'une méthode retournant la couleur des yeux.

**2)** Deux personnes donnent naissance à une troisième qui hérite des gênes de ses deux parents : son premier gêne est choisi aléatoirement parmi les deux de sa mère, son second gêne est choisi aléatoirement parmi les deux de son père. Plutôt que de créer une méthode, on utilise l'opérateur `__add__` de l'addition pour donner naissance à un troisième individu. Le hasard en anglais se dit *random*, une recherche sur Internet à l'aide de ce mot-clé vous permettra de savoir comment tirer un nombre aléatoirement en langage *Python*.

**3)** On désire simuler la progression ou la récession du gêne récessif bleus sur une population de 1000 individus. On dispose au départ de 1000 individus. A l'étape suivante ou l'itération suivante, ces 1000 individus vont être remplacés par 1000 enfants, chacun nés de parents aléatoirement choisis de la génération précédente. L'objectif de cette question est de programmer le passage d'une génération à l'autre.

**4)** Le programme est presque terminé, il ne reste plus qu'à faire en sorte que le programme effectue quelques centaines d'itérations.

**5)** Le programme est terminé, on peut maintenant jouer avec, c'est-à-dire observer la proportion de gens aux yeux bleus génération après génération. Etant donné que le programme utilise le hasard à chaque itération, les résultats sont-ils à peu près stables si on exécute le programme plusieurs fois ? Sont-ils plus stables si on réduit ou si on augmente le nombre d'individus, si on réduit ou on augmente le nombre d'itérations ? En fonction de la proportion de gênes bleus au départ, comment évolue la proportion de gens aux yeux bleus après plusieurs centaines d'itérations ?

*A ce sujet, un article plus récent prédit l'extinction des blondes vers 2100.*

***Le Monde du 15/07/2007, La fin des vraies blondes ?, par Laure Belot, Véronique Lorelle***

C'est peu après 2200, en Finlande, que naîtra la dernière blonde de la planète. Cette affirmation a été relayée par différents médias, du New York Times à la BBC. Depuis 2002 pourtant, l'Organisation mondiale de la santé - citée comme la source de cette étude annonçant "l'extinction du gène caractérisant les vrais

blonds" - ne cesse de clamer qu'elle n'a "jamais conduit de recherches sur ce sujet".

Pour autant, le canular n'est pas sans fondements. La planète compte 6,5 milliards d'individus, dont environ 1 milliard en Afrique et 4 milliards en Asie (de la Turquie au Japon), selon le Population Reference Bureau. En 2050, la population mondiale passera à 9 milliards d'âmes. Les 3 milliards de personnes supplémentaires seront principalement issues d'Afrique, d'Asie et d'Amérique du Sud.

Mécaniquement, la fréquence des blonds, rares en dehors des pays occidentaux - même si on en trouve de façon sporadique en Afrique du Nord et au Moyen Orient - va diminuer. Et cela d'autant plus que cette teinte de cheveux, comme les yeux bleus, est due à des gènes récessifs. "Pour être blond, il faut hériter de deux gènes commandant ce caractère, l'un venu du père, l'autre de la mère", rappelle le généticien Axel Kahn. "Si, dans le monde, le nombre de gènes blonds reste constant alors que celui de gènes bruns augmente, il y aura une probabilité décroissante que deux personnes héritent de deux gènes blonds, condition sine que non de sa blondeur." Selon M. Kahn, une hypothèse réaliste serait celle "d'une augmentation du brassage des populations, et l'émergence d'un profil d'humain métissé, type brésilien".

L'Oréal, le numéro un mondial de la beauté, a déjà anticipé. Il construit à Saint-Ouen (Seine-Saint-Denis) un centre mondial dédié à la recherche capillaire, où 600 chercheurs, à l'horizon 2010, travailleront sur "une nouvelle forme de beauté, le cheveu métissé", explique Patricia Pineau, directrice de la communication de la recherche. D'ores et déjà, selon une étude conduite sur des milliers de chevelures, qui sera publiée dans Human Biology à l'automne, L'Oréal a recensé huit différents types de cheveux au lieu de trois à l'échelon mondial. "Jusqu'ici, on distinguait le cheveu caucasien, africain et asiatique. Nous avons mis au point une classification plus respectueuse de la géographie humaine, et basée sur la forme et la frisure du cheveu : du très raide (Asie, Inde, Mexique...) au crépu intense (Ghana, Nordeste brésilien). De quoi s'adapter à la diversité des chevelures induite par le métissage des populations."

Paradoxe : dans les prochaines décennies, la proportion de blond naturel va aller en s'amenuisant, et, pourtant, le mythe pourrait bien grandir. Déjà, en Europe, une femme sur trois se teint les cheveux dans une nuance de blond, alors que, selon Joanna Pitman, auteur de l'ouvrage Les Blondes (Autrement, 2005), seul "un adulte sur vingt aux Etats-Unis et à peu près la même proportion en Europe" ont cette vraie couleur. A quoi tient cette fascination ? L'histoire remonte, d'après l'écrivain, à Vénus-Aphrodite, déesse de l'amour à la chevelure claire... "Chaque époque a ensuite conféré (à la blondeur) des valeurs qui reflétaient ses préoccupations. Elle devint un vice pendant les âges obscurs, une obsession à la Renaissance, une mystique dans l'Angleterre élisabéthaine (...), une idéologie dans les années 1930, un signal d'invite sexuelle dans les années 1950...", écrit Joanna Pitman. Et à notre époque ? Pour l'écrivain britannique Marina Warner, spécialiste des mythes, "être blond, c'est rappeler la couleur de l'enfance, entrer dans la quête de la jeunesse éternelle".

Pour le psychanalyste Samuel Lepastier, le mythe de la blondeur est intimement lié à l'histoire européenne. Le continent, envahi par des peuples blonds - les Doriens, les Francs - a associé cette couleur au vainqueur : "C'est un mythe fondateur de l'inégalité des races humaines, dont la version la plus délirante a été le racisme hitlérien", souligne-t-il.

Mais l'attirance pour le blond s'expliquerait au-delà même du phénomène sociologique ou culturel, selon l'anthropologue canadien Peter Frost. Pour le chercheur, qui a étudié comment le gène a perduré à certains endroits de la planète, cela serait lié à la "sélection sexuelle qui privilégie les teintes vives mais aussi les couleurs rares. L'attirance sexuelle s'oriente vers les variantes moins fréquentes et plus accrocheuses. Ainsi s'installe un équilibre dynamique qui mène à une diversité de plus en plus grande de l'espèce."

Contrairement à des idées reçues, ce serait donc moins la couleur des blés qui séduit que sa rareté dans un groupe de population donnée. Et le chercheur canadien de citer les travaux du biologiste américain Thomas Thelen. "Ce dernier a préparé trois séries de diapositives montrant des femmes attirantes : la première présentait 6 brunes ; la deuxième 1 brune et 5 blondes ; et la troisième 1 brune et 11 blondes. Il a ensuite demandé à des hommes de choisir, pour chaque série, la femme qu'ils voudraient épouser. Résultat : plus la même brune était rare dans une série, plus ils la choisissaient."

Les sociétés aux cheveux de jais s'amusent déjà d'une chevelure différente de la leur, à l'instar des lolitas japonaises. "On crée non seulement des blonds peroxydés mais aussi des cheveux pourpres ou verts, des couleurs tout à fait novatrices, dont le but demeure le même : attirer l'attention", assure Peter Frost.

Pour autant, selon Samuel Lepastier, rien ne dit que le mythe blond ne sera pas remplacé par un autre, avec le basculement de la planète vers l'Asie et d'autres cultures dominantes. La force symbolique du blond, rappelle-t-il, est avant tout le fruit de deux cents ans de suprématie du monde occidental, depuis la colonisation jusqu'à l'expansion de la culture américaine, avec ses déesses platine.

Les généticiens, eux, sont sûrs d'une chose : le caractère "phénotype blond" ne s'éteindra jamais, dit Axel Kahn. "Il pourra resurgir ici ou là, un peu comme le font aujourd'hui les maladies rares." La planète n'est pas à l'abri de nouvelles surprises. A ce jour, des populations de blonds naturels se développent parmi les Inuits de l'ouest de l'Arctique canadien et les Aborigènes d'Australie centrale. "Dans ces deux cas, il s'agit peut-être, analyse Peter Frost, des débuts d'une évolution par sélection sexuelle, semblable à celle qu'ont connue les Européens."

**Etymologie**

**BLOND :** du latin blandus (charmant) et du latin médiéval blundus (jaune). Le mot apparaît au XIIe siècle en français et entre dans le vocabulaire courant vers 1930 avec les déesses hollywoodiennes. (Source : Joanna Pitman, v. ci-dessous)

**GENETIQUE :** La couleur des cheveux est déterminée par le gène MC1R, qui régule l'eumélanine (pigment brun-noir) et la phéomélanine (blond-roux). La proportion de ces pigments diffère dans le bulbe pileux, l'iris et la peau, d'où l'existence, par exemple, de bruns aux yeux bleus. Selon la méthodologie L'Oréal, un cheveu blond a au moins 50% de pigments clairs.

À LIRE Les Blondes. Une drôle d'histoire, d'Aphrodite à Madonna, de Joanna Pitman (Autrement, 2005).

From the Beast to the Blonde : On Fairy Tales and Their Tellers, de Marina Warner (Chatto and Windus, 1994)

### 26.2.2 Correction

```
import random

class Couleur :

    def __init__ (self, g1,g2) :
        self.g1 = g1
        self.g2 = g2

    def couleur (self) :
        """ deux genes differents : marron,
        deux genes egaux a marron : marron
        deux genes egaux a bleu : bleu"""
        if self.g1 != self.g2 : return "marron"
        elif self.g1 == "marron" : return "marron"
        else : return "bleu"

    def __str__ (self) :
        return self.couleur ()

    def __add__ (self, autre) :
        """creation d'un fils, on tire au hasard le premier gene
        parmi ceux de self, on recommence pour le second gene parmi ceux de autre"""

        n = random.randint (0,1)
        if n == 0 : g1 = self.g1
        else : g1 = self.g2

        n = random.randint (0,1)
        if n == 0 : g2 = autre.g1
        else : g2 = autre.g2

        return Couleur (g1,g2)
```

```
def creation_hasard_couleur (proportion_gene_bleu) :
    """cree un individu au hasard,
    proportion_gene_bleu est la proportion du gene bleu"""
    x = random.random ()
    if x <= proportion_gene_bleu : g1 = "bleu"
    else : g1 = "marron"

    x = random.random ()
    if x <= proportion_gene_bleu : g2 = "bleu"
    else : g2 = "marron"

    return Couleur (g1,g2)


def simulation (proportion_gene_bleu, nb_couleur, nb_iteration) :
    """effectue une simulation en tenant du nombre d'invidu (nb_couleur),
    du nombre d'iteration (nb_iteration),
    de la proportion de gene bleu initiale (proportion_gene_bleu),
    cette proportion est differente de la proportion de gens aux yeux bleus,
    proportion_gene_bleu est la probabilite du gene bleu face au gene marron"""

    l = [ creation_hasard_couleur (proportion_gene_bleu) for i in range (0,nb_couleur) ]

    # on compte les proportions des couleurs des gens au debut
    # on utilise un dictionnaire
    prop_couleur = { "bleu":0 , "marron":0 }
    for c in l : prop_couleur [ c.couleur () ] += 1

    print "distribution des couleurs d'yeux a la premiere generation "
    print prop_couleur

    for it in range (0, nb_iteration) :

        e = []
        for i in range (0, len (l)) :
            n1 = random.randint (0, len (l)-1)
            n2 = random.randint (0, len (l)-1)
            e.append ( l [n1] + l [n2] )

        if it % 100 == 0 :
            print it / 100, # pour l'avancement

        l = e

    # on compte les proportions des couleurs des gens a la fin
    # on utilise un dictionnaire
    prop_couleur_fin = { "bleu":0 , "marron":0 }
    for c in l : prop_couleur_fin [ c.couleur () ] += 1

    print "\ndistribution des couleurs d'yeux a la derniere generation "
    print prop_couleur_fin


l = [ 0.1 * (i+1) for i in range (0,9) ]
for x in l :
    print "\nproportion_gene_bleu = ", x
    simulation (x,1000,1000)

# ce programme effectue une simulation de 1000 iterations pour 1000 individus
# pour toutes les proportions de gene bleu [ 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
# l'objectif est de determiner a partir de quel seuil les gens aux yeux bleus ont une chance
# de ne pas disparaitre
#
# pour chaque seuil, une seule simulation de permet de conclure, il faudrait
```

```
# faire plusieurs centaines de simulation pour chaque seuil et ne conserver que la moyenne
# pour avoir un resultat fiable
```

**fin correction TD 26.2.1** □

## 26.3 Calculer des espérances de vie

### 26.3.1 Enoncé

Le programme est ici assez proche de celui réalisé en seconde séance de cours. Les données se résume à une table de mortalité [1]. Cette table consiste en une matrice :

| âge | homme | femme | ensemble |
|---|---|---|---|
| 0 | 100000 | 100000 | 100000 |
| 1 | 99571 | 99658 | 99613 |
| 2 | 99537 | 99631 | 99583 |
| 3 | 99514 | 99613 | 99562 |
| 4 | 99495 | 99599 | 99546 |
| 5 | 99479 | 99587 | 99532 |
| ... | | | |

On considère la dernière colonne qui forme une suite $(N_i)_{i\geqslant 0}$. $N_i$ est le nombre de survivants à l'âge $i$. Par exemple pour $i \leqslant j$, $p_{ij}^{+} = \frac{N_j}{N_i}$ est la probabilité d'être encore en vie à l'âge $j$ sachant qu'on a l'âge $i$. Autrement, l'espérance de vie est en fait l'âge moyen de mourir. D'une autre manière, $p_{ij} = \frac{N_j - N_{j+1}}{N_i}$ est la probabilité de mourir à l'âge $j$ sachant qu'on a l'âge $i$ [2]. En continuant ce raisonnement, on peut écrire que l'espérance de vie à l'âge $i$ est définie par :

$$E_i = \frac{\sum_{j=i}^{\infty} j\, p_{ij}}{\sum_{j=i}^{\infty} p_{ij}} \tag{26.1}$$

**1)** Ecrire une fonction qui retourne la probabilité de mourir à un âge $j$ sachant qu'on a l'âge $i$ [3].

**2)** Ecrire une fonction qui retourne l'espérance de vie à l'âge $i$.

**3)** Transformer ce programme de façon à ce qu'il fonctionne avec une classe.

**4)** Les données contiennent la table de mortalité pour les hommes, les femmes et l'ensemble de ces deux populations. Quelle est la proportion de femmes de plus d'un an ? Ecrire la méthode associée. En aparté, quelle est la proportion de femmes ?

**5)** Une personne de 50 ans souhaite acheter un appartement en viager à une personne de 70 ans. Combien de temps pourra-t-elle vivre dans cet appartement ?

**6)** On suppose que la mortalité infantile va être divisée par deux (mortalité à un an). On se demande ce que deviendra l'espérance de vie dans ce cas. Pour ce faire, on crée une seconde classe `PopulationPlus`

1. téléchargeable depuis http ://www.xavierdupre.fr/html/programmation/langage_py_thon/python_td_simple/index.html, données 2003-2005 France
2. $N_j > N_{j+1}$ car $N_j$ désigne le nombre de personnes encore vivantes à l'âge $j$, il y en a moins l'année d'après.
3. faire attention aux divisions entières

qui hérite de la précédente et pour laquelle, on remplace les fonctions adéquates pour que le calcul de l'espérance de vie soit correct.

**Pour les indomptables :**

**7)** Et si cette réduction du taux de mortalité n'était appliquée qu'aux femmes ?

**Pour les furieux :**

**8)** Le taux de fécondité est environ de deux enfants par femmes. Si on considère d'une manière simplifiée qu'une femme aura un enfant entre 25 et 34 ans et que l'âge importe peu, la probabilité qu'une femme ait un enfant entre 25 et 34 ans et de 0,2. Cette hypothèse très simple permet de calculer le nombre d'enfants l'année suivante et de faire évoluer la population sur plusieurs années. Au bout de cent ans, où nous mène cette hypothèse ? Quelle serait le taux de fécondité assurant une population constante au bout de 100 ans ? La proportion homme/femme change-t-elle ?

**Les Japonais menacés de disparition, par Frédéric Lemaître, Le Monde, 28 septembre 2007**

Les ordinateurs des démographes japonais sont formels. Au rythme actuel de décroissance de la population, le dernier Japonais naîtra dans huit cents ans. Bien sûr, une nouvelle ruse de l'histoire se chargera sans doute de démentir ce scénario catastrophe. Néanmoins les perspectives sont inquiétantes. De plus en plus, même. Il y a un an, les mêmes ordinateurs avaient fixé la date fatidique à l'an 3 200. Dans environ mille deux cents ans, donc. Mais les données officielles publiées à la fin de l'année 2006 les ont amenés à rapprocher l'échéance. Depuis 2005, ce pays de 127 millions d'habitants perd chaque année 1 million d'habitants. Dans moins d'un demi-siècle - en 2050 -, le pays aura perdu environ 37 millions d'habitants. Du jamais-vu dans un pays riche en temps de paix.

Certains Japonais ne s'en inquiètent pas outre mesure. Après tout, ce pays montagneux est essentiellement peuplé sur la bande côtière, où la densité de population dans certaines villes comme Tokyo apparaît comme un frein à la qualité de la vie. Mais cette opinion semble largement minoritaire, et les "déclinistes" ont le vent en poupe. Car non seulement la population va se réduire, mais elle va vieillir.

Le pays compte aujourd'hui 11 millions d'habitants de plus de 75 ans. Ce nombre devrait doubler dans les vingt ans. Signe des temps, le nouveau premier ministre, Yasuo Fakuda, est âgé de 71 ans. Dix-neuf ans de plus que son prédécesseur, Shinzo Abe. Le 18 septembre, le Japon a célébré, comme il se doit, le 112e anniversaire de Tomoji Tanabe, doyen mondial de la gent masculine. Selon les statistiques, le pays comptera dès le mois d'octobre 32 000 centenaires, un nombre en rapide augmentation.

Ce double phénomène - déclin et vieillissement - a d'importantes conséquences. Economiques, tout d'abord. Même si le lien entre population et croissance est complexe, la réduction de la population active devrait, à partir de 2030, priver chaque année le pays de plus d'un demi-point de croissance, a calculé l'institut de recherche de la banque Mizuho. Moins d'actifs, plus de retraités : l'équation pour les régimes de retraite est facile à deviner. L'âge de la cessation d'activité est progressivement porté de 60 à 65 ans, et ce n'est sans doute qu'un début. Du coup, le Japon ouvre trois pistes de réflexion longtemps restées taboues : la politique familiale à mener, la place des femmes dans le monde du travail et l'immigration.

Si la population décline, c'est en partie parce que le nombre de Japonaises âgées de 20 à 39 ans diminue. "Même si le nombre d'enfants par femme augmente, cela sera insuffisant pour maintenir la population au niveau actuel", affirme le professeur Akihiko Matsutani, un des chantres du déclinisme. Mais la raison essentielle est que le taux de fécondité des Japonaises est l'un des plus bas du monde : 1,3, bien loin des 2,1 requis pour assurer le renouvellement des générations. Même si les raisons d'un tel phénomène sont nombreuses, la modestie des aides publiques figure sans doute en bonne place. Le Japon y consacrait, en 2003, 1,1% de son PIB, contre 2,4% en moyenne dans les pays de l'OCDE et 3,6% en France, un des pays où la fécondité est la plus élevée. S'ils veulent faire démentir les démographes, les gouvernements à venir devront aider financièrement les familles à avoir des enfants. Question : des responsables confrontés à un déficit public colossal (180% du PIB, contre 64% en France) sont-ils prêts à investir dans une politique dont les effets ne se feront sentir qu'à moyen terme, souvent bien après qu'ils auront eux-mêmes pris leur retraite ?

Les entreprises ont aussi leur part de responsabilité. Le nombre de femmes cadres est extrêmement réduit, et, en raison de l'amplitude des journées de travail - fréquemment douze heures par jour pour un cadre

-, concilier vie professionnelle et vie familiale relève de l'exploit. Pourtant, pour limiter la baisse à venir de la population active, les entreprises n'auront d'autre choix que de favoriser l'activité des femmes, de leur offrir les mêmes perspectives de carrière qu'aux hommes, et sans doute de revoir les organisations du travail.

L'INNOVATION PLUTÔT QUE L'IMMIGRATION

Reste l'immigration. Aujourd'hui, le pays compte 2 millions d'étrangers, dont la moitié de Coréens, installés depuis plusieurs décennies et dont les descendants n'ont pas la nationalité japonaise, ainsi que 500 000 Chinois. Comparé aux pays européens, ce nombre est très faible. Néanmoins, pour des raisons culturelles, et notamment linguistiques, personne n'imagine ouvrir grandes les portes du pays. "Au Japon, on accueille les immigrés très qualifiés. Pas les autres", résume Kuniko Inoguchi, ancienne ministre pour l'égalité hommes-femmes et les affaires sociales, aujourd'hui députée. Mais le système montre vite ses limites. "99,99% des chercheurs employés dans le pays sont japonais, alors qu'aux Etats-Unis la moitié ne sont pas américains", affirme le professeur Matsunami. Si le Japon peut espérer puiser dans certaines réserves - notamment dans l'importante communauté japonaise qui réside au Brésil -, celles-ci ne sont pas inépuisables. Une tentative d'autoriser l'immigration de quelques centaines d'infirmières philippines est en cours. Mais cela restera marginal par rapport aux besoins.

Plutôt que l'immigration, le Japon a choisi d'innover. Privilégier les robots plutôt que les femmes de ménage chinoises ou philippines. Alors que la planète se couvre de têtes grises, le Japon, qui reste la deuxième puissance économique mondiale, prend une avance qui peut être décisive dans les technologies facilitant la vie des personnes âgées. Mais cela ne règle évidemment pas tous les problèmes, et la capacité d'innovation d'un pays vieillissant reste à prouver.

Une chose est sûre : le Japon ne pourra pas, dans les années qui viennent, prétendre encore peser dans les relations internationales - en particulier dans le dialogue qui l'obsède tant entre la Chine et les Etats-Unis - s'il ne s'ouvre pas davantage aux autres. Aux hommes, mais aussi aux capitaux et aux marchandises. Aujourd'hui, les investissements étrangers ne représentent que 5% du PIB (contre 27% en France) et le commerce avec les autres pays 10% (contre environ 40% en France). Alors que le dynamisme démographique américain pourrait permettre aux Etats-Unis de se replier sur eux-mêmes sans forcément décliner, ce n'est pas le cas du Japon. Seule relative bonne nouvelle pour Tokyo : les deux autres principales puissances asiatiques, la Chine et la Corée du Sud, connaissent également d'importants problèmes démographiques.

### 26.3.2 Correction

**1) 2)**

```
# colonne 1 : age
# colonne 2 : nombre d'hommes en vie a cet age depuis la naissance
# colonne 3 : nombre de femmes en vie a cet age depuis la naissance
# colonne 4 : nombre de gens en vie a cet age depuis la naissance
mortalite = [\                                                                  5
                [       0       ,       100000      ,       100000      ,       100000      ], \
                [       1       ,       99571       ,       99658       ,       99613       ], \
                [       2       ,       99537       ,       99631       ,       99583       ], \
                [       3       ,       99514       ,       99613       ,       99562       ], \
                [       4       ,       99495       ,       99599       ,       99546       ], \  10
                [       5       ,       99479       ,       99587       ,       99532       ], \
                [       6       ,       99465       ,       99577       ,       99520       ], \
                [       7       ,       99453       ,       99567       ,       99509       ], \
                [       8       ,       99442       ,       99558       ,       99499       ], \
                [       9       ,       99433       ,       99550       ,       99490       ], \  15
                [       10      ,       99422       ,       99543       ,       99481       ], \
                [       11      ,       99411       ,       99535       ,       99472       ], \
                [       12      ,       99400       ,       99527       ,       99462       ], \
                [       13      ,       99387       ,       99518       ,       99451       ], \
                [       14      ,       99371       ,       99507       ,       99437       ], \  20
```

```
[    15    ,    99350    ,    99495    ,    99421    ], \
[    16    ,    99320    ,    99480    ,    99398    ], \
[    17    ,    99280    ,    99461    ,    99368    ], \
[    18    ,    99227    ,    99438    ,    99330    ], \
[    19    ,    99157    ,    99410    ,    99281    ], \
[    20    ,    99076    ,    99382    ,    99225    ], \
[    21    ,    98989    ,    99353    ,    99167    ], \
[    22    ,    98899    ,    99324    ,    99106    ], \
[    23    ,    98808    ,    99295    ,    99046    ], \
[    24    ,    98718    ,    99267    ,    98986    ], \
[    25    ,    98632    ,    99239    ,    98928    ], \
[    26    ,    98545    ,    99212    ,    98871    ], \
[    27    ,    98455    ,    99180    ,    98809    ], \
[    28    ,    98367    ,    99148    ,    98748    ], \
[    29    ,    98279    ,    99116    ,    98687    ], \
[    30    ,    98187    ,    99080    ,    98623    ], \
[    31    ,    98085    ,    99039    ,    98551    ], \
[    32    ,    97979    ,    98998    ,    98476    ], \
[    33    ,    97871    ,    98951    ,    98398    ], \
[    34    ,    97756    ,    98899    ,    98314    ], \
[    35    ,    97633    ,    98846    ,    98225    ], \
[    36    ,    97500    ,    98783    ,    98126    ], \
[    37    ,    97355    ,    98716    ,    98020    ], \
[    38    ,    97201    ,    98642    ,    97905    ], \
[    39    ,    97033    ,    98558    ,    97777    ], \
[    40    ,    96849    ,    98464    ,    97637    ], \
[    41    ,    96645    ,    98363    ,    97483    ], \
[    42    ,    96427    ,    98249    ,    97316    ], \
[    43    ,    96186    ,    98128    ,    97134    ], \
[    44    ,    95912    ,    97987    ,    96924    ], \
[    45    ,    95606    ,    97837    ,    96695    ], \
[    46    ,    95272    ,    97675    ,    96445    ], \
[    47    ,    94901    ,    97493    ,    96166    ], \
[    48    ,    94483    ,    97295    ,    95855    ], \
[    49    ,    94024    ,    97087    ,    95518    ], \
[    50    ,    93532    ,    96858    ,    95155    ], \
[    51    ,    92993    ,    96615    ,    94760    ], \
[    52    ,    92428    ,    96358    ,    94346    ], \
[    53    ,    91835    ,    96084    ,    93909    ], \
[    54    ,    91201    ,    95794    ,    93442    ], \
[    55    ,    90533    ,    95487    ,    92950    ], \
[    56    ,    89824    ,    95168    ,    92432    ], \
[    57    ,    89070    ,    94828    ,    91880    ], \
[    58    ,    88266    ,    94472    ,    91295    ], \
[    59    ,    87407    ,    94086    ,    90666    ], \
[    60    ,    86496    ,    93667    ,    89995    ], \
[    61    ,    85546    ,    93226    ,    89294    ], \
[    62    ,    84556    ,    92766    ,    88562    ], \
[    63    ,    83504    ,    92272    ,    87783    ], \
[    64    ,    82388    ,    91749    ,    86956    ], \
[    65    ,    81202    ,    91188    ,    86075    ], \
[    66    ,    79938    ,    90587    ,    85135    ], \
[    67    ,    78599    ,    89936    ,    84131    ], \
[    68    ,    77184    ,    89236    ,    83065    ], \
[    69    ,    75653    ,    88467    ,    81906    ], \
[    70    ,    74012    ,    87633    ,    80659    ], \
[    71    ,    72245    ,    86710    ,    79304    ], \
[    72    ,    70350    ,    85692    ,    77837    ], \
[    73    ,    68317    ,    84567    ,    76247    ], \
[    74    ,    66155    ,    83344    ,    74543    ], \
[    75    ,    63859    ,    81995    ,    72709    ], \
[    76    ,    61404    ,    80490    ,    70718    ], \
[    77    ,    58802    ,    78823    ,    68572    ], \
[    78    ,    56060    ,    76968    ,    66263    ], \
[    79    ,    53165    ,    74899    ,    63771    ], \
```

```
                [       80        ,       50171       ,       72622       ,       61127       ], \
                [       81        ,       47013       ,       70078       ,       58269       ], \
                [       82        ,       43733       ,       67275       ,       55222       ], \
                [       83        ,       40329       ,       64168       ,       51963       ], \
                [       84        ,       36830       ,       60763       ,       48509       ], \
                [       85        ,       33321       ,       57061       ,       44906       ], \
                [       86        ,       29785       ,       53099       ,       41162       ], \
                [       87        ,       26244       ,       48845       ,       37273       ], \
                [       88        ,       22775       ,       44371       ,       33314       ], \
                [       89        ,       19407       ,       39804       ,       29360       ], \
                [       90        ,       16252       ,       35111       ,       25455       ], \
                [       91        ,       13351       ,       30448       ,       21694       ], \
                [       92        ,       10752       ,       25917       ,       18153       ], \
                [       93        ,       8489        ,       21596       ,       14885       ], \
                [       94        ,       6563        ,       17564       ,       11931       ], \
                [       95        ,       4919        ,       13959       ,       9331        ], \
                [       96        ,       3588        ,       10756       ,       7086        ], \
                [       97        ,       2562        ,       8079        ,       5254        ], \
                [       98        ,       1791        ,       5918        ,       3805        ], \
                [       99        ,       1254        ,       4245        ,       2714        ], \
                [       100        ,       879        ,       3014        ,       1921        ], \
                [       101        ,       628        ,       2110        ,       1351        ], \
                [       102        ,       467        ,       1482        ,       962        ], \
                [       103        ,       375        ,       1056        ,       707        ], \
                [       104        ,       279        ,       752        ,       510        ], \
               ]

def proba_vie_sup (mortalite, age_vie, age_mort) :
    nb1 = mortalite [age_vie][3]
    if age_mort < len (mortalite) : nb2 = mortalite [age_mort][3]
    else : nb2 = 0
    return float (nb2) / float (nb1)

def proba_mort_egale (mortalite, age_vie, age_mort) :
    p1 = proba_vie_sup  (mortalite, age_vie, age_mort)
    p2 = proba_vie_sup  (mortalite, age_vie, age_mort+1)
    return p1 - p2

def esperance_vie (mortalite, age_vie) :
    s = 0.0
    w = 0.0
    for i in range (age_vie, len (mortalite)) :
        s += i * proba_mort_egale (mortalite, age_vie, i)
        w += proba_mort_egale (mortalite, age_vie, i)
    return s / w - age_vie

print proba_vie_sup (mortalite, 20, 40)
print esperance_vie (mortalite, 20), esperance_vie (mortalite, 0)
```

### 3) 4)

On considère $\alpha_i$ la porportion de femme à l'âge $i$, $F_i$ le nombre femmes vivantes à l'âge $i$, $H_i$ le nombre d'hommes vivant et $N_i$ le nombre de gens vivant à l'âge $i$.

$$N_i = \alpha_i F_i + (1 - \alpha_i) H_i \tag{26.2}$$

$$\Longrightarrow \alpha_i = \frac{N_i - H_i}{F_i - H_i} \tag{26.3}$$

En aparté, on connaît la proportion d'hommes et de femmes de plus d'un an et le taux de mortalité à un an. On note $p_f$ la proportion cherchée, $m_{1f}$ le taux de mortalité des femmes à un an, $m_{1h}$ le taux de mortalité des hommes à un an, $p_{1f}$ la proportion de femmes de plus d'un an, $p_{1h}$ la proportion d'hommes

à un an : (voir la fonction `proportion_femme_tout`)

$$p = \frac{p_{1f}(1-m_{1f})^{-1}}{p_{1f}(1-m_{1f})^{-1} + p_{1h}(1-m_{1h})^{-1}}$$

**5)** On calcule les espérances de vie à 50 et 70 ans. Il suffit de faire la différence.

**6)** Il suffit de remplacer la fonction `proba_mort_egale` de telle que l'espérance de vie n'utilise plus le coefficient $p_{00}$ mais le coefficient $p'_{00} = 1 - \frac{1-p_{00}}{2}$.

**Pour les indomptables :**

**7)** Si la réduction de mortalité n'était appliquée qu'aux femmes, il faudrait utiliser la proportion de femmes pour reconstruire la table de mortalité de l'ensemble de la population avec les même proportions mais avec une table de mortalité féminine modifiée.

**Pour les furieux :**

**8)** Voir la classe `Simulation`. D'après ce modèle, le taux de fécondité devrait se situer entre 0.2106 et 0.2107. La proportion homme/femme ne change pas de manière significative.

```
# colonne 1 : age
# colonne 2 : nombre d'hommes en vie a cet age depuis la naissance
# colonne 3 : nombre de femmes en vie a cet age depuis la naissance
# colonne 4 : nombre de gens en vie a cet age depuis la naissance
mortalite = [\                                                                  5
                [       0       ,       100000       ,       100000       ,       100000       ], \
                [       1       ,       99571        ,       99658        ,       99613        ], \
                [       2       ,       99537        ,       99631        ,       99583        ], \
                [       3       ,       99514        ,       99613        ,       99562        ], \
                [       4       ,       99495        ,       99599        ,       99546        ], \ 10
                [       5       ,       99479        ,       99587        ,       99532        ], \
                [       6       ,       99465        ,       99577        ,       99520        ], \
                [       7       ,       99453        ,       99567        ,       99509        ], \
                [       8       ,       99442        ,       99558        ,       99499        ], \
                [       9       ,       99433        ,       99550        ,       99490        ], \ 15
                [       10       ,       99422       ,       99543        ,       99481        ], \
                [       11       ,       99411       ,       99535        ,       99472        ], \
                [       12       ,       99400       ,       99527        ,       99462        ], \
                [       13       ,       99387       ,       99518        ,       99451        ], \
                [       14       ,       99371       ,       99507        ,       99437        ], \ 20
                [       15       ,       99350       ,       99495        ,       99421        ], \
                [       16       ,       99320       ,       99480        ,       99398        ], \
                [       17       ,       99280       ,       99461        ,       99368        ], \
                [       18       ,       99227       ,       99438        ,       99330        ], \
                [       19       ,       99157       ,       99410        ,       99281        ], \ 25
                [       20       ,       99076       ,       99382        ,       99225        ], \
                [       21       ,       98989       ,       99353        ,       99167        ], \
                [       22       ,       98899       ,       99324        ,       99106        ], \
                [       23       ,       98808       ,       99295        ,       99046        ], \
                [       24       ,       98718       ,       99267        ,       98986        ], \ 30
                [       25       ,       98632       ,       99239        ,       98928        ], \
                [       26       ,       98545       ,       99212        ,       98871        ], \
                [       27       ,       98455       ,       99180        ,       98809        ], \
                [       28       ,       98367       ,       99148        ,       98748        ], \
                [       29       ,       98279       ,       99116        ,       98687        ], \ 35
                [       30       ,       98187       ,       99080        ,       98623        ], \
                [       31       ,       98085       ,       99039        ,       98551        ], \
                [       32       ,       97979       ,       98998        ,       98476        ], \
                [       33       ,       97871       ,       98951        ,       98398        ], \
                [       34       ,       97756       ,       98899        ,       98314        ], \ 40
                [       35       ,       97633       ,       98846        ,       98225        ], \
                [       36       ,       97500       ,       98783        ,       98126        ], \
                [       37       ,       97355       ,       98716        ,       98020        ], \
                [       38       ,       97201       ,       98642        ,       97905        ], \
```

```
[      39      ,      97033      ,      98558      ,      97777      ], \
[      40      ,      96849      ,      98464      ,      97637      ], \
[      41      ,      96645      ,      98363      ,      97483      ], \
[      42      ,      96427      ,      98249      ,      97316      ], \
[      43      ,      96186      ,      98128      ,      97134      ], \
[      44      ,      95912      ,      97987      ,      96924      ], \
[      45      ,      95606      ,      97837      ,      96695      ], \
[      46      ,      95272      ,      97675      ,      96445      ], \
[      47      ,      94901      ,      97493      ,      96166      ], \
[      48      ,      94483      ,      97295      ,      95855      ], \
[      49      ,      94024      ,      97087      ,      95518      ], \
[      50      ,      93532      ,      96858      ,      95155      ], \
[      51      ,      92993      ,      96615      ,      94760      ], \
[      52      ,      92428      ,      96358      ,      94346      ], \
[      53      ,      91835      ,      96084      ,      93909      ], \
[      54      ,      91201      ,      95794      ,      93442      ], \
[      55      ,      90533      ,      95487      ,      92950      ], \
[      56      ,      89824      ,      95168      ,      92432      ], \
[      57      ,      89070      ,      94828      ,      91880      ], \
[      58      ,      88266      ,      94472      ,      91295      ], \
[      59      ,      87407      ,      94086      ,      90666      ], \
[      60      ,      86496      ,      93667      ,      89995      ], \
[      61      ,      85546      ,      93226      ,      89294      ], \
[      62      ,      84556      ,      92766      ,      88562      ], \
[      63      ,      83504      ,      92272      ,      87783      ], \
[      64      ,      82388      ,      91749      ,      86956      ], \
[      65      ,      81202      ,      91188      ,      86075      ], \
[      66      ,      79938      ,      90587      ,      85135      ], \
[      67      ,      78599      ,      89936      ,      84131      ], \
[      68      ,      77184      ,      89236      ,      83065      ], \
[      69      ,      75653      ,      88467      ,      81906      ], \
[      70      ,      74012      ,      87633      ,      80659      ], \
[      71      ,      72245      ,      86710      ,      79304      ], \
[      72      ,      70350      ,      85692      ,      77837      ], \
[      73      ,      68317      ,      84567      ,      76247      ], \
[      74      ,      66155      ,      83344      ,      74543      ], \
[      75      ,      63859      ,      81995      ,      72709      ], \
[      76      ,      61404      ,      80490      ,      70718      ], \
[      77      ,      58802      ,      78823      ,      68572      ], \
[      78      ,      56060      ,      76968      ,      66263      ], \
[      79      ,      53165      ,      74899      ,      63771      ], \
[      80      ,      50171      ,      72622      ,      61127      ], \
[      81      ,      47013      ,      70078      ,      58269      ], \
[      82      ,      43733      ,      67275      ,      55222      ], \
[      83      ,      40329      ,      64168      ,      51963      ], \
[      84      ,      36830      ,      60763      ,      48509      ], \
[      85      ,      33321      ,      57061      ,      44906      ], \
[      86      ,      29785      ,      53099      ,      41162      ], \
[      87      ,      26244      ,      48845      ,      37273      ], \
[      88      ,      22775      ,      44371      ,      33314      ], \
[      89      ,      19407      ,      39804      ,      29360      ], \
[      90      ,      16252      ,      35111      ,      25455      ], \
[      91      ,      13351      ,      30448      ,      21694      ], \
[      92      ,      10752      ,      25917      ,      18153      ], \
[      93      ,      8489      ,      21596      ,      14885      ], \
[      94      ,      6563      ,      17564      ,      11931      ], \
[      95      ,      4919      ,      13959      ,      9331      ], \
[      96      ,      3588      ,      10756      ,      7086      ], \
[      97      ,      2562      ,      8079      ,      5254      ], \
[      98      ,      1791      ,      5918      ,      3805      ], \
[      99      ,      1254      ,      4245      ,      2714      ], \
[      100      ,      879      ,      3014      ,      1921      ], \
[      101      ,      628      ,      2110      ,      1351      ], \
[      102      ,      467      ,      1482      ,      962      ], \
[      103      ,      375      ,      1056      ,      707      ], \
```

```
                [       104         ,       279         ,       752         ,       510         ], \
                ]

class Population :

    def __init__ (self, mortalite) :
        self.mortalite = mortalite


    def proba_vie_sup (self, age_vie, age_mort) :
        nb1 = self.mortalite [age_vie][3]
        if age_mort < len (mortalite) : nb2 = self.mortalite [age_mort][3]
        else : nb2 = 0
        return float (nb2) / float (nb1)

    def proba_mortalite_sexuee (self, age_vie, age_mort, indice) :
        nb1 = self.mortalite [age_vie][indice]
        if age_mort < len (mortalite) : nb2 = self.mortalite [age_mort][indice]
        else : nb2 = 0
        return 1.0 - float (nb2) / float (nb1)

    def proba_mort_egale (self, age_vie, age_mort) :
        p1 = self.proba_vie_sup  (age_vie, age_mort)
        p2 = self.proba_vie_sup  (age_vie, age_mort+1)
        return p1 - p2

    def esperance_vie (self, age_vie) :
        s = 0.0
        w = 0.0
        for i in range (age_vie, len (mortalite)) :
            s += i * self.proba_mort_egale (age_vie, i)
            w += self.proba_mort_egale (age_vie, i)
        if w == 0 : return 0.0
        else : return s / w - age_vie

    def proportion_femme (self, age_vie) :
        d1 = self.mortalite [age_vie][3] - self.mortalite [age_vie][1]
        d2 = self.mortalite [age_vie][2] - self.mortalite [age_vie][1]
        if d1 == 0 and d2 == 0 : return 0.5
        else : return float (d1) / float (d2)

    def annee_viager (self, i, j) :
        return self.esperance_vie (i) - self.esperance_vie (j)

    def proportion_femme_tout (self) :
        p1f = self.proportion_femme (1)
        p1h = 1.0 - p1f
        m1f = self.proba_mortalite_sexuee (0, 1, 1)
        m1h = self.proba_mortalite_sexuee (0, 1, 2)
        cf  = 1.0 / (1.0 - m1f)
        ch  = 1.0 / (1.0 - m1h)
        return p1f * cf / ( p1f * cf + p1h * ch )


class PopulationPlus (Population) :

    def proba_mort_egale (self, age_vie, age_mort) :
        if age_mort > 0 :
            return Population.proba_mort_egale (self, age_vie, age_mort)
        else :
            p1 = self.proba_vie_sup  (age_vie, age_mort)
            p2 = 1.0 - (1.0 - self.proba_vie_sup  (age_vie, age_mort+1)) / 2
            return p1 - p2
```

```
class Simulation (Population)  :

    def __init__ (self, popu, fecondite = 0.2, age1 = 25, age2 = 34) :
        Population.__init__ (self, popu)
        self.fecondite = fecondite
        self.age1 = age1
        self.age2 = age2

    def prepare (self) :
        p = self.proportion_femme_tout ()
        self.sim = [ [ t [1] * p, t [2] * (1.0 - p) ] for t in self.mortalite ]

    def proportion_bebe_fille (self) :
        p1 = self.proportion_femme_tout ()
        #p2 = self.proportion_femme (1)
        return p1 # approximation

    def bebe (self) :
        b = 0
        for a in range (self.age1, self.age2 + 1) :
            b += self.sim [a][0] * self.fecondite
        p = self.proportion_bebe_fille ()
        return int (b * p), int (b * (1.0 - p))

    def iteration (self) :
        for i in range (len (self.sim)-1, 0, -1) : # boucle a l'envers
            self.sim [i][0] = self.sim [i-1][0] * (1.0 - self.proba_mortalite_sexuee (i-1, i, 1))
            self.sim [i][1] = self.sim [i-1][1] * (1.0 - self.proba_mortalite_sexuee (i-1, i, 2))
        f,g = self.bebe ()
        self.sim [0][0] = f
        self.sim [0][1] = g

    def simulation (self, nb) :
        self.prepare ()
        for i in range (0, nb) :
            self.iteration ()
        return self.sim

    def nb (self, sim) :
        f,g = 0,0
        for i in sim :
            f += i [0]
            g += i [1]
        return int (f), int (g)



p = Population (mortalite)
print p.proba_vie_sup (20, 40)
print p.esperance_vie (20), p.esperance_vie (0)
for i in range (0, 5) :
    print p.proportion_femme (i)

print "--------------------"
pp = PopulationPlus (mortalite)
print pp.esperance_vie (20), pp.esperance_vie (0)
print pp.proportion_femme_tout ()

for x in range (0, 200) :
    fecondite = 0.2 + x * 0.0001
    s = Simulation (mortalite, fecondite = fecondite)
    r = s.simulation (0)
    z = s.nb (r)
    r = s.simulation (100)
    print "fecondite = ", fecondite, " population H,F ", s.nb (r), " initiale ", z
```

240

**fin correction TD 26.3.1** □

## 26.4 Trouver une personne ayant les mêmes goûts que vous

### 26.4.1 Enoncé

Et si on refaisait la cérémonie des Césars 2005 ? Ou plutôt, on cherche à savoir quels sont les personnes qui ont des goûts proches ou éloignés en comparant leurs palmarès lors de cette cérémonie. La première étape de ce TD est que chacun construise son palmarès et l'envoie par mail à une personne qui recueillera l'ensemble des informations dans un fichier texte. Une fois ce fichier constitué, celle-ci le renverra à tout le monde.

Toutefois, il y a quelques petits détails à régler avant de commencer. Il est nécessaire de s'entendre sur une manière commune d'écrire les réponses. Ce fichier texte doit également contenir le prénom ou le nom de chaque personne afin qu'on puisse retrouver l'auteur d'un palmarès. Il est possible aussi d'autoriser une réponse "indécis", c'est une décision commune à prendre. Enfin, il est conseillé de lire le TD dans son ensemble afin de comprendre à quoi ces informations sont destinées et peut-être orienter différemment les premiers choix de programmation et de format.

**Meilleur Film français de l'année :**

1. 36, QUAI DES ORFÈVRES (Olivier Marchal)
2. UN LONG DIMANCHE DE FIANÇAILLES (Jean-Pierre Jeunet)
3. ROIS et REINE (Arnaud Desplechin)
4. LES CHORISTES (Christophe Barratier)
5. L' ESQUIVE (Abdellatif Kechiche)

**Meilleur Réalisateur :**

1. OLIVIER MARCHAL pour 36, quai des Orfèvres
2. JEAN-PIERRE JEUNET pour Un long dimanche de fiançailles
3. ARNAUD DESPLECHIN pour Rois et reine
4. CHRISTOPHE BARRATIER pour Les Choristes
5. ABDELLATIF KECHICHE pour L' Esquive

**Meilleur Acteur :**

1. DANIEL AUTEUIL dans 36, quai des Orfèvres
2. MATHIEU AMALRIC dans Rois et reine
3. GÉRARD JUGNOT dans Les Choristes
4. BENOÎT POELVOORDE dans Podium
5. PHILIPPE TORRETON dans L'Équipier

**Meilleure Actrice :**

1. AUDREY TAUTOU dans Un long dimanche de fiançailles
2. EMMANUELLE DEVOS dans Rois et reine
3. MAGGIE CHEUNG dans Clean

4. YOLANDE MOREAU dans Quand la mer monte
5. KARIN VIARD dans Le Rôle de sa vie

**Meilleur Acteur dans un second rôle :**

1. ANDRÉ DUSSOLLIER dans 36, quai des Orfèvres
2. CLOVIS CORNILLAC dans Mensonges et trahisons et plus si affinités...
3. JEAN-PAUL ROUVE dans Podium
4. FRANÇOIS BERLÉAND dans Les Choristes
5. MAURICE GARREL dans Rois et reine

**Meilleure Actrice dans un second rôle :**

1. ÉMILIE DEQUENNE dans L' Équipier
2. ARIANE ASCARIDE dans Brodeuses
3. JULIE DEPARDIEU dans Podium
4. MARION COTILLARD dans Un long dimanche de fiançailles
5. MYLÈNE DEMONGEOT dans 36, quai des Orfèvres

**Meilleur Espoir masculin :**

1. GASPARD ULLIEL dans Un long dimanche de fiançailles
2. OSMAN ELKHARRAZ dans L' Esquive
3. DAMIEN JOUILLEROT dans Les Fautes d'orthographe
4. JÉRÉMIE RENIER dans Violence des échanges en milieu tempéré
5. MALIK ZIDI dans Les temps qui changent

**Meilleur Espoir féminin :**

1. MARILOU BERRY dans Comme une image
2. SARA FORESTIER dans L' Esquive
3. LOLA NAYMARK dans Brodeuses
4. SABRINA OUAZANI dans L' Esquive
5. MAGALI WOCH dans Rois et reine

**Meilleur Premier Film :**

1. QUAND LA MER MONTE (Gilles Porte et Yolande Moreau)
2. BRODEUSES (Éleonore Faucher)
3. LES CHORISTES (Christophe Barratier)
4. PODIUM (Yann Moix)
5. VIOLENCE DES ÉCHANGES EN MILIEU TEMPÉRÉ (Jean-Marc Moutout)

**Meilleur Scénario original ou adaptation :**

1. AGNÈS JAOUI et JEAN-PIERRE BACRi pour Comme une image
2. ARNAUD DESPLECHIN et ROGER BOHBOT pour Rois et reine
3. JEAN-PIERRE JEUNET et GUILLAUME LAURANT pour Un long dimanche de fiançailles
4. ABDELLATIF KECHICHE et GHALYA LACROIX pour L' Esquive
5. OLIVIER MARCHAL, FRANCK MANCUSO et JULIEN RAPPENEAU pour 36, quai des Orfèvres

**Meilleure musique écrite pour un film :**

1. TONY GATLIF et DELPHINE MANTOULET pour Exils
2. ANGELO BADALAMENTI pour Un long dimanche de fiançailles
3. BRUNO COULAIS pour Les Choristes

4. NICOLA PIOVANI pour L'Équipier

**Meilleure Photo :**

1. BRUNO DELBONNEL pour Un long dimanche de fiançailles
2. JEAN-MARIE DREUJOU pour Deux frères
3. ÉRIC GAUTIER pour Clean

**Meilleur Son :**

1. PIERRE MERTENS, FRANÇOIS MAUREL, SYLVAIN LASSEUR et JOËL RANGON pour 36, quai des Orfèvres
2. DANIEL SOBRINO, NICOLAS CANTIN et NICOLAS NAEGELEN pour Les Choristes
3. JEAN UMANSKY, GÉRARD HARDY et VINCENT ARNARDI pour Un long dimanche de fiançailles

**Meilleurs Costumes :**

1. CATHERINE BOUCHARD pour Podium
2. MADELINE FONTAINE pour Un long dimanche de fiançailles
3. PIERRE-JEAN LARROQUE pour Arsène Lupin

**Meilleurs Décors :**

1. ALINE BONETTO pour Un long dimanche de fiançailles
2. FRANÇOIS CHAUVAUD pour Les Choristes
3. JEAN-PIERRE FOUILLET pour Immortel (ad vitam)

**Meilleur Montage :**

1. NOËLLE BOISSON pour Deux frères
2. HACHDÉ pour 36, quai des Orfèvres
3. HERVÉ SCHNEID pour Un long dimanche de fiançailles

**Meilleur Court Métrage :**

1. COUSINES (Lyes Salem)
2. HYMNE À LA GAZELLE (Stéphanie Duvivier)
3. LA MÉTHODE BOURCHNIKOV (Grégoire Sivan)
4. LES PARALLÈLES (Nicolas Saada)

**Meilleur Film de l'Union européenne :**

1. LA MAUVAISE ÉDUCATION (Pedro Almodovar)
2. JUST A KISS (Ken Loach)
3. MONDOVINO (Jonathan Nossiter)
4. SARABAND (Ingmar Bergman)
5. LA VIE EST UN MIRACLE (Emir Kusturica )

**Meilleur Film étranger :**

1. ETERNAL SUNSHINE OF THE SPOTLESS MIND (Michel Gondry)
2. 21 GRAMMES (Alejandro González Inárritu)
3. CARNETS DE VOYAGE (Walter Salles)
4. FAHRENHEIT 9/11 (Michael Moore)
5. LOST IN TRANSLATION (Sofia Coppola)

**1)** Une fois que le fichier des palmarès est constitué, il faut écrire un programme permettant de le lire afin de pouvoir manipuler les informations qu'il contient. Un indice avant de se lancer dans l'écriture : les chaînes de caractères possèdent une méthode `split` assez intéressante mais l'aide sur Internet sera sans doute plus bavarde.

**2)** On suppose que deux personnes sont proches si elles ont beaucoup de réponses en commun. Il faut construire une fonction qui compte le nombre de réponses communes entre deux personnes.

**3)** Chacun d'entre vous peut maintenant déterminer quelles sont les deux personnes qui ont le plus et le moins de réponses communes avec vous.

**4)** Vos réponses sont-elles réciproques : lorsqu'une personne A a déterminé l'autre personne la plus proche B, est-ce que A est aussi la personne la plus proche pour B ?

**5)** Il est possible de surcharger l'opérateur `__cmp__`, ceci permet de trier aisément une liste d'instances de la même classe avec la méthode `sort`, dans notre cas, il s'agit de trier la liste par nombre de réponses communes croissant.

**6)** Une fois la liste triée, on écrit dans un fichier texte les prénoms et le nombre de réponses communes de façon à conserver ce résultat.

### 26.4.2 Correction

```
import random

def genere_fichier_exemple (nom) :
    """cette fonction cree un fichier d'exemple contenant des prenoms
    et des reponses, le format est le suivant :

    prenom, espace, liste des reponses mise bout a bout, passage a la ligne"""
    prenom = [  "jean", "cecile", "juliette", "daniel", "julie", \
                "gerard", "amelie", "charles", "alix", "frederic", \
                "claire", "richard" ]
    f = open (nom, "w")
    for p in prenom :
        f.write (p)
        f.write (" ")
        for i in range (0,20) :
            a = random.randint (0,5) + 1
            f.write (str (a))
        f.write ("\n")
    f.close ()

class Personne :
    """on definit une personne"""
    def __init__ (self, prenom, reponse) :
        self.prenom  = prenom
        self.reponse = reponse
        self.commun  = 0    # compte le nombre de points communs

    def __cmp__ (self, autre) :
        """operateur de comparaison, reponse a la question 5"""
        if self.commun == autre.commun : return 0
        if self.commun < autre.commun : return -1
        return 1

    def __str__ (self) :
        """pour l'affichage"""
        s = self.prenom + "\t" + str (self.commun)
        return s

    def point_commun (self, autre) :
```

```
        """calcul le nombre de points commun entre deux personnes"""
        s = 0
        for i in range (0, len (self.reponse)) :
            if self.reponse [i] == autre.reponse [i] : s += 1
        return s


def lit_reponse (nom) :
    """on relie le fichier ecrit avec la fonction precedente"""
    f = open (nom, "r")
    line = f.readlines ()
    f.close ()

    rep = []
    for l in line :
        s = l.replace ("\n", "")  # on enleve les fins de ligne
        spl = s.split ()    # on coupe la ligne en deux, les deux morceaux
                            # separes par des espaces
        rep.append ( Personne ( spl [0], spl [1] ))
    return rep

def point_commun (n, rep) :
    """calcule le nombre de points communs entre rep [n] et rep [i],
    stocke le resultat dans rep [i].commun"""
    for r in rep :
        d = r.point_commun (rep [n])
        r.commun = d

############################################################
# debut du programme
############################################################
# creation du fichier
genere_fichier_exemple ("reponses.txt")

# relecture du fichier
rep = lit_reponse ("reponses.txt")

# calcule les points communs
point_commun (0, rep)

# on les trie par ordre croissant
rep.sort ()

print "le ou la plus proche ", rep [len (rep)-1]
print "le ou la plus loin ", rep [0]

# liste complete
for r in rep :
    print r
```

**fin correction TD 26.4.1** □

## 26.5 Calcul d'un histogramme

### 26.5.1 Enoncé

L'objectif de ce TD est de vérifier numériquement pour un exemple le théorème central limite. Soit $k \in \mathbb{N}$ et $n \in \mathbb{N}$, on suppose que $(X_1, \ldots, X_{nk})$ sont des variables aléatoires de Bernouilli telles que :

$$\forall i,\ \mathbb{P}(X_i = 0) = \frac{1}{2} \text{ et } \mathbb{P}(X_i = 1) = \frac{1}{2}$$

On construit la suite de variables $(Y_1, \ldots, Y_n)$ telles que :

$$\forall i,\ Y_i = \sum_{j=1}^{k} X_{ij}$$

On vérifie que :

$$\forall i,\ \forall l \in \{0, \ldots, k\},\ \mathbb{P}(Y_i = l) = \frac{C_k^l}{2^k}$$

$Y_i$ suit une loi binomiale $\mathcal{B}\left(k, \frac{1}{2}\right)$. Le théorème centrale limite permet d'affirmer que :

$$\frac{1}{\sqrt{n}} \sum_{i=1}^{n} \frac{\left(Y_i - \frac{k}{2}\right)}{\sqrt{\frac{k}{4}}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1)$$

Pour vérifier numériquement ce théorème, l'idée est de comparer graphiquement l'histogramme de la loi normale et celui des variables $Y_i$. Pour ce faire, on tire aléatoirement $n$ réels $(Z_i)$ suivant la loi normale. Un histogramme calculé à partir de cet échantillon est une suite de nombres entiers $(h_m)$ telle que :

$$h_m = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{Z_i \in [m\epsilon, (m+1)\epsilon[\}}$$

Autrement dit, $h_m$ est le nombre de nombres $Z_i$ compris dans l'intervalle $[m\epsilon, (m+1)\epsilon[$.

**1)** La première étape consiste à construire l'histogramme de la loi normale. On pourra prendre comme valeur :

$$n = 10000\,;\ \epsilon = 0,1$$

| **Fonctions utiles :** | |
|---|---|
| `gauss(mean, var)` | Permet de tirer un nombre au hasard selon une loi normale de moyenne `mean` et de variance `var`. Cette fonction fait partie du module `random`. Pour l'utiliser, il faut ajouter au début du programme la ligne `import random` puis écrire `h = random . gauss(0, 1)`, par exemple. |
| `round(x, d)` | Permet d'arrondir le nombre `x` à `d` décimales. |

L'histogramme pourra être représenté à l'aide d'un dictionnaire. Il sera ensuite affiché puis recopié dans le tableur Excel afin de le faire apparaître graphiquement (par l'intermédiaire d'un graphe de type XY). Cette courbe sera comparée à celle de la fonction [4] :

$$f(x) = \frac{1}{\sqrt{2\pi}}\, e^{-\frac{x^2}{2}},\ \text{qui est la fonction densité d'une loi normale.}$$

4. On compare en fait les courbes, $(m\epsilon, h_m)_m$ et $(m\epsilon, f(m\epsilon)\epsilon)_m$. La valeur de l'histogramme correspondant à la valeur de l'intégrale de la densité sur l'intervalle $[m\epsilon, (m+1)\epsilon]$.

**2)** La seconde étape consiste à construire l'histogramme de la variable $Y' = \sqrt{\frac{4}{k}}\left(Y - \frac{k}{2}\right)$ avec $Y = \sum_{j=1}^{k} X_j$ et $X_j$ un entiers aléatoire, 0 ou 1. On pourra prendre comme valeur :

$$n = 10000\,;\ \epsilon = 0,1\,;\ k = 400$$

On construit donc un nombre $Y$ comme étant la somme de $k$ entiers aléatoires choisis dans l'ensemble $\{0,1\}$ et on en déduit $Y'$. Puis on construit un autre histogramme de la même manière que celui de la question 1 mais cette fois-ci pour $n$ nombres $Y'_i$ construits comme le précédent. On récupère le résultat sous Excel pour vérifier, en principale, que la courbe obtenue suit à peu près le même dessin que celui de l'histogramme obtenu à la question 1.

| **Fonctions utiles :** | |
|---|---|
| `randint(a, b)` | Permet de tirer un entier au hasard entre `a` et `b`. Cette fonction fait partie du module `random`. Pour l'utiliser, il faut ajouter au début du programme la ligne `import random` puis écrire `h = random . randint(1, 20)`, par exemple. |
| `sqrt(x)` | Calcule la racine carrée du nombre `x`. Cette fonction fait partie du module `math`. Pour l'utiliser, il faut ajouter au début du programme la ligne `import math` puis écrire `r = math . sqrt(6)`, par exemple. |

**Travail à faire :**

Que se passe-t-il si on choisit $k = 100$ ? Pourquoi les courbes obtenues pour les deux questions ne coïncident-elles plus ?

**Indice :** Les deux courbes ont-elles le même nombre de points ? Quelle est la véritable valeur $\epsilon$ de l'histogramme obtenu pour $k = 100$ ?

### 26.5.2 Correction

**1)**

```
# -*- coding: cp1252 -*-
import random
import math

n  = 10000
hn = dict ()
for i in xrange (1,n):
    x  = random.gauss (0,1)
    xx = round (x, 1)
    if xx not in hn:                                    # si h [xx] n'existe pas encore
        hn [xx] = 0
    hn [xx] += 1

# affichage de l'histogramme
for w in hn:
    print w, "    \t", float (hn [w]) / float (n)
```

L'histogramme est ensuite recopié depuis l'interpréteur Python vers Excel où il sera dessiné (voir figure 26.1).

**2)**

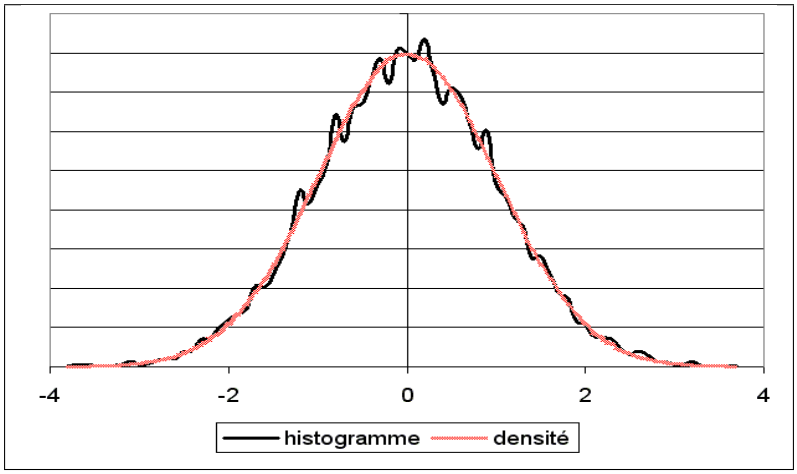



**Figure 26.1 :** *Histogramme sous Excel d'une variable suivant la loi normale (courbe noire), et sa densité (courbe rouge, grisée). Les abscisses ont été triées par ordre croissant. Si Excel est en version française, il est nécessaire de remplacer le symbol "." par ",". Les deux courbes comparées sont* $(m\epsilon, h_m)_m$ *pour l'histogramme et* $(m\epsilon, f(m\epsilon)\epsilon)_m$ *pour la densité. La valeur de l'histogramme correspondant à la valeur de l'intégrale de la densité sur l'intervalle* $[m\epsilon, (m+1)\epsilon]$.

```
# -*- coding: cp1252 -*-
import random
import math

n  = 10000
hb = dict ()
k  = 400
for i in xrange (1,n):
          if (i % 500 == 0) :               # comme le programme est assez lent,
              print i                 # cela permet de suivre l'avancement
    s = 0
    for j in xrange (1,k):
        a  = random.randint (0,1)
        s += a
    x  = float (s - k/2) / math.sqrt (float (k * 0.25))
    xx = round (x, 1)

    if xx not in hb:
        hb [xx] = 0
    hb [xx] += 1

# affichage de l'histogramme
for w in hb:
    print w, "    \t", float (hb [w]) / float (n)
```

Comme pour la question 1, l'histogramme est ensuite recopié depuis l'interpréteur *Python* vers Excel où il sera dessiné (voir figure 26.2). Lorsque $k = 100$, la variable $Y' = \frac{\sum_{i=1}^{k} \left(X_i - \frac{k}{2}\right)}{\sqrt{\frac{k}{4}}}$ où $X_i \in \{0,1\}$ est une variable aléatoire de binomiale $\mathcal{B}\left(k, \frac{1}{2}\right)$ définie par :

$$Y' = \frac{\sum_{i=1}^{k} (X_i - 50)}{5}$$

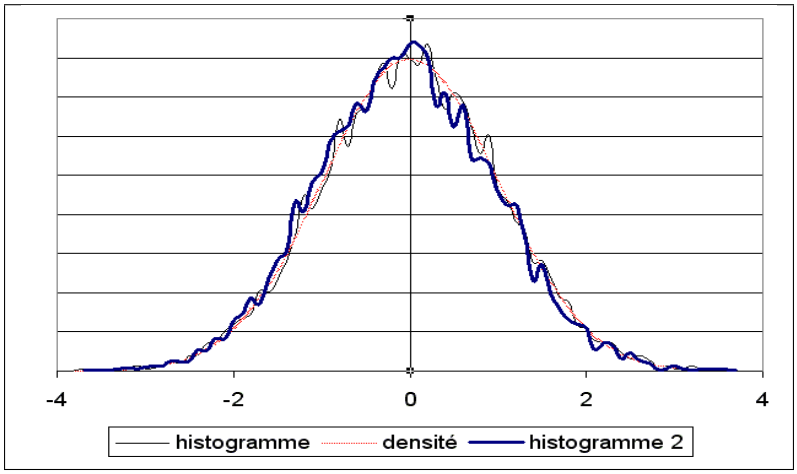



**Figure 26.2 :** *Graphique regroupant les deux histogrammes et la densité. Les trois courbes suivent le même chemin. Plus n sera grand, plus les histogrammes vont se rapprocher de la courbe de la densité.*

Par conséquent, $Y'$ est un nombre entier divisé par 5, c'est donc un multiple entier de $0,2$. Or la largeur d'un intervalle est de $\epsilon = 0,1 < 0,2$. L'histogramme calculé pour $k = 100$ ne peut avoir une largeur d'intervalle $\epsilon$ inférieure à $0,2$. Par conséquent, les histogrammes obtenus pour les deux questions ne sont plus comparables (voir figure 26.4).

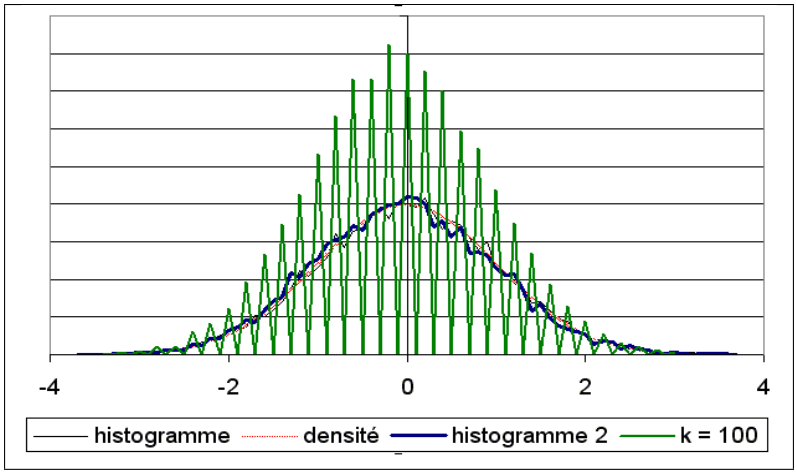



**Figure 26.3 :** *Pour $k = 100$, $\epsilon$ doit être supérieur à $0,2$. L'histogramme obtenu a une amplitude deux fois plus grande mais, pour la moitié des intervalles $[m\epsilon,(m+1)\epsilon[$, la valeur $h_m$ de l'histogramme est nul. On vérifie que pour tous les histogrammes, $\sum_m h_m = 1 = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$.*

### Pour aller plus loin :

Il est possible d'éviter le passage par Excel et d'afficher directement depuis *Python* la représentation graphique d'une distribution, ce que le code suivant permet de faire (voir figure 26.4). Cette solution nécessite l'installation de trois modules, le module `IaGraph`[5], `Numeric`[6], `ScientificPython`[7]. Ceux-ci sont parfois fournis avec un fichier exécutable qui effectue l'installation de manière automatique. L'installation est parfois manuelle et suit le même schéma que celle présentée au TD **??**, question **??**.

5. Module permettant de tracer des graphes de types XY ou des courbes de niveaux, adresse : http ://www.johnny-lin.com/py_pkgs/IaGraph/Doc/

6. Module proposant des fonctionnalités mathématiques, manipulation de tableaux, transformée de Fourier rapide, ... adresse : http ://sourceforge.net/projects/numpy

7. Module proposant d'autres fonctionnalités scientifiques, utilisation de VRML, multi threading, librairie MPI, dérivés, interpolation, quaternion, histogramme, ... adresse : http ://starship.python.net/ hinsen/ScientificPython/

```
x = hn.keys ()
x.sort ()
y = [ hn [a] for a in x ]
import IaGraph.plot
IaGraph.plot(x, y)
import Tkinter
Tkinter.mainloop ()
```

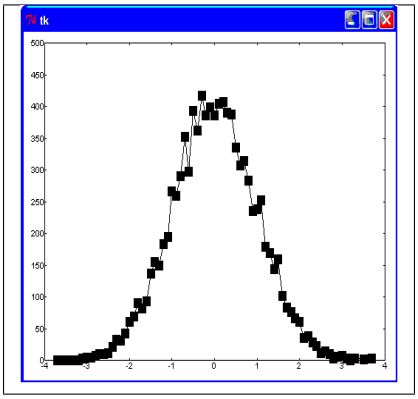



**Figure 26.4 :** *Représentation de l'histogramme obtenu à partir d'une loi normale en utilisant le module* `IaGraph.plot`.

Il est aussi possible d'utiliser la librairie `SciPy`[8]. Ce module inclut un sous-module `gplt`. Le programme fait apparaître une fenêtre (voir figure 26.5) qui disparaît lorsqu'on termine une session de *Python*.

```
from scipy import gplt
x1 = hn.keys ()
x1.sort ()
y = [ hn [a] for a in x1 ]
x2 = hb.keys ()
x2.sort ()
z = [ hb [a] for a in x2 ]
gplt.plot (x1,y,x2,z)
```

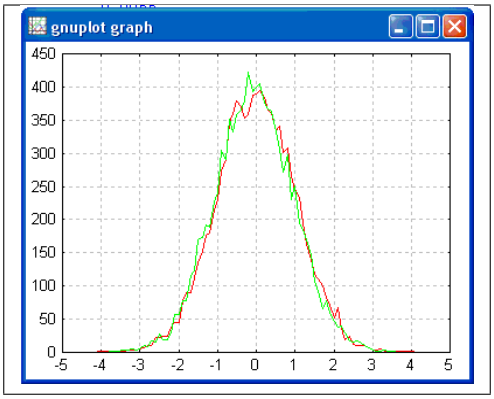



**Figure 26.5 :** *Représentation des histogrammes obtenus en utilisant le module* `scipy.gplt`.

8. Le module SciPy propose de nombreux algorithmes, algorithmes de classification, intégration, optimisation, génération de nombres aléatoires, traitement du signal, interpolation, algorithmes génétiques, algèbre linéaire, transformée de Fourier, dessin de courbe 2D, 3D, adresse : http ://www.scipy.org/

Un autre module `biggles` permet de dessiner une courbe. Il a l'avantage de pouvoir convertir les graphes directement sous forme d'images. Il est accessible depuis l'adresse *http ://biggles.sourceforge.net/*. Le résultat obtenu est illustré par la figure 26.6.

```
x1 = hn.keys ()
x1.sort ()
y = [ hn [a] for a in x1 ]
x2 = hb.keys ()
x2.sort ()
z = [ hb [a] for a in x2 ]

import biggles
p = biggles.FramedPlot()
p.title = "Histogramme"
p.xlabel = "x"
p.ylabel = "y"
p.add( biggles.Curve(x1, y, color="red") )
p.add( biggles.Curve(x2, z, color="blue") )
p.show()
```

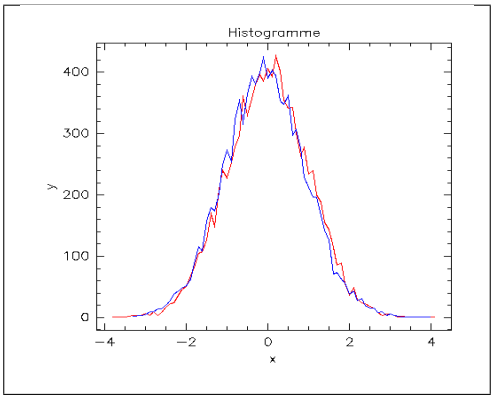



**Figure 26.6 :** *Représentation des histogrammes obtenus en utilisant le module* `biggles`.

Finalement, le programme est :

```
# -*- coding: cp1252 -*-
import random
import math

n  = 10000
hn = dict ()

for i in xrange (1,n):
    x  = random.gauss (0,1)
    xx = round (x, 1)
    if xx not in hn:                                    # si h [xx] n'existe pas encore
        hn [xx] = 0
    hn [xx] += 1

# affichage de l'histogramme
for w in hn:
    print w, "    \t", float (hn [w]) / float (n)


n  = 10000
hb = dict ()
k  = 400
```

```
for i in xrange (1,n):
    if (i % 500 == 0) :             # comme le programme est assez lent,
        print i              # cela permet de suivre l'avancement
    s = 0
    for j in xrange (1,k):
        a  = random.randint (0,1)
        s += a
    x  = float (s - k/2) / math.sqrt (float (k * 0.25))
    xx = round (x, 1)

    if xx not in hb:
        hb [xx] = 0
    hb [xx] += 1

# affichage de l'histogramme
for w in hb:
    print w, "    \t", float (hb [w]) / float (n)


x1 = hn.keys ()
x1.sort ()
y = [ hn [a] for a in x1 ]
x2 = hb.keys ()
x2.sort ()
z = [ hb [a] for a in x2 ]

import biggles
p = biggles.FramedPlot()
p.title = "Histogramme"
p.xlabel = "x"
p.ylabel = "y"
p.add( biggles.Curve(x1, y, color="red") )
p.add( biggles.Curve(x2, z, color="blue") )
p.show()
```

**fin correction TD 26.5.1** □

Septième partie

# DOMAINE HORS SUJETS

# Chapitre 27

# Sujets mathématiques

---

## 27.1 File d'attente

### 27.1.1 Enoncé

Cet énoncé s'inspire du livre *Devenez sorciers, devenez savants* de Georges Charpak et Henri Broch dont est tiré l'extrait suivant.

> Le présentateur se tourne vers la caméra principale, et d'un air très sérieux et enjôleur, regarde le téléspectateur droit dans les yeux en déclarant : "Allez-y ! Allumez cinq ou six lampes à côté de vous :" Puis il se tourne vers le médium et demande : "Vous pensez réellement pouvoir le faire ?" Après quelques moments d'hésitation, le mage se prononce : "J'espère avoir suffisamment de concentration ce soir, mais je ne suis pas dans les conditions vraiment idéales ; pour produire ce genre de phénomène à distance, d'habitude, je me retire pendant plusieurs jours dans une solitude totale et une profonde obscurité, après un jeûne strict."
>
> Si jamais il échoue, le public le mettra au compte des circonstances et non pas de ces compétences. Et, pourtant, le médium n'échoue pas. Des ampoules grillent chez les téléspectateurs qui regardent cette émission. Ils font part au standard téléphonique de la chaîne qui diffuse en direct cet extraordinaire moment de culture. Le médium a donc bien réussi - comme il le prétendait -, en concentrant sa puissance spirituelle sur la matière, à griller des ampoules électriques à distance.

---

> Supposons que cette émission soit suivie par environ un million de téléspectateurs et qu'elle dure une heure ou plus. Cela signifie qu'environ cinq à six millions d'ampoules ont été allumées pendant une heure ou plus. Supposons que ce nombre soit de deux millions.
>
> La durée de vie moyenne d'une ampoule à incandescence est de mille heures. Ce qui signifie que, pendant la durée de l'émission, il y aura environ deux milles lampes grillées.

Comment pourrait-on réaliser en *Python* une simulation numérique de cette expérience ?

Il est préférable de commencer les simulations avec de petits valeurs, 10000 ampoules pour une durée de vie de 100 heures et une expérience qui s'étale sur 500 heures.

### 27.1.2 Correction

Ce problème ressemble à un problème de files d'attente. Ce dernier consiste à déterminer un nombre adéquat de guichets en fonction de la vitesse de remplissage d'une file d'attente afin de limiter le temps d'attente. On désigne souvent cette problématique par un sigle du type $M/M/S$. Le premier $M$ signifie

qu'on suppose que la probabilité que $n$ personnes arrivent pendant une durée $t$ suit une loi de Poisson de paramètre $\lambda t$. Le second $M$ signifie qu'on suppose que le temps de traitement d'une personne par un guichetier suit une loi exponentielle de paramètre $\mu$. $S$ désigne le nombre de guichets. Pour de tels problèmes, on cherche à déterminer la probabilité que le système (file d'attente + guichets) contienne $n$ personnes. De cette probabilité, on peut déduire par exemple le temps d'attente moyen pour une personne qui entre dans la file d'attente. On suppose que le système est stationnaire, les probabilités ne dépendent pas du temps.

**Rappel 27.1 : loi de Poisson et loi exponentielle**
Si une variable $X$ suit une loi de Poisson de paramète $\lambda t$, elle a pour densité :

$$\mathbb{P}(X = n) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \tag{27.1}$$

Si une variable $X$ suit une loi exponentielle de paramètre $\mu$, elle a pour densité :

$$f(t) = \mu\, e^{-\mu t} \text{ et } \mathbb{P}(X \leqslant t) = \int_0^t \mu\, e^{-\mu x} dx = 1 - e^{-\mu t} \tag{27.2}$$

#### 27.1.2.1 File d'attente, cas M/M/1

On s'intéresse d'abord à un système $M/M/1$. Il n'y a donc qu'un seul guichet. $\lambda$ est le nombre moyen d'arrivée par unité de temps tandis que $\mu$ est le nombre moyen de clients traités par unité de temps. On suppose également que $\lambda < \mu$. Dans le cas contraire, la file d'attente devient infinie. On désigne par $p_n(t)$ la probabilité que la file d'attente contiennent $n$ personne. Que vaut cette probabilité à l'instant $t + dt$ ?

On considère que pendant la durée $t + dt$, au plus une personne peut s'ajouter à la file d'attente et au plus une personne peut être traitée par un guichetier. Les autres cas sont négligeables. On désigne par $B(x, t, dt)$ le fait qu'une personne quitte un guichet pendant les instants $t$ et $t + dt$ sachant qu'elle est arrivée au guichet à l'instant $x$. On cherche à déterminer la probabilité $\mathbb{P}(B(x, t, dt))$. On peut dire aussi que $\mathbb{P}(B(x, t, dt)$ est la probabilité que le temps de traitement d'une personne est inférieur à $t + dt - x$ sachant qu'il est supérieur à $t - x$. Si $D$ est une variable de durée suivant une loi exponentielle, alors :

$$\begin{aligned}
\mathbb{P}(B(x,t,dt)) &= \mathbb{P}(D \leqslant t + dt - x \mid D > t - x) = \frac{\mathbb{P}(t + dt - x \geqslant D > t - x)}{\mathbb{P}(D > t - x)} \\
&= \frac{\int_{t-x}^{t+dt-x} \mu e^{-\mu u} du}{\int_{t-x}^{\infty} \mu e^{-\mu u} du} = \frac{e^{-\mu(t-x)} - e^{-\mu(t-x+dt)}}{e^{-\mu(t-x)}} \\
&= 1 - e^{-\mu dt} \\
\mathbb{P}(B(x,t,dt)) &= -\mu dt + o(dt)
\end{aligned} \tag{27.3}$$

Cette probabilité ne dépend ni de $x$, ni de $t$. En ce qui concerne les arrivées, la probabilité qu'une personne arrive pendant les instants $t$ et $t + dt$ ne dépend pas du passé et suit une loi de Poisson de paramètre $\lambda$. Cette constation et l'équation (27.3) nous permettent d'écrire que :

$$\mathbb{P}(\text{une personne arrive pendant } dt) = \lambda dt\, e^{-\lambda dt} \sim \lambda dt + o(dt) \tag{27.4}$$

$$\mathbb{P}(\text{une personne part pendant } dt) = 1 - e^{-\mu dt} \sim \mu dt + o(dt) \tag{27.5}$$

De plus, pendant la durée $dt$, quatre cas sont possibles :

1. Une personne peut arriver sans qu'aucune ne parte d'un guichet.
2. Une personne peut partir d'un guichet sans qu'aucune autre n'arrive.
3. Une personne peut arriver et une autre partir d'un guichet.
4. Aucune personne n'arrive et aucune ne part d'un guichet.

Ces quatre cas permettent d'écrire la relation suivante :

$$\begin{aligned} p_n(t+dt) = & \; p_{n-1}(t)\,\lambda dt + \\ & p_{n+1}(t)\,\mu dt + \\ & p_n(t)\,(\mu dt \lambda dt) + \\ & p_n(t)\,(1-\mu dt)\,(1-\lambda dt) \end{aligned} \tag{27.6}$$

On néglige les termes du second degré en $(dt)^2$ :

$$p_n(t+dt) = p_{n-1}(t)\,\lambda dt + p_{n+1}(t)\,\mu dt + p_n(t)\,(1-\mu dt-\lambda dt) \tag{27.7}$$

$$\iff \frac{p_n(t+dt)-p_n(t)}{dt} = \lambda p_{n-1}(t) + \mu p_{n+1}(t) - (\mu+\lambda)\,p_n(t) \tag{27.8}$$

Cette relation n'est vraie que lorsque $n > 0$, lorsque $n = 0$, aucune personne déjà présente ne peut être traitée par un guichetier, on a donc :

$$\frac{p_0(t+dt)-p_0(t)}{dt} = \mu p_1(t) - \lambda p_0(t) \tag{27.9}$$

Comme le système est stationnaire, toutes les dérivées sont nulles. Les probabilités ne dépendent pas du temps. Ceci donne les relations suivantes :

$$\left\{ \begin{array}{lcl} \lambda p_{n-1} + \mu p_{n+1} - (\mu+\lambda)\,p_n & = & 0 \\ \mu p_1 - \lambda p_0 & = & 0 \end{array} \right. \tag{27.10}$$

$$\iff \left\{ \begin{array}{lcl} \mu p_{n+1} & = & (\mu+\lambda)\,p_n - \lambda\, p_{n-1} \\ \mu p_1 & = & \lambda p_0 \end{array} \right. \tag{27.11}$$

On vérifie par récurrence que :

$$p_n = \left(\frac{\lambda}{\mu}\right)^n p_0 \tag{27.12}$$

Il reste à déterminer $p_0$. Or, comme : $\sum_0^\infty p_n = 1 = p_0 \sum_0^\infty \left(\frac{\lambda}{\mu}\right)^n$, on obtient que :

$$\frac{p_0}{1-\frac{\lambda}{\mu}} = 1 \iff p_0 = 1 - \frac{\lambda}{\mu} \tag{27.13}$$

**Exemple :**

On suppose qu'un médecin traite en moyenne quatre patients par heure tandis qu'il accepte trois rendez-vous par heure. Donc $\lambda = 3$ et $\mu = 4$. Le nombre moyen $\overline{n}$ de patients dans sa salle d'attente est donné par :

$$\overline{n} = \sum_0^\infty np_n = \left(1-\frac{\lambda}{\mu}\right)\sum_0^\infty n\left(\frac{\lambda}{\mu}\right)^n = \frac{\frac{\lambda}{\mu}}{1-\frac{\lambda}{\mu}} = \frac{\lambda}{\mu-\lambda} \tag{27.14}$$

Il y a donc en moyenne trois personnes dans la salle d'attente de ce médecin. Comme le temps moyen de traitement de chacun est $\frac{1}{\mu}$, le temps moyen d'attente d'un patient arrivant dans la salle d'attente est : $\frac{\lambda\mu}{\mu-\lambda}$. Ce temps est égal à trois quarts d'heure pour cet exemple.

#### 27.1.2.2 File d'attente, cas M/M/S

Le système contient dorénavant $S$ guichets, on suppose que la vitesse $\mu$ de traitement des clients est commune à tous les guichets. On cherche tout d'abord la probabilité qu'une personne s'en aille parmi les $k$ qui occupent un guichet. On désigne par $(D_1,\ldots,D_k)$ $k$ variables indépendantes suivant une loi exponentielle de paramètre $\mu$, pendant un temps $dt$, la probabilité qu'une personne parmi $k$ quitte un guichet est :

$$\begin{aligned}
\mathbb{P}\left(\min\left\{D_1,\ldots,D_k\right\}\leqslant dt\right) &= 1-\mathbb{P}\left(\min\left\{D_1,\ldots,D_k\right\}>dt\right)\\
&= 1-\left[\prod_{n=1}^k \mathbb{P}\left(D_n>dt\right)\right]\\
&= 1-\left[\prod_{n=1}^k 1-\mathbb{P}\left(D_n\leqslant dt\right)\right]\\
&= 1-\left[\prod_{n=1}^k e^{-\mu dt}\right]\\
&= 1-e^{-k\mu dt}\sim k\mu dt+o(dt)
\end{aligned} \tag{27.15}$$

Pour déterminer les probabilités $(p_n)_n$, on applique le même raisonnement que pour un système $M/M/1$ en distinguant les cas $n\leqslant S$ et $n>S$. On adapte la récurrence donnée par le système d'équations (27.10) au cas $M/M/S$ :

$$\left\{\begin{array}{lcl}
\mu p_1-\lambda p_0 & = & 0\\
\lambda p_{n-1}+(n+1)\,\mu p_{n+1}-(n\mu+\lambda)\,p_n & = & 0 \text{ si } 1\leqslant n<S\\
\lambda p_{n-1}+S\mu p_{n+1}-(S\mu+\lambda)\,p_n & = & 0 \text{ si } n\geqslant S
\end{array}\right. \tag{27.16}$$

On suppose que $\frac{\lambda}{S\mu}<1$ afin que la file d'attente ne devienne infinie. Comme pour un système $M/M/1$, ces formules de récurrences et le fait que $\sum_0^\infty p_n=1$ permet de déduire que :

$$\left\{\begin{array}{lcl}
p_0 & = & \dfrac{1}{\frac{\left(\frac{\lambda}{\mu}\right)^S}{S!\left(1-\frac{\lambda}{S\mu}\right)}+\sum\limits_{k=1}^{S-1}\frac{\left(\frac{\lambda}{\mu}\right)^n}{n!}}\\
p_n & = & \dfrac{1}{n!}\left(\dfrac{\lambda}{\mu}\right)^n p_0 \quad \text{si } n<S\\
p_n & = & \dfrac{1}{S^{n-S}S!}\left(\dfrac{\lambda}{\mu}\right)^n p_0 \quad \text{si } n\geqslant S
\end{array}\right. \tag{27.17}$$

Ces calculs sont utilisés pour optimiser le nombre de guichets. Chaque guichetier a un coût qui doit être comparé avec le coût associé au temps d'attente des clients. Ces résultats sont extraits du livre [Faure2000].

> "La théorie des files d'attente remonte aux premiers travaux de K. Erlang (1917), sur le calcul du nombre d'organes de chaque type à installer dans un central téléphonique automatique. Développée aussi par Engset (1918), cette théorie s'est amplifiée sous l'impulsion de nombreux chercheurs (E. Borel, D. Kendall, A. Kolmogorov, Khintchine, LC. Palm, F. Pollaczek, L. Feller, ...). Les informaticiens l'utilisent notamment pour l'évaluation de performances, à titre prévisionnel, des systèmes ou des réseaux informatiques.

#### 27.1.2.3 Retour aux ampoules

La durée de traitement d'un client fait penser à la durée de vie d'une ampoule. Les lampes font office de guichets tandis que le rôle des clients est joué par des lumières. Toutefois, ce n'est pas le temps d'attente moyen ni la longueur moyenne de la file d'attente qui nous intéresse mais, en quelque sorte, le nombre de clients qui sont traités pendant une durée $T$. En fait, plus exactement, c'est le nombre de guichets qui auront traités au moins un client pendant une durée $T$ qui nous intéresse. Il correspond exactement au nombre d'ampoules qui vont griller pendant cette même période $T$. Il reste à définir ce qu'est une file d'attente d'ampoules et surtout son paramètre $\lambda$.

Lorsqu'une ampoule grille, elle est a priori changée dans les plus brefs délais, comme si la file d'attente des ampoules était infinie. Ceci signifie que $\lambda >> \mu$, configuration qui sort du champ d'application des files d'attente $M/M/S$. Le paramètre $\lambda$ n'a a priori aucun rôle à jouer. On peut néanmoins s'inspirer de la méthode développée dans les paragraphes précédants pour aborder le problème des ampoules.

On suppose que la durée de vie d'une ampoule suit toujours une loi exponentielle de paramètre $\mu$ et qu'il y en a exactement $S$ qui brillent en même temps. On note $p_n(t)$ la probabilité que $n$ ampoules aient grillées à l'instant $t$. Si $N(t)$ est le nombre d'ampoules grillées à l'instant $t$ : $p_n(t) = \mathbb{P}(N(t) = n)$. Cette fonction n'est plus stationnaire et décroît avec le temps à partir d'un certain moment [1]. On utilise un raisonnement similaire à celui qui a permis d'écrire les équations (27.4), (27.5), (27.6) pour obtenir :

$$\begin{aligned} p_n(t+dt) &= (1 - S\mu dt)\, p_n(t) + S\mu p_{n-1}(t)dt \\ \iff \frac{p_n(t+dt) - p_{n-1}(t)}{dt} &= -S\mu p_n(t) + S\mu p_{n-1}(t) \\ \iff p'_n(t) &= -S\mu p_n(t) + S\mu p_{n-1}(t) \end{aligned} \tag{27.18}$$

On connaît la fonction $p_0(t)$ puisqu'elle correspond à la probabilité qu'aucune des $S$ ampoules n'ait grillé depuis l'instant 0 jusqu'à l'instant $t$, par conséquent :

$$\begin{aligned} p_0(t) &= \mathbb{P}(\text{ durée de vie des } S \text{ ampoules soient toutes supérieures à } t\ ) \\ \implies p_0(t) &= \left[\int_t^\infty \mu e^{-\mu u} du\right]^S \\ \implies p_0(t) &= e^{-S\mu t} \end{aligned} \tag{27.19}$$

L'équation (27.18) permet de définir une suite d'équations différentielles du premier degré :

1. Plus on avance dans le temps, plus le nombre d'ampoules grillées augmente. Par conséquent, la probabilité qu'il y ait $n$ ampoules grillées augmente tout d'abord puis, à partir d'un moment $t$, elle diminue.

$$\begin{aligned}
p_0(t) &= e^{-S\mu t} \\
p_1'(t) &= -S\mu p_1(t) + S\mu e^{-S\mu t} \\
p_2'(t) &= -S\mu p_2(t) + p_1(t) \\
&\dots \\
p_n'(t) &= -S\mu p_n(t) + S\mu p_{n-1}(t)
\end{aligned}$$

On peut donc calculer par récurrence la suite de fonction $(p_n(t))_n$. La solution de l'équation différentielle homogène est $e^{-S\mu t}$. On utilise la méthode de la variation de la constante en posant $p_n(t) = C_n(t)e^{-S\mu t}$. On aboutit à l'équation :

$$\begin{aligned}
p_n'(t) &= -S\mu p_n(t) + S\mu p_{n-1}(t) = -S\mu p_n(t) + C_n'(t)e^{-S\mu t} \\
\Longrightarrow C_n'(t)e^{-S\mu t} &= S\mu p_{n-1}(t) \\
\Longrightarrow C_n'(t) &= S\mu p_{n-1}(t)e^{S\mu t}
\end{aligned}$$

Pour $n = 1$, on obtient $C_1'(t) = S\mu \Longrightarrow C_1'(t) = S\mu t + A_1$. Par conséquent, $p_1(t) = (S\mu t + A_1)\, e^{-S\mu t}$. On sait que $\forall t,\ \sum_0^\infty p_n(t) = 1$ mais cela ne permet pas de déterminer la constante $A_1$. Néanmoins, en faisant tendre $t \longrightarrow 0$, nécessairement $p_1(t) \longrightarrow 0$. Par conséquent : $A_1 = 0$ et $p_1(t) = S\mu t\ e^{-S\mu t}$. On pose maintenant $p_2(t) = C_2(t)e^{-S\mu t}$. La résolution de l'équation différentielle (27.20) pour $n = 2$ aboutit à :

$$\begin{aligned}
C_2'(t) &= S\mu p_1(t)e^{S\mu t} = (S\mu t)^2 \\
\Longrightarrow C_2(t) &= \frac{1}{2}S^2\mu^2t^2 + A_2 \\
\Longrightarrow p_2(t) &= \left(\frac{1}{2}S^2\mu^2t^2 + A_2\right)e^{-S\mu t}
\end{aligned}$$

De même, en faisant tendre $t \longrightarrow 0$, on démontre que $A_2 = 0$. En poursuivant ce raisonnement, par récurrence, on démontre que :

$$p_n(t) = \frac{(S\mu t)^n}{n!}\, e^{-S\mu t} \tag{27.20}$$

$p_n(t) = \mathbb{P}\left(N(t) = n\right)$ et d'après l'équation (27.20), la variable aléatoire $N(t)$ suit une loi de Poisson de paramètre $S\mu t$. $N$ est aussi appelé *processus de Poisson*. L'espérance de $N$ est égale à : $\mathbb{E}\left(N(t)\right) = S\mu t$. Pendant une durée $T$, le nombre moyen d'ampoules grillées est :

$$\mathbb{E}\left(N(t) - N(t-T)\right) = \mathbb{E}\left(N(T)\right) - \mathbb{E}\left(N(t-T)\right) = S\mu T \tag{27.21}$$

Ce nombre est indépendant du temps $t$.

**Application numérique :**

Pour des ampoules d'une durée de 1000 heures, le paramètre $\mu = \frac{1}{1000}$, $T = 1$. S'il y a deux millions d'ampoules, le nombre moyen d'ampoules grillées par heure est : $S\mu T = 2000$. On retrouve le résultat énoncé.

#### 27.1.2.4 Programme informatique

La durée de vie de chaque ampoule suit une loi exponentielle de paramètre $\mu$. Il faut donc simuler une telle variable dont la fonction de répartition est : $F_\mu(x) = 1 - e^{\mu x}$. On utilise pour cela une propriété qui découle de la fonction de répartition. On note $F_\mu^{-1}(x) = -\frac{\ln(1-x)}{\mu}$. Cette fonction vérifie $F_\mu^{-1}\left(F_\mu(x)\right) = 1$. Or si $U$ est une variable aléatoire uniforme sur $[0,1]$, alors la variable $V = F_\mu^{-1}(U)$ suit la loi exponentielle avec $\mu$ pour paramètre [2]. La première fonction simule une variable exponentielle de paramètre $\mu$ :

```
import math
import random
def generate_expo (mu):
    x = 0
    while x == 0: x = random.random()
    y = - math.log(1-x) / mu
    return y
```

Le module `random` propose aussi une fonction qui simule automatiquement une variable exponentielle :

```
import random
def generate_expo (mu):
    return random.expovariate(mu)
```

Pour réaliser cette simulation, on suppose qu'on a un tableau de $S$ ampoules. Chaque case de ce tableau contient la durée de vie restante d'une ampoule, simulée selon une loi exponentielle. Au départ, toutes les durées de vie sont nulles. On considère qu'à chaque itération, une heure passe. Lors de chaque itération, pour chaque ampoule, on vérifie sa durée de vie restante. Si celle-ci est nulle, on la remplace par une autre dont on choisit aléatoirement la durée de vie (arrondie à l'entier le plus proche). Si la durée de vie n'est pas nulle, on la diminue d'une heure. A chaque itération, on compte le nombre d'ampoules grillées pour en faire la moyenne au bout de $n$ itérations. Pour effectuer cette simulation, les valeurs choisies sont :

$$S = 10000 \quad \mu = \frac{1}{100} \quad n = 500$$

Le programme suivant réalise cette simulation. On calcule la moyenne du nombre d'ampoules grillées par heure sur les 500 itérations excepté la première pour laquelle toutes les ampoules sont grillées - configuration aberrante ou tout du moins très peu probable -. La valeur obtenue est proche de $S\mu = 100$.

```
# -*- coding: cp1252 -*-

import math
import random

def generate_expo (mu):
    return random.expovariate(mu)

S           = 10000
iteration   = 500
mu          = 1.0 / 100

# création d'un tableau de S ampoule qui contient la durée de
# vide restante d'une ampoule
ampoule         = [0 for a in range(0,S)]
moyenne_grille  = 0
for i in range(0,iteration):
```

2. Effectivement : $\mathbb{P}\left(V \leqslant t\right) = \mathbb{P}\left(F_\mu^{-1}(U) \leqslant t\right) = \mathbb{P}\left(U \leqslant F_\mu(t)\right) = F_\mu(x)$. La fonction de répartition de la variable $V$ est $F_\mu$, $V$ est donc une loi exponentielle de paramètre $\mu$.

```
    grille = 0
    mean   = 0
    for n in range(0,S):
        mean += ampoule [n]
        if ampoule[n] == 0:
            # remplacement d'une ampoule grillée
            grille += 1
            # on détermine la durée de vie de cette ampoule
            # on arrondit à l'entier le plus proche
            ampoule [n] = int (generate_expo(mu))
        else :
            # on enlève une heure à la durée de vie de l'ampoule
            ampoule [n] -= 1
    mean /= S
    if i > 0: moyenne_grille += grille
    print "itération : ", i, " moyenne durée : ", mean, " grillées :", grille

moyenne_grille = float (moyenne_grille) / float (iteration - 1)
print "nombre moyen d'ampoules grillées :", moyenne_grille
```

## 27.2 Simulation génétique

### 27.2.1 Enoncé

L'article qui suit est extrait du *Courrier International* daté du Jeudi 16 septembre 2004 qui reprend un article du journal *The Independent*. Il concerne une province finlandaise dont la population est en grande partie issue d'un noyau de 40 à 60 familles qui vivaient là au XVI$^{e}$ siècle. Quelques siècles plus tard, la fréquence de certaines maladies est anormalement élevée.

> "La population de l'est de la Finlande est bien connue pour être malade", note The Independent. Dans cette région isolée, on constate non seulement l'un des plus forts taux de maladies cardio-vasculaires au monde, mais aussi des taux anormalement élevés de diabète, de schizophrénie et de dépression, énumère le quotidien britannique. Alors que la mortalité infantile y est l'une des plus faibles au monde, dès que les gens atteignent la quarantaine, "leur espérance de vie est bien inférieure à celle des autres Européens".
> Cette spécificité régionale intrigue la communauté scientifique depuis plus de vingt ans. Certains croyaient tenir l'explication en pointant les mauvaises habitudes alimentaires des Finlandais, ainsi que leur consommation d'alcool. En effet, "alors qu'en moyenne ils boivent moins que les Français ou les Italiens, par exemple, ils consomment de grandes quantités d'alcool en peu de temps, et ce dans un seul but : se soûler".
> Mais la véritable explication de la mauvaise santé de cette population serait d'ordre génétique. Descendant d'un tout petit groupe de colons installés dans la région au XVIe siècle par le roi de Suède - "les historiens estiment qu'il s'agissait de 40 à 60 familles", précise The Independent -, les habitants de la région ont ensuite pratiqué des mariages consanguins pendant plusieurs centaines d'années. "Cela veut dire que les Finlandais de l'Est possèdent aujourd'hui des traits génétiques extraordinairement homogènes."
> Cette homogénéité "s'est maintenue pendant des siècles parce que très peu de gens ont immigré dans la région". Autrement dit, "l'isolement géographique, religieux et linguistique des Finlandais de l'Est a abouti à un héritage génétique commun" qui en fait "un matériau parfait pour la science", explique le quotidien. Et, de fait, les chercheurs ont déjà identifié 31 gènes différents qui seraient impliqués dans des pathologies caractéristiques de la région. Selon une scientifique de l'université d'Helsinki, chaque habitant est porteur d'au moins un de ces gènes.

Comment pourrait-on reproduire en *Python* cette micro-population ?

L'ensemble des chromosomes d'une personne est réduit à un gêne récessif porteur d'une maladie. Chaque personne est définie par un couple de booléen $(b_1, b_2)$. Si $b_1 = b_2 = true$ alors la personne porteuse de ce couple tombera malade. Si $b_1 \neq b_2$, la personne sera dit porteuse. Dans le dernier cas, la personne sera considérée comme saine. Pour simplifier, on suppose que la population de départ contient 40 couples qui ont à chaque génération un garçon et une fille. Chaque enfant prendra au hasard un gêne de sa mère et un gêne de son père. A la génération suivante, les enfants forment des couples au hasard et ont à leur tour des enfants tandis que leurs parents ont à nouveau des enfants. Chaque couple ne peut avoir d'enfants que pendant deux générations. La population évolue presque (sans tenir compte des malades) comme la suite de Fibonacci [3] :

$$\begin{array}{rcl} u_0 & = & 40 \\ u_1 & = & 80 \\ u_2 & = & 120 = u_1 + u_0 \\ & ... & \\ u_n & = & u_{n-1} + u_{n-2} \end{array}$$

Si on suppose qu'il existait à la première génération, un porteur du gêne récessif, comment évolue par la suite la proportion de malades, de porteurs du gêne récessif et de personnes saines ?

### 27.2.2 Correction

La figure 27.1 présente les résultats d'une simulation. Au début, une seule personne parmi 80 est porteuse du gêne de la maladie. On suppose que les mariages sont consanguins pour la plupart. Selon l'article, ces maladies ne surviennent qu'à un âge avancé. Il est fort probable qu'elles soient longtemps passées inaperçues. Il n'est pas absurde que des personnes malades aient eu des enfants tout autant que les personnes saines.

Les résultats obtenus peuvent être assez variables d'une simulation à l'autre. Ceci est dû au fait que la première génération ne contient qu'un seul porteur du gêne. Ce gêne peut tout-à-fait disparaître après quelques générations ou se répandre très rapidement. Mais après quelques génération, la progression se statibilise. Il serait néanmoins nécessaire d'agréger les résultats de plusieurs simulations pour obtenir des estimateurs fiables de la population finale. Pour cette simulation, à peu près 1,6 % de la population est porteuse de gêne déficient. Cela paraît peu en comparaison des conclusions de l'article mais il faudrait étoffer cette maquette informatique avec 31 gêne et non plus un seul.

3. La suite de Fibonacci (ou Léonard de Pise, son vrai nom) répond au *problème des lapins* : "Possédant au départ un couple de lapins, combien de couples de lapins obtient-on en douze mois si chaque couple engendre tous les mois un nouveau couple à compter du second mois de son existence ?". La réponse débouche sur la suite dite de Fibonacci définie par récurrence : $u_n = u_{n-1} + u_{n-2}$. Au mois $n$, la population est égale à la population du mois précédent $n-1$ et les enfants du mois $n-2$. Ce mathématicien a réuni dans son livre *Liber Abacci*, écrit en 1202, l'ensemble des connaissances mathématiques de l'époque, notamment le signe zéro venu d'Inde à travers l'Orient, secoué par les Croisades. Pendant cette période, les marchands pisans, génois, vénitien, suivirent les soldats et étendirent leur zone d'influence aux ports de la Méditerrannée et de la mer Noire. "Alors qu'il était jeune garçon, son père, qui dirigeait pour le compte de l'Ordre des Marchands de Pise, le bureau des douanes de Bougie en Algérie, le rappela près de lui et lui fit suivre les meilleurs cours sur les méthodes de calcul indo-arabes. C'est ainsi qu'il s'initia aux mathématiques ; lors de ses fréquents voyages professionnels pour le compte des marchands pisans, il rencontrait des mathématiciens en Egypte, en Syrie, en Provence, en Grèce, et en Sicile. Il en relevait au cours de *disputes*, des défis mathématiques et étudiait en profondeur les *Eléments* d'Euclide qu'il a toujours considérés comme un modèle de style et de rigueur logique.", extrait de *Les mathématiciens*, collection Bibliothèque scientifique, Belin pour la science. Son livre servit de référence à l'enseignement des mathématiques pendant près de trois siècles.

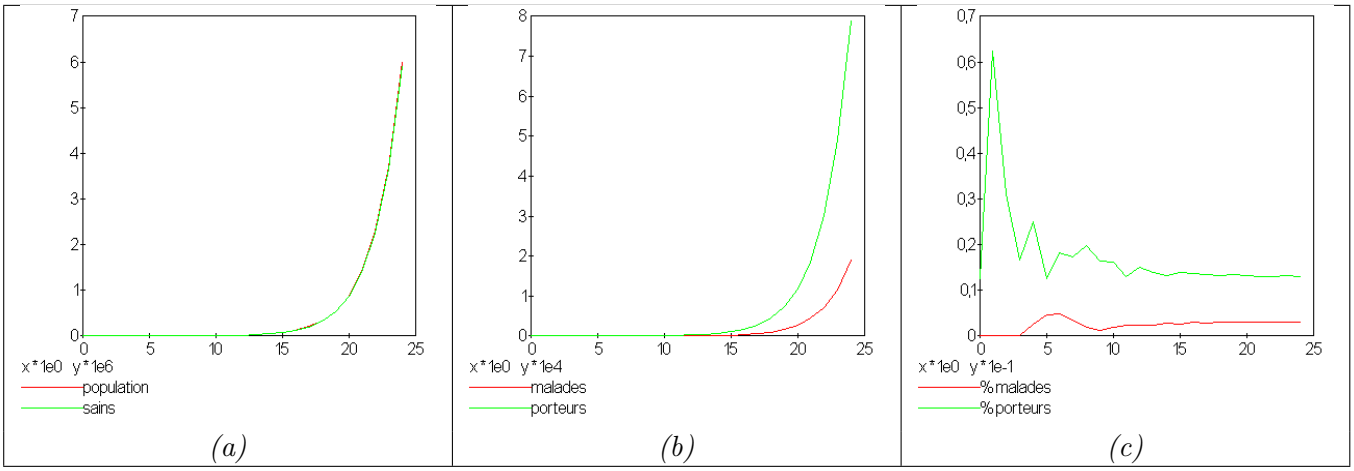



**Figure 27.1 :** *Simulation de la progression du gêne malade. L'image (a) montre l'évolution de la population et des personnes non porteuses du gênes. L'image (b) montre l'évolution des porteurs et des malades. L'image (c) montre l'évolution de ces mêmes populations mais rapportées à la population globale. On suppose que les mariages sont consanguins à 60%, et que les malades ont des enfants pendant deux générations. La première génération contient 80 personnes dont une porteuse du gêne. La population finale est composée de six millions de personnes dont 78710 porteurs du gêne et 18903 malades, soit respectivement 1,3% et 0,3% de la population. Ces deux chiffres semblent se stabiliser.*

## 27.3 Tracer une ligne sans multiplication

### 27.3.1 Enoncé

L'objectif de cet exercice est de dessiner une ligne à l'écran en n'utilisant que des additions et des soustractions.

#### 27.3.1.1 L'algorithme de Bresenham

Ce problème cache deux difficultés. La première consiste à déterminer un ensemble de pixels voisins (voir les figures 27.2 et 27.3) uniformément répartis le long d'une droite. L'ensemble des pixels sélectionnés doit aussi être connexe car il doit être possible de passer d'une extremité à une autre, sautant d'un pixel vers un pixel voisin, sans emprunter un pixel ne faisant pas partie du segment. La seconde difficulté est de se passer des multiplications et des divisions, opérations plus onéreuses en temps de calcul qu'une simple addition ou soustraction. A l'époque où les ordinateurs n'étaient pas aussi rapides qu'aujourd'hui, ce gain avait son importance.

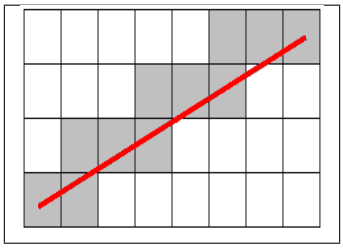


**Figure 27.2 :** *Tracer une ligne à l'écran revient à déterminer un ensemble de pixels voisins recouvrant la droite à dessiner. La ligne de cette figure vérifie la 4-connexité : deux pixels sont considérés comme voisins s'ils ont une arête en commun.*

En regardant la figure 27.3, il paraît assez intuitif comme premirèe approche de calculer le vecteur directeur $(1, m)$ de la droite puis d'en déduire le points de passage du segment pour chaque colonne verticale de pixels : $y = mx + p$. Mais cette méthode utilise une multiplication pour le calcul des ordonnées.

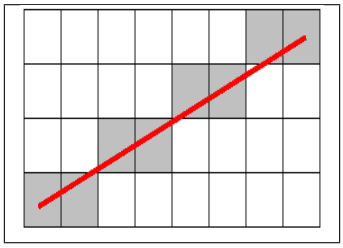


**Figure 27.3 :** *Même ligne que pour la figure 27.2 mais cette ligne vérifie la 8-connexité : deux pixels sont considérés comme voisins s'ils ont un sommet en commun. Par rapport à la ligne de la figure 27.2 des pixels voisins ont disparu car ils ne sont pas nécessaires pour passer du pixel* $(0,0)$ *au pixel* $(7,3)$ *sans perte de 8-connexité. Le segment obtenu est décrit par l'ensemble de pixels suivant :* $[(0,0),(1,0),(2,1),(3,1),(4,2),(5,2),(6,3),(7,3)]$.

L'algorithme de Bresenham (voir [Bresenham1965]) propose une solution s'appuyant uniquement sur des additions et des soustractions. La figure 27.3 montre un segment reliant deux pixels obtenus par l'algorithme qui suit.

**Algorithme 27.2 : tracé d'une ligne (Bresenham)**
On suppose qu'on doit tracer la ligne rejoignant les pixels de coordonnées $(x_1, y_1)$ à $(x_2, y_2)$. On suppose de plus que : $x_2 > x_1$, $y_2 \geqslant y_1$ et $x_2 - x_1 \geqslant y_2 - y_1$. Les autres cas sont obtenus par symétrie par rapport aux axes et aux premières bissectrices. On suppose que $(x_1, y_1, x_2, y_2) \in \mathbb{N}^4$.

**Etape A : initialisation**

$v_x \longleftarrow x_2 - x_1$
$v_y \longleftarrow y_2 - y_1$
$\beta \longleftarrow \frac{v_x}{2}$
$y \longleftarrow y_1$

initialisation

**Etape B : étape principale**

**pour** $x = x_1$ **à** $x_2$ **faire**
    dessiner le point de coordonnées $(x, y)$
    $\beta \longleftarrow \beta - v_y$
    **si** $\beta < 0$ **alors**
        $y \longleftarrow y + 1$
        $\beta \longleftarrow \beta + v_x$
    **fin si**
**fin pour**

On vérifie qu'à tout moment : $\beta = yv_x - xv_y + \frac{v_x}{2}$ et $0 \leqslant \beta < v_x$. Ceci implique :

$$\begin{array}{ccccc} -\frac{v_x}{2} & \leqslant & yv_x - xv_y & < & \frac{v_x}{2} \\ -\frac{1}{2} & \leqslant & y - x\frac{v_y}{v_x} & < & \frac{1}{2} \end{array}$$

Autrement dit, le pixel choisi pour représenter la droite n'est jamais éloigné de plus d'un demi-pixel de la vraie droite. L'objectif de cet exercice est d'implémenter une fonction qui utilise l'algorithme 27.2 pour retourner la liste des pixels par lesquels passe une droite reliant les pixels $(x_1, y_1)$ et $(x_2, y_2)$ en 8-connexité.

Toutefois avant de passer à la partie programmation, il peut être utile de s'interroger sur la valeur de la constante choisie comme première valeur de la variable $\beta$. La figure 27.4 montre la même ligne que celle de la figure 27.3 mais obtenue pour deux autres valeurs d'initialisation de la variable $\beta$. De manière imagée, lorsque la valeur d'initialisation choisie pour $\beta$ est $\frac{v_x}{2}$, le segment passe par le milieu des pixels.

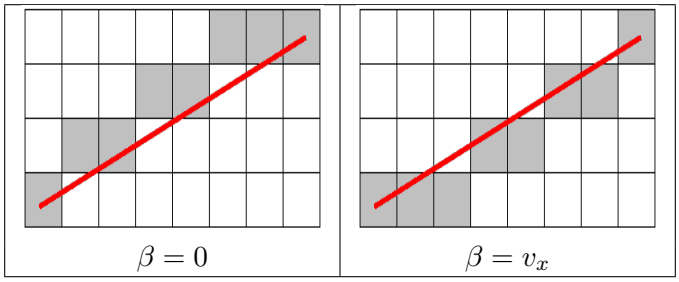



**Figure 27.4 :** *Ces deux lignes sont obtenus pour deux autres valeurs d'initialisation de la variable $\beta$. Pour la première image, $\beta = 0$ et l'ensemble des pixels est légèrement décalé vers le haut. Lorsque $\beta = v_x$, l'ensemble des pixels est légèrement décalé vers le bas. Pour la figure 27.3, $\beta = \frac{v_x}{2}$.*

#### 27.3.1.2 Extension aux ellipses

Bresenham a également décliné une version de cet algorithme pour le tracé des arcs circulaires (voir [Bresenham1977]). Ce paragraphe présente une version de cet algorithme qui permet de tracer une ellipse dont les axes sont confondues avec les axes des abscisses et des ordonnées. L'équation cartésienne d'une ellipse de demi-axe $a$ et $b$ est :

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{27.22}$$

A partir de cette équation, on en déduit l'équation de la tangente au point $(x_0, y_0)$ :

$$x\ \frac{2x_0}{a^2} + y\ \frac{2y_0}{b^2} = 0 \Longleftrightarrow x\ x_0\ b^2 + y\ y_0\ a^2 = 0 \tag{27.23}$$

L'algorithme qui suit permet de tracer sans multiplication ni division une ellipse dont l'équation (27.22).

**Algorithme 27.3 : tracé d'une ellipse (Bresenham)**
On suppose qu'on doit tracer l'ellipse d'équation $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$. On suppose que $a \neq 0$ et $b \neq 0$ auquel cas, les ellipses sont des segments. On suppose $(a, b) \in \mathbb{N}^2$.
**Etape A : initialisation**

$A \longleftarrow a^2$
$B \longleftarrow b^2$
$x \longleftarrow a$
$y \longleftarrow 0$
$v_x \longleftarrow aB$
$v_y \longleftarrow 0$
$\beta \longleftarrow \frac{v_x}{2}$

Le point $(x, y)$ est la position du pixel qui fera partie de l'ellipse. Le vecteur $(v_x, v_y)$ représente le vecteur normal à l'ellipse au point $(x, y)$. Ce vecteur est constamment égal à $(v_x, v_y) = (xB, yA) = \left(xb^2, ya^2\right)$.

**Etape B : tracé du premier huitième**
**tant que** $(v_x \geqslant v_y)$ **faire**
dessiner le point de coordonnées $(x, y)$
$y \longleftarrow y + 1$
$v_y \longleftarrow v_y + A$
$\beta \longleftarrow \beta - v_y$
**si** $\beta < 0$ **alors**
$x \longleftarrow x - 1$
$v_x \longleftarrow v_x - B$
$\beta \longleftarrow \beta + v_x$
**fin si**
**fin tant que**

**Etape C : tracé du second huitième**
**tant que** $(x \geqslant 0)$ **faire**
dessiner le point de coordonnées $(x, y)$
$x \longleftarrow x - 1$
$v_x \longleftarrow v_x - B$
$\beta \longleftarrow \beta + v_x$
**si** $\beta > 0$ **alors**
$y \longleftarrow y + 1$
$v_y \longleftarrow v_y + A$
$\beta \longleftarrow \beta - v_y$
**fin si**
**fin tant que**

**Etape D : autres huitièmes**
Le reste de l'ellipse est obtenu par symétrie par rapport aux deux axes du répère orthonormé du plan.

Avant de démontrer pourquoi le tracé de l'ellipse est bien obtenu, on vérifie que dès que $x$ ou $y$ sont modifiés, $v_x$ et $v_y$ sont également modifiés de sorte que $(v_x, v_y) = \left(xb^2, ya^2\right)$. La figure 27.5 explique pourquoi l'algorithme contient deux étapes de tracé. L'étape B incrémente $y$ à chaque itération. Comme $x$ ne peut décroître plus vite que $y$ ne croît, cette étape ne peut tracer des courbes dont les tangentes sont plus verticales qu'horizontales (voir figure 27.6). C'est pourquoi on ajoute une seconde étape C symétrique de la première.

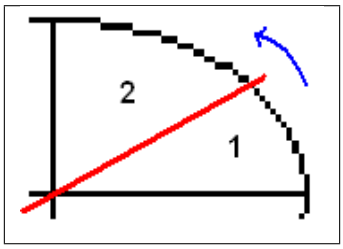



**Figure 27.5 :** *Algorithme 29.2, les deux premiers huitièmes sont scindés en deux parties. La partie 1 pour laquelle $v_x > v_y$, c'est-à-dire qu'on se déplace plus souvent vers le haut que vers la gauche. $y$ croît à chaque itération de l'étape B tandis que $x$ ne décroît qu'occasionnellement. C'est l'inverse pour la partie 2, $v_x < v_y$ et $x$ décroît à chaque itération de l'étape C tandis que $y$ ne croît qu'occasionnellement.*

La première partie de l'ellipse (zone 1 de la figure 27.5), la normale à l'ellipse dirigée par le vecteur $(v_x, v_y)$ vérifie $v_x > v_y$. Ce cas est traité par l'étape B de l'algorithme 29.2. On réécrit cette étape en fonction d'un indice $t$ égal au nombre d'itérations :

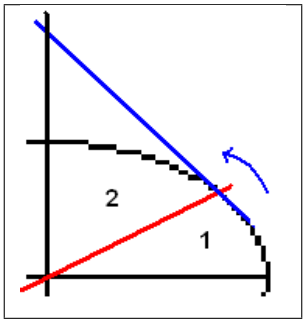



**Figure 27.6 :** *Algorithme 29.2, si le premier quart était tracé par l'étape B seule, le tracé suivrait bien l'ellipse pour commencer puis suivrait ensuite la droite bleue de vecteur directeur* $(-1,1)$ *et tangente à l'ellipse.*

$$y(t+1) = y(t) + 1 \text{ et } y(0) = 0 \tag{27.24}$$

$$v_y(t+1) = v_y(t) + a^2 \text{ et } v_y(0) = 0 \tag{27.25}$$

$$x(t+1) = \begin{cases} x(t) & \text{si } \beta(t) - v_y(t+1) \geqslant 0 \\ x(t) - 1 & \text{sinon} \end{cases} \tag{27.26}$$

$$v_x(t+1) = \begin{cases} v_x(t) & \text{si } \beta(t) - v_y(t+1) \geqslant 0 \\ v_x(t) - b^2 & \text{sinon} \end{cases} \text{ et } v_x(0) = a\, b^2 \tag{27.27}$$

$$\beta(t+1) = \begin{cases} \beta(t) - v_y(t+1) & \text{si } \beta(t) - v_y(t+1) \geqslant 0 \\ \beta(t) - v_y(t+1) + v_x(t+1) & \text{sinon} \end{cases} \text{ et } \beta(0) = \frac{v_x(0)}{2} \tag{27.28}$$

Tout d'abord, les équations (27.25) et (27.27) nous permettent de vérifier que :

$$\forall t,\ v_x(t) = b^2 x(t) \text{ et } v_y(t) = y(t)\, a^2$$

De plus, on sait qu'à tout instant :

$$\begin{aligned} & 0 \leqslant \beta(t) \leqslant v_x(t+1) \\ \Longrightarrow\ & 0 \leqslant \beta(t) \leqslant b^2 x(t) \end{aligned} \tag{27.29}$$

Tout repose sur la variable $\beta$. L'objectif des lignes qui suivent est de comprendre ce que $\beta$ représente. Les équations (27.24) à (27.28) permettent d'écrire que :

$$\begin{aligned} \beta(t+1) &= \beta(t) - v_y(t+1)\,[y(t+1) - y(t)] - v_x(t+1)\,[x(t+1) - x(t)] \\ \Longleftrightarrow \beta(t+1) - \beta(t) &= -y(t+1)a^2\,[y(t+1) - y(t)] - x(t+1)b^2\,[x(t+1) - x(t)] \\ \Longleftrightarrow \beta(t+1) - \beta(t) &= \begin{pmatrix} x(t+1)b^2 \\ y(t+1)a^2 \end{pmatrix} \cdot \begin{pmatrix} x(t+1) - x(t) \\ y(t+1) - y(t) \end{pmatrix} \end{aligned} \tag{27.30}$$

L'équation (27.30) permet d'affirmer que :

$$\beta(t+1) - \beta(0) = \sum_{i=1}^{t+1} \begin{pmatrix} x(i+1)b^2 \\ y(i+1)a^2 \end{pmatrix} \cdot \begin{pmatrix} x(i+1) - x(i) \\ y(i+1) - y(i) \end{pmatrix} \tag{27.31}$$

On note le point $M(t)$ de coordonnées $M(t) = (x(t), y(t))$ et $\overrightarrow{dM(t+1)}$ le vecteur $\overrightarrow{dM(t+1)} = \overrightarrow{OM(t+1)} - \overrightarrow{OM(t)} = (x(t+1) - x(t),\ y(t+1) - y(t))$. L'équation (27.31) peut s'écrire sous la forme :

$$\beta(t+1) - \beta(0) = \sum_{i=1}^{t+1} \begin{pmatrix} x(i+1)b^2 \\ y(i+1)a^2 \end{pmatrix} \cdot \overrightarrow{dM(t+1)} \tag{27.32}$$

On définit maintenant la fonction $f$ par :

$$f(x,y) = \frac{b^2x^2 + a^2y^2}{2} \tag{27.33}$$

L'ellipse correspond à l'ensemble $\left\{(x,y) \in \mathbb{R}^2 \mid f(x,y) = \frac{a^2b^2}{2}\right\}$. On remarque également que :

$$\begin{aligned} \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) &= (b^2x, a^2y) \\ \overrightarrow{N(t+1)} = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)(x(t), y(t)) &= \big(b^2x(t+1), a^2y(t+1)\big) \end{aligned} \tag{27.34}$$

$\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$ est un vecteur orthonogonal à la courbe de niveau de $f$ passant par le point $(x,y)$. En regroupant les équations (27.32) et (27.34), on peut écrire que :

$$\beta(t+1) - \beta(0) = \sum_{i=1}^{t+1} \overrightarrow{N(t+1)} \cdot \overrightarrow{dM(t+1)} \tag{27.35}$$

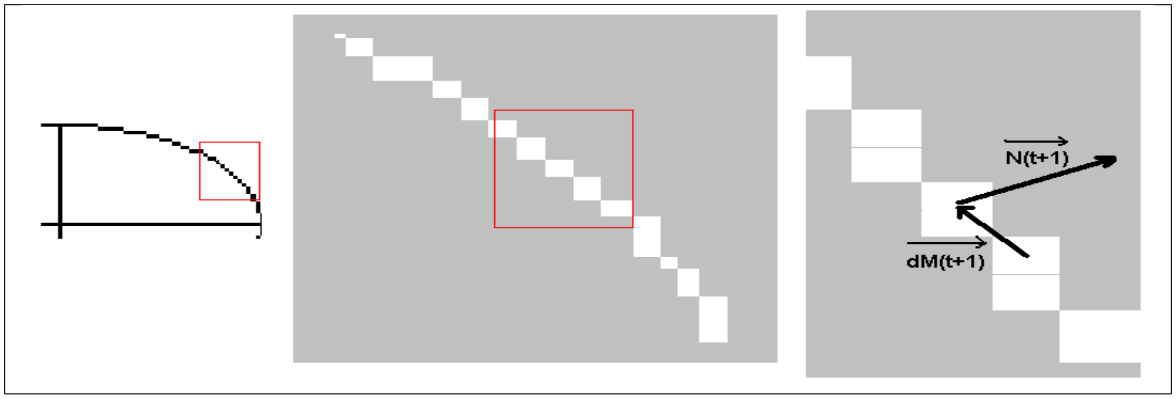



**Figure 27.7 :** *La seconde image est un grossissement d'une partie de la première image. La troisième image est aussi un grossissement de la seconde où sont représentés le vecteur $dM(t+1)$ et le vecteur normal $N(t+1)$ orthognonal à la courbe de niveau de la fonction $f$ passant par le point $M(t+1)$. $\beta(t+1)$ est la somme des produits scalaires entre la normale $N(t+1)$ et des petits déplacements $dM(t+1)$.*

La figure 27.7 représente les vecteurs $\overrightarrow{N(t+1)}$ et $\overrightarrow{dM(t+1)}$. $\beta(t)$ est en quelques sorte la somme des écarts à la normale. L'inéquation (27.29) devient :

$$\begin{aligned} (27.29) &\iff -\beta(0) \leqslant \beta(t+1) - \beta(0) \leqslant b^2x(t+1) - \beta(0) \\ &\iff -\frac{b^2x(t+1)}{2} \leqslant \beta(t+1) - \beta(0) \leqslant \frac{b^2x(t+1)}{2} \\ &\iff -\frac{1}{2} \leqslant \frac{\beta(t+1) - \beta(0)}{b^2x(t+1)} \leqslant \frac{1}{2} \end{aligned} \tag{27.36}$$

Dans le cas du tracé d'une droite (algorithme 27.2), le vecteur $\overrightarrow{N(t+1)}$ est constant et l'équation (27.35) peut être simplifiée. Pour une ellipse, ce vecteur est différent à chaque nouvelle position $M(t+1)$.

On sait que $\overrightarrow{N(t+1)} = \begin{pmatrix} x(t+1)b^2 \\ y(t+1)a^2 \end{pmatrix}$. De plus pour la première partie de la courbe (zone 1 de la figure 27.5), on sait que $x(t+1)b^2 > y(t+1)a^2$. On en déduit que :

$$\left\| \frac{\overrightarrow{N(t+1)}}{x(t+1)b^2} \right\| \leqslant 1 \tag{27.37}$$

On définit $\alpha(t+1) = \overrightarrow{dM(t+1)}.\dfrac{\overrightarrow{N(t+1)}}{x(t+1)b^2}$. L'inégalité (27.37) signifie que pour un déplacement $\overrightarrow{dM(t+1)}$, le nombre $\alpha(t+1)$ sous-estime le nombre de pixels dont on se rapproche ou dont on s'éloigne de la courbe de l'ellipse. En additionnant ces petits déplacements, on peut considérer que $\dfrac{\beta(t+1)}{x(t+1)b^2}$ est une approximation de la distance en pixels entre le points $M(t+1)$ et la courbe de l'ellipse. Et d'après l'inégalité (27.36), cette distance est toujours inférieure à un demi-pixel.

La conclusion de cette démonstration n'est pas rigoureuse mais elle explique l'étape B de l'algorithme 29.2 et le rôle de la variable $\beta(t+1)$. Lors de l'étape C, les rôles de $x$ et $y$ sont inversées. Il est facile d'adapter la démonstration précédente pour démontrer que les centres des pixels choisis pour représenter l'ellipse ne sont pas distants du plus d'un demi-pixel de la courbe de l'ellipse lors de l'étape C

Le terme $\beta(0)$ est choisi selon le même raisonnement que celui développé dans le cas d'une droite (voir figure 27.4). La figure 27.8 reprend cette expérience dans le cas d'un cercle. Les différentes valeurs déplacent dans un sens ou dans l'autre les bornes de l'encadrement (27.36) de $\beta(t+1) - \beta(0)$ selon que l'on souhaite que l'ellipse passe par le centre des pixels ou un sommet (voir figure 27.9).

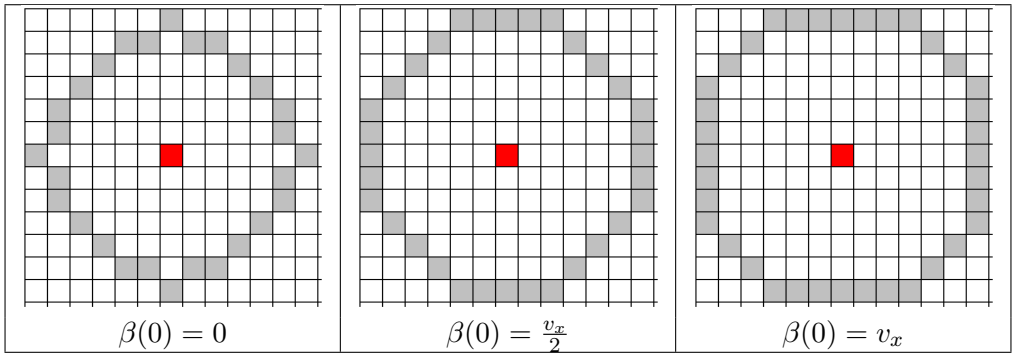



**Figure 27.8 :** *Ces trois ellipses sont obtenus pour trois valeurs différentes d'initialisation de la variable $\beta$. Pour la première image, $\beta = 0$ et l'ensemble des pixels est légèrement décalé vers le centre du cercle. Lorsque $\beta = v_x$, l'ensemble des pixels est légèrement décalé vers l'extérieur. Le cercle le plus vraisemblant est obtenu pour $\beta = \frac{v_x}{2}$.*

#### 27.3.1.3 Extension aux coniques

Le paragraphe précédent a montré comment l'algorithme 29.2 a été conçu et pourquoi le tracé de l'ellipse ne s'éloigne pas de plus d'un demi-pixel de la courbe. Il serait possible de concevoir un algorithme similaire pour le tracé de toute courbe pour laquelle les coordonnées du vecteur normal dépendent linéairement de $(x, y)$. Il est donc possible de représenter n'importe quelle conique par un ensemble de pixels sans multiplication ni division (voir [Bresenham1977]).

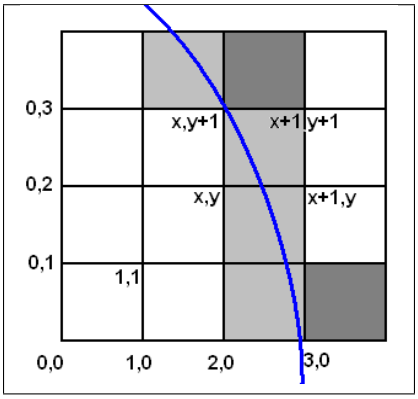



**Figure 27.9 :** *Les pixels choisis pour représenter la courbe de l'ellipse sont ceux (couleur gris clair) dont le centre est le plus près de cette courbe et non ceux (gris foncé) dont le coin inférieur gauche est le plus près de cette courbe. C'est pour cela qu'on définit* $\beta(0) = \frac{v_x(0)}{2}$ *et non* $\beta(0) = 0$.

### 27.3.2 Correction

La fonction qui permet de tracer une ligne se décompose en deux fonctions. La première partie `trace_ligne_simple` permet de tracer les lignes pour lesquels $x_1 < x_2$ et $y_1 < y_2$. La seconde fonction `trace_ligne` trace les lignes pour tous les cas. Dans un premier temps, cette fonction échange les couples $(x_1, x_2)$ et/ou $(y_1, y_2)$ pour se ramener à un cas accepté par la fonction `trace_ligne_simple`. Dans un second temps, à partir du résultat obtenu à cette première étape, elle en déduit la ligne pour le cas initial.

```
# -*- coding: cp1252 -*-
"""ce module contient la fonction trace_ligne qui retourne l'ensemble des pixels
concernés par le tracé d'une ligne en 8-connexité entre deux pixels"""
import pygame                 # pour les affichages
import random

def trace_ligne_simple (x1,y1,x2,y2):
    """trace une ligne entre les points de coordonnées (x1,y1) et (x2,y2),
    on suppose que x2 > x1, y2 >= y1,
    retourne la ligne sous la forme d'un ensemble de pixels (x,y)"""

    if y2 - y1 <= x2 - x1 :  # droite en dessous de la première bissectrice
        vx = x2 - x1
        vy = y2 - y1
        b  = vx / 2
        y  = y1
        x  = x1

        ligne = []
        while x <= x2 :
            ligne.append ((x,y))
            b -= vy
            x += 1
            if b < 0:
                b += vx
                y += 1
        return ligne
    else :                    # droite au dessus de la première bissectrice
        vx = x2 - x1
        vy = y2 - y1
        b  = vy / 2
        y  = y1
        x  = x1

        ligne = []
```

```
        while y <= y2 :
            ligne.append ((x,y))
            b -= vx
            y += 1
            if b < 0:
                b += vy
                x += 1
        return ligne

def trace_ligne (x1,y1,x2,y2):
    """trace une ligne entre les points de coordonnées (x1,y1) et (x2,y2),
    aucune contrainte sur les coordonnées,
    retourne la ligne sous la forme d'un ensemble de pixels (x,y)"""

    if x1 == x2 :
        if y1 <= y2: return [ (x1, i) for i in xrange (y1,y2+1) ]
        else : return [ (x1, i) for i in xrange (y2,y1+1) ]

    if y1 == y2 :
        if x1 <= x2: return [ (i, y1) for i in xrange (x1,x2+1) ]
        else : return [ (i, y1) for i in xrange (x2,x1+1) ]

    if x1 < x2 :
        if y1 < y2 :
            return trace_ligne_simple (x1,y1,x2,y2)
        else :
            ligne = trace_ligne_simple (x1,y2,x2,y1)
            return [ (x,y1 + y2 - y) for (x,y) in ligne ]

    if x2 < x1 :
        if y1 < y2 :
            ligne = trace_ligne_simple (x2,y1,x1,y2)
            return [ (x1 + x2 - x, y) for (x,y) in ligne ]
        else :
            ligne = trace_ligne_simple (x2,y2,x1,y1)
            return [ (x1 + x2 - x, y1 + y2 - y) for (x,y) in ligne ]

def display_ligne (ligne, screen):
    """affiche une ligne à l'écran"""
    color = 0,0,0
    for p in ligne:
        pygame.draw.line (screen, color, p,p)
    pygame.display.flip ()


def attendre_clic (screen):
    """attend la pression d'un clic de souris pour continuer"""
    reste = True
    while reste:
        for event in pygame.event.get():
            if event.type == pygame.MOUSEBUTTONUP :
                reste = False
                break

if __name__ == "__main__" :
    pygame.init ()

    size      = width, height = x,y = 200, 200
    black     = 0, 0, 0
    white     = 255,255,255
    screen    = pygame.display.set_mode(size)
    screen.fill (white)

    print trace_ligne (0,0, 7,3)
    # affiche [(0, 0), (1, 0), (2, 1), (3, 1), (4, 2), (5, 2), (6, 3), (7, 3)]
```

```
for n in xrange (0,10):
    x1      = random.randint (0,x-1)
    y1      = random.randint (0,y-1)
    x2      = random.randint (0,x-1)
    y2      = random.randint (0,y-1)
    ligne   = trace_ligne (x1,y1,x2,y2)
    display_ligne (ligne, screen)

attendre_clic (screen)
```

L'algorithme 29.2 décrivant le tracé d'une ellipse est le sujet du TD 5.

## 27.4 Simulation d'une corde

### 27.4.1 Enoncé

#### 27.4.1.1 Equation théorique de la corde en équilibre

On cherche dans cette partie à calculer l'équation de la courbe $y = f(x)$ correspondant à une corde suspendue comme celle de la figure 27.10. On suppose que la corde a pour masse $m$ et pour longueur $L$. Elle est attachée aux points de coordonnées $(x_1, y_1)$ et $(x_2, y_2)$. La corde est modélisée par une infinité de billes de masse $dm = \frac{m}{L}\ dl$ reliées par des segments de longueur $dl$. Trois forces s'exercent sur chaque masse, le poids et deux tensions dues aux voisins.

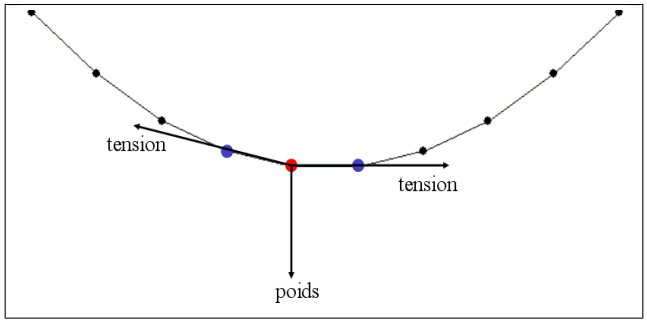



**Figure 27.10 :** *Dessin d'une corde suspendue entre ses deux extrémités, cette corde est modélisée par un ensemble de masses ponctuelles reliées par des morceaux de corde sans masse. Trois forces s'exercent sur chaque masse, son poids et les deux tensions dues à ses deux voisins.*

A l'équilibre, la somme des forces qui s'exercent sur une masse est nulle. Le poids est égale à $gdm$ où $g$ est la constante de gravitation. Les deux tensions sont colinéaires aux segments de cordes reliés de part et d'autre de la masse ponctuelle $dm$. La tension a pour amplitude $T$ et sa projection sur l'axe des abscisses est constante tout le long de la corde.

$$\begin{pmatrix} 0 \\ -gdm \end{pmatrix} + \frac{T}{dx}\begin{pmatrix} -dx \\ f(x-dx) - f(x) \end{pmatrix} + \frac{T}{dx}\begin{pmatrix} dx \\ f(x+dx) - f(x) \end{pmatrix} = 0 \tag{27.38}$$

On peut approcher $f(x+dx) - f(x)$ par $f'(x+dx)dx$. De même, $f(x-dx) - f(x) = -f'(x)dx$. Cette remarque permet de transformer la relation (27.38) pour obtenir la relation suivante :

$$-\frac{gm}{L}\ dl + \frac{T}{dx}\left(f'(x+dx)dx - f'(x)dx\right) = 0 \tag{27.39}$$

Comme $dl = \sqrt{(dx)^2 + (dy)^2} = dx\sqrt{1 + f'^2(x)}$, on obtient :

$$-\frac{gm}{L}\sqrt{1 + f'^2(x)}dx + Tf''(x)dx = 0 \tag{27.40}$$

On en déduit que :

$$f''(x) = \frac{gm}{TL}\sqrt{1 + f'^2(x)} \tag{27.41}$$

On pose $\alpha = \frac{gm}{TL}$, $f'$ est solution de l'équation différentielle $g' = \alpha\sqrt{1 + g^2}$. $g$ est également solution de l'équation $g'^2 = \alpha^2\left(1 + g^2\right)$. En dérivant, on obtient :

$$g'' = \alpha^2 g \Longrightarrow g(x) = Ae^{\alpha x} + Be^{-\alpha x} \tag{27.42}$$

Il reste à vérifier à quelle condition sur les constantes $A$ et $B$, la fonction $g$ est solution de l'équation $g'^2 = \alpha^2\left(1 + g^2\right)$. Ceci donne :

$$\alpha^2\left[A^2e^{2\alpha x} + B^2e^{-2\alpha x} - 2AB\right] = \alpha^2\left[1 + A^2e^{2\alpha x} + B^2e^{-2\alpha x} + 2AB\right] \tag{27.43}$$

Par conséquent :

$$4AB + 1 = 0 \tag{27.44}$$

On en déduit que $g(x) = Ae^{\alpha x} - \frac{1}{4A}e^{-\alpha x}$ et :

$$f(x) = \frac{1}{\alpha}\left[Ae^{\alpha x} + \frac{1}{4A}e^{-\alpha x}\right] + C \tag{27.45}$$

On peut transformer cette expression, comme $A > 0$, on pose $x_0 = \ln 2A \Longleftrightarrow 2A = e^{x_0}$. On note également $x \longrightarrow \cosh(x)$ la fonction cosinus hyperbolique.

$$\begin{aligned} f(x) &= \frac{1}{2\alpha}\left[2Ae^{\alpha x} + \frac{1}{2A}e^{-\alpha x}\right] + C \\ f(x) &= \frac{1}{2\alpha}\left[e^{\alpha x + x_0} + e^{-\alpha x - x_0}\right] + C \\ f(x) &= \frac{1}{\alpha}\cosh\left(\alpha x + x_0\right) + C \end{aligned} \tag{27.46}$$

Il reste à déterminer les constantes $x_0$, $C$ et $T$ à l'aide des conditions initiales :

$$\left\{\begin{array}{lcl} f(x_1) & = & y_1 \\ f(x_2) & = & y_2 \\ \int_{x_1}^{x_2}\sqrt{1 + f'^2(x)}dx & = & L \end{array}\right. \tag{27.47}$$

Cette dernière condition signifie que la longueur de la corde est $L$. Comme $f'$ vérifie l'équation (27.41), on peut simplifier cette dernière contrainte par $\int_{x_1}^{x_2} f''(x)dx = \alpha L \iff f'(x_2) - f'(x_1) = \alpha L$. Le système (27.47) devient :

$$\left\{ \begin{array}{lcl} f(x_1) & = & y_1 \\ f(x_2) & = & y_2 \\ f'(x_2) - f'(x_1) & = & \alpha L \end{array} \right. \tag{27.48}$$

#### 27.4.1.2 Solution dans un cas particulier

On résoud le système (27.48) dans un cas particulier pour lequel $x_1 = -x_2$, $x_2 > 0$ et $y_1 = y_2 = 0$. La fonction $f$ est donc paire et $\forall x,\ f(-x) = f(x)$. On en déduit que $x_0 = 0$.

Par conséquent, $f(x) = \frac{1}{\alpha}\ \cosh(\alpha x) + C$. Comme $f(x_1) = f(x_2) = 0$, la constance $C$ vaut $C = -\frac{1}{\alpha}\cosh(\alpha x_1)$. Il reste à déterminer la valeur du paramètre $\alpha$ qui englobe la constante $T$. Comme $f'(x) = \sinh(\alpha x)$, la dernière condition du système (27.48) équivaut à :

$$\begin{aligned} \sinh(\alpha x_2) - \sinh(\alpha x_1) &= \alpha L \\ \iff 2\sinh(\alpha x_2) &= \alpha L \\ \iff \frac{1}{\alpha}\sinh(\alpha x_2) &= \frac{L}{2} \end{aligned} \tag{27.49}$$

Cette dernière équation n'est pas évidente à résoudre. On peut toutefois remarquer que la fonction $h(\alpha) = \frac{1}{\alpha}\sinh(\alpha x_2)$ est croissante [4]. Il est possible de déterminer numériquement une valeur approchée pour $\alpha$ en utilisant par exemple un algorithme de recherche par dichotomie.

#### 27.4.1.3 Simulation informatique

On cherche ici à retrouver ce résultat théorique à l'aide d'une simulation informatique. La corde est représentée par une suite de $n$ masses ponctuelles reliées par des ressorts de masse nulle qui exercent une traction lorsque leur longueur dépasse un certain seuil. On considère une corde de longueur $L$ et de masse $M$, elle est divisée en $n$ masses de masses $\frac{M}{n}$. Ces masses sont reliés par $n-1$ ressorts qui exercent une traction sur les masses qu'ils relient lorsque leur longueur dépasse le seuil $\frac{L}{n-1}$. Les masses sont initialement placées sur une droite reliant les deux extremités fixes de la corde.

Chaque masse est soumis à une accélération égale à la somme de trois forces qui sont son poids et les tractions exercées par les ressorts auxquels elle est attachée. A chaque instant $t$, on peut calculer cette accélération, en déduire la vitesse puis la position à l'instant $t + dt$. Ce mécanisme permet d'obtenir une animation de la corde atteignant sa position d'équilibre (voir figure 27.11). On construit l'algorithme suivant.

4. On vérifie que la dérivée seconde est positive sur $\mathbb{R}^+$ et que la dérivée est nulle à l'origine, donc croissante et positive.

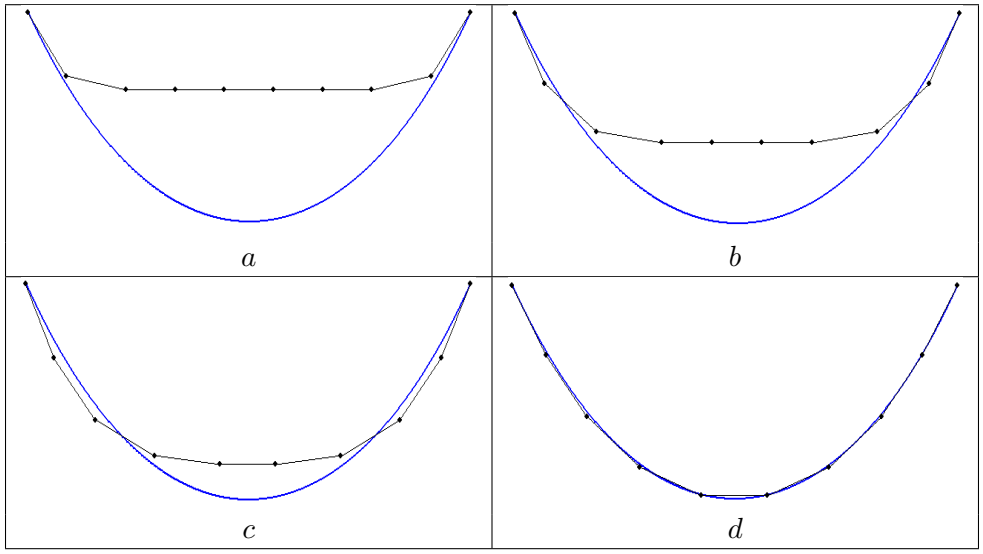



**Figure 27.11 :** *Dessin d'une corde suspendue entre ses deux extrémités, cette corde est modélisée par un ensemble de masses ponctuelles reliées par des morceaux de corde élastiques sans masse. Ces quatre images a, b, c, d représentent la corde lorsqu'on la laisse tomber. La courbe bleue est la courbe théorique de la corde calculée au paragraphe 27.4.1.2. Ces images ont été obtenues avec l'algorithme 27.4.*

**Algorithme 27.4 : mouvement d'une corde**

Soit une corde de longueur $L$ suspendue entre les points d'abscisse $(-x_0, 0)$ et $(x_0, 0)$ de telle sorte que $2x_0 < L$. Cette corde à une masse $M$ et une raideur $k$. La pesanteur est notée $g$. On divise cette corde en $n$ masses ponctuelles de masse $\forall i \in \{1, \ldots, n\}$, $m_i = \frac{M}{n}$. Chaque masse a une position $p_i$. On note pour chaque point $p_i$ sa vitesse $v_i$. On désigne par $f$ un coefficient de freinage, plus cette vitesse est grande, plus la corde convergera rapidement vers sa position d'équilibre. $dt$ désigne un pas de temps.

**Etape A : initialisation**

$\forall i \in \{1, \ldots, n\}$, $p_i = -x_0 + 2x_0 \frac{i-1}{n-1}$

**Etape B : calcul des forces**

Pour chaque masse, on calcule une accélération $a_i \in \mathbb{R}^2$. $a_0 = a_n = 0$ (les extrémités sont fixes).

**pour** $i = 1$ **à** $n-1$ **faire**
  $a_i \longleftarrow (0, -m_i g) - f v_i$
  **si** $\|\overrightarrow{p_{i-1}, p_i}\| > \frac{L}{n-1}$ **alors**
    $a_i \longleftarrow a_i + k \left[ \|\overrightarrow{p_{i-1}, p_i}\| - \frac{L}{n-1} \right] \frac{\overrightarrow{p_i p_{i-1}}}{\|\overrightarrow{p_{i-1}, p_i}\|}$
  **fin si**
  **si** $\|\overrightarrow{p_{i+1}, p_i}\| > \frac{L}{n-1}$ **alors**
    $a_i \longleftarrow a_i + k \left[ \|\overrightarrow{p_{i+1}, p_i}\| - \frac{L}{n-1} \right] \frac{\overrightarrow{p_i p_{i+1}}}{\|\overrightarrow{p_{i+1}, p_i}\|}$
  **fin si**
**fin pour**

**Etape C : mise à jour des positions**

**pour** $i = 1$ **à** $n-1$ **faire**
  $p_i \longleftarrow p_i + v_i dt$
  $v_i \longleftarrow v_i + a_i dt$
**fin pour**

**Etape D : terminaison**
Tant que l'algorithme n'a pas convergé, on retourne à l'étape B.

Cet algorithme reproduit la chute libre de masses ponctuelles reliées par des ressorts. Ce type de modèle est parfois utilisé pour reproduire le comportement d'un vêtement autour d'un corps en mouvement. Chaque masse n'est plus reliée à deux voisins mais à quatre voisins.

### 27.4.2 Correction

Le programme qui suit implémente l'algorithme 27.4. La classe `point` définit ce qu'est un point dans plan. La classe `corde` définit ce qu'est une corde. Le constructeur reproduit l'étape A d'initialisation. La méthode `force_point` calcule les forces qui s'appliquent à chaque masse ponctuelle, c'est-à-dire l'étape B. La méthode `iteration` met à jour les positions de chaque masse selon les formules décrites lors de l'étape C.

La méthode `display` affiche la corde, elle appelle également la méthode `display_courbe_theorique` qui dessine la courbe théorique de la corde (voir paragraphe 27.4.1.2) lorsque les extrémités ont même ordonnée. La dernière méthode `__str__` permet d'écrire entre autres les positions de chaque masse, la distance entre chaque masse, la distance attendue.

La méthode `display_courbe_theorique` nécessite la détermination du paramètre $\alpha$ qui vérifient l'équation $\frac{1}{\alpha}\sinh(\alpha x_2) - \frac{L}{2} = 0$ (voir paragraphe 27.4.1.2). Cette détermination est faite grâce à l'algorithme qui suit. Il s'appuie sur le faite que la fonction $\alpha \longleftarrow \frac{1}{\alpha}\sinh(\alpha x_2) - \frac{L}{2}$ est croissante pour $x_2 > 0$.

**Algorithme 27.5 : recherche dichotomique**
Soit $f$ une fonction croisante, soient $a, b \in \mathbb{R}$ tels que $a < b$ et $f(a) \leqslant 0 \leqslant f(b)$. On cherche à approcher $x \in [a, b]$ tel que $f(x) = 0$. Soit $s \in \mathbb{R}_+^*$ une petite valeur. $x \longleftarrow \frac{a+b}{2}$
**tant que** $(|f(x)| > s)$ **faire**
    **si** $f(x) > 0$ **alors**
        $b \longleftarrow x$
    **sinon**
        $a \longleftarrow x$
    **fin si**
    $x \longleftarrow \frac{a+b}{2}$
**fin tant que**

Cet algorithme divise l'intervalle de recherche en deux à chaque itération.

```
# -*- coding: cp1252 -*-
import pygame                 # pour les affichages
import math                   # pour les racines carrées


class point (object) :
    """définition d'un point : deux coordonnées et une masse"""
    __slots__ = "x","y","m"

    def __init__ (self, x,y,m) :
        """définit un point de la corde, de coordonnées (x,y)
        et de masse m"""
        self.x, self.y, self.m = float (x), float (y), float (m)

    def deplace (self, dep, dt) :
        """déplace un point"""
        self.x += dep [0] * dt
```

```
        self.y += dep [1] * dt

    def deplace_point (self, dep, dt) :
        """déplace un point"""
        self.x += dep.x * dt
        self.y += dep.y * dt

    def difference (self, p) :
        """retourne le vecteur qui relie deux points,
        retourne le couple de ses coordonnées"""
        return p.x - self.x, p.y - self.y

    def __str__ (self) :
        """afficher le point"""
        return "(x,y) = (%4.1f,%4.1f) masse %f" % (self.x,self.y,self.m)

class corde (object) :
    """définition d'une corde, une liste de points"""

    def __init__(self, nb, p1, p2, m, k, g, f, l) :
        """initialisation d'une corde
        @param          nb          nombre de points
        @param          p1 = x1,y1  coordoonnées du premier point (fixe)
        @param          p2 = x2,y2  coordoonnées du dernier point (fixe)
        @param          m           masse de la corde,
                                    répartie entre tous les pointstous les points
        @param          k           raideur de l'élastique
        @param          g           intensité de l'apesanteur,
                                    valeur positive
        @param          f           vitesse de freinage
        @param          l           longueur de la corde"""
        x1,y1           = p1[0], p1[1]
        x2,y2           = p2[0], p2[1]
        self.list       = []
        self.vitesse    = []
        for i in range (0,nb) :
            x = x1 + float (i) * (x2 - x1) / float (nb - 1)
            y = y1 + float (i) * (y2 - y1) / float (nb - 1)
            self.list.append ( point (x,y, float (m) / nb))
            self.vitesse.append (point (0,0,0))
        self.k = k * nb
        self.g = g
        self.l = float (l) / (nb-1)
        self.f = f

    def display (self, screen) :
        """affichage de la corde à l'aide du module pyagame"""
        self.display_courbe_theorique (screen)
        x,y     = screen.get_size ()
        color   = (0,0,0)
        for p in self.list :
            pygame.draw.circle (screen, color, (int (p.x), int (y - p.y)), int (p.m+1))
        for i in xrange (0,len (self.list)-1) :
            pygame.draw.line (screen, color, \
                              (int (self.list [i].x),      int (y - self.list [i].y)), \
                              (int (self.list [i+1].x),    int (y - self.list [i+1].y)))

    def display_courbe_theorique (self, screen) :

        def cosh (x) :
            """retourne le cosinus hyperbolique de x"""
            return math.cosh (x)

        def sinh (x) :
            """retourne le sinus hyperbolique de x"""
```

```
            return math.sinh (x)

        def asinh (x) :
            """retourne la réciproque du sinus hyperbolique de x"""
            t = x + math.sqrt (x*x + 1)
            return math.log (t)

        def acosh (x) :
            """retourne la réciproque du cosinus hyperbolique de x"""
            t = abs (x) + math.sqrt (x*x - 1)
            return math.log (t)

        def trouve_alpha (x2, L) :
            """détermine la valeur de alpha une seule fois"""
            if self.__dict__.has_key ("alpha") : return self.alpha

            def fonction (a,x2,L) :
                if a == 0.0 : return 0.0
                else : return 1.0 / a * sinh (a * x2) - L / 2

            a1      = 1.0 / x2
            a2      = 1.0 / x2
            m       = 1.0 / x2
            diff    = fonction (m, x2, L)
            while abs (diff) > 1e-3 :
                if diff > 0 : # faire décroître alpha
                    if fonction (a1, x2, L) > 0 : a1 /= 2.0
                    else : a2 = m
                else :
                    if fonction (a2, x2, L) < 0 : a2 *= 2.0
                    else : a1 = m
                m       = (a1 + a2) / 2
                diff    = fonction (m, x2, L)
            self.alpha = m
            return m

        p1 = self.list [0]
        p2 = self.list [ len (self.list) - 1 ]
        if p1.y != p2.y : return None # cas non prévu
        # on trace la courbe y = cosh (alpha x)
        L       = self.l * (len (self.list) - 1)
        mx      = (p1.x + p2.x) / 2
        x1      = p1.x - mx
        x2      = p2.x - mx
        alpha   = trouve_alpha (x2, L) # est solution de 1/alpha sinh (alpha x1) = L/2
        C       = - 1.0 / alpha * cosh (alpha * x2)

        sx,sy   = screen.get_size ()
        color   = (0,0,255)

        ix1     = int (x1)
        ix2     = int (x2)
        x0,y0   = p1.x,p1.y
        for i in xrange (ix1, ix2+1) :
            x   = int (mx + i)
            y   = 1.0 / alpha * cosh (alpha * float (i)) + C + p1.y
            if 0 < y < sy and 0 < y0 < sy :
                pygame.draw.line (screen, color, (x0,sy-y0), (x, sy-y), 2)
            x0,y0 = x,y


    def force_point (self, i) :
        """calcule les forces qui s'exerce en un point,
        retourne un couple x,y"""
        x,y = 0,0
```

```
        # poids
        y -= self.g * self.list [i].m
        # voisin de gauche
        dx, dy = self.list [i].difference (self.list [i-1])
        d = math.sqrt (dx *dx + dy *dy)
        if d > self.l :
            dx = (d - self.l) / d * dx
            dy = (d - self.l) / d * dy
            x += self.k * dx
            y += self.k * dy
        # voisin de droite
        dx, dy = self.list [i].difference (self.list [i+1])
        d = math.sqrt (dx *dx + dy *dy)
        if d > self.l :
            dx = (d - self.l) / d * dx
            dy = (d - self.l) / d * dy
            x += self.k * dx
            y += self.k * dy
        # freinage
        x += - self.f * self.vitesse [i].x
        y += - self.f * self.vitesse [i].y

        return x,y

    def iteration (self, dt) :
        """calcule les déplacements de chaque point et les met à jour,
        on ne déplace pas les points situés aux extrémités,
        retourne la somme des vitesses et des accélérations au carré"""
        force = [ (0,0) ]
        for i in xrange (1, len (self.list)-1) :
            x,y = self.force_point (i)
            force.append ((x,y))
        force.append ((0,0))

        # déplacement
        for i in xrange (1, len (self.list)-1) :
            self.vitesse [i].deplace ( force [i], dt )
            self.list [i].deplace_point ( self.vitesse [i], dt )

        d = 0
        for f in force :
            d += self.vitesse [i].x ** 2 + force [i][0] **2
            d += self.vitesse [i].y ** 2 + force [i][1] **2

        return d

    def __str__ (self):
        """affiche chaque point de la corde"""
        s = ""
        l = 0
        for i in xrange (0, len (self.list)) :
            s +=  "point " + str (i) + " : " + str (self.list [i])
            if i < len (self.list) -1 :
                x,y     = self.list [i].difference (self.list [i+1])
                d       = math.sqrt (x*x + y*y)
                s += "\t segment : %4.0f" % d
                s += " ( %4.0f )" % self.l
            if i != 0 and i != len (self.list)-1 :
                x,y = self.force_point (i)
                s += "\t force en ce point (%f,%f) " % (x,y)
            s += "\n"
            if i > 0 :
                x,y = self.list [i].difference (self.list [i-1])
                l += math.sqrt (x*x + y*y)
        s += "longueur de la corde " + str (l) + "\n"
```

```
        s += "longueur attendue " + str ((len (self.list)-1) * self.l) + "\n"
        return s

def attendre_clic (screen,x,y):
    """dessine un rectangle rouge sur l'écran et
    attend la pression d'un clic de souris"""
    color   = 255,0,0
    pygame.draw.line (screen, color, (10,10), (x-10,10), 2)
    pygame.draw.line (screen, color, (x-10,10), (x-10,y-10), 2)
    pygame.draw.line (screen, color, (x-10,y-10), (10,y-10), 2)
    pygame.draw.line (screen, color, (10,y-10), (10,10), 2)
    pygame.display.flip ()
    reste = True
    while reste:
        for event in pygame.event.get():
            if event.type == pygame.MOUSEBUTTONUP :
                reste = False
                break

if __name__ == "__main__" :

    pygame.init ()
    size      = width, height = x,y = 800, 500
    black     = 0, 0, 0
    white     = 255,255,255
    screen    = pygame.display.set_mode(size)
    nb        = 10
    c         = corde (nb, (100,450), (700,450), 100, 1, 0.1, 0.05, 800)
    dt        = 0.1
    print "corde initial"
    print c

    iter      = 0
    dep       = len (c.list) * (x*x + y*y)
    while True and dep > 1e-6 :

        for event in pygame.event.get():
            if event.type == pygame.QUIT: sys.exit()
            if event.type == pygame.MOUSEBUTTONUP:
                print c
                attendre_clic (screen,x,y)

        if iter % 10 == 0 :
            screen.fill (white)
            c.display (screen)
            pygame.display.flip ()

        iter += 1

        if iter == 1: attendre_clic (screen,x,y)

        dep = c.iteration (dt)
        #print "dep =", dep

        pygame.display.flip ()

    print c
    attendre_clic (screen,x,y)
```

# Chapitre 28

# Inclassables

---

## 28.1 Filtrage du signal sonore

### 28.1.1 Enoncé

Un signal sonore est une série temporelle où chaque valeur est une amplitude qui définit le son pendant une très courte durée - quelques dizaines de microsecondes - (voir la figure 28.1). Cette information est celle qu'on trouve sur des supports numériques comme le compact disque. Les algorithmes de compression condensent cette information en minimisant la dégradation du son. Via des formats de codage tels que le format MP3, le son peut être stocké sous une forme compressée. Compressé, le son peut aussi voyager plus facilement. La téléphonie mobile utilise sans cesse des algorithmes de compression afin de réduire les coûts de transmission.

#### 28.1.1.1 Transformée de Fourier discrète

La transformée de Fourier est une des méthodes utilisée pour compresser le son. Le son est composé d'ondes, sa nature se prête bien à sa décomposition en somme de fonctions sinusoïdales. La transformée de Fourier discrète permet de décomposer une série temporelle de période $T$, $(Y_0, \dots, Y_{T-1})$ :

$$\forall t \geqslant 0,\ Y_t = \frac{1}{T} \sum_{n=0}^{T-1} \widehat{Y_n} \exp\left(\frac{2i\pi tn}{T}\right) \tag{28.1}$$

Les coefficients $\left(\widehat{Y_n}\right)_n$ sont calculés comme suit :

$$\forall t \geqslant 0,\ \widehat{Y_p} = \sum_{n=0}^{T-1} Y_n \exp\left(-\frac{2i\pi pn}{T}\right) \tag{28.2}$$

Le calcul des coefficients nécessite $N^2$ additions et multiplications complexes.

#### 28.1.1.2 Transformée de Fourier rapide

La transformée de Fourier rapide ou Fast Fourier Transform (FFT) permet de calculer les coefficients de la transformée en un coût de $O(T \ln T)$ au lieu de $O(T^2)$ comme le suggère le paragraphe 28.1.1.1. L'algorithme est itératif, on calcule des transformée de Fourier sur des échantillons de taille croissante jusqu'à obtenir les coefficients de la transformée de Fourier sur l'échantillon initiale. On considère la série temporelle $(Y_0, \dots, Y_{T-1})$, on suppose que $T$ est paire et $T = 2R$.

$$\begin{aligned}
\forall t \geqslant 0,\ \widehat{Y_p} &= \sum_{n=0}^{T-1} Y_n \exp\left(-\frac{2i\pi pn}{T}\right) \\
&= \sum_{n=0}^{R-1} Y_{2n} \exp\left(-\frac{2i\pi p(2n)}{2R}\right) + \sum_{n=0}^{R-1} Y_{2n+1} \exp\left(-\frac{2i\pi p(2n+1)}{2R}\right) \\
&= \sum_{n=0}^{R-1} Y_{2n} \exp\left(-\frac{2i\pi pn}{R}\right) + \exp\left(-\frac{2i\pi p}{2R}\right) \sum_{n=0}^{R-1} Y_{2n+1} \exp\left(-\frac{2i\pi p(2n)}{2R}\right) \\
&= \sum_{n=0}^{R-1} Y_{2n} \exp\left(-\frac{2i\pi pn}{R}\right) + \exp\left(-\frac{2i\pi p}{2R}\right) \sum_{n=0}^{R-1} Y_{2n+1} \exp\left(-\frac{2i\pi pn}{R}\right) \qquad (28.3)
\end{aligned}$$

On construit les deux sous-suites extraites de la suite $(Y_0, \ldots, Y_{T-1})$. $T$ est toujours pair, $T = 2R$. La première suite contient les termes pairs $(Y_{2t})_{0 \leqslant t < R}$, la seconde contient les termes impairs, $(Y_{2t+1})_{0 \leqslant t < R}$. Pour ces deux sous-suites, on calcule la transformée de Fourier pour obtenir les coefficients de la série paire $\left(\widehat{Y_p^a}\right)_{0 \leqslant t < R}$ et de la série impaire $\left(\widehat{Y_p^b}\right)_{0 \leqslant t < R}$. L'équation (28.3) peut être écrite sous la forme :

$$\forall t \geqslant 0,\ \widehat{Y_p} = \begin{cases} \widehat{Y_p^a} + \exp\left(-\frac{2i\pi p}{2R}\right) \widehat{Y_p^b} & \text{si } 0 \leqslant p < R \\ \widehat{Y_{R-p}^a} + \exp\left(-\frac{2i\pi p}{2R}\right) \widehat{Y_{R-p}^b} & \text{si } R \leqslant p < 2R \end{cases} \qquad (28.4)$$

Si on suppose que $T$ est une puissance de deux, $T = 2^q$, il est possible de réitérer ce calcul de manière récursive jusqu'à obtenir des sous suites de deux valeurs. Et à chaque étape, l'expression (28.3) utilise $T$ additions et multiplications. Le coût global du calcul de la transformée de Fourier est en $O(T \log_2 T)$. Ce gain est très intéressant lorsqu'on fait du traitement du signal en temps réel.

Le livre [Mallat2000] dresse sommairement le portrait de la transformée de Fourier rapide, il fait référence au livre [Nussbaumer1982] ou encore à l'article [Duhamel1990] qui s'intéresse particulièrement à ce sujet car il existe des variantes plus rapide encore de la transformée de Fourier rapide.

#### 28.1.1.3 Filtrage

La méthode de filtrage proposée est assez simple.

1. On décompose tout d'abord le signal sonore avec la transformée de Fourier rapide.
2. On élimine ensuite une partie des coefficients. Un filtre passe haut éliminera les fréquences basses, un filtre passe-bas éliminera les hautres fréquences. On peut choisir de conserver uniquement les fréquences pour lesquels les coefficients $\widehat{Y_n}$ sont de module élevé.
3. Cette étape terminée, on recompose le signal avec les coefficients restants.

L'objectif de la compression est de supprimer le plus de coefficients sans trop dégrader le signal lors de sa recomposition. Pour des signaux longs - une conversion téléphonique par exemple -, il est impossible de traiter l'ensemble du signal. C'est pourquoi, on découpe la série temporelle en petits segments de longueur fixe, on applique le processus décrit ci-dessus sur chacun des segments. Cette étape est appelée *échantillonnage*. On choisira de préférence des segments de longueur une puissance de deux, 256 ou 512 par exemple, ils correspondent à des signaux sonores de 10 ou 20 millisecondes. La longueur des échantillons est souvent choisie en fonction des capacités de calcul. L'échantillonnage limite l'estimation des coefficients associés aux basses fréquences, ceci a pour conséquence que l'estimation des coefficients associés aux hautes fréquences est elle aussi faussée. Ce phénomène est appelé *aliasing*. On dit que les hautes fréquences se replie sur les basses fréquences. Ce phénomène a tendance à produire des discontinuités entre deux segments lorsque le signal est reconstitué. Pour corriger ce phénomène, le signal est souvent modifié ou modulé avant d'être échantillonné.

#### 28.1.1.4 Horizons

La transformée de Fourier date du début du XVIII° siècle, à l'époque de Napoléon (voir [Math2000]). Fourier essayait de prévoir la propagation de la chaleur dans les corps solides. Il eut l'idée de cette décomposition en 1807 et la publia sous une forme achevée en 1822 dans son livre intitulée *Théorie analytique de la chaleur*. Fourier y affirmait que toute fonction, éventuellement discontinue, pouvait être décomposée sous la forme d'une somme de fonctions sinusoïdales. Fourier ne démontrait pas la convergence de sa série, problème qui trouva sa solution à la fin du XIX° siècle lorsque lord Kelvin inventa un calculateur mécanique permettant de prévoir la hauteur des marées.

De grands noms jalonnent l'histoire de l'analyse de Fourier comme Josiah Willard Gibbss qui se pencha sur le problème de la convergence de la série de Fourier en un point de discontinuité en 1899. Laurant Schwartz au milieu du XX° siècle rapprocha l'analyse de Fourier avec la théorie de la distribution. James Cooley inventa en 1965 la transformée de Fourier rapide. Cette théorie a permis aussi de découvrir la structure de l'ADN, utilisée pour décrypter la diffraction des rayons X.

Entre les deux guerres, Hartley émit l'idée que l'homme localisait la provenance d'un son grâce au déphasage relatif des ondes atteignant ses deux oreilles. Ses recherches aboutirent à la transformée de Hartley, qui décompose elle-aussi une fonction réelles en somme de fonctions sinusoïdales sans recourir aux nombres complexes.

$$\forall t \in \mathbb{N} \cap [-T+1, T-1]\,,\ \widehat{Y_t^H} = \sum_{n=0}^{T-1} Y_n \left[ \cos \frac{2i\pi kn}{N} + \sin \frac{2i\pi kn}{N} \right] \tag{28.5}$$

Les coefficients de la transformée de Fourier peuvent se déduire de ceux obtenus pour la transformée de Hartley qui présente l'avantage d'être plus rapide à calculer. Il existe également une version rapide de la transformée de Hartley mise au point en 1984 (voir [Bracewell1986]).

### 28.1.2 Correction

Le programme suivant filtre un signal sonore. Il décompose le signal à l'aide d'une transformée de Fourier discrète puis le recompose après qu'une partie de son spectre a été enlevée. Les figures 28.1 et 28.2 illustrent les signaux originaux et modifiée ainsi que le spectre du signal pour un court extrait du signal sonore choisi au milieu du `sound10.wav`. Les moyennes fréquences ont été conservées, intervalles $[20, 30]$.

La principale fonction du programme est la fonction `filtre_son_extrait` qui décompose un signal par l'intermédiaire de la fonction `fft` du module `Numeric.FFT`. Le signal est filtré, seules sont conservées les fréquences comprises entre $a$ et $b$. Puis le signal est recomposé grâce à la fonction `inverse_fft`.

La fonction `filtre_son` se contente de découper le signal par période de longueur `div` puis envoie chaque segment à la fonction précédemment décrite. Les autres fonctions effectuent les tâches annexes, lire un fichier contenant un signal sonore, le jouer, afficher un segment, afficher les coefficients de sa décomposition.

```
import pygame
import pygame.mixer
import pygame.sndarray
import FFT
import math
import numpy as Numeric
import string
import copy
import pylab
import numpy

pygame.mixer.init ()
pygame.init ()
```

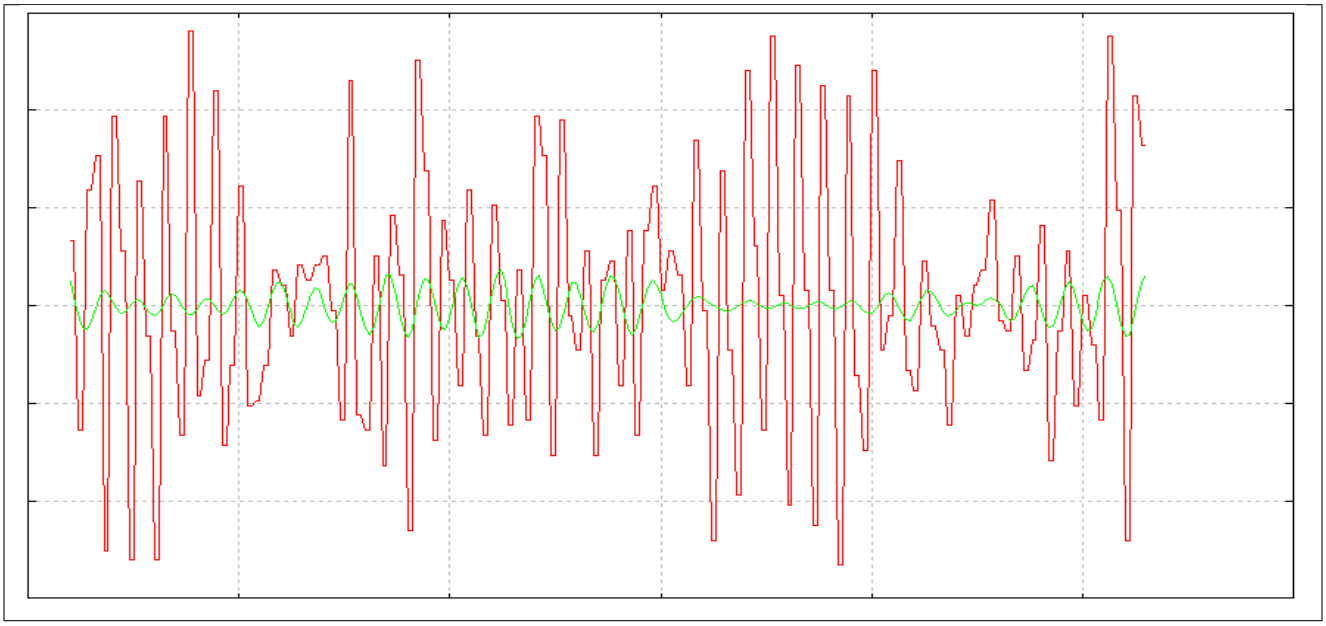


**Figure 28.1 :** *Signal original et signal filtrée. Les basses et hautes fréquences ont été enlevées. Le spectre correspond est donné par la figure 28.2.*

```
fourier = None
indice = None

def get_sound ():
    """charge le son sound010.wav"""
    s = pygame.mixer.Sound ("sound010.wav")
    t = pygame.sndarray.array (s)
    return s

def play_sound(s):
    """joue un son"""
    s.play ()

def convert_array(t, s):
    """joue un son decrit dans un tableau a une dimension"""

    s = pygame.sndarray.array (s)
    for i in range (0, len (s)) :
        s [i]= t [i]
    #tt = Numeric.array ([ [x, x] for x in t] )
    #print tt [0:10]
    s = pygame.sndarray.make_sound (s)
    return s

def array_sound(s):
    """convertit un son en un tableau d'entiers"""
    a = pygame.sndarray.array(s)
    t = Numeric.array([i for i in xrange(0,len(a))])
    for i in xrange(0,len(a)): t [i] = a [i][0]
    return t

def dessine_son(mem,t,fourier,ind,a,b):
    """dessine une partie du son, limite la taille a 512"""
    m = len (mem)
    if m > 256 : m = 256
    x  = [ i for i in xrange (ind,ind+m) ]
```

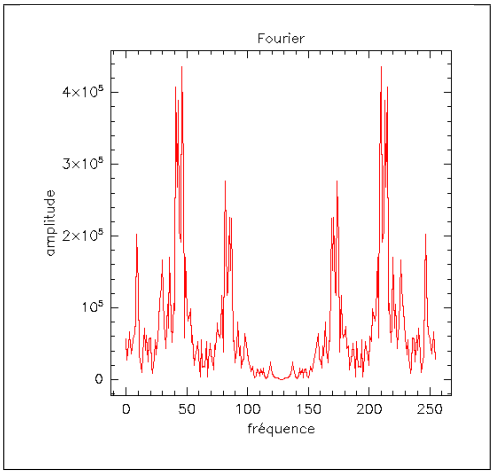



**Figure 28.2 :** *Transformée de Fourier du signal de la figure 28.1.*

```
    y1 = [ mem[i] for i in xrange (ind,ind+m) ]
    y2 = [ t[i] for i in xrange (ind,ind+m) ]
    pylab.figure (1)
    p1 = pylab.plot (x,y1)
    p2 = pylab.plot (x,y2)
    pylab.title ("Fourier")
    pylab.xlabel ("frequence")
    pylab.ylabel ("amplitude")
    pylab.legend ( ("son", "son + filtre"))

    m = len (fourier)
    if m > 256 : m = 256
    #x  = [ i for i in xrange (0,m) ]
    pylab.figure (2)
    x  = [ i for i in xrange (0,m) ]
    y1 = [ abs(fourier[i]) for i in xrange (0,m) ]
    y2 = []
    for i in x :
        if a <= i <= b : y2.append (450000.0)
        else : y2.append (0.0)
    p3 = pylab.plot (x,y1)
    p4 = pylab.plot (x,y2)
    pylab.legend ( ("fourrier", "filtre"))
    pylab.show()

def filtre_son_extrait(t,a,b):
    """calcul de la transformee de Fourier, application du filtre [a,b],
    recomposition du signal"""
    fft = FFT.fft (t)
    global fourier
    if fourier == None and indice != None : fourier = copy.copy(fft)
    for i in xrange(0,len(t)):
        if a <= i <= b:
            pass
        else:
            fft [i] = complex(0,0)
    tt = FFT.inverse_fft(fft)
    for i in xrange(0,len(t)):
        t [i] = int(tt [i].real)


def filtre_son(t,a,b,div = 256):
    """filtre un son par tranche de div frequences, ne garde que les
    frequences comprises entre a et b"""
```

```
    global indice
    nb = len (t) / div
    for i in xrange (0,nb):
        if i == nb / 2 : indice = i * div
        ext = t [i * div : (i+1) * div]
        filtre_son_extrait (ext,a,b)

def essai():
    print "chargement du son"
    s = get_sound ()
    print "duree : ", s.get_length (), " secondes"
    print "musique"
    play_sound (s)
    pygame.time.delay (6000)
    t = array_sound (s)
    mem = copy.copy(t)
    print "nombre de donnees ", len (t)
    print "duree d'une donnee ", s.get_length () * 1000 / len (t), "millisecondes"


    rep = ""
    if rep != "n" :
        #rep = string.split (rep, ",")
        #a = int (rep [0])
        #b = int (rep [1])
        a,b = 10,100

        print "filtrage [%d,%d]" % (a,b)
        filtre_son (t,a,b)

        print "dessin des premiers instants, son filtre"
        dessine_son (mem,t,fourier,indice, a,b)

        print "son filtre"
        s = convert_array (t, s)
        play_sound (s)

        pygame.time.delay (6000)

essai ()
```

## 28.2 Images de synthèse, modèle d'illumination

### 28.2.1 Enoncé

Le lancer de rayon est une méthode couramment utilisée pour réaliser des images de synthèse. L'exposé se limite au dessin de décor incluant des sphères et des facettes triangulaires ou quadrilaètres. La méthode peut être aisément étendue à d'autres figures.

#### 28.2.1.1 La scène

La *scène* est l'ensemble des objets et des sources de lumière qui doivent être représentés par l'image de synthèse. La surface de ces objets peut avoir différentes propriétés (surface mate, réfléchissante, création d'un rayon réfracté). L'image de synthèse est une vue possible de cette scène définie par la position de l'œil d'un obervateur, matérialisé par un point (le centre de l'œil), un angle de vue et un écran de projection qui sert de rétine.

Un œil - ou un appareil photographique - capte tous les rayons de lumière qui traversent l'objectif jusqu'à la pellicule (voir figure 28.3). La première idée pour construire une image de synthèse est simplement de prendre une à une les sources de lumière, de calculer tous les rayons qu'elles produisent et de s'intéresser

à ceux qui interceptent notre écran (voir figure 28.4). Malheureusement, le nombre de rayons qui viennent toucher l'écran est négligeable par rapport à ceux émis par les sources de lumière. Il est préférable de ne considérer que ceux qui nous sont utiles pour calculer l'image de synthèse, c'est-à-dire ceux qui atteignent la pellicule.

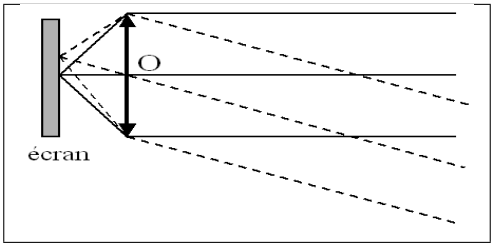



**Figure 28.3 :** *Principe des appareils photographiques, la lentille fait converger tous les rayons parallèles en un seul point de l'écran. Lors du calcul d'une image de synthèse, on se concentre sur le seul rayon qui passe par le centre de la lentille.*

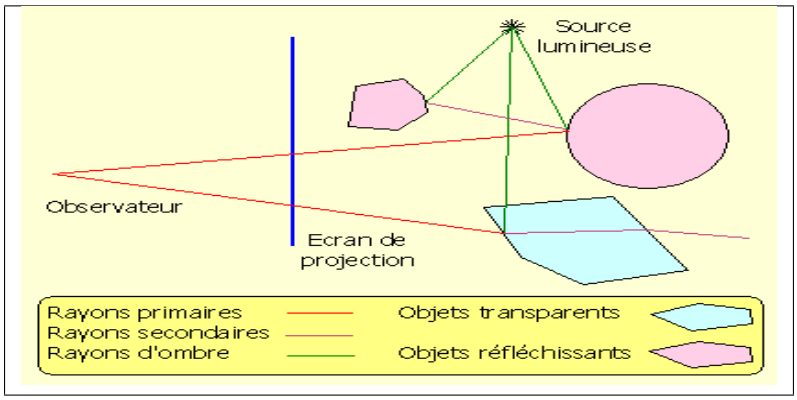



**Figure 28.4 :** *Peu nombreux les rayons qui partent de la source lumineuse et qui atteignent l'œil. L'idée du lancer de rayon consiste donc à partir de l'œil pour atteindre la source lumineuse.*

On considère que tous les rayons utiles pour l'image arrivent en un même point $O$ qui est le centre de l'œil de l'observateur. C'est le principe de la chambre noire. L'idée du lancer de rayon est de parcourir le même chemin que la lumière mais en sens inverse (voir figure 28.5), c'est-à-dire de l'œil vers la source de lumière.

Pour cela, un écran quadrillé en pixel est placé devant le point $O$ (ou derrière comme sur la figure 28.3). On lance un rayon partant de $O$, passant par le pixel de coordonnées $(i, j)$ de l'image et on calcule la trajectoire de ce rayon afin de déterminer la couleur de ce pixel. Si le rayon n'intercepte aucun objet et se perd à l'infini (le ciel par exemple), la couleur de l'image au point de l'écran traversé par le rayon sera donc noire (ou bleue si le rayon atteint le ciel). En revanche, si le rayon intercepte un objet de couleur rouge en un point $P$, on regarde si le point P est éclairé par une source de lumière :

- Si c'est le cas, la couleur de l'image au point de l'écran traversé par le rayon sera donc rouge.
- Si ce n'est pas le cas, cela veut dire que ce point de l'objet n'est pas éclairé (il y a un objet entre la source et le point P ou la source n'éclaire pas dans cette direction), l'image au point de l'écran traversé par le rayon sera noire.

Une couleur est composée de trois couleurs primaires : rouge, vert, bleu. La définition d'une couleur repose donc sur trois intensités. Par soucis d'économie de mémoire et de traitement, les ordinateurs utilisent en général 256 niveaux d'intensité pour chaque couleur primaire, soit $256 \times 256 \times 256 = 2^{24} = 16 \times 2^{20} \approx 16$ millions de couleurs. Le tableau suivant contient quelques couleurs triplets correspondant à des couleurs usuelles.

| | | | |
|---|---|---|---|
| (0,0,0) | noir | (0,0,255) | bleu |
| (255,255,255) | blanc | (100,100,100) | gris foncé |
| (255,0,0) | rouge | (200,200,200) | gris clair |
| (0,255,0) | vert | (200,200,0) | jaune clair |

Selon cette définition, une source de lumière ponctuelle sera composée en fait de trois sources de lumières ponctuelles. Dans l'algorithme du lancer de rayon, chaque couleur sera traitée séparément. Une source de lumière est en fait la juxtaposition de trois sources de couleurs indépendantes. De cette manière un objet parfaitement bleu éclairé par une lumière parfaitement rouge sera noir.

La méthode décrite plus haut ne considère que des objets avec des surfaces régulières et mates. Le rayon de lumière ne peut les traverser (réfration dans une goutte d'eau) ou rebondir comme dans un miroir (réflexion). Le schéma 28.5 (voir également la figure 28.10) illustre de quelle manière ce problème peut être résolu partiellement.

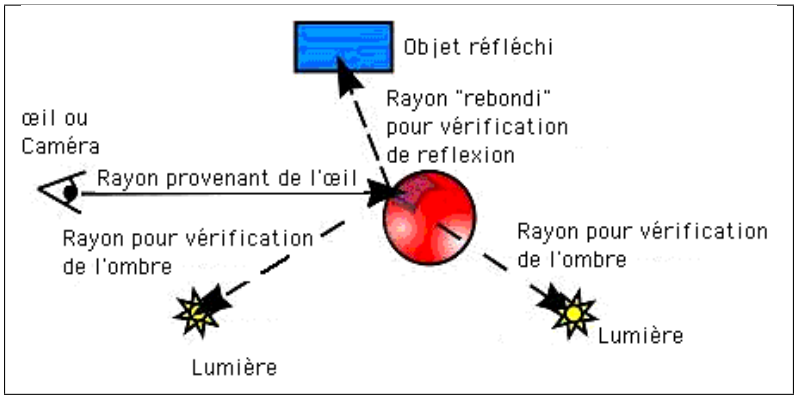



**Figure 28.5 :** *Le rayon réfracté et le rayon réfléchi peuvent être considérés comme deux autres rayons émanents de l'œil même s'ils n'en viennent pas directement. On peut de cette manière voir un objet à travers un miroir, mais il n'est pas possible qu'un objet mate soit éclairé à travers un miroir.*

#### 28.2.1.2 Modèle d'illumination

Les images produites par la méthode précédente ne sont pas satisfaisantes. Les surfaces non directement accessibles depuis une source lumière n'apparaissent pas à l'écran alors que dans la réalité, il existe une lumière diffuse qui illumine toute la pièce et qui fait que l'on peut voir sous une table même si les lumières sont fixées au plafond. De même, un plan faisant face à une source lumière sera beaucoup plus éclairé qu'un plan incliné par rapport à cette même source.

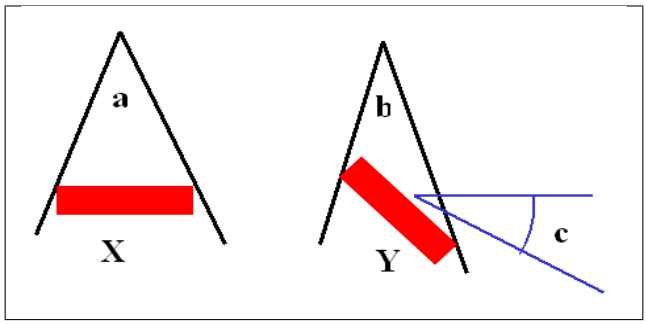



**Figure 28.6 :** *Les deux plans de cette figure ont la même taille mais le second est incliné, il reçoit moins de lumière et en renvoie d'autant moins.*

La figure 28.6 permet de comparer l'illumination d'un plan $Y$ incliné d'un angle $c$ par rapport à un plan $X$ non incliné. La quantité de lumière $I$ reçue par les plans $X$ et $Y$ est proportionnelle aux angles $a$ et $b$.

Le plan $Y$ est incliné et sur le schéma $b < a$, par conséquent $Y$ recevra moins de lumière que $X$.

Les modèles d'illumination proposent des moyens de calculer l'intensité lumineuse de chaque rayon de manière à se rapprocher le plus possible de la réalité. Plus le modèle est complexe, plus le temps de calcul d'une image est long. Les modèles d'illumination sont des compromis entre temps de calcul et réalisme. Par exemple :

- Une source de lumière réelle est rarement ponctuelle et éclaire rarement dans toute les directions mais sa modélisation peut l'être car cette simplification ne nuit pas trop à la qualité de l'image.
- Une surface peut être parfaitement réfléchissante comme un miroir mais aussi simplement brillante ou mate, ces trois différents types de surfaces ne captent pas la lumière de la même manière. Si un miroir est facile à modéliser, une surface brillante n'est jamais tout-à-fait réfléchissante ni tout-à-fait mate : c'est un compromis entre les deux. On pourra considérer qu'une partie $k$ du rayon est réfléchie, l'autre $1 - k$ est diffusée dans toutes les directions.

#### 28.2.1.3 Géométrie et sphère

Le technique du lancer de rayon nécessite le calcul d'intersection entre une droite et tout type d'objet représenté. Par la suite, le modèle d'illumination choisi (voir paragraphe 28.2.1.7) requiert le calcul du vecteur normal en tout point de la surface de l'objet. Ce paragraphe s'intéresse au cas particulier où les objets de la scène sont tous des sphères.

> **Définition 28.1 : rayon**
> Un rayon est simplement une demi-droite avec une origine et un sens de direction. L'origine est en général le centre de l'œil.

Dans le cas du lancer de rayon, chaque rayon partira de l'œil de l'observateur, ira ou non intercepter un objet. Dans le cas où le rayon est intercepté, on regarde si un rayon provenant d'une source de lumière peut atteindre ce point donc si le rayon reliant ce point à la source de lumière n'intercepte pas un autre objet.

Les calculs du point d'intersection sont présentés dans le cas où cet objet est une sphère. Un rayon est représenté de manière paramétrique comme suit :

$$\begin{cases} x = \lambda v_x + O_x \\ y = \lambda v_y + O_y \\ z = \lambda v_z + O_z \end{cases} , \lambda \geqslant 0 \text{ où } O \text{ est l'origine du rayon et } \overrightarrow{V} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \text{ sa direction} \tag{28.6}$$

L'équation d'une sphère est la suivante :

$$(x - C_x)^2 + (y - C_y)^2 + (z - C_z)^2 = R^2 \tag{28.7}$$

$R$ est le rayon et $C$ le centre de la sphère. Un point $M = (x, y, z)$ est l'intersection d'une droite et d'une sphère s'il appartient à la fois à la droite et à la sphère. On obtient l'équation :

$$(\lambda v_x + O_x - C_x)^2 + (\lambda v_y + O_y - C_y)^2 + (\lambda v_z + O_z - C_z)^2 = R^2 \tag{28.8}$$

On pose $\overrightarrow{OC} = (C_x - O_x, C_y - O_y, C_z - O_z)$ et :

$$\Leftrightarrow \lambda^2 \left\| \overrightarrow{V} \right\|^2 - 2\lambda \overrightarrow{V}.\overrightarrow{OC} + \left\| \overrightarrow{OC} \right\|^2 - R^2 = 0 \tag{28.9}$$

Cette équation admet des solutions si son discriminant réduit vérifie :

$$\Delta' = \left(\overrightarrow{V} \,.\, \overrightarrow{OC}\right)^2 - \left\|\overrightarrow{V}\right\|^2 \left(\left\|\overrightarrow{OC}\right\|^2 - R^2\right) \geqslant 0 \tag{28.10}$$

C'est une équation du second degré qui a au plus deux solutions $\lambda_1$ et $\lambda_2$ :

$$\begin{cases} \lambda_1 = \dfrac{\overrightarrow{V} \,.\, \overrightarrow{OC} - \sqrt{\Delta'}}{\left\|\overrightarrow{V}\right\|^2} = \dfrac{\overrightarrow{V}\cdot\overrightarrow{OC} - \sqrt{\left(\overrightarrow{V} \,.\, \overrightarrow{OC}\right)^2 - \left\|\overrightarrow{V}\right\|^2\left(\left\|\overrightarrow{OC}\right\|^2 - R^2\right)}}{\left\|\overrightarrow{V}\right\|^2} \\ \lambda_2 = \dfrac{\overrightarrow{V} \,.\, \overrightarrow{OC} + \sqrt{\Delta'}}{\left\|\overrightarrow{V}\right\|^2} = \dfrac{\overrightarrow{V}\cdot\overrightarrow{OC} + \sqrt{\left(\overrightarrow{V} \,.\, \overrightarrow{OC}\right)^2 - \left\|\overrightarrow{V}\right\|^2\left(\left\|\overrightarrow{OC}\right\|^2 - R^2\right)}}{\left\|\overrightarrow{V}\right\|^2} \end{cases} \tag{28.11}$$

En remplaçant $\lambda$ dans la représentation paramètrique de la droite, on obtient les coordonnées des intersections $T(\lambda_1)$ et $T(\lambda_2)$ si elles existent. Dans ce cas, on sait que $T(\lambda_1)$ est placé devant $T(\lambda_2)$ et devant l'observateur si :

$$0 \leqslant \overrightarrow{V}.\overrightarrow{OT(\lambda_1)} \leqslant \overrightarrow{V}.\overrightarrow{OT(\lambda_2)} \Longleftrightarrow 0 \leqslant \lambda_1 \leqslant \lambda_2 \tag{28.12}$$

Pour un rayon, on calcule l'intersection de ce rayon avec chaque objet de la scène. L'observateur ne pourra percevoir que l'objet le plus proche placé devant lui, c'est-à-dire le plus proche des points d'intersection. De plus, ce point le plus proche ne sera visible que s'il éclairé (voir figure 28.7). Ceci signifie qu'il existe un autre rayon qui relie une source lumineuse à ce point, qui n'intercepte aucun objet y compris celui dont fait partie ce point.

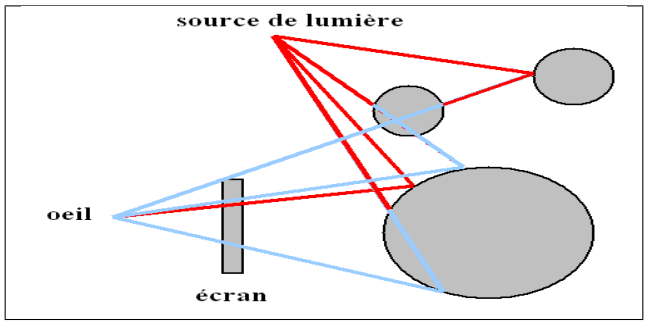



**Figure 28.7 :** *Partie non éclairées par la source lumineuse, la plus petite des trois sphères s'interposent entre la plus grande et la source lumineuse. La grande sphère arborera une zone d'ombre. Elle s'interpose également entre l'œil et l'autre petite sphère dont une partie seulement sera visible.*

A partir du moment où on connaît le centre $C$ d'une sphère, on peut déterminer la normale à cette sphère en tout point $P$ de cette surface :

$$\overrightarrow{N} = \frac{\overrightarrow{CP}}{CP} = \frac{\overrightarrow{CP}}{R} \tag{28.13}$$

#### 28.2.1.4 Géométrie et facettes

Le paragraphe précédent étudiait l'intersection d'un rayon et d'une sphère. Celui-ci étudie l'intersection d'un rayon et d'une triangle ou d'un quadrilatère plan. Ces résultats peuvent être étendus à tout polygone inscrit dans un même plan.

Il faut d'abord calculer l'intersection entre une droite et un plan de l'espace. Tout d'abord, on suppose que les points A,B,C forment un triangle non réduit à une droite. $A$ a pour coordonnées $(x_a, y_a, z_a)$, $B$ a pour coordonnées $(x_b, y_b, z_b)$, $C$ a pour coordonnées $(x_c, y_c, z_c)$. Un vecteur normal au plan qui contient ce triangle est obtenu par :

$$\overrightarrow{AB} \wedge \overrightarrow{AC} = \begin{pmatrix} (y_b - y_a)(z_c - z_a) - (y_c - y_a)(z_b - z_a) \\ (z_b - z_a)(x_c - x_a) - (z_c - z_a)(x_b - x_a) \\ (x_b - x_a)(y_c - y_a) - (x_c - x_a)(y_b - y_a) \end{pmatrix} = \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = \overrightarrow{n} \tag{28.14}$$

Soit $M$ un point de coordonnées $M(x, y, z)$, le plan $P$ qui contient le triangle $ABC$ a pour équation cartésienne :

$$\overrightarrow{AB} \wedge \overrightarrow{AC} \cdot \overrightarrow{AM} = 0 \Longleftrightarrow n_x (x - x_a) + n_y (y - y_a) + n_z (z - z_n) = 0 \tag{28.15}$$

On suppose que la droite $D$ passant par $O = (x_o, y_o, z_o)$ et de vecteur directeur $\overrightarrow{v} = (v_x, v_y, v_z)$ a pour équation paramétrique :

$$D = \begin{cases} x_o + \lambda v_x \\ y_o + \lambda v_y \\ z_o + \lambda v_z \end{cases}, \ \lambda \in \mathbb{R} \tag{28.16}$$

L'intersection entre le plan $P$ et la droite $D$ est la solution de l'équation :

$$\begin{aligned} n_x (x_o + \lambda v_x - x_a) + n_y (y_o + \lambda v_y - y_a) + n_z (z_o + \lambda v_z - z_a) &= 0 \\ \lambda (n_x v_x + n_y v_y + n_z v_z) + n_x (x_o - x_a) + n_y (y_o - y_a) + n_z (z_o - z_a) &= 0 \\ \lambda \overrightarrow{n} \cdot \overrightarrow{v} + \overrightarrow{n} \cdot \overrightarrow{AO} &= 0 \end{aligned} \tag{28.17}$$

On suppose que la droite n'est pas parallèle au plan (c'est-à-dire $\overrightarrow{n} \cdot \overrightarrow{v} \neq 0$). Dans ce cas, on peut écrire que :

$$\lambda = -\frac{\overrightarrow{n} \cdot \overrightarrow{AO}}{\overrightarrow{n} \cdot \overrightarrow{v}} \tag{28.18}$$

Le point d'intersection $I$ entre le plan $P$ et la droite $D$ est donc :

$$I = O - \frac{\overrightarrow{n} \cdot \overrightarrow{AO}}{\overrightarrow{n} \cdot \overrightarrow{v}} \overrightarrow{v} = O + \frac{\overrightarrow{n} \cdot \overrightarrow{OA}}{\overrightarrow{n} \cdot \overrightarrow{v}} \overrightarrow{v} \tag{28.19}$$

Après avoir calculé l'intersection entre la droite $D$ et le plan $P$, il faut déterminer si le point d'intersection appartient au triangle $ABC$. Si c'est le cas, la somme des angles est égale à $2\pi$ (voir figure 28.8) :

$$\left(\overrightarrow{IA}, \overrightarrow{IB}\right) + \left(\overrightarrow{IB}, \overrightarrow{IC}\right) + \left(\overrightarrow{IC}, \overrightarrow{IA}\right) = 2\pi \tag{28.20}$$

Pour savoir si un point $I$ appartient à un quadrilatère plan $ABCD$, la condition précédente devient :

$$\left(\overrightarrow{IA}, \overrightarrow{IB}\right) + \left(\overrightarrow{IB}, \overrightarrow{IC}\right) + \left(\overrightarrow{IC}, \overrightarrow{ID}\right) + \left(\overrightarrow{ID}, \overrightarrow{IA}\right) = 2\pi \tag{28.21}$$

Il est facile d'adapter l'expression précédente pour tout polygone inscrit dans un plan dont l'intérieur ne forme qu'une seule composante connexe.

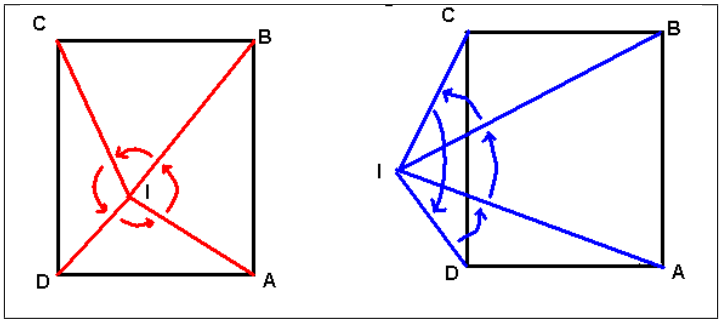



**Figure 28.8 :** *Si $I$ est à l'intérieur du rectangle, la somme des angles de centre $I$ est égale à $2\pi$. Si $I$ est à l'extérieur du rectangle, la somme des angles ce centre $I$ est nulle.*

#### 28.2.1.5 Lancer de rayon

Cette partie n'utilise plus le fait que les objets soient des sphères, seuls comptent les coordonnées des points d'intersection et la normale à la surface des objets en ces points. Il ne reste plus qu'à placer l'œil de l'observateur dans la scène et construire une image en fonction d'un angle de vue $\hat{a}$ qui décrit l'ouverture de l'œil. Nous allons pour le moment lancer un seul rayon par pixel. Ce rayon a pour origine l'œil $O$ et pour direction un vecteur que nous allons constuire. Les figures 28.9, 28.10, 28.11 illustrent cela.

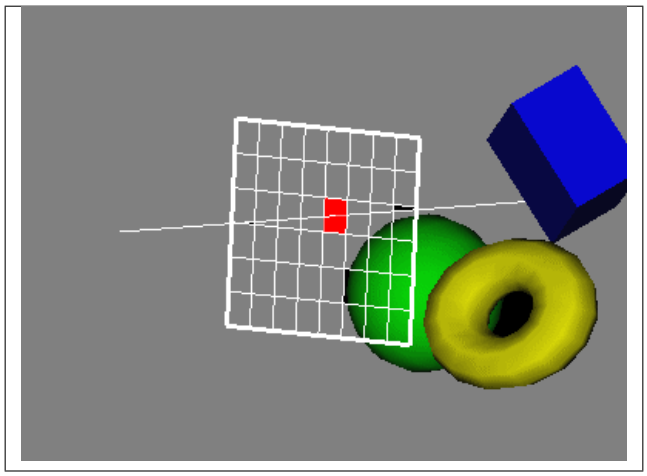


**Figure 28.9 :** *Lancer d'un rayon, il part du centre de l'œil, traverse l'écran avant d'atteindre un objet.*

Pour caractériser précisement le point de vue en plus de l'angle de vue, il est nécessaire d'avoir une direction de visée $\overrightarrow{D}$ qui détermine un cône de visée et un axe $\overrightarrow{A}$ orthogonal au précédent qui détermine la verticale. $\left(\overrightarrow{D}, \overrightarrow{A}, \overrightarrow{D} \wedge \overrightarrow{A}\right)$ forme une base orthonormée de l'espace, $\left(\overrightarrow{A}, \overrightarrow{D} \wedge \overrightarrow{A}\right)$ forme une base de l'écran. On suppose que l'image à construire admet une résolution de $n$ pixels sur $m$ pixels. La figure 28.12 illustre ces notations. On considère l'ensemble de points :

$$\mathbf{P}_n = \left\{ P_{ij}^{nm} = O + \overrightarrow{D} + \frac{i \tan \frac{\hat{a}}{2}}{n} \overrightarrow{A} + \frac{j \tan \frac{\hat{a}}{2}}{n} \overrightarrow{D} \wedge \overrightarrow{A}, \quad (i,j) \in \left\{ -\frac{n}{2}, \ldots, \frac{n}{2} \right\} \times \left\{ -\frac{m}{2}, \ldots, \frac{m}{2} \right\} \right\} \quad (28.22)$$

Le rayon $\left(O, \overrightarrow{OP_{ij}^{nm}}\right)$ est le rayon qui partira de l'œil de l'observateur en $O$ et passera par le pixel de

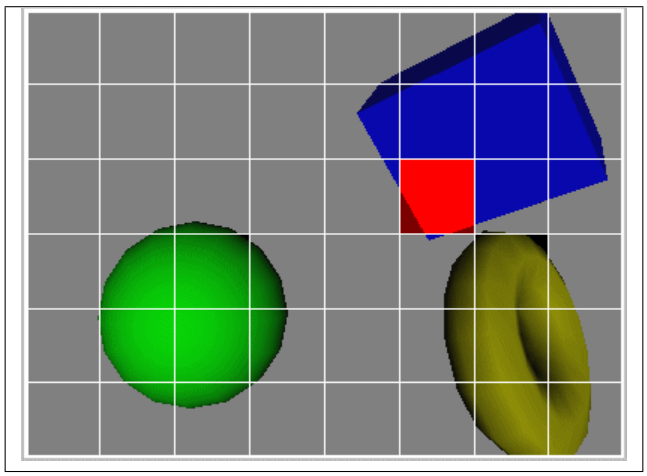


**Figure 28.10 :** *Lancer d'un rayon, autre vue, le quadrillage matérialise les pixels. Le pixel prendra la couleur de l'objet qui recouvra le centre du pixel.*

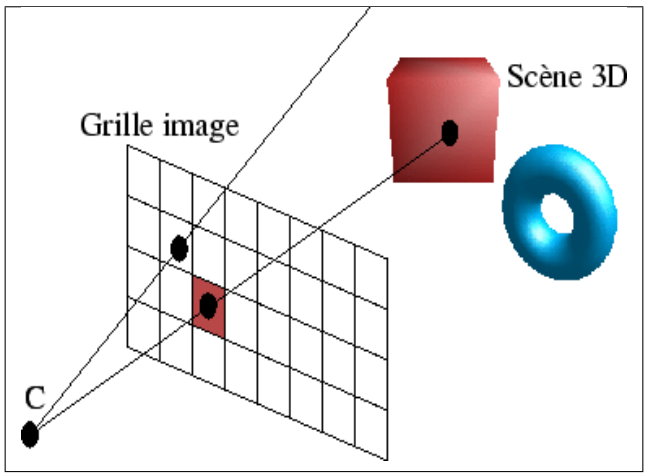



**Figure 28.11 :** *Lancer d'un rayon, le rayon part du centre de l'œil, coupe l'écran en un pixel puis s'en va intercepter un objet.*

---

l'image de coordonnées $\left(i+\frac{n}{2}, j+\frac{m}{2}\right)$. On peut maintenant décrire l'algorithme du lancer de rayon.

**Algorithme 28.2 : lancer de rayon**

On suppose que l'œil est placé à l'origine. Le vecteur $\overrightarrow{D} = (0,0,1)$ sépare l'œil du centre de l'écran composé de $n \times m$ pixels. $\overrightarrow{A} = (1,0,0)$ et $\overrightarrow{D} \wedge \overrightarrow{A} = (0,1,0)$. La scène contient $N$ objets $(O_1, \ldots, O_N)$ et $M$ sources lumineuses $(S_1, \ldots, S_M)$.

**pour** $i = 0$ **à** $n-1$ **faire**
  **pour** $j = 0$ **à** $m-1$ **faire**
    On calcule les coordonnées du point $P(x,y,z)$.

$$x \longleftarrow \frac{\left(i-\frac{n}{2}\right)\tan\frac{\hat{a}}{2}}{n}$$
$$y \longleftarrow \frac{\left(j-\frac{m}{2}\right)\tan\frac{\hat{a}}{2}}{n}$$
$$z \longleftarrow 1$$

    **pour** $k = 1$ **à** $N$ **faire**
      On calcul le point d'intersection entre l'objet $O_k$ et le rayon $\left(O, \overrightarrow{OP}\right)$. On obtient le point $P_k$ et la normale à la surface de l'objet au point d'intersection $\overrightarrow{N_k}$.
    **fin pour** On s'intéresse au point $P_{k^*}$ qui minimise $\overrightarrow{OP} \cdot \overrightarrow{OP_k}$ et vérifie $\overrightarrow{OP} \cdot \overrightarrow{OP_k} > 0$.
    $I = (0,0,0)$ est l'intensité lumineuse du pixel de coordonnées $(i,j)$.
    **pour** $l = 1$ **à** $S$ **faire**
      Si la source lumineuse $S_l$ éclaire le point $P_k$, donc si le rayon $\left(P_{k^*}, \overrightarrow{P_{k^*}S_l}\right)$ n'est intercepté par aucun objet, on calcule uen intensité $I'$ selon le modèle d'illumination

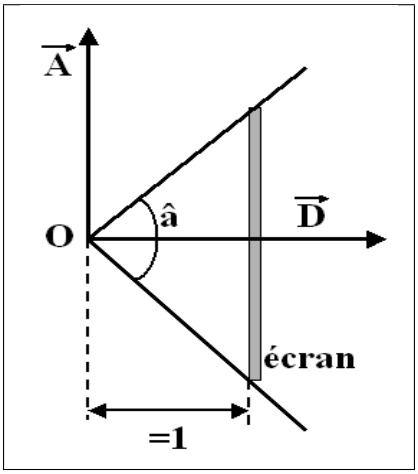



**Figure 28.12 :** *Choix du repère, le vecteur $\overrightarrow{D} \wedge \overrightarrow{A}$ pointe vers le lecteur.*

#### 28.2.1.6 Antialiasing

L'antialiasing a pour but de palier aux défauts de la discrétisation des images en pixels. Il est possible que deux rayons très proches l'un de l'autre aient des trajectoires très différentes, il suffit que l'un d'eux intercepte un objet et l'autre non, que l'un deux soit presque tangente à la surface de l'objet et l'autre ne fasse que l'effleurer. Dans ce cas, l'image montrera quelques irrégularités. Pour y remédier, on lance plusieurs rayons par pixel et l'intensité d'un pixel sera une moyenne des intensités obtenues pour chaque rayon. Sans antialiasing, l'arête oblique d'un cube sera en forme d'escalier, l'antialiasing permet d'introduire un peu de flou plus discret pour l'œil (voir figure 28.13).

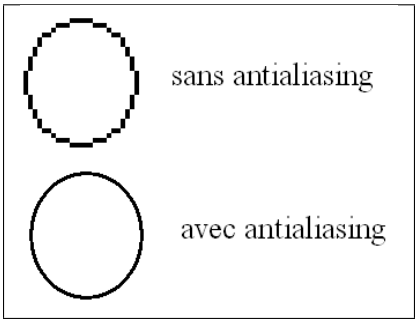



**Figure 28.13 :** *Utilisation de l'antialiasing, les contours ne sont plus saccadés. Le nombre de pixels de l'image n'a pas augmenté mais on ne passe plus brutalement de la couleur du fond à celle de l'objet, ce passage est lissé par l'antialiasing.*

L'antialiasing consiste à découper chaque pixel en un un carré de $p$ petits pixels de côté. On note $c^{nmp}_{ip+p_1 j, jp+p_2}$ le vecteur des intensités des trois registres de couleurs obtenus par lancer de rayon pour le rayon $\left(O, \overrightarrow{OP^n_{ij}}\right)$ pour une image de $np \times mp$ pixels. Le vecteur $\overrightarrow{a^{n,m,p}_{ij}}$ obtenu par antialiasing d'ordre $p$ sur une image de $n \times m$ pixels est :

$$a^{n,m,p}_{ij} = \frac{1}{p^2} \sum_{0 \leqslant p_1, p_2 \leqslant p} c^{nmp}_{pi+p_1, pj+p_2} \tag{28.23}$$

#### 28.2.1.7 Modèle d'illumination de Phong

Selon le modèle d'illumination de Phong, l'intensité (ou quantité de lumière) de chacune des trois composantes (rouge, vert, bleu) d'un rayon interceptant un objet est calculée comme suit :

$$I = \underbrace{k_a C_s}_{I_1} + \sum_j S_j \frac{\overbrace{k_d\, C_s \overrightarrow{N}.\overrightarrow{L_j}}^{I_2} + \overbrace{k_s\, C_r \left( \overrightarrow{N}.\frac{\overrightarrow{L_j}+\overrightarrow{E}}{\left\|\overrightarrow{L_j}+\overrightarrow{E}\right\|} \right)^n}^{I_3}}{C + D_j} \tag{28.24}$$

Chaque intensité modélise une facette de la réalité :

- $I_1$ modélise la lumière ambiente : même en pleine nuit, une faible lumière éclaire les objets en masquant les reliefs : c'est cette lumière qui est modélisée.
- $I_2$ modélise les impressions de relief : un plan incliné par rapport à la source de lumière reçoit moins de lumière qu'un plan faisant face à cette source.
- $I_3$ modélise les reflets : certaines surfaces ont des propriétés réfléchissantes, une surface brillante éblouit plus qu'une surface mate.

Chaque paramètre à un sens physique dans le modèle choisi :

- $k_a, k_d, k_s$ sont des coefficients.
- $C_s$ est la couleur de l'objet.
- $C_r$ est la couleur de réflexion de la surface, en générale égale à celle de la source.
- $S_j$ est l'intensité de la $j^\circ$ source de lumière.
- $\overrightarrow{N}$ est la normale à la surface de l'objet au point d'interception.
- $\overrightarrow{L_j}$ est la direction de la droite passant par le point d'interception et la source de lumière.
- $\overrightarrow{E}$ est la direction de la droite passant par l'œil et le point d'interception.
- $\frac{\overrightarrow{L_j}+\overrightarrow{E}}{\left\|\overrightarrow{L_j}+\overrightarrow{E}\right\|}$ est alors la bissectrice entre $\overrightarrow{L_j}$ et $\overrightarrow{E}$.
- $n$ est un paramètre (égal à 6 dans le programme) ce paramètre modélise l'étendue du reflet ($n$ grand $\Rightarrow$ faible étendue).
- $D_j$ est un coefficient lié à l'objet.
- $C$ est une constante.

La figure 28.14 illustre deux exemples obtenus avec le modèle d'illumination de Phong.

#### 28.2.1.8 Interpolation de Gouraud

Les surfaces complexes des objets sont souvent approchés à l'aide d'une multitude de facettes triangulaires. Ce modèle introduit des discontinuités visibles lors du calcul de l'image de synthèse. Ces points de discontinuités sont les arêtes et les sommets des surfaces. Gouraud propose dans un premier temps de définir la normale à une jonction entre facettes comme la moyenne des vecteurs normaux de chaque facette.

Chaque facette est un triangle plan, la normale le long de cette surface est constante. Dans le modèle de Gourand, afin de réduire encore les discontinuités, la couleur est le barycentre des couleurs calculées à l'aide des normales estimées en chaque sommet des facettes. Par extension, il est possible d'associer les modèles d'illumination de Phong et de Gouraud en supposant que la normale en un point d'une facette est le barycentre des normales en chaque sommet.

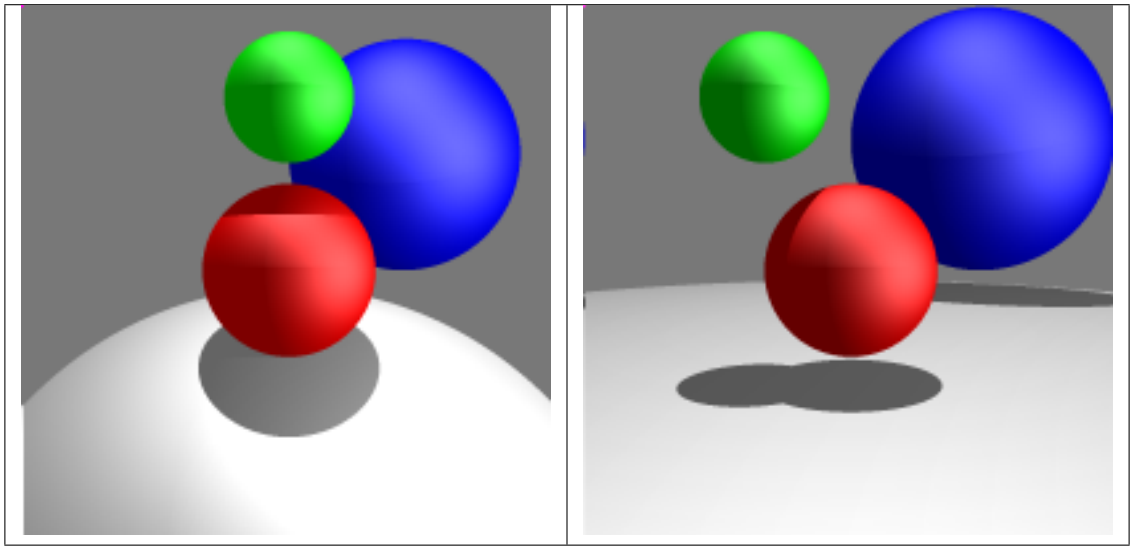


**Figure 28.14 :** *Deux exemples d'image de synthèse obtenues avec le modèle d'illumination de Phong.*

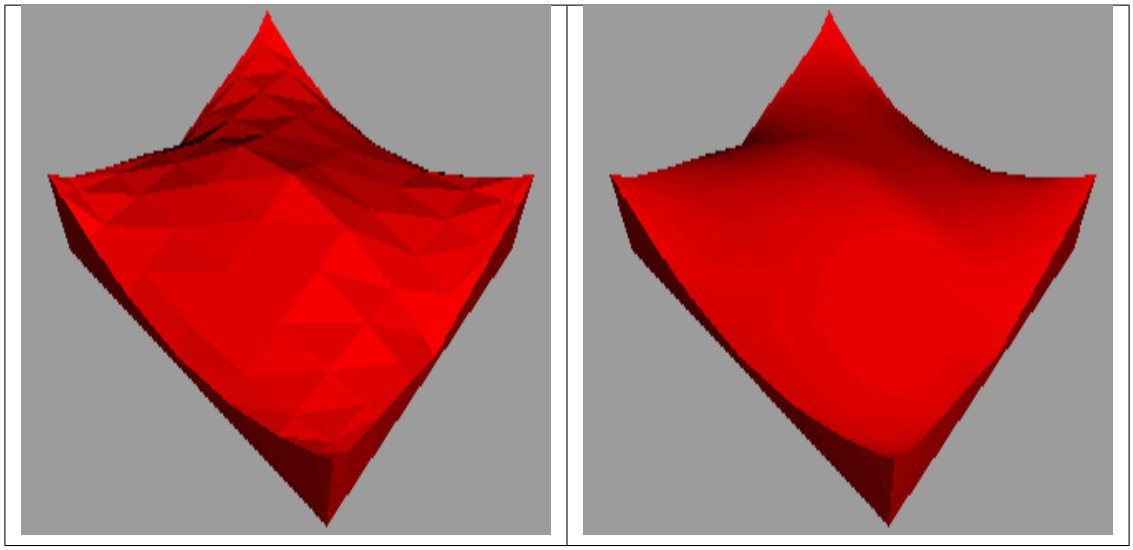


**Figure 28.15 :** *La première est obtenue sans utiliser la méthode Gouraud, la seconde l'utilise.*

#### 28.2.1.9 Horizons

La méthode du lancer de rayon permet d'obtenir des images de synthèse de bonne qualité. Cependant, elle gère difficilement les multiples réflexions qui découle de la présence d'un miroir dans la scène. Lorsqu'un rayon issu de l'œil touche une surface, comment savoir quels seront les rayons qui atteindront une source de lumière après plusieurs reflexions dans des miroirs.

Une autre technique, complémentaire à celle du lancer de rayon permet de traiter ce genre de problème ainsi que celui de la lumière ambiente. C'est la radiosité. Gourmande en calcul, elle découpe les surfaces en morceaux infinitésimaux afin d'estimer une intensité lumineuse ambiente. Il ne s'agit plus d'un rayon qui traverse l'espace mais d'une densité lumineuse qui occupe l'espace.

### 28.2.2 Correction

Le programme est scindé en cinq fichiers. Le premier implémente les objets de base tels que les vecteurs, les pixels, les couleurs, les rayons. Un rayon est défini par une origine et une direction. Cette classe contient aussi sa couleur et le pixel de l'image auquel il correspond. Ce fichier définit aussi l'interface pour un objet et une source. Une source n'est qu'un point. La classe `objet` définit cinq fonctions que tout objet doit posséder pour pouvoir être dessiné.

– La méthode `intersection` calcule l'intersection entre l'objet et un rayon, elle retourne `None` si elle n'existe pas.
– Les méthodes `normale` et `couleur` retourne la normale et la couleur en un point de la surface de l'objet.
– Les méthodes `rayon_refracte` et `rayon_reflechi` retournent les rayons réfractés et réfléchis s'ils existent.

#### 28.2.2.1 Fichier : `image_synthese_base.py`

```
# -*- coding: cp1252 -*-
"""définition des objets permettant de construire une image de synthèse"""
import math
import copy

class vecteur (object) :
    """définit ce qu'est un point"""
    __slots__ = "x","y","z"

    def __str__ (self):
        """pour l'affichage"""
        return "(%3.2f,%3.2f,%3.2f)" % (self.x, self.y, self.z)

    def __init__(self,x,y,z):
        """initialisation"""
        self.x, self.y, self.z = float (x), float (y), float (z)

    def __add__ (self,p):
        """addition de deux points"""
        return vecteur (self.x + p.x, self.y + p.y, self.z + p.z)

    def __neg__ (self):
        """retourne l'opposé d'un vecteur"""
        return vecteur (-self.x,-self.y,-self.z)

    def __iadd__ (self,p):
        """addition de deux points"""
        self.x += p.x
        self.y += p.y
        self.z += p.z
        return self

    def __sub__ (self,p):
        """soustraction de deux points"""
        return vecteur (self.x - p.x, self.y - p.y, self.z - p.z)

    def __isub__ (self,p):
        """soustraction de deux points"""
        self.x -= p.x
        self.y -= p.y
        self.z -= p.z
        return self

    def __mul__ (self,x):
        """multiplication par un scalaire"""
        return vecteur (self.x * x, self.y * x, self.z * x)

    def __imul__ (self,x):
```

```
        """multiplication par un scalaire"""
        self.x *= x
        self.y *= x
        self.z *= x
        return self

    def __div__ (self,x):
        """division par un scalaire"""
        return vecteur (self.x / x, self.y / x, self.z / x)

    def __idiv__ (self,x):
        """division par un scalaire"""
        self.x /= x
        self.y /= x
        self.z /= x
        return self

    def norme2 (self) :
        """retourne la norme du vecteur au carrée"""
        return self.x * self.x + self.y * self.y + self.z * self.z

    def scalaire (self, v) :
        """calcule le produit scalaire entre self et v"""
        return self.x * v.x + self.y * v.y + self.z * v.z

    def vectoriel (self, v) :
        """calcule le produit vectoriel entre self et v"""
        res   = vecteur (0,0,0)
        res.x = self.y * v.z - self.z * v.y
        res.y = self.z * v.x - self.x * v.z
        res.z = self.x * v.y - self.y * v.x
        return res

    def norme (self) :
        """retourne la norme du vecteur"""
        return math.sqrt (self.norme2 ())

    def renorme (self) :
        """renorme ce vecteur"""
        n = self.norme ()
        if n > 0 : return self / n
        else : return copy.copy (self)

    def cosinus (self, v) :
        """retourne le cosinus de entre le vecteur self et le vecteur r"""
        sc = self.scalaire (v)
        n1 = self.norme ()
        n2 = v.norme ()
        n  = n1 * n2
        if n == 0 : return 0
        return float (sc) / float (n)

    def sinus (self, v, norm) :
        """retourne le sinus de entre le vecteur self et le vecteur r,
        norm est un vecteur normal et de norme 1 permettant d'orienter
        le plan dans lequel se trouve les deux vecteurs dont il faut mesurer le sinus"""
        sc = self.vectoriel (v)
        n1 = self.norme ()
        n2 = v.norme ()
        n  = n1 * n2
        if n == 0 : return 0
        return sc.scalaire (norm) / float (n)

    def angle (self, v, norm) :
        """retourne l'angle entre les vecteur self et v,
```

```
        retourne un angle compris entre -pi et pi,
        norm est la direction du vecteur normal au plan des deux vecteurs"""
        cos     = self.cosinus (v)
        sin     = self.sinus (v, norm)
        angle   = math.atan2 (sin, cos)
        if angle > math.pi : angle -= math.pi * 2
        return angle

    def diff_abs (self, v):
        """retourne la somme des valeurs absolues des différentes entre coordonnées"""
        r  = abs (self.x - v.x)
        r += abs (self.y - v.y)
        r += abs (self.z - v.z)
        return r

    def __eq__ (self, v) :
        """définit l'égalité entre deux vecteurs"""
        if v == None : return False
        return self.diff_abs (v) < 1e-10

    def __ne__ (self, v) :
        """définit l'égalité entre deux vecteurs"""
        if v == None : return True
        return self.diff_abs (v) > 1e-10


class couleur (vecteur) :
    """une couleur est un vecteur dont les coordonnées sont comprises entre 0 et 1,
    x <--> rouge, y <--> vert, z <--> bleu"""

    def __init__ (self, x,y,z) :
        vecteur.__init__(self, x,y,z)
        self.borne ()

    def borne (self) :
        """si une couleur est hors bornes, réajuste la couleur, prend le maximum devient 1,
        les autres intensités sont ajustées selon ce facteur d'échelle"""
        if self.x < 0 : self.x = 0
        if self.y < 0 : self.y = 0
        if self.z < 0 : self.z = 0
        m = max (self.x, self.y)
        m = max (m, self.z)
        if m > 1 :
            self.x /= m
            self.y /= m
            self.z /= m

    def __add__ (self,p):
        """addition de deux couleurs"""
        return couleur (self.x + p.x, self.y + p.y, self.z + p.z)

    def produit_terme (self, v) :
        """effectue un produit terme à terme"""
        return couleur (self.x * v.x, self.y * v.y, self.z * v.z)

    def __mul__ (self,x):
        """multiplication par un scalaire"""
        return couleur (self.x * x, self.y * x, self.z * x)



class repere (object) :
    """définition d'un repère orthonormé"""

    def __init__ (self,   origine = vecteur (0,0,0), \
```

```
                          axex    = vecteur (1,0,0), \
                          axey    = vecteur (0,1,0), \
                          axez    = vecteur (0,0,1)) :
        """initialisation, origine et les trois axes"""
        self.origine    = origine
        self.x          = axex
        self.y          = axey
        self.z          = axez

    def coordonnees (self, v) :
        """on suppose que les coordonnées de v sont exprimées dans ce repère,
        calcule les coordonnées de v dans le repère d'origine"""
        res      = copy.copy (self.origine)
        res.x  += v.x * self.x.x +  v.y * self.y.x + v.z * self.z.x
        res.y  += v.x * self.x.y +  v.y * self.y.y + v.z * self.z.y
        res.z  += v.x * self.x.z +  v.y * self.y.z + v.z * self.z.z
        return res

    def __str__ (self):
        """affichage"""
        s  = "origine : " + str (self.origine) + "\n"
        s += "axe des x : " + str (self.x) + "\n"
        s += "axe des y : " + str (self.y) + "\n"
        s += "axe des z : " + str (self.z) + "\n"
        return s

class pixel (object) :
    """définit ce qu'est un pixel"""
    __slots__ = "x","y"

    def __init__(self,x,y):
        """initialisation"""
        self.x, self.y = int (x), int (y)

    def __str__ (self):
        """pour l'affichage"""
        return "(%d, %d)" % (self.x, self.y)




class rayon (object) :
    """définit ce qu'est un rayon"""
    __slots__ = "origine", "direction", "pixel", "couleur"

    def __init__ (self, origine, direction, pixel, couleur):
        """initialisation"""
        self.origine, self.direction, self.pixel, self.couleur = \
                origine, direction, pixel, couleur

    def __str__ (self):
        """pour l'affichage"""
        s  = "origine   : "     + str (self.origine)
        s += " direction : "    + str (self.direction)
        s += " pixel : "        + str (self.pixel)
        s += " couleur : "      + str (self.couleur)
        return s


class objet (object):
    """définit l'interface pour un objet à dessiner dans une image de synthese"""

    def intersection (self, r) :
        """retourne le point d'intersection avec le rayon r,
        retourne None s'il n'y pas d'intersection"""
```

```
        return None

    def normale (self, p, rayon) :
        """retourne la normale au point de coordonnée p, et connaissant le rayon"""
        return None

    def couleur_point (self, p) :
        """retourne la couleur au point de coordonnée p"""
        return None

    def rayon_refracte (self, rayon, p) :
        """retourne le rayon réfracté au point p de la surface,
        si aucune, retourne None"""
        return None

    def rayon_reflechi (self, rayon, p) :
        """retourne le rayon réfléchi au point p de la surface,
        si aucune, retourne None"""
        return None

    def phong_coefficient (self):
        """retourne un coefficient propre à l'objet pour
        le modèle d'illumination de Phong"""
        return float (0)



class source (object) :
    """définition d'une source ponctuelle"""
    __slots__ = "origine", "couleur"

    def __init__ (self, origine, couleur):
        """initialisation"""
        self.origine, self.couleur = origine, couleur

    def __str__ (self) :
        """affichage"""
        return "source : " + str (self.origine) + " couleur : " + str (self.couleur)

if __name__ == "__main__" :
    v = vecteur (0,1,2)
    u = vecteur (0,1,2)
    w = u + v
    print u,v,w
    print w * 6
    p = pixel (5,5)
    print p
    c = couleur (1,1,1)
    print c
    r = rayon (u,w,p,c)
    print r
    s = source (v, c)
    print s
```

#### 28.2.2.2 Fichier : image_synthese_sphere.py

Le fichier suivant décrit une sphère. Cette classe hérite de `objet` et surcharge les méthodes `intersection`, `normale`, `couleur`. Cette sphère ne réfléchit aucun rayon et ne produit aucun rayon réfracté.

```
# -*- coding: cp1252 -*-
"""définition d'une sphère"""
import image_synthese_base as base
```

```
import math

class sphere (base.objet):
    """définit une sphère"""
    __slots__ = "centre", "rayon", "couleur"

    def __init__ (self, centre, rayon, couleur):
        """initialisation"""
        self.centre, self.rayon, self.couleur = centre, float (rayon), couleur

    def intersection (self, r) :
        """retourne le point d'intersection avec le rayon r,
        retourne None s'il n'y pas d'intersection"""
        oc      = self.centre - r.origine
        vn      = r.direction.norme2 ()
        s       = r.direction.scalaire (oc)
        delta   = s*s - vn * (oc.norme2 () - self.rayon * self.rayon)
        if delta < 0 : return None
        delta   = math.sqrt (delta)
        l1      = (s - delta) / vn
        l2      = (s + delta) / vn

        if      0 < l1 < l2 :   l = l1
        elif    l1 < 0 < l2 :   l = l2
        elif    0 < l2 < l1 :   l = l2
        elif    l2 < 0 < l1 :   l = l1
        else :                  l = None

        if l == None : return None

        v = r.origine + r.direction * l
        return v

    def normale (self, p, rayon) :
        """retourne la normale au point de coordonnée p"""
        v = (p - self.centre) / self.rayon
        return v

    def couleur_point (self, p) :
        """retourne la couleur au point de coordonnée p"""
        return self.couleur

    def __str__ (self):
        """affichage"""
        s  = "sphère --- centre : " + str (self.centre)
        s += " rayon : " + str (self.rayon)
        s += " couleur : " + str (self.couleur)
        return s



if __name__ == "__main__" :
    s = sphere (base.vecteur (0,0,0), 5, base.couleur (0,1,0))
    r = base.rayon (    base.vecteur (10,0,0), base.vecteur (1,0,0), \
                        base.pixel (0,0), base.couleur (0,0,0))
    print s
    print r
    p = s.intersection (r)
    print p
```

#### 28.2.2.3 Fichier : image_synthese_scene.py

Le troisième fichier définit ce qu'est une scène, elle implémente les fonctions qui permettent de calculer les intersections entre un rayon et les objets. La méthode `modele_illumination` implémente un modèle d'illumination simplifiée pour lequel l'intensité des couleurs est proportionnelle au cosinus entre la normale à un objet et la provenance de la lumière. Ce petit exemple, qui ne dépend d'aucun paramètre, permet de corriger les différentes fonctions jusqu'ici implémentées. En effet, il est préférable de tester son programme avant que celui-ci ne soit terminé. La détection des erreurs est une étape parfois aussi longue que sa conception. Il est important de réduire le plus possible les sources possibles d'erreurs en ne testant que des petites portions de codes.

```
# -*- coding: cp1252 -*-
"""définition d'une scène"""
import image_synthese_base as base
import image_synthese_sphere as obj
import math
import pygame

class scene (object):
    """définit une scène, les axes x,y sont ceux de l'écran,
    z-1 est la distance à l'écran du point (x,y,z)"""

    def __init__ (self, repere, alpha, x,y) :
        """définit la position de l'oeil, l'angle d'ouverture,
        et la taille de l'écran"""
        self.repere = repere
        self.alpha  = float (alpha)
        self.dim    = (int (x), int (y))

    def ajoute_source (self, source):
        """ajoute une source ponctuelle de lumière"""
        if not self.__dict__.has_key ("sources") : self.sources = []
        self.sources.append (source)

    def ajoute_objet (self, objet):
        """ajoute un objet à la scène"""
        if not self.__dict__.has_key ("objets") : self.objets = []
        self.objets.append (objet)

    def __str__ (self) :
        """affichage"""
        s  = "scène ----------------------------\n"
        s += "repère : " + str (self.repere) + "\n"
        s += "angle d'ouverture : " + str (self.alpha) + "\n"
        s += "dimension de l'écran : " + str (self.dim) + "\n"
        if self.__dict__.has_key ("sources") :
            for a in self.sources : s += "   " +str (a) + "\n"
        if self.__dict__.has_key ("objets") :
            for a in self.objets :  s += "   " + str (a) + "\n"
        return s

    def intersection (self, rayon) :
        """calcule le point d'intersection entre un rayon et le plus proche des objets,
        retourne l'objet et le point d'intersection"""
        if not self.__dict__.has_key ("objets") : return None, None
        p       = rayon.origine
        sp,so   = None, None
        for o in self.objets :
            i = o.intersection (rayon)
            if i == None : continue
            if rayon.direction.scalaire (i - p) <= 0 : continue
            if i == rayon.origine : continue
            if sp == None :
```

```
            sp  = i
            so  = o
        else :
            v = i - p
            d = sp - p
            if v.norme2 () < d.norme2 () :
                sp = i
                so = o
    return so, sp

def sources_atteintes (self, p) :
    """retourne la liste des sources atteintes depuis une position p de l'espace,
    vérifie qu'aucun objet ne fait obstacle"""
    res = []
    for s in self.sources:
        r   = base.rayon (s.origine, p - s.origine, base.pixel (0,0), s.couleur)
        o,i = self.intersection (r)
        if i == None : continue
        if (i - p).norme2 () < 1e-10 :   # possible problème d'arrondi
            res.append (s)
            continue
    return res

def construit_rayon (self, pixel) :
    """construit le rayon correspondant au pixel pixel"""
    x = (pixel.x - self.dim [0] / 2) * math.tan (self.alpha / 2) / min (self.dim)
    y = (pixel.y - self.dim [1] / 2) * math.tan (self.alpha / 2) / min (self.dim)
    v = base.vecteur (x,y,1)
    r = base.rayon (self.repere.origine, self.repere.coordonnees (v), \
                        pixel, base.couleur (1,1,1))
    return r

def modele_illumination (self, rayon, p, obj, source) :
    """calcule la couleur pour un rayon donné, un point p, un objet obj,
    et une source de lumière source"""
    n   = obj.normale (p, rayon)
    cos = n.cosinus (source.origine - p)
    cl  = obj.couleur_point (p) * cos
    cl  = cl.produit_terme (rayon.couleur)
    return cl

def couleur_fond (self) :
    """retourne la couleur du fond"""
    return base.couleur (0,0,0)

def rayon_couleur (self, rayon, ref = True) :
    """retourne la couleur d'un rayon connaissant les objets,
    cette fonction doit être surchargée pour chaque modèle d'illumination,
    si ref == True, on tient compte des rayons réfracté et réfléchi"""

    list_rayon = [ rayon ]
    c = base.couleur (0,0,0)
    b = False
    while len (list_rayon) > 0 :
        r   = list_rayon.pop ()
        o,p = self.intersection (r)

        if p == None : continue

        if ref :
            t   = o.rayon_refracte (r, p)
            if t != None : list_rayon.append (t)
            t   = o.rayon_reflechi (r, p)
            if t != None : list_rayon.append (t)
```

```
            sources = self.sources_atteintes (p)
            if len (sources) == 0 : return base.couleur (0,0,0)
            for s in sources :
                cl  = self.modele_illumination (r, p, o, s)
                c  += cl
                b   = True

        if not b : c = self.couleur_fond ()
        else : c.borne ()
        return c

    def construit_image (self, screen):
        """construit l'image de synthèse où screen est un objet du module pygame"""
        count   = 0
        nbpixel = int (self.dim [0] * self.dim [1] / 100)
        for y in xrange (0, self.dim [1]) :
            for x in xrange (0, self.dim [0]) :
                p = base.pixel (x,y)
                r = self.construit_rayon (p)
                c = self.rayon_couleur (r, True)
                q = (p.x,self.dim [1] - p.y - 1)
                d = (int (c.x * 255), int (c.y * 255), int (c.z * 255))
                pygame.draw.line (screen, d, q,q)
                count += 1
                if count % 150 == 0 : pygame.display.flip ()
                if count % nbpixel == 0 : print "avancement " , count // nbpixel , "%"
        pygame.display.flip ()


def attendre_clic ():
    """dessine une croix sur l'écran et attend la pression d'un clic de souris"""
    reste = True
    while reste:
        for event in pygame.event.get():
            if event.type == pygame.MOUSEBUTTONUP :
                reste = False
                break


if __name__ == "__main__" :
    s = scene (base.repere (), math.pi / 1.5, 400, 300)
    s.ajoute_source ( base.source (base.vecteur (0,10,10), \
                        base.couleur (1,1,1) ) )
    s.ajoute_source ( base.source (base.vecteur (10,10,5), \
                        base.couleur (0.5,0.5,0.5) ) )
    s.ajoute_objet  ( obj.sphere  (base.vecteur (0,0,12), \
                        3, base.couleur (1,0,0) ) )
    s.ajoute_objet  ( obj.sphere  (base.vecteur (0,-400,12), \
                        396, base.couleur (0.5,0.5,0.5) ) )
    print s

    screen  = pygame.display.set_mode (s.dim)
    screen.fill ((255,255,255))
    s.construit_image (screen)

    print "image terminée"
    attendre_clic ()
```

#### 28.2.2.4 Fichier : image_synthese_phong.py

Le programme précédent permet d'obtenir l'image de la figure 28.16. La grande sphère et le bas de la sphère centrale sont plutôt réussis. En revanche, le haut de sphère centrale est uniformément rouge, le

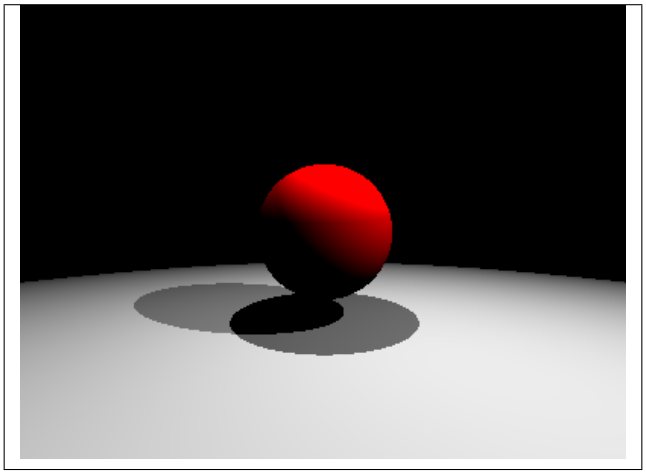


**Figure 28.16 :** *Image de synthèse créée lors de l'éxecution du fichier* `image_synthese_scene.py`. *Le modèle d'illumination est plus simple que celui de Phong. L'intensité des couleurs perceptible par l'observateur est proportionnelle au cosinus entre la normale à la sphère et la direction de la lumière. La scène est composée de deux sources de lumière, une petite sphère, une grande qui joue le rôle du sol. Aucun anti-aliasing n'a été implémenté et on peut observer que le contour de la sphère est en escalier.*

modèle est encore éloigné de la réalité. Le dernier fichier construit la classe `scene_phong` qui hérite de `scene`. Elle redéfinit la méthode `modele_illumination` qui calcule la couleur d'un rayon lancé depuis l'œil.

```
# -*- coding: cp1252 -*-
"""implémentation du modèle d'illumination de Phong"""
import image_synthese_scene as scene
import image_synthese_base as base
import image_synthese_sphere as obj
import math
import pygame


class scene_phong (scene.scene):
    """définit une scène et utilise le modèle d'illumination de Phong
    pour construire l'image de synthèse"""

    def __init__ (self, repere, alpha, x,y,
                  ka          = 0.1,
                  kb          = 0.8,
                  kc          = 0.3,
                  reflet      = 6,
                  fond        = base.couleur (200,200,200)) :
        """définit la position de l'oeil, l'angle d'ouverture,
        et la taille de l'écran"""
        scene.scene.__init__ (self, repere, alpha, x, y)
        self.ka, self.kb, self.kc   = ka,kb,kc
        self.reflet                 = reflet
        self.fond                   = fond
        self.constante              = float (1)

    def __str__ (self) :
        """affichage"""
        s  = scene.scene.__str__ (self)
        s += "ka = %1.3f   kb = %1.3f   kc = %1.3f\n" % (self.ka,self.kb,self.kc)
        s += "reflet " + str (self.reflet) + "\n"
        s += "couleur du fond " + str (self.fond) + "\n"
        return s

    def couleur_fond (self) :
```

```
        """retourne la couleur du fond"""
        return self.fond * self.ka

    def modele_illumination (self, rayon, p, obj, source) :
        """calcule la couleur pour un rayon donné, un point p, un objet obj,
        et une source de lumière source"""
        n   = obj.normale (p, rayon).renorme ()
        vr  = rayon.direction.renorme ()
        vr *= float (-1)
        vs  = source.origine - p
        vs  = vs.renorme ()
        bi  = vs + vr
        bi  = bi.renorme ()

        # premier terme
        cos      = n.scalaire (vs)
        couleur  = source.couleur.produit_terme (obj.couleur_point (p)) * (cos * self.kb)

        # second terme : reflet
        cos      = n.scalaire (bi) ** self.reflet
        couleur += source.couleur.produit_terme (source.couleur) * (cos * self.kc)
        couleur  = couleur.produit_terme (rayon.couleur)

        return couleur


if __name__ == "__main__" :
    s = scene_phong (base.repere (), math.pi / 1.5, 400, 300)

    s.ajoute_source ( base.source (base.vecteur (0,10,10), \
                        base.couleur (1,1,1) ) )
    s.ajoute_source ( base.source (base.vecteur (10,10,5), \
                        base.couleur (0.5,0.5,0.5) ) )
    s.ajoute_objet  ( obj.sphere  (base.vecteur (0,0,12), \
                        3, base.couleur (1,0,0) ) )
    s.ajoute_objet  ( obj.sphere  (base.vecteur (0,-400,12), \
                        396, base.couleur (0.5,0.5,0.5) ) )
    print s

    screen  = pygame.display.set_mode (s.dim)
    screen.fill ((255,255,255))
    s.construit_image (screen)

    print "image terminée"
    scene.attendre_clic ()
```

La figure 28.17 est la même scène que la figure 28.16 mais calculée avec le modèle d'illumination de Phong implémenté dans le dernier fichier. La sphère rouge paraît plus pâle, la couleur est attenuée par le reflet.

#### 28.2.2.5 Fichier : `image_synthese_facette.py`

Le dernier fichier implémente les facettes triangulaires et quadrilatères. Une facette triangulaire dérive de la classe `objet` définie dans le module `image_synthese_base.py`. Elle définit l'intersection entre un rayon et une facette. L'objet qui implémente une facette en forme de quadrilatère dérive à son tour d'une facette triangulaire. Le résultat de ce programme est donné par la figure 28.18.

Lorsque les interfaces des objets sont bien conçues, l'ajout de classe d'objets devient une tâche indépendante du choix d'un modèle d'illumination. Pour construire un nouvel objet, il n'est pas nécessaire de connaître l'objet `scene` ou `scene_phong`, il suffit de surcharger les méthodes de la classe `objet` pour adapter son comportement à celui du nouvel objet.

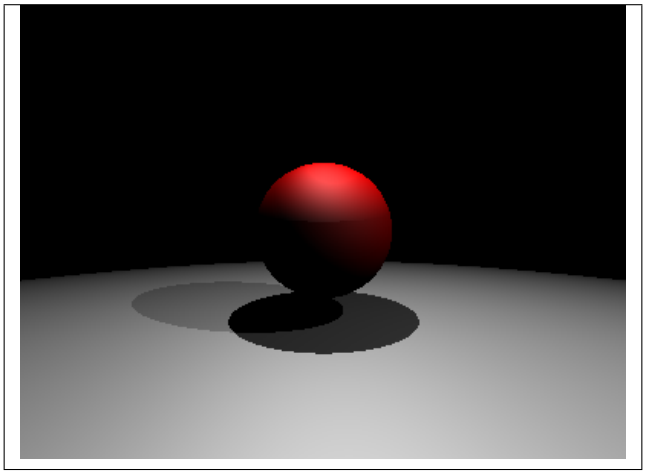


**Figure 28.17 :** *Cette scène est la même que celle de la figure 28.16 mais calculée avec le modèle d'illumination de Phong.*

```
# -*- coding: cp1252 -*-
"""définition d'une sphère"""
import image_synthese_base as base
import image_synthese_sphere as obj
import image_synthese_phong as scene
import image_synthese_scene as scene_p
import pygame
import math

class facette (base.objet):
    """définit un triangle dans l'espace"""

    def __init__ (self, a,b,c, couleur):
        """initialisation"""
        self.a, self.b, self.c = a,b,c
        ab = b - a
        ac = c - a
        self.vnorm = ab.vectoriel (ac)
        self.vnorm = self.vnorm.renorme ()
        self.couleur = couleur

    def intersection_plan (self, r) :
        """retourne le point d'intersection entre le plan et le rayon r"""
        if r.direction.scalaire (self.vnorm) == 0 :
            return None
        oa      = self.a - r.origine
        l       = self.vnorm.scalaire (oa) / self.vnorm.scalaire (r.direction)
        p       = r.origine + r.direction * l
        return p

    def point_interieur (self, p) :
        """dit si un point appartient à l'intérieur du triangle"""
        pa      = self.a - p
        pb      = self.b - p
        pc      = self.c - p
        theta   = pa.angle (pb, self.vnorm)
        theta  += pb.angle (pc, self.vnorm)
        theta  += pc.angle (pa, self.vnorm)
        theta   = abs (theta)
        if theta >= math.pi * 0.9 : return True
        else : return False
```

```
    def intersection (self, r) :
        """retourne le point d'intersection avec le rayon r,
        retourne None s'il n'y pas d'intersection"""
        p = self.intersection_plan (r)
        if p == None : return None
        if self.point_interieur (p) : return p
        else : return None

    def normale (self, p, rayon) :
        """retourne la normale au point de coordonnée p et connaissant le rayon"""
        if rayon.direction.scalaire (self.vnorm) < 0 :
            return self.vnorm
        else :
            return - self.vnorm

    def couleur_point (self, p) :
        """retourne la couleur au point de coordonnée p"""
        return self.couleur

    def __str__ (self):
        """affichage"""
        s  = "facette --- a : " + str (self.a)
        s += " b : " + str (self.b)
        s += " c : " + str (self.c)
        s += " couleur : " + str (self.couleur)
        return s


class rectangle (facette):
    """définit un rectangle dans l'espace"""

    def __init__ (self, a,b,c,d, couleur):
        """initialisation, si d == None, d est calculé comme étant
        le symétrique de b par rapport au milieu du segment [ac]"""
        facette.__init__(self, a,b,c, couleur)
        if d != None : self.d = d
        else :
            i       = (a + c) / 2
            self.d  = b + (i-b) * 2

    def point_interieur (self, p) :
        """dit si un point appartient à l'intérieur du triangle"""
        pa      = self.a - p
        pb      = self.b - p
        pc      = self.c - p
        pd      = self.d - p
        theta   = pa.angle (pb, self.vnorm)
        theta  += pb.angle (pc, self.vnorm)
        theta  += pc.angle (pd, self.vnorm)
        theta  += pd.angle (pa, self.vnorm)
        theta   = abs (theta)
        if theta >= math.pi * 0.9 : return True
        else : return False

    def __str__ (self):
        """affichage"""
        s  = "rectangle --- a : " + str (self.a)
        s += " b : " + str (self.b)
        s += " c : " + str (self.c)
        s += " d : " + str (self.d)
        s += " couleur : " + str (self.couleur)
        return s
```

```
if __name__ == "__main__" :

    s = scene.scene_phong (base.repere (), math.pi / 1.5, 400, 300)

    s.ajoute_source ( base.source (base.vecteur (0,8,8), \
                        base.couleur (0.6,0.6,0.6) ) )
    s.ajoute_source ( base.source (base.vecteur (10,0,0), \
                        base.couleur (0.6,0.6,0.6) ) )
    s.ajoute_source ( base.source (base.vecteur (8,8,4.5), \
                        base.couleur (0.6,0.6,0.6) ) )
    s.ajoute_objet  ( obj.sphere  (base.vecteur (1,0,5), \
                        1, base.couleur (1,0,0) ) )
    s.ajoute_objet  ( obj.sphere  (base.vecteur (0,-400,12), \
                        396, base.couleur (0.5,0.5,0.5) ) )
    s.ajoute_objet (facette ( base.vecteur (0,-2.5,6),   \
                              base.vecteur (-2,-2.5,3), \
                              base.vecteur (1,-3.5,4.5), \
                              base.couleur (0.2,0.8,0)))
    s.ajoute_objet (rectangle ( base.vecteur (0,-2.5,6),   \
                              base.vecteur (-2,-2.5,3), \
                              base.vecteur (-2,2.8,3.5), \
                              None, \
                              base.couleur (0,0,1)))
    print s

    screen  = pygame.display.set_mode (s.dim)
    screen.fill ((255,255,255))
    s.construit_image (screen)

    print "image terminée"
    scene_p.attendre_clic ()
```

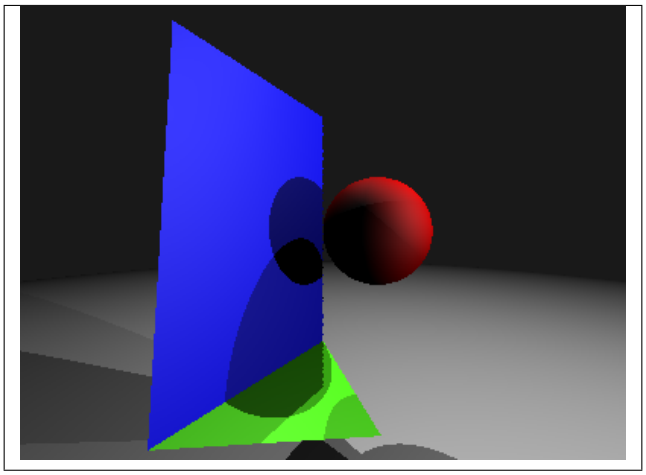


**Figure 28.18 :** *Cette scène est la même que celle de la figure 28.17, une facette triangulaire et une facette en forme de quadrilatère ont été ajoutées.*

Huitième partie

# EXERCICES AUTOUR DE SUJETS DÉJÀ ABORDÉS

# Chapitre 29

# Exercices plutôt mathématiques

---

## 29.1 Tracé d'une ligne sans multiplication

### 29.1.1 Enoncé

**1)**

L'objectif est ici de modifier un programme existant afin de l'adapter à une tâche sensiblement équivalente. On utilise pour cela le programme référencé à l'adresse internet *http ://perso.wanadoo.fr/x.dupre/programmation/python_cours_exemple_programme_html/bresenham_ligne.html*. Bien qu'il soit repris dans cet énoncé, l'algorithme qu'il implémente est décrit en détail dans le document fourni à l'adresse *http ://perso.wanadoo.fr/x.dupre/programmation/python_cours_exemple.pdf*

Ce programme implémente l'algorithme de Bresenham qui permet de tracer une ligne sans effectuer ni multiplication ni division. La droite obtenue est 8-connexe, c'est-à-dire que deux pixels voisins de cette droite ont au moins un sommet en commun. L'objectif de ce TD est d'adapter le programme précédent afin que la droite obtenue soit 4-connexe, c'est-à-dire que deux pixels voisins ont une arête en commun. L'algorithme suivant permet de tracer une ligne en 8-connexité.

**Algorithme 29.1 : tracé d'une ligne (Bresenham)**
On suppose qu'on doit tracer la ligne rejoignant les pixels de coordonnées $(x_1, y_1)$ à $(x_2, y_2)$. On suppose de plus que : $x_2 > x_1$, $y_2 \geqslant y_1$ et $x_2 - x_1 \geqslant y_2 - y_1$. Les autres cas sont obtenus par symétrie par rapport aux axes et aux premières bissectrices. On suppose que $(x_1, y_1, x_2, y_2) \in \mathbb{N}^4$.

**Etape A : initialisation**

$v_x \longleftarrow x_2 - x_1$
$v_y \longleftarrow y_2 - y_1$
$b \longleftarrow \frac{v_x}{2}$
$y \longleftarrow y_1$

initialisation

**Etape B : étape principale**

**pour** $x = x_1$ **à** $x_2$ **faire**
    dessiner le point de coordonnées $(x, y)$
    $b \longleftarrow b - v_y$
    **si** $b < 0$ **alors**
        $y \longleftarrow y + 1$
        $b \longleftarrow b + v_x$
    **fin si**
**fin pour**

Le programme[1] qui suit implémente cet algorithme qui permet de tracer une ligne en 8-connexité.

```
# -*- coding: cp1252 -*-
"""ce module contient la fonction trace_ligne qui retourne l'ensemble des pixels
```

1. Ce programme utilise pour les affichages le module `pygame` disponible à l'adresse http ://www.pygame.org/.

```
concernés par le tracé d'une ligne en 8-connexité entre deux pixels"""
import pygame                 # pour les affichages
import random

def trace_ligne_simple (x1,y1,x2,y2):
    """trace une ligne entre les points de coordonnées (x1,y1) et (x2,y2),
    on suppose que x2 > x1, y2 >= y1,
    retourne la ligne sous la forme d'un ensemble de pixels (x,y)"""

    if y2 - y1 <= x2 - x1 :  # droite en dessous de la première bissectrice
        vx = x2 - x1
        vy = y2 - y1
        b  = vx / 2
        y  = y1
        x  = x1

        ligne = []
        while x <= x2 :
            ligne.append ((x,y))
            b -= vy
            x += 1
            if b < 0:
                b += vx
                y += 1
        return ligne
    else :                   # droite au dessus de la première bissectrice
        vx = x2 - x1
        vy = y2 - y1
        b  = vy / 2
        y  = y1
        x  = x1

        ligne = []
        while y <= y2 :
            ligne.append ((x,y))
            b -= vx
            y += 1
            if b < 0:
                b += vy
                x += 1
        return ligne

def trace_ligne (x1,y1,x2,y2):
    """trace une ligne entre les points de coordonnées (x1,y1) et (x2,y2),
    aucune contrainte sur les coordonnées,
    retourne la ligne sous la forme d'un ensemble de pixels (x,y)"""

    if x1 == x2 :
        if y1 <= y2: return [ (x1, i) for i in xrange (y1,y2+1) ]
        else : return [ (x1, i) for i in xrange (y2,y1+1) ]

    if y1 == y2 :
        if x1 <= x2: return [ (i, y1) for i in xrange (x1,x2+1) ]
        else : return [ (i, y1) for i in xrange (x2,x1+1) ]

    if x1 < x2 :
        if y1 < y2 :
            return trace_ligne_simple (x1,y1,x2,y2)
        else :
            ligne = trace_ligne_simple (x1,y2,x2,y1)
            return [ (x,y1 + y2 - y) for (x,y) in ligne ]

    if x2 < x1 :
        if y1 < y2 :
            ligne = trace_ligne_simple (x2,y1,x1,y2)
```

```
            return [ (x1 + x2 - x, y) for (x,y) in ligne ]
        else :
            ligne = trace_ligne_simple (x2,y2,x1,y1)
            return [ (x1 + x2 - x, y1 + y2 - y) for (x,y) in ligne ]

def display_ligne (ligne, screen):
    """affiche une ligne à l'écran"""
    color = 0,0,0
    for p in ligne:
        pygame.draw.line (screen, color, p,p)
    pygame.display.flip ()


def attendre_clic (screen):
    """attend la pression d'un clic de souris pour continuer"""
    reste = True
    while reste:
        for event in pygame.event.get():
            if event.type == pygame.MOUSEBUTTONUP :
                reste = False
                break

if __name__ == "__main__" :
    pygame.init ()

    size      = width, height = x,y = 200, 200
    black     = 0, 0, 0
    white     = 255,255,255
    screen    = pygame.display.set_mode(size)
    screen.fill (white)

    print trace_ligne (0,0, 7,3)
    # affiche [(0, 0), (1, 0), (2, 1), (3, 1), (4, 2), (5, 2), (6, 3), (7, 3)]

    for n in xrange (0,10):
        x1      = random.randint (0,x-1)
        y1      = random.randint (0,y-1)
        x2      = random.randint (0,x-1)
        y2      = random.randint (0,y-1)
        ligne   = trace_ligne (x1,y1,x2,y2)
        display_ligne (ligne, screen)

    attendre_clic (screen)
```

**2)**

L'objectif de cette seconde question est de s'inspirer de l'algorithme 29.1 afin de tracer - sans multiplication ni division - une ellipse de centre $(x_0, y_0)$, de demi grand axe $a$ et de demi petit axe $b$. Son équation cartésienne est la suivante :

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

L'algorithme est décrit dans le document fourni à l'adresse *http ://perso.wanadoo.fr/x.dupre/programmation/python_ cours_ exemple.pdf*. Il y est également démontré que le tracé obtenu ne s'écarte pas de plus d'un

demi-pixel de la courbe de l'ellipse.

**Algorithme 29.2 : tracé d'une ellipse (Bresenham)**
On suppose qu'on doit tracer l'ellipse d'équation $\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$. On suppose que $a\neq 0$ et $b\neq 0$ auquel cas, les ellipses sont des segments. On suppose $(a,b)\in\mathbb{N}^2$.
**Etape A : initialisation**
$A \longleftarrow a^2$
$B \longleftarrow b^2$
$x \longleftarrow a$
$y \longleftarrow 0$
$v_x \longleftarrow aB$
$v_y \longleftarrow 0$
$\beta \longleftarrow \frac{v_x}{2}$

Le point $(x,y)$ est la position du pixel qui fera partie de l'ellipse. Le vecteur $(v_x,v_y)$ représente le vecteur normal à l'ellipse au point $(x,y)$. Ce vecteur est constamment égal à $(v_x,v_y)=(xB,yA)=\left(xb^2,ya^2\right)$.

**Etape B : tracé du premier huitième**
**tant que** $(v_x\geqslant v_y)$ **faire**
dessiner le point de coordonnées $(x,y)$
$y \longleftarrow y+1$
$v_y \longleftarrow v_y+A$
$\beta \longleftarrow \beta-v_y$
**si** $\beta<0$ **alors**
$x \longleftarrow x-1$
$v_x \longleftarrow v_x-B$
$\beta \longleftarrow \beta+v_x$
**fin si**
**fin tant que**

**Etape C : tracé du second huitième**
**tant que** $(x\geqslant 0)$ **faire**
dessiner le point de coordonnées $(x,y)$
$x \longleftarrow x-1$
$v_x \longleftarrow v_x-B$
$\beta \longleftarrow \beta+v_x$
**si** $\beta>0$ **alors**
$y \longleftarrow y+1$
$v_y \longleftarrow v_y+A$
$\beta \longleftarrow \beta-v_y$
**fin si**
**fin tant que**

**Etape D : autres huitièmes**
Le reste de l'ellipse est obtenu par symétrie par rapport aux deux axes du répère orthonormé du plan.

### 29.1.2 Correction

**1)**

Par rapport au programme initial, deux lignes ont été ajoutées, deux lignes ont été déplacées, elles sont suivies d'un commentaires commençant par # − − − − − − − −. Un exemple d'exécution de ce programme apparaît à la figure 29.1.

La première fonction `trace_ligne_simple` contient deux parties presque identiques. La première, lignes 12 à 28, trace des lignes partant de l'origine (`x1`, `y1`) vers un point situé entre l'axe des abscisses et la première bissectrice. A chaque itération, on se déplace d'un pixel vers la droite et parfois d'un pixel vers le haut. La seconde partie, lignes 30 à 45, trace des lignes partant de l'origine (`x1`, `y1`) vers un point situé entre l'axe des ordonnées et la première bissectrice. A chaque itération, on se déplace d'un pixel vers le haut et parfois d'un pixel vers la droite.

La seconde fonction, `trace_ligne`, trace toutes les lignes en se ramenant par translation et/ou symétrie axiale à l'un des deux cas traités par la première fonction.

```
# -*- coding: cp1252 -*-
"""ce module contient la fonction trace_ligne qui retourne l'ensemble des pixels
concernés par le tracé d'une ligne en 8-connexité entre deux pixels"""
import pygame                # pour les affichages
import random

def trace_ligne_simple (x1,y1,x2,y2):
    """trace une ligne entre les points de coordonnées (x1,y1) et (x2,y2),
    on suppose que x2 > x1, y2 >= y1,
    retourne la ligne sous la forme d'un ensemble de pixels (x,y)"""

    if y2 - y1 <= x2 - x1 :  # droite en dessous de la première bissectrice
        vx = x2 - x1
        vy = y2 - y1
        b  = vx / 2
        y  = y1
        x  = x1

        ligne = []
        while x <= x2 :
            ligne.append ((x,y))
            b -= vy
            if b < 0:
                b += vx
                y += 1
                ligne.append ((x,y))    # ------ ligne ajoutée
            x += 1                      # ------ ligne déplacée
        return ligne
    else :                  # droite au dessus de la première bissectrice
        vx = x2 - x1
        vy = y2 - y1
        b  = vy / 2
        y  = y1
        x  = x1

        ligne = []
        while y <= y2 :
            ligne.append ((x,y))
            b -= vx
            if b < 0:
                b += vy
                x += 1
                ligne.append ((x,y))   # ------ ligne ajoutée
            y += 1                     # ------ ligne déplacée
        return ligne

def trace_ligne (x1,y1,x2,y2):
    """trace une ligne entre les points de coordonnées (x1,y1) et (x2,y2),
    aucune contrainte sur les coordonnées,
    retourne la ligne sous la forme d'un ensemble de pixels (x,y)"""
```

```
    if x1 == x2 :
        if y1 <= y2: return [ (x1, i) for i in xrange (y1,y2+1) ]
        else : return [ (x1, i) for i in xrange (y2,y1+1) ]

    if y1 == y2 :
        if x1 <= x2: return [ (i, y1) for i in xrange (x1,x2+1) ]
        else : return [ (i, y1) for i in xrange (x2,x1+1) ]

    if x1 < x2 :
        if y1 < y2 :
            return trace_ligne_simple (x1,y1,x2,y2)
        else :
            ligne = trace_ligne_simple (x1,y2,x2,y1)
            return [ (x,y1 + y2 - y) for (x,y) in ligne ]

    if x2 < x1 :
        if y1 < y2 :
            ligne = trace_ligne_simple (x2,y1,x1,y2)
            return [ (x1 + x2 - x, y) for (x,y) in ligne ]
        else :
            ligne = trace_ligne_simple (x2,y2,x1,y1)
            return [ (x1 + x2 - x, y1 + y2 - y) for (x,y) in ligne ]

def display_ligne (ligne, screen):
    """affiche une ligne à l'écran"""
    color = 0,0,0
    for p in ligne:
        pygame.draw.line (screen, color, p,p)
    pygame.display.flip ()


def attendre_clic (screen):
    """attend la pression d'un clic de souris pour continuer"""
    reste = True
    while reste:
        for event in pygame.event.get():
            if event.type == pygame.MOUSEBUTTONUP :
                reste = False
                break

if __name__ == "__main__" :
    pygame.init ()

    size      = width, height = x,y = 200, 200
    black     = 0, 0, 0
    white     = 255,255,255
    screen    = pygame.display.set_mode(size)
    screen.fill (white)

    print trace_ligne (0,0, 7,3)
    # affiche [(0, 0), (1, 0), (1, 1), (2, 1), (3, 1),
    #          (3, 2), (4, 2), (5, 2), (5, 3), (6, 3), (7, 3)]
    # rappel : résultat en 8-connexité :
    # [(0, 0), (1, 0), (2, 1), (3, 1), (4, 2), (5, 2), (6, 3), (7, 3)]

    for n in xrange (0,10):
        x1      = random.randint (0,x-1)
        y1      = random.randint (0,y-1)
        x2      = random.randint (0,x-1)
        y2      = random.randint (0,y-1)
        ligne   = trace_ligne (x1,y1,x2,y2)
        display_ligne (ligne, screen)

    attendre_clic (screen)
```

**2)**

Les fonctions `display_ligne`, `attendre_clic` sont identiques aux programmes précédent. La partie principale (celle qui suit l'instruction `if __name__ == "__main__"` :) est quasiment identique. La fonction qui permet de tracer une ellipse s'intitule `trace_ellipse`. Un exemple d'exécution de ce programme apparaît à la figure 29.1.

```
# -*- coding: cp1252 -*-
"""ce module contient la fonction trace_ellipse qui retourne l'ensemble des pixels
concernés par le tracé d'une ellipse en 8-connexité"""
import pygame                 # pour les affichages
import random
import math

def trace_ellipse (xc,yc,a,b):
    """dessine une ellipse de centre xc,yc, de demi axe horizontal a,
    de demi-axe vertical b, l'ellipse a pour équation xš/aš + yš/bš = 1
    si l'origine est placée en xc,yc,
    l'équation de la tangente au point x0, y0 est :  x x0 / aš + y y0 / bš = 0,
    ou x x0 bš + y y0 aš = 0"""

    # on évite les cas litigieux
    if a == 0 :
        return [ (xc, yc + y) for y in xrange (-b,b) ]
    if b == 0 :
        return [ (xc + x, yc) for x in xrange (-a,a) ]

    bb = b * b
    aa = a * a

    # on trace l'ellipse de centre 0,0
    ellipse = []
    # premier huitième
    vx = a * bb
    vy = 0
    x  = a
    y  = 0
    bl = vx / 2

    while vx >= vy and x >= 0 :
        ellipse.append ((x,y))
        y  += 1
        vy += aa        # vy  = y * aa
        bl -= vy
        if bl < 0 :
            x  -= 1
            vx -= bb    # vx  = x * bb
            bl += vx

    # second huitième
    while x >= 0 :
        ellipse.append ((x,y))
        x  -= 1
        vx -= bb        # vx  = x * bb
        bl += vx
        if bl > 0 :
            y  += 1
            vy += aa    # vy  = y * aa
            bl -= vy

    # second quart, symétrique par rapport à l'axe des ordonnées
    ellipse2 = [ (-x,y) for (x,y) in ellipse ]
    ellipse2.reverse ()
    ellipse.extend (ellipse2)
```

```
    # troisième et quatrième quarts : symétrique par rapport à l'axe des abscisse
    ellipse2 = [ (x,-y) for (x,y) in ellipse ]
    ellipse2.reverse ()
    ellipse.extend (ellipse2)

    return [ (x + xc, y + yc) for (x,y) in ellipse ]

def display_ligne (ensemble, screen):
    """affiche un ensemble de pixels à l'écran"""
    color = 0,0,0
    for p in ensemble:
        pygame.draw.line (screen, color, p,p)
    pygame.display.flip ()

def attendre_clic (screen):
    """attend la pression d'un clic de souris pour continuer"""
    reste = True
    while reste:
        for event in pygame.event.get():
            if event.type == pygame.MOUSEBUTTONUP :
                reste = False
                break

if __name__ == "__main__" :
    pygame.init ()

    size      = width, height = x,y = 300, 300
    black     = 0, 0, 0
    white     = 255,255,255
    screen    = pygame.display.set_mode(size)
    screen.fill (white)

    print trace_ellipse (0,0,6,6)
    # pour le premier quart, affiche
    # [(6, 0), (6, 1), (6, 2), (5, 3), (4, 4), (3, 5), (2, 6), (1, 6), (0, 6), ...

    for n in xrange (0,10):
        x1      = random.randint (0,x-1)
        y1      = random.randint (0,y-1)
        x2      = random.randint (0,x-1)
        y2      = random.randint (0,y-1)
        xc,yc   = (x1 + x2) / 2 , (y1 + y2) / 2
        a,b     = abs (x2 - x1) / 2 , abs (y2 - y1) / 2
        ell     = trace_ellipse (xc,yc,a,b)
        display_ligne (ell, screen)

    attendre_clic (screen)
```

**fin correction TD 29.1.1** □

## 29.2 Un portrait lointain à l'aide du TSP

### 29.2.1 Enoncé

Cette idée est tirée d'un article[2] qui décrit l'image d'un portrait grâce un seul trait contigü (voir figure 29.2). L'algorithme qui permet d'obtenir les résultats de la figure 29.2 s'inspire de l'optimisation de

2. Robert Bosch, Adrianne Herman, *Continuous line drawings via the traveling salesman problem*, Operations Research Letters (32), page 302-303, 2004.

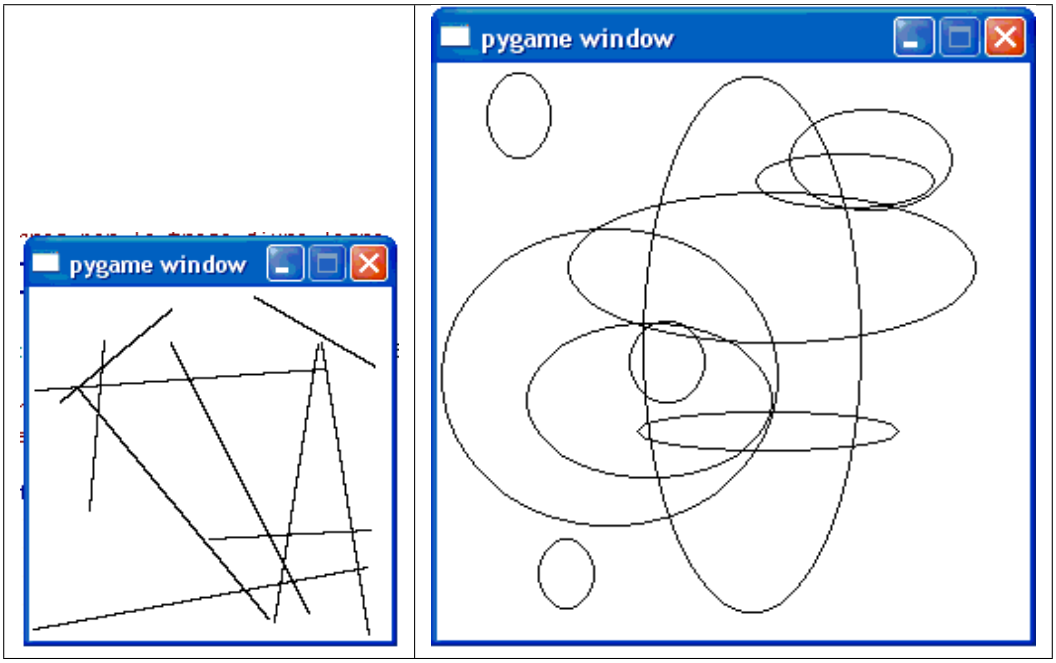



**Figure 29.1 :** *Illustration des deux programmes de tracé de lignes (4-connexité) et d'ellipses (8-connexité), algorithmes de Bresenham, sans multiplication ni division.*

la tournée d'un voyageur de commerce. Des villes sont disséminées sur toute l'image, plus une zone de l'image est sombre, plus elle acceuillera de villes. Une fois les villes placées, il suffit d'utiliser un algorithme permettant de déterminer le plus court chemin le plus passant par toutes les villes. Les valeurs qui ont permis d'obtenir ces images sont $k = 3$ et $\gamma = 9$.

**Algorithme 29.3 : portrait**
Soit $I(n, m)$ une image ayant $n$ lignes et $m$ colonnes. On suppose que $k, \gamma \in \mathbb{N}^*$.

**Etape A : zoom**
L'image est agrandie pour obtenir la taille $(kn, km)$. Chaque pixel de cette image contient une intensité en niveau de gris, 0 pour noir, 255 pour blanc.

**Etape B : partitionnement**
On partitionne l'image en carrée de $k \times k$ pixels. Sur chaque carrée $(i, j)$, on détermine l'intensité moyenne $\mu_{ij}$. On calcule ensuite le nombre $g_{ij}$ :

$$g_{ij} = \gamma - \left\lfloor \frac{\gamma \mu_{ij}}{256} \right\rfloor$$

$\frac{g_{ij}}{\gamma}$ est l'obscurité moyenne sur ce petit carré.

**Etape C : voyageur de commerce**
On répartit de manière aléatoire et uniforme $g_{ij}$ villes dans le carré de coordonnées $(i, j)$ puis on optimise la tournée d'un voyageur de commerce.

L'objectif de ce TD est de se servir du programme développé pour la résolution du voyageur de commerce disponible à l'adresse *http ://perso.wanadoo.fr/x.dupre/programmation/python_cours_exemple_programme_html/tsp_kruskal.html* et de l'utiliser pour créer l'image figure 29.2. Il n'est pas nécessaire de connaître la méthode qui permet de résoudre ce problème, elle est néanmoins détaillée dans le document disponible à l'adresse *http ://perso.wanadoo.fr/x.dupre/programmation/python_cours_exemple.pdf*.

L'extrait de programme qui suit définit les paramètres de l'algorithme 29.3, charge une image représentant

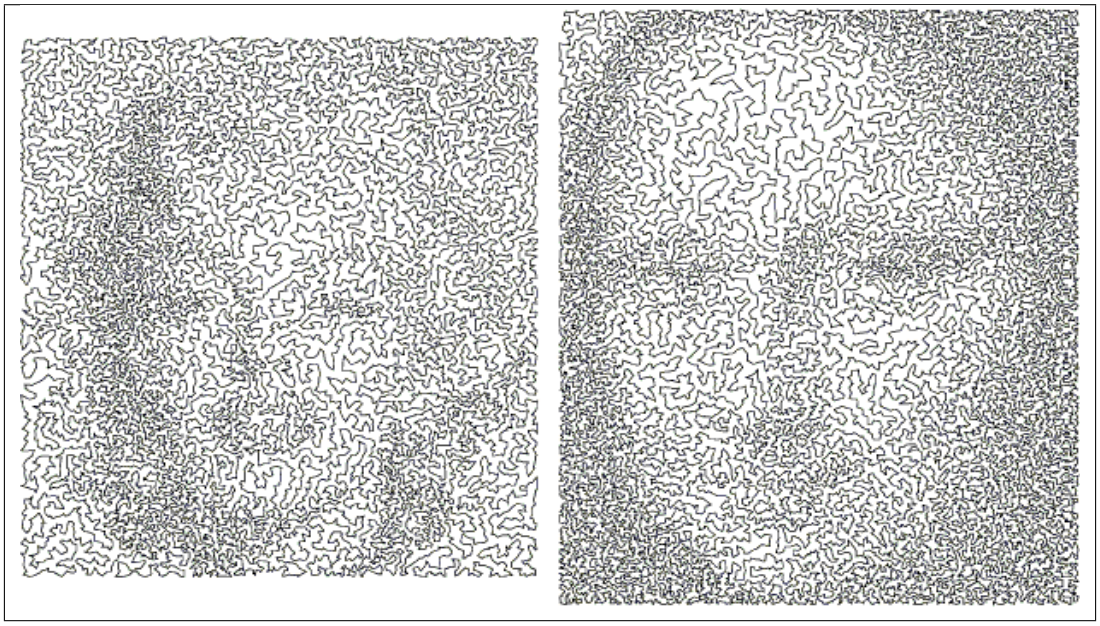


**Figure 29.2 :** *Dessin de Marylin Monroe et de la Joconde à l'aide d'un seul trait contigü. Le portrait de Marylin contient 20332 neurones, Mona Lisa 27486. Pour ces deux images, $k = 3$ et $\gamma = 9$. De loin, l'unique chemin disparaît pour laisser place au portrait.*

un portrait puis exécute la séquence d'instructions permettant d'obtenir le meilleur chemin.

```
gamma     = 9    # paramètre gamma
park      = 30   # paramètre k, pas trop petit pour commencer,
                 # de plus en plus petit par la suite (jusqu'à 3)

# chargement d'une image
image     = pygame.image.load("bette_davis.png")
size      = image.get_size ()
xx,yy     = size [0], size [1]
X,Y       = int (xx // park) + 1, int (yy // park) + 1

# à cet endroit, il faut déterminer l'emplacement des villes
# en utilisant l'image, la fonction construit_ville
# du programme initial doit être réécrite
villes    =  construit_ville (image, gamma, park)

# les lignes qui suivent décrivent la séquence à écrire pour
# obtenir le plus court chemin passant pour toutes les villes
arbre     = arbre_poids_minimal (villes,park)
chemin    = circuit_eulerien (villes ,arbre)
neurone   = circuit_hamiltonien (chemin)
neurones  = [ villes [i] for i in neurone ]
amelioration_chemin (neurones, park, X,Y, 6, screen)
```

La recherche d'un plus court chemin pour quelques dizaines de milliers de villes est onéreuse en temps de calcul. Il est conseillé dans un premier temps de prendre des valeurs pour les paramètres $\gamma$ et $k$ qui ne génère pas trop de villes. Les étapes s'enchaînent rapidement jusqu'au résultat final. Et une fois le programme mis au point pour un problème de petite taille, il sera possible de le faire tourner pendant quelques heures avec un grand nombre de points afin d'obtenir une image proche de la figure 29.2. Afin d'éviter de construire un trop grand nombe de villes, l'étape A a été éludée.

Le programme qui résoud le problème du voyageur de commerce utilise le module `pygame` qui permet facilement de réaliser des affichages. Il est disponible à l'adresse *http ://www.pygame.org/*. Il utilise également

le module `psyco` qui permet d'accélérer la vitesse d'exécution d'un programme. Ce module est disponible à l'adresse `http` : //`psyco.sourceforge.net`/.

### 29.2.2 Correction

Le programme qui suit reprend celui disponible à l'adresse *http ://perso.wanadoo.fr/x.dupre/programmation/python_cours_exemple_programme_html/tsp_kruskal.html*. Les fonctions d'affichage ont été modifiées pour prendre en compte l'image chargée au début du programme. La fonction `amelioration_chemin` a été également modifiée pour pouvoir arrêter l'optimisation et la reprendre d'un clic de souris. La pression de la touche `ECHAP` permet d'interrompre l'optimisation et de terminer le programme par l'affichage du meilleur chemin à une échelle double. Le résultat obtenu avec le programme est présenté figures 29.3 et 29.4.

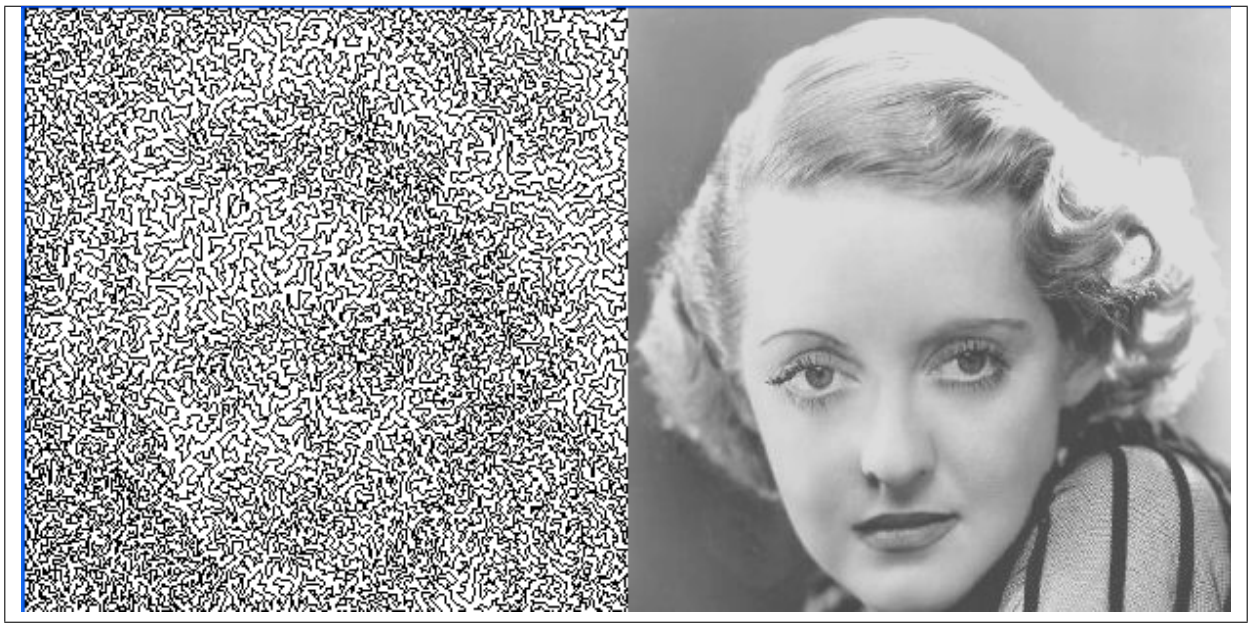


**Figure 29.3 :** *Dessin de Bette Davies obtenu avec le programme inclus dans le paragraphe 29.2.2. Les paramètres choisis sont $k = 3$ et $\gamma = 4$, soit 18000 villes. Le programme a tourné une heure et demi sur un Pentium IV cadencé à 1,6 GHz.*

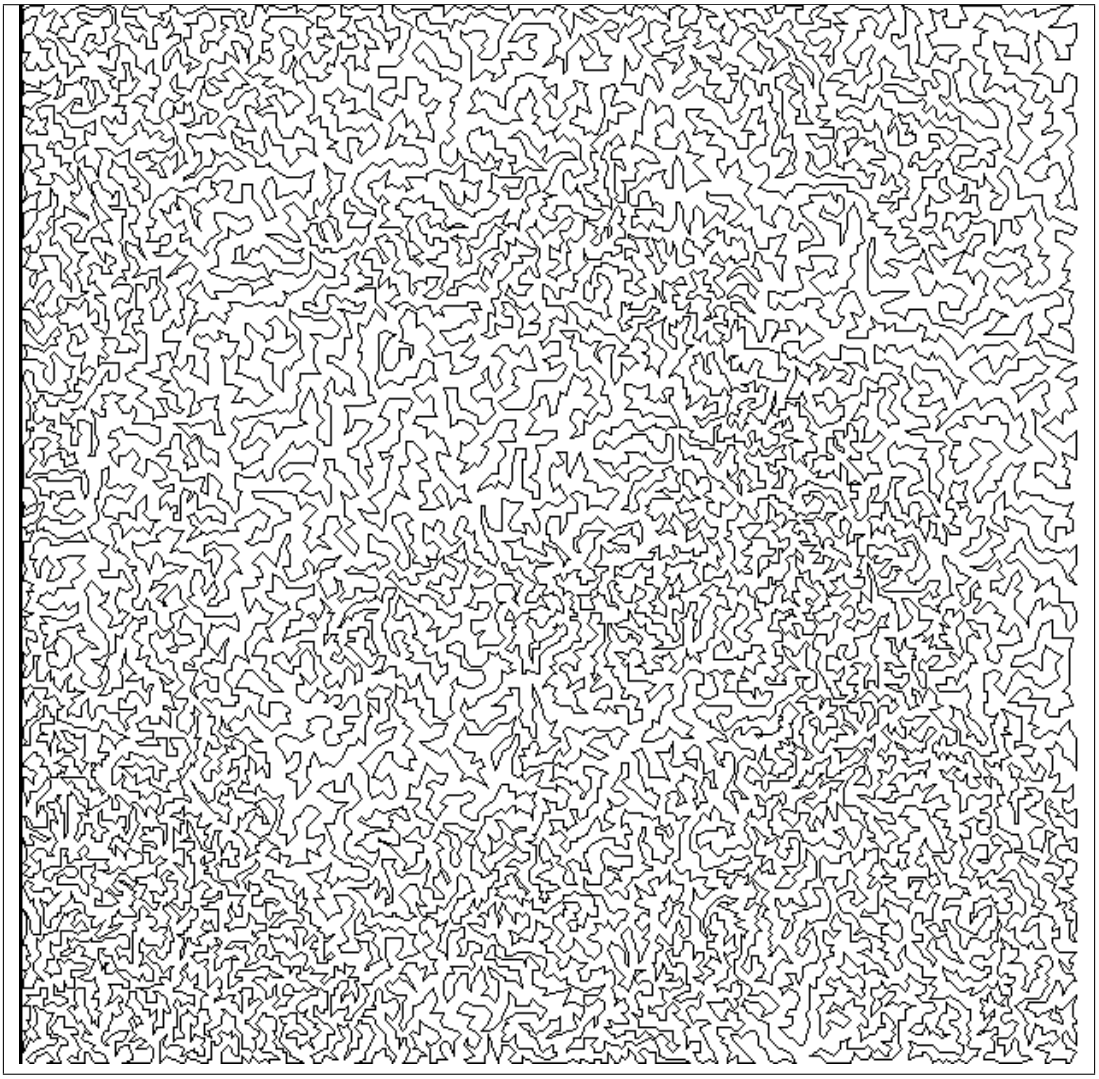


**Figure 29.4 :** *Même dessin que celui de la figure 29.3 mais agrandi. Le visage apparaît lorsqu'on s'éloigne de quelques mètres de l'écran.*

```
# -*- coding: cp1252 -*-
import random                 # pour tirer aléatoirement des nombres
import math                   # fonctions cos, sin
import pygame                 # pour les affichages
import copy
import bresenham_ligne4 as brl  # pour dessiner des lignes
import psyco                  # pour accélérer le programme

def attendre_clic (screen,x,y):
    """dessine une croix sur l'écran et attend la pression d'un clic de souris"""
    color   = 0,0,0
    pygame.draw.line (screen, color, (0,0), (x-1,y-1))
    pygame.draw.line (screen, color, (x-1,0), (0,y-1))
    pygame.display.flip ()
    reste = True
    while reste:
        for event in pygame.event.get():
            if event.type == pygame.MOUSEBUTTONUP :
                reste = False
                break
```

```
def ajuste_image (image, park):
    """arrondi les dimensions d'une images à park, retourne une partie de l'image"""
    x,y = image.get_size ()
    x = (x // park) * park
    y = (y // park) * park
    image2 = image.subsurface (((0,0),(x,y)))
    image3 = image2.copy ()
    return image3

def construit_ville(image, gamma, park):
    """tire aléatoirement des villes,
    ces villes sont d'autant plus nombreuses que l'intensité de l'image
    est faible, évite les villes confondues"""
    x,y = image.get_size ()
    nbx = x // park
    nby = y // park

    villes = []
    for i in xrange(0,nbx):
        for j in xrange(0,nby):
            x1 = i * park
            x2 = (i+1) * park
            y1 = j * park
            y2 = (j+1) * park

            # calcul de l'obscurité moyenne
            moy = 0
            for k in xrange(x1,x2):
                for l in xrange(y1,y2):
                    int = image.get_at ((k,l))
                    moy += int [0] + int [1] + int [2]
            moy /= park * park * 3

            # nombre de villes
            nbv = gamma - gamma * moy / 256
            pl  = []

            for k in xrange(0,nbv):
                xx = random.randint (x1,x2-1)
                yy = random.randint (y1,y2-1)
                if (xx,yy) not in pl :
                    villes.append ((xx,yy))
                    pl.append ((xx,yy))

    return villes

def display_ville(villes,screen):
    """dessine les villes à l'écran"""
    color  = 0,0,0
    for v in villes:
        pygame.draw.line (screen, color, v, v)


def display_neurone(neurones_,screen,mult = 1):
    """dessine les neurones à l'écran"""
    color       = 0,0,0
    color2      = 0,255,0
    if mult != 1 :
        neurones    = [ (n [0] * mult, n [1] * mult ) for n in neurones_ ]
        if mult == 2 : l = 1
        else : l = 1
        for n in neurones :
            pygame.draw.line (screen, color, n, n)
        #pygame.draw.circle (screen, color2, neurones [0], 8)
        if len (neurones) > 1 :
```

```
            color3 = 150,0,0
            #pygame.draw.circle (screen, color3, neurones [1], 6)
        if len (neurones) > 1:
            pygame.draw.lines (screen, color, True, neurones, l)
    else :
        neurones    = [ (n [0] * mult, n [1] * mult ) for n in neurones_ ]
        if mult == 2 : l = 1
        else : l = 1
        for n in neurones_ :
            pygame.draw.line (screen, color, n, n)
        #pygame.draw.circle (screen, color2, neurones [0], 8)
        if len (neurones_) > 1 :
            color3 = 150,0,0
            #pygame.draw.circle (screen, color3, neurones [1], 6)
        if len (neurones_) > 1:
            pygame.draw.lines (screen, color, True, neurones_, l)

def display_arbre (villes, arbre, mult = 1):
    """dessine le graphe de poids minimal défini par arbre"""
    if mult == 2 :
        color = 0,255,0
        l = 4
    else :
        l = 1
        color = 0,0,255

    for i in xrange (0, len (villes)):
        for j in arbre [i]:
            v  = (villes [i][0] * mult, villes [i][1] * mult)
            vv = (villes [j][0] * mult, villes [j][1] * mult)
            pygame.draw.line (screen, color, v, vv, l)


def repartition_zone (villes, zone_taille, ask_zone = False) :
    """répartit les villes en zones, retourne les villes rangées par zones,
    chaque éléments zones [z][k] contient :
        - les coordonnées de la ville
        - ses coordonnées en zone, (zx, zy)
        - son indice dans la liste villes
    la fonction retourne également le nombre de zones
    selon l'axe des abscisses et l'axe des ordonnées,
    retourne aussi le nombre de zones, si ask_zone est True,
    retourne un paramètre supplémentaire : zone"""
    print "gestion des zones"
    X,Y = 0,0
    for v in villes:
        X = max (v [0] // zone_taille, X)
        Y = max (v [1] // zone_taille, Y)
    X += 1
    Y += 1

    # attribution des zones
    zone    = []
    nb      = len (villes)
    Zmax    = 0
    for i in xrange (0,len (villes)):
        v       = villes [i]
        x       = int (v [0] // zone_taille)
        y       = int (v [1] // zone_taille)
        z       = int (y * X + x)
        Zmax    = max (z,Zmax)
        zone.append ((z, v, (x,y), i))

    # rangement par zone
    Zmax   += 1
```

```
    zones = [ [] for i in xrange (0,Zmax) ]
    for z in zone:
        zones [ z [0] ].append ((z [1], z [2], z [3]))

    if ask_zone :
        return zones,X,Y,Zmax,zone
    else :
        return zones,X,Y,Zmax

def voisinage_zone (z,Zmax,X,Y):
    """retourne la liste des voisins d'une zone z
    sachant qu'il y a X zones sur l'axe des abscisses et Y zones sur l'axe des ordonnées,
    Zmax est le nombre de zones,
    inclus z dans cette liste"""
    x       = z % X
    y       = z // X
    voisin_ = [z]
    if x > 0: voisin_.append (z-1)
    if x < X: voisin_.append (z+1)
    if y > 0: voisin_.append (z-X)
    if y < Y: voisin_.append (z+X)
    if x > 0 and y > 0 : voisin_.append (z-1-X)
    if x > 0 and y < Y : voisin_.append (z-1+X)
    if x < X and y > 0 : voisin_.append (z+1-X)
    if x < X and y < Y : voisin_.append (z+1+X)
    voisin = [ int (i) for i in voisin_ if i >= 0 and i < Zmax ]
    return voisin

def distance (p1,p2):
    """calcule la distance entre deux villes"""
    x = p1 [0] - p2[0]
    y = p1 [1] - p2[1]
    return math.sqrt (x*x + y*y)

def arbre_poids_minimal(villes, zone_taille):
    """construit l'arbre de poids minimal, retourne une liste de
    listes, chaque sous-liste associée à une ville contient la liste des ses voisins,
    zone_taille permet de découper l'image en zones,
    les distances ne seront calculées que si
    deux éléments sont dans la même zone ou dans une zone voisine"""

    def tri_distance (u,v):
        if u [2] < v [2] : return -1
        elif u [2] > v [2] : return 1
        else : return 0

    zones,X,Y,Zmax = repartition_zone (villes, zone_taille)

    # calcul des distances
    print "calcul des distances"
    li      = []

    for z in xrange (0,len (zones)):
        if z % 1000 == 0 : print "zone ", z, len(zones)
        voisin  = voisinage_zone (z,Zmax,X,Y)
        for v in zones [z]:
            for zz in voisin:
                for u in zones [zz]:
                    d = distance (v [0], u [0])
                    li.append ((v [2], u [2], d))

    print "tri ", len(li)
    li.sort (tri_distance)

    # composantes connexes
```

```
    print "construction de l'arbre"

    # nombre de composantes connexes
    nb_comp     = len (villes)

    # indice de la composante d'une ville
    num_comp    = [ i for i in xrange(0,len(villes)) ]

    # liste des voisins pour chaque ville
    arbre       = [ [] for i in xrange(0,len(villes)) ]

    # liste des villes par composante connexe
    list_comp   = [ [i] for i in xrange(0,len(villes)) ]

    iii = 0
    for c in li:
        if iii % 10000 == 0 : print "arête ", iii, len(li)
        iii += 1
        i,j = c [0], c [1]
        if num_comp [i] != num_comp [j]:
            # on relie les villes i et j car elles appartiennent
            # à des composantes connexes différentes
            arbre [i].append (j)        # i est voisine de j
            arbre [j].append (i)        # j est voisine de i
            cl = num_comp [i]           # composante connexe restante
            ki = num_comp [j]           # composante connexe à aggréger à la précédente
            for k in list_comp [ki]:
                num_comp [k] = cl
                list_comp [cl].append (k)
            list_comp [ki] = []
            nb_comp -= 1                # une composante connexe en moins

        if nb_comp == 0:
            break  # il n'y a plus qu'une seule composante connexe, inutile de continuer

    return arbre

def vecteur_points (p1,p2):
    """retourne le vecteur entre les points p1 et p2"""
    return ( p2 [0] - p1 [0], p2 [1] - p1 [1])

def vecteur_norme (vec):
    """retourne la norme d'un vecteur"""
    return math.sqrt (vec [0] * vec [0] + vec [1] * vec [1])

def vecteur_cosinus (vec1, vec2):
    """retourne le cosinus entre deux vecteurs, utilise le produit scalaire"""
    norm1 = vecteur_norme (vec1)
    norm2 = vecteur_norme (vec2)
    if norm1 == 0: return 1
    if norm2 == 0: return 1
    scal  = vec1 [0] * vec2 [0] + vec1 [1] * vec2 [1]
    return scal / (norm1 * norm2)

def vecteur_sinus (vec1, vec2):
    """retourne le sinus entre deux vecteurs, utilise le produit vectoriel"""
    norm1 = vecteur_norme (vec1)
    norm2 = vecteur_norme (vec2)
    if norm1 == 0:  return 0
    if norm2 == 0:  return 0
    scal  = vec1 [0] * vec2 [1] - vec1 [1] * vec2 [0]
    return scal / (norm1 * norm2)

def oppose_vecteur (vec):
    """retourne le vecteur opposé"""
```

```
    return (- vec [0], - vec [1])

def circuit_eulerien (villes, arbre):
    """définit un circuit eulérien, villes contient la liste des villes,
    tandis que arbre est une liste de listes, arbre [i] est la liste des villes
    connectées à la ville i,
    on suppose que arbre est un graphe de poids minimal non orienté,
    l'algorithme ne marche pas s'il existe des villes confondues"""

    # on choisit une ville qui est une extrémité et parmi celle-là on la choisit au hasard
    has = []
    for i in xrange (0,len (villes)):
        n = len (arbre [i])
        if n == 1: has.append (i)

    bm = random.randint (0, len (has) -1)
    bm = has [bm]

    # vecteur, le circuit eulérien contient
    # nécessairement 2 * len (villes) noeuds puisque c'est
    # le graphe eulérien d'un arbre de poids minimal non orienté
    vec     = (1,1)
    chemin  = [bm]
    while len (chemin) < 2 * len (villes)-1 :

        v = villes [bm]

        ma      = - math.pi - 1
        bvec    = vec
        opvec   = oppose_vecteur (vec)
        for k in xrange (0, len (arbre [bm])) :
            l     = arbre [bm][k]
            vec2  = vecteur_points (v, villes [l])
            if opvec == vec2 :
                angle = -math.pi
            else :
                cos   = vecteur_cosinus (vec, vec2)
                sin   = vecteur_sinus   (vec, vec2)
                angle = math.atan2 (sin, cos)
            if angle > ma :
                ma      = angle
                bl      = k
                bvec    = vec2

        b   = arbre [bm][bl]
        chemin.append (b)
        del arbre [bm][bl]  # on supprime l'arc pour ne plus l'utiliser
        bm  = b
        vec = bvec

    return chemin

def circuit_hamiltonien (chemin):
    """extrait un circuit hamiltonien depuis un circuit eurlérien"""
    nb      = max (chemin) + 1
    res     = []
    coche   = [ False for i in xrange (0, nb) ]
    for c in chemin :
        if coche [c] : continue
        res.append (c)
        coche [c] = True

    return res

def equation_droite (p1,p2) :
```

```
    """retourne l'équation d'une droite passant par p1 et p2,
    ax + by + c = 0, retourne les coefficients a,b,c"""
    vec = vecteur_points (p1,p2)
    a = vec [1]
    b = - vec [0]
    c = - a * p1 [0] - b * p1 [1]
    return a,b,c

def evaluation_droite (a,b,c,p):
    """l'équation d'une droite est : ax + by + c, retourne la valeur
    de cette expression au point p"""
    return a * p [0] + b * p [1] + c

def intersection_segment (p1,p2,p3,p4):
    """dit si les segments [p1 p2] et [p3 p4] ont une intersection,
    ne retourne pas l'intersection"""
    # équation de la droite (p1 p2)
    a1,b1,c1 = equation_droite (p1,p2)
    a2,b2,c2 = equation_droite (p3,p4)
    s1 = evaluation_droite (a2,b2,c2,p1)
    s2 = evaluation_droite (a2,b2,c2,p2)
    s3 = evaluation_droite (a1,b1,c1,p3)
    s4 = evaluation_droite (a1,b1,c1,p4)
    if s1 * s2 <= 0 and s3 * s4 <= 0 : return True
    else : return False

def longueur_chemin (chemin):
    """retourne la longueur d'un chemin"""
    s   = 0
    nb  = len (chemin)
    for i in xrange (0,nb) :
        s += distance ( chemin [i], chemin [ (i+1) % nb ])
    return s

def retournement_essai (chemin, i,j, negligeable = 1e-5):
    """dit s'il est judicieux de parcourir le chemin entre les sommets i et j
    en sens inverse, si c'est judicieux, change le chemin et retourne True,
    sinon, retourne False,
    si j < i, alors la partie à inverser est
    i, i+1, i+2, ..., len(chemin) -1, 0, 1, ..., j"""

    nb = len (chemin)
    if i == j : return False
    if j - i == -1 : return False
    if j - i - nb == -1 : return False

    ia = (i - 1 + nb) % nb
    ja = (j + 1 ) % nb
    # arcs enlevés
    d_ia_i = distance (chemin [ia], chemin [i])
    d_j_ja = distance (chemin [j],  chemin [ja])
    # arcs ajoutés
    d_ia_j = distance (chemin [ia], chemin [j])
    d_i_ja = distance (chemin [i],  chemin [ja])
    # amélioration ?
    d = d_ia_j + d_i_ja - d_j_ja - d_ia_i
    if d >= - negligeable : return False

    # si amélioration, il faut retourner le chemin entre les indices i et j
    jp = j
    if jp < i : jp = j + nb
    ip = i

    while ip < jp :
        i           = ip % nb
```

```
        j           = jp % nb
        ech         = chemin [i]
        chemin [i]  = chemin [j]
        chemin [j]  = ech
        ip = ip+1
        jp = jp-1

    return True

def retournement (chemin, taille) :
    """amélioration du chemin par un algorithme simple,
    utilise des retournements de taille au plus <taille>,
    retourne le nombre de modifications"""

    # traitement des petits retournements
    nb = len (chemin)
    nb_change = 1
    nbtout    = 0
    retour      = { }
    while nb_change > 0 :
        nb_change   = 0
        for t in xrange (1,taille+1):
            retour [t] = 0
            for i in xrange (0, nb) :
                j = (i + t) % nb
                b = retournement_essai (chemin, i, j)
                if b :
                    retour [t] += 1
                    nb_change  += 1
        nbtout += nb_change
    print "nombre de retournements %d longueur : \t %10.0f détail \t" \
                % (nbtout, longueur_chemin (chemin)), " détail : ", retour
    return nbtout

def echange_position_essai (chemin, a,b, x, inversion, negligeable = 1e-5):
    """échange la place des villes ka à kb pour les placer entre les villes i et i+1,
    si inversion est True, on inverse également le chemin inséré, si inversion est False,
    on ne l'inverse pas,
    si cela améliore, déplace les villes et retourne True, sinon, retourne False"""

    nb = len(chemin)
    xa = (x + 1) % nb
    ka = (a - 1 + nb) % nb
    kb = (b + 1 ) % nb

    if not inversion :

        if x == ka : return False
        if x == kb : return False
        if xa == ka : return False
        if b < a :
            if a <= x <= b + nb : return False
        elif a <= x <= b : return False
        if b < a :
            if a <= x + nb <= b + nb : return False
        elif a <= x + nb <= b : return False

        # arcs enlevés
        d_x_xa = distance (chemin [x],  chemin [xa])
        d_ka_a = distance (chemin [ka], chemin [a])
        d_b_kb = distance (chemin [b],  chemin [kb])
        # arcs ajoutés
        d_ka_kb = distance (chemin [ka], chemin [kb])
        d_x_a   = distance (chemin [x],  chemin [a])
        d_b_xa  = distance (chemin [b],  chemin [xa])
```

```
    d = d_ka_kb + d_x_a + d_b_xa - d_x_xa - d_ka_a - d_b_kb
    if d >= -negligeable : return False

    # villes à déplacer
    ech = []
    bp = b
    if bp < a : bp = b + nb
    for i in xrange (a,bp+1):
        ech.append (chemin [i % nb] )
    diff = bp - a + 1

    xp = x
    if xp < b : xp += nb

    for l in xrange (b+1,xp+1) :
        ll              = l % nb
        bp              = (a + l - b-1) % nb
        chemin [bp]     = chemin [ll]

    for l in xrange (0,len(ech)) :
        chemin [ (x+l-diff+1 + nb) % nb ] = ech [l]

    return True

else :

    if x == ka : return False
    if x == kb : return False
    if xa == ka : return False
    if b < a :
        if a <= x <= b + nb : return False
    elif a <= x <= b : return False
    if b < a :
        if a <= x + nb <= b + nb : return False
    elif a <= x + nb <= b : return False


    # arcs enlevés
    d_x_xa = distance (chemin [x],  chemin [xa])
    d_ka_a = distance (chemin [ka], chemin [a])
    d_b_kb = distance (chemin [b],  chemin [kb])
    # arcs ajoutés
    d_ka_kb = distance (chemin [ka], chemin [kb])
    d_x_b   = distance (chemin [x],  chemin [b])
    d_a_xa  = distance (chemin [a],  chemin [xa])

    d = d_ka_kb + d_x_b + d_a_xa - d_x_xa - d_ka_a - d_b_kb
    if d >= -negligeable  : return False

    # villes à déplacer
    ech = []
    bp = b
    if bp < a : bp = b + nb
    for i in xrange (a,bp+1):
        ech.append (chemin [i % nb] )
    ech.reverse ()
    diff = bp - a + 1

    cop = copy.copy (chemin)

    xp = x
    if xp < b : xp += nb

    for l in xrange (b+1,xp+1) :
```

```
            ll              = l % nb
            bp              = (a + l - b-1) % nb
            chemin [bp]     = chemin [ll]

        for l in xrange (0,len(ech)) :
            chemin [ (x+l-diff+1 + nb) % nb ] = ech [l]

        return True

def dessin_arete_zone (chemin, taille_zone, X,Y):
    """retourne une liste de listes de listes,
    res [i][j] est une liste des arêtes passant près de la zone (x,y) = [i][j],
    si k in res [i][j], alors l'arête k,k+1 est dans la zone (i,j),
    X est le nombre de zones horizontalement, Y est le nombre de zones verticalement,
    taille_zone est la longueur du côté du carré d'une zone"""
    res = [ [ [] for j in xrange (0,Y+1) ]  for i in xrange (0,X+1) ]
    nb = len (chemin)
    for i in xrange (0,nb):
        a       = chemin [i]
        b       = chemin [(i+1) % nb]
        x1,x2   = int (a [0] // taille_zone), int (b [0] // taille_zone)
        y1,y2   = int (a [1] // taille_zone), int (b [1] // taille_zone)
        line = brl.trace_ligne (x1,y1,x2,y2)
        for x,y in line:
            res [x][y].append (i)
    return res

def voisinage_zone_xy (x,y,X,Y):
    """retourne la liste des voisins d'une zone (x,y)
    sachant qu'il y a X zones sur l'axe des abscisses et Y zones sur l'axe des ordonnées,
    inclus z dans cette liste"""
    voisin = [(x,y)]
    if x > 0   : voisin.append ((x-1,y))
    if x < X-1 : voisin.append ((x+1,y))
    if y > 0   : voisin.append ((x,y-1))
    if y < Y-1 : voisin.append ((x,y+1))
    if x > 0   and y > 0   : voisin.append ((x-1,y-1))
    if x > 0   and y < Y-1 : voisin.append ((x-1,y+1))
    if x < X-1 and y > 0   : voisin.append ((x+1,y-1))
    if x < X-1 and y < Y-1 : voisin.append ((x+1,y+1))
    return voisin

def echange_position (chemin, taille, taille_zone, X,Y, grande = 0.5) :
    """regarde si on ne peut pas déplacer un segment de longueur taille
    pour supprimer les arêtes les plus longues,
    au maximum <grande> longues arêtes,
    retourne le nombre de changement effectués,
    X est le nombre de zones horizontalement,
    Y est le nombre de zones verticalement,
    taille_zone est la longueur d'un côté du carré d'une zone"""

    nb = len(chemin)

    def tri_arete (x,y):
        """pour trier la liste l par ordre décroissant"""
        if x [2] < y [2] : return 1
        elif x [2] > y [2] : return -1
        else : return 0

    # list des arêtes triés par ordre décroissant
    la = []
    for i in xrange (0,nb) :
        im = (i+1) % nb
        la.append ( (i, im, distance (chemin [i], chemin [im]) ) )
    la.sort (tri_arete)
```

```
# zone associé à chaque arête
zone = dessin_arete_zone (chemin, taille_zone, X, Y)

dseuil    = la [ int (nb * grande) ] [ 2 ]
nbtout    = 0
nb_change = 0
iarete    = 0
retour    = { }
for t in xrange (1, taille+1): retour [t] = 0

while iarete < nb :
    nb_change   = 0
    arete       = la [iarete]
    iarete     += 1
    x           = arete [0]
    xm          = arete [1]
    a           = chemin [x]
    b           = chemin [xm]
    d           = distance ( a,b )
    if d < dseuil : break    # arête trop petite

    # zone traversée par la ligne
    x1,x2   = int (a [0] // taille_zone), int (b [0] // taille_zone)
    y1,y2   = int (a [1] // taille_zone), int (b [1] // taille_zone)
    ens     = brl.trace_ligne (x1,y1,x2,y2)
    ville   = []
    for k,l in ens:
        voisin = voisinage_zone_xy (k,l,X,Y)
        for u,v in voisin :
            ville.extend (zone [u][v])

    # on supprime les doubles
    ville.sort ()
    if len (ville) == 0 : continue
    sup = []
    mx = -1
    for v in ville:
        if v == mx : sup.append (v)
        mx = v
    for s in sup:
        ville.remove (s)

    # on étudie les possibilités de casser l'arête (x,xm) aux alentours des villes
    # comprise dans l'ensemble ville
    for t in xrange (1, taille+1):

        for i in ville:

            # on essaye d'insérer le sous-chemin (x- t + 1 + nb)  --> x
            # au milieu de l'arête i,i+1
            b = echange_position_essai (chemin, (x- t + 1 + nb) % nb,x, i, False)
            if b :
                nb_change  += 1
                retour [t] += 1
                continue

            # on essaye d'insérer le sous-chemin (xm+ t - 1)  --> xm
            # au milieu de l'arête i,i+1
            b = echange_position_essai (chemin, (xm + t - 1) % nb, xm, i, False)
            if b :
                nb_change  += 1
                retour [t] += 1
                continue
```

```
                # on essaye de casser l'arête x,xm en insérant
                # le sous-chemin i --> (i+t) % nb
                b = echange_position_essai (chemin, i, (i+t) % nb, x, False)
                if b :
                    nb_change  += 1
                    retour [t] += 1
                    continue
                # idem
                b = echange_position_essai (chemin, i, (i+t) % nb, x, True)
                if b :
                    retour [t] += 1
                    nb_change  += 1
                    continue
                # idem
                b = echange_position_essai (chemin, (i-t+nb) % nb, i, x, False)
                if b :
                    nb_change  += 1
                    retour [t] += 1
                    continue
                # idem
                b = echange_position_essai (chemin, (i-t + nb) % nb, i, x, True)
                if b :
                    retour [t] += 1
                    nb_change  += 1
                    continue

        nbtout += nb_change

    print "nombre de déplacements %d longueur :   \t %10.0f détail \t" \
                    % (nbtout, longueur_chemin (chemin)), " détail : ", retour
    return nbtout

def supprime_croisement (chemin, taille_zone, X,Y) :
    """supprime les croisements d'arêtes,
    retourne le nombre de changement effectués,
    X est le nombre de zones horizontalement,
    Y est le nombre de zones verticalement,
    taille_zone est la longueur d'un côté du carré d'une zone"""

    nb = len(chemin)

    # zone associé à chaque arête
    zone    = dessin_arete_zone (chemin, taille_zone, X, Y)
    nbtout  = 0

    for i in xrange (0,nb) :
        im      = (i+1) % nb
        a       = chemin [i]
        b       = chemin [im]

        # zone traversée par la ligne
        x1,x2   = int (a [0] // taille_zone), int (b [0] // taille_zone)
        y1,y2   = int (a [1] // taille_zone), int (b [1] // taille_zone)
        ens     = brl.trace_ligne (x1,y1,x2,y2)
        ville   = []
        for k,l in ens:
            voisin = voisinage_zone_xy (k,l,X,Y)
            for u,v in voisin :
                ville.extend (zone [u][v])

        # on supprime les doubles
        ville.sort ()
        if len (ville) == 0 : continue
        sup = []
        mx = -1
```

```
        for v in ville:
            if v == mx : sup.append (v)
            mx = v
        for s in sup:
            ville.remove (s)

        nb_change = 0
        for v in ville :
            b = retournement_essai (chemin, i,v)
            if b :
                nb_change += 1
                continue
            b = retournement_essai (chemin, im,v)
            if b :
                nb_change += 1
                continue

        nbtout += nb_change

    print "nombre de croisements %d longueur :      \t %10.0f détail \t" \
                    % (nbtout, longueur_chemin (chemin))
    return nbtout

def amelioration_chemin (chemin, taille_zone, X, Y, taille = 10, \
                             screen = None, image = None):
    """amélioration du chemin par un algorithme simple,
    utilise des retournements de taille au plus taille,
    traite les arcs qui se croisent,
    traite les grands arcs, utilise un quadrillage de taille window,
    X est le nombre de zones horizontalement,
    Y est le nombre de zones verticalement,
    taille_zone est la longueur d'un côté du carré d'une zone"""

    white = 255,255,255

    #première étape rapide
    nb = 1
    while nb > 0:
        nb  = retournement (chemin, taille)
        if screen != None:
            screen.fill (white)
            display_neurone(chemin, screen, 1)
            pygame.display.flip ()

    # amélioration
    nb      = 1
    size    = image.get_size ()
    dec     = size [0]
    fin     = False
    while nb > 0 and not fin :
        nb  = retournement (chemin, taille)
        if screen != None:
            screen.fill (white)
            if image != None : screen.blit (image, (dec,0))
            display_neurone(chemin, screen)
            pygame.display.flip ()
        nb += echange_position (chemin, taille / 2, taille_zone, X, Y)
        if screen != None:
            screen.fill (white)
            if image != None : screen.blit (image, (dec,0))
            display_neurone(chemin, screen)
            pygame.display.flip ()
        nb += supprime_croisement (chemin, taille_zone,X,Y)
        if screen != None:
            screen.fill (white)
```

```
            if image != None : screen.blit (image, (dec,0))
            display_neurone(chemin, screen)
            pygame.display.flip ()

        if screen != None :
            for event in pygame.event.get():
                if event.type == pygame.MOUSEBUTTONUP :
                    print "attendre.........................................."
                    attendre_clic (screen,0,0)
                    print "reprise"
                    break
                if event.type == pygame.KEYDOWN and event.key == 27 :
                    # si la touche ECHAP est pressée, l'optimisation s'arrête
                    print ".............arrêter l'optimisation"
                    fin = True
                    break


################################################################################

if __name__ == "__main__" :
    pygame.init ()
    psyco.full ()

    gamma     = 4
    park      = 3

    image2 = pygame.image.load("bette_davis.png")
    image = ajuste_image (image2, park)

    size      = image.get_size ()
    xx,yy     = size [0], size [1]
    X,Y       = int (xx // park) + 1, int (yy // park) + 1
    dec       = size [0]
    size      = (size [0] * 2, size [1])
    size      = width, height = x,y = size
    black     = 0, 0, 0
    white     = 255,255,255
    screen    = pygame.display.set_mode(size)
    villes    = construit_ville (image, gamma, park)

    print "taille de l'image : ", size
    print "nombre de villes : ", len(villes)

    screen.fill (white)
    screen.blit (image, (dec,0))
    display_ville (villes, screen)
    pygame.display.flip ()

    print "calcul"
    arbre = arbre_poids_minimal (villes,park)
    arbre2 = copy.deepcopy (arbre)

    print "dessin"
    display_arbre (villes, arbre)
    attendre_clic(screen,0,0)

    print "circuit eulérien"
    chemin = circuit_eulerien (villes ,arbre)

    print "circuit hamiltonien"
    neurone = circuit_hamiltonien (chemin)

    print "dessin"
    neurones = [ villes [i] for i in neurone ]
```

```
    display_neurone(neurones, screen)
    attendre_clic(screen,0,0)

    print "amélioration"
    amelioration_chemin (neurones, park, X,Y, 6, screen, image)
    attendre_clic(screen,0,0)

    # taille double
    print "taille double"
    size      = (size [0], size [1]*2)
    screen    = pygame.display.set_mode(size)
    screen.fill (white)
    #display_arbre (villes, arbre2, 2)
    display_neurone (neurones, screen, 2)
    attendre_clic(screen,0,0)
```

**fin correction TD 29.2.1** □

## 29.3 Image de synthèse

### 29.3.1 Enoncé

Le document *http ://perso.wanadoo.fr/x.dupre/programmation/python_cours_exemple.pdf* présente la méthode du lancer de rayon pour construire des images de synthèse. Celle-ci est implémentée au moyen de cinq fichiers accessibles à l'adresse *http ://perso.wanadoo.fr/x.dupre/programmation/python_cours_exemple_programme_html/index.html*. Les seuls objets que ce programme peut dessiner sont des sphères et des facettes triangulaires ou en forme de parallélogrammes plan. L'objectif de ce TD est d'ajouter de nouveaux objets qui puissent être dessinés sur des images de synthèse par simple dérivation d'une classe.

**1)** Le premier problème à résoudre consiste à construire une facette rectangulaire dont la surface est celle d'une image. Ce nouvel objet est modélisé par une nouvelle classe qui hérite de la classe `rectangle` (fichier `image_synthese_facette.py`. La figure 29.5 montre un exemple de résultat qu'on veut obtenir.

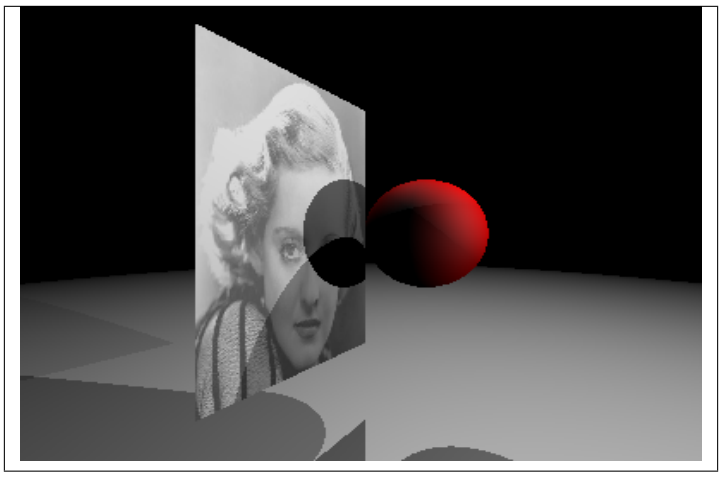


**Figure 29.5 :** *Portrait de Bette Davis partiellement caché par l'ombre d'une sphère.*

Soit une facette en forme de quadrilatère dont les sommets sont $ABCD$, ses diagonales sont $AC$ et $BD$. Un repère possible pour cette image est le repère $\left(A, \frac{\overrightarrow{AB}}{\left\|\overrightarrow{AB}\right\|}, \frac{\overrightarrow{AD}}{\left\|\overrightarrow{AD}\right\|}\right)$. La figure 29.6 illustre ce repère. Le problème consiste à trouver la correspondance entre :

– le point d'intersection $P$ entre le rayon et la facette
– le pixel de coordonnées $(x, y)$ de l'image qui lui est associé

Cette image a pour dimension $(X, Y)$. Cette correspondance est traduite par les expressions suivantes :

$$\begin{cases} x &= \overrightarrow{AP} \cdot \overrightarrow{AB} \dfrac{X}{\left\|\overrightarrow{AB}\right\|^2} \\ y &= \overrightarrow{AP} \cdot \overrightarrow{AD} \dfrac{Y}{\left\|\overrightarrow{AD}\right\|^2} \end{cases} \tag{29.1}$$

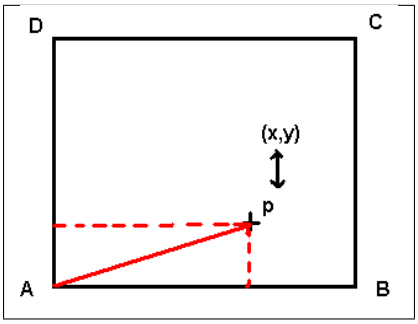



**Figure 29.6 :** *Repère intuitif pour une facette en forme de quadrilatère.*

**2)** Le second problème consiste à construire une surface qui reflète une partie des rayons comme dans l'image 29.8. La figure 29.7 montre le rayon réfléchi. On considère que le rayon incident est défini par une origine $O$ et un vecteur directeur $\overrightarrow{v}$, le point d'intersection est $P$ et la normale à la surface au point $P$ est le vecteur $\overrightarrow{N}$. Alors le rayon réfléchi a pour origine le point $P$ et pour vecteur directeur $\overrightarrow{v'}$ défini par :

$$\overrightarrow{v'} = \overrightarrow{v} - (2\overrightarrow{v} \cdot \overrightarrow{n})\,\overrightarrow{n} \tag{29.2}$$

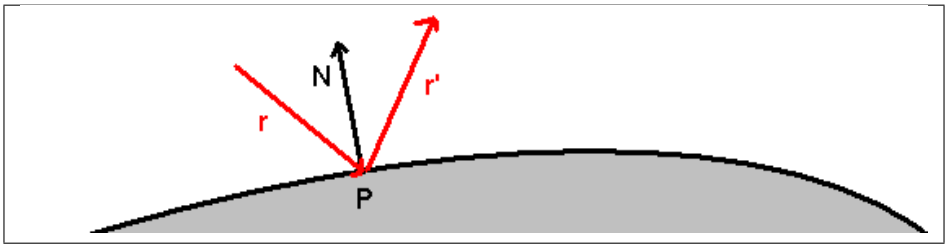



**Figure 29.7 :** *Rayon réfléchi, $N$ est le vecteur normal, $r$ est le rayon incident, $r'$ est le rayon réfléchi construit à l'aide de l'expression (29.2). Les vecteurs $r$ et $r'$ sont symétriques par rapport à la droite portée par $n$.*

Le programme permettant de construire des images de synthèse contenant des sphères et des facettes est réparti sur cinq fichiers. Le premier décrit les objets de base tels que les vecteurs, les couleurs, les objets, les sources de lumière, les rayons. Les autres fichiers se concentrent sur une classe en particulier, deux contiennent des objets `image_synthese_sphere.py` et `image_synthese_facette.py`. Les deux derniers implémentent chacun un modèle d'illumination différent `image_synthese_scene.py` et `image_synthese_phong.py`. Ils sont plus amplement détaillés dans le document *http ://perso.wanadoo.fr/x.dupre/programmation/python_ cours_ exemple.pdf.*

**Fichier :** `image_synthese_base.py`

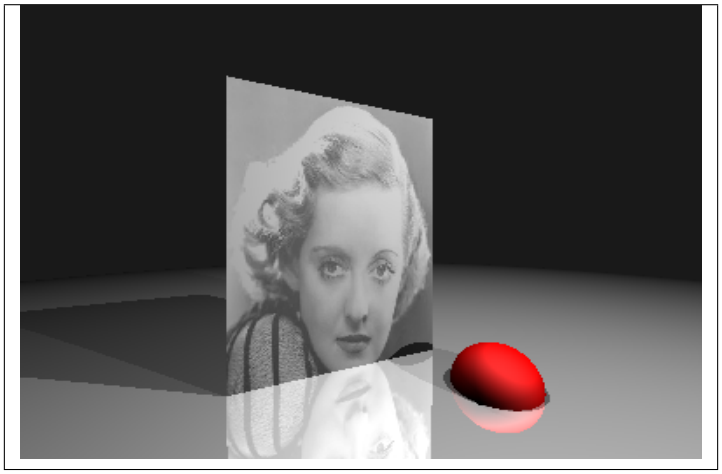


**Figure 29.8 :** *Portrait de Bette Davis partiellement caché par l'ombre d'une sphère et reflété par la plus grande des sphères.*

```
# -*- coding: cp1252 -*-
"""définition des objets permettant de construire une image de synthèse"""
import math
import copy

class vecteur (object) :
    """définit ce qu'est un point"""
    __slots__ = "x","y","z"

    def __str__ (self):
        """pour l'affichage"""
        return "(%3.2f,%3.2f,%3.2f)" % (self.x, self.y, self.z)

    def __init__(self,x,y,z):
        """initialisation"""
        self.x, self.y, self.z = float (x), float (y), float (z)

    def __add__ (self,p):
        """addition de deux points"""
        return vecteur (self.x + p.x, self.y + p.y, self.z + p.z)

    def __neg__ (self):
        """retourne l'opposé d'un vecteur"""
        return vecteur (-self.x,-self.y,-self.z)

    def __iadd__ (self,p):
        """addition de deux points"""
        self.x += p.x
        self.y += p.y
        self.z += p.z
        return self

    def __sub__ (self,p):
        """soustraction de deux points"""
        return vecteur (self.x - p.x, self.y - p.y, self.z - p.z)

    def __isub__ (self,p):
        """soustraction de deux points"""
        self.x -= p.x
        self.y -= p.y
        self.z -= p.z
        return self
```

```
    def __mul__ (self,x):
        """multiplication par un scalaire"""
        return vecteur (self.x * x, self.y * x, self.z * x)

    def __imul__ (self,x):
        """multiplication par un scalaire"""
        self.x *= x
        self.y *= x
        self.z *= x
        return self

    def __div__ (self,x):
        """division par un scalaire"""
        return vecteur (self.x / x, self.y / x, self.z / x)

    def __idiv__ (self,x):
        """division par un scalaire"""
        self.x /= x
        self.y /= x
        self.z /= x
        return self

    def norme2 (self) :
        """retourne la norme du vecteur au carrée"""
        return self.x * self.x + self.y * self.y + self.z * self.z

    def scalaire (self, v) :
        """calcule le produit scalaire entre self et v"""
        return self.x * v.x + self.y * v.y + self.z * v.z

    def vectoriel (self, v) :
        """calcule le produit vectoriel entre self et v"""
        res   = vecteur (0,0,0)
        res.x = self.y * v.z - self.z * v.y
        res.y = self.z * v.x - self.x * v.z
        res.z = self.x * v.y - self.y * v.x
        return res

    def norme (self) :
        """retourne la norme du vecteur"""
        return math.sqrt (self.norme2 ())

    def renorme (self) :
        """renorme ce vecteur"""
        n = self.norme ()
        if n > 0 : return self / n
        else : return copy.copy (self)

    def cosinus (self, v) :
        """retourne le cosinus de entre le vecteur self et le vecteur r"""
        sc = self.scalaire (v)
        n1 = self.norme ()
        n2 = v.norme ()
        n  = n1 * n2
        if n == 0 : return 0
        return float (sc) / float (n)

    def sinus (self, v, norm) :
        """retourne le sinus de entre le vecteur self et le vecteur r,
        norm est un vecteur normal et de norme 1 permettant d'orienter
        le plan dans lequel se trouve les deux vecteurs dont il faut mesurer le sinus"""
        sc = self.vectoriel (v)
        n1 = self.norme ()
        n2 = v.norme ()
```

```
        n  = n1 * n2
        if n == 0 : return 0
        return sc.scalaire (norm) / float (n)

    def angle (self, v, norm) :
        """retourne l'angle entre les vecteur self et v,
        retourne un angle compris entre -pi et pi,
        norm est la direction du vecteur normal au plan des deux vecteurs"""
        cos     = self.cosinus (v)
        sin     = self.sinus (v, norm)
        angle   = math.atan2 (sin, cos)
        if angle > math.pi : angle -= math.pi * 2
        return angle

    def diff_abs (self, v):
        """retourne la somme des valeurs absolues des différentes entre coordonnées"""
        r  = abs (self.x - v.x)
        r += abs (self.y - v.y)
        r += abs (self.z - v.z)
        return r

    def __eq__ (self, v) :
        """définit l'égalité entre deux vecteurs"""
        if v == None : return False
        return self.diff_abs (v) < 1e-10

    def __ne__ (self, v) :
        """définit l'égalité entre deux vecteurs"""
        if v == None : return True
        return self.diff_abs (v) > 1e-10


class couleur (vecteur) :
    """une couleur est un vecteur dont les coordonnées sont comprises entre 0 et 1,
    x <--> rouge, y <--> vert, z <--> bleu"""

    def __init__ (self, x,y,z) :
        vecteur.__init__(self, x,y,z)
        self.borne ()

    def borne (self) :
        """si une couleur est hors bornes, réajuste la couleur, prend le maximum devient 1,
        les autres intensités sont ajustées selon ce facteur d'échelle"""
        if self.x < 0 : self.x = 0
        if self.y < 0 : self.y = 0
        if self.z < 0 : self.z = 0
        m = max (self.x, self.y)
        m = max (m, self.z)
        if m > 1 :
            self.x /= m
            self.y /= m
            self.z /= m

    def __add__ (self,p):
        """addition de deux couleurs"""
        return couleur (self.x + p.x, self.y + p.y, self.z + p.z)

    def produit_terme (self, v) :
        """effectue un produit terme à terme"""
        return couleur (self.x * v.x, self.y * v.y, self.z * v.z)

    def __mul__ (self,x):
        """multiplication par un scalaire"""
        return couleur (self.x * x, self.y * x, self.z * x)
```

```
class repere (object) :
    """définition d'un repère orthonormé"""

    def __init__ (self,   origine = vecteur (0,0,0), \
                            axex    = vecteur (1,0,0), \
                            axey    = vecteur (0,1,0), \
                            axez    = vecteur (0,0,1)) :
        """initialisation, origine et les trois axes"""
        self.origine    = origine
        self.x          = axex
        self.y          = axey
        self.z          = axez

    def coordonnees (self, v) :
        """on suppose que les coordonnées de v sont exprimées dans ce repère,
        calcule les coordonnées de v dans le repère d'origine"""
        res      = copy.copy (self.origine)
        res.x  += v.x * self.x.x +  v.y * self.y.x + v.z * self.z.x
        res.y  += v.x * self.x.y +  v.y * self.y.y + v.z * self.z.y
        res.z  += v.x * self.x.z +  v.y * self.y.z + v.z * self.z.z
        return res

    def __str__ (self):
        """affichage"""
        s  = "origine : " + str (self.origine) + "\n"
        s += "axe des x : " + str (self.x) + "\n"
        s += "axe des y : " + str (self.y) + "\n"
        s += "axe des z : " + str (self.z) + "\n"
        return s

class pixel (object) :
    """définit ce qu'est un pixel"""
    __slots__ = "x","y"

    def __init__(self,x,y):
        """initialisation"""
        self.x, self.y = int (x), int (y)

    def __str__ (self):
        """pour l'affichage"""
        return "(%d, %d)" % (self.x, self.y)




class rayon (object) :
    """définit ce qu'est un rayon"""
    __slots__ = "origine", "direction", "pixel", "couleur"

    def __init__ (self, origine, direction, pixel, couleur):
        """initialisation"""
        self.origine, self.direction, self.pixel, self.couleur = \
                origine, direction, pixel, couleur

    def __str__ (self):
        """pour l'affichage"""
        s  = "origine   : "     + str (self.origine)
        s += " direction : "    + str (self.direction)
        s += " pixel : "        + str (self.pixel)
        s += " couleur : "      + str (self.couleur)
        return s
```

```
class objet (object):
    """définit l'interface pour un objet à dessiner dans une image de synthese"""

    def intersection (self, r) :
        """retourne le point d'intersection avec le rayon r,
        retourne None s'il n'y pas d'intersection"""
        return None

    def normale (self, p, rayon) :
        """retourne la normale au point de coordonnée p, et connaissant le rayon"""
        return None

    def couleur_point (self, p) :
        """retourne la couleur au point de coordonnée p"""
        return None

    def rayon_refracte (self, rayon, p) :
        """retourne le rayon réfracté au point p de la surface,
        si aucune, retourne None"""
        return None

    def rayon_reflechi (self, rayon, p) :
        """retourne le rayon réfléchi au point p de la surface,
        si aucune, retourne None"""
        return None

    def phong_coefficient (self):
        """retourne un coefficient propre à l'objet pour
        le modèle d'illumination de Phong"""
        return float (0)



class source (object) :
    """définition d'une source ponctuelle"""
    __slots__ = "origine", "couleur"

    def __init__ (self, origine, couleur):
        """initialisation"""
        self.origine, self.couleur = origine, couleur

    def __str__ (self) :
        """affichage"""
        return "source : " + str (self.origine) + " couleur : " + str (self.couleur)

if __name__ == "__main__" :
    v = vecteur (0,1,2)
    u = vecteur (0,1,2)
    w = u + v
    print u,v,w
    print w * 6
    p = pixel (5,5)
    print p
    c = couleur (1,1,1)
    print c
    r = rayon (u,w,p,c)
    print r
    s = source (v, c)
    print s
```

**Fichier :** `image_synthese_sphere.py`

```
# -*- coding: cp1252 -*-
"""définition d'une sphère"""
```

```
import image_synthese_base as base
import math

class sphere (base.objet):
    """définit une sphère"""
    __slots__ = "centre", "rayon", "couleur"

    def __init__ (self, centre, rayon, couleur):
        """initialisation"""
        self.centre, self.rayon, self.couleur = centre, float (rayon), couleur

    def intersection (self, r) :
        """retourne le point d'intersection avec le rayon r,
        retourne None s'il n'y pas d'intersection"""
        oc      = self.centre - r.origine
        vn      = r.direction.norme2 ()
        s       = r.direction.scalaire (oc)
        delta   = s*s - vn * (oc.norme2 () - self.rayon * self.rayon)
        if delta < 0 : return None
        delta   = math.sqrt (delta)
        l1      = (s - delta) / vn
        l2      = (s + delta) / vn

        if      0 < l1 < l2 :   l = l1
        elif    l1 < 0 < l2 :   l = l2
        elif    0 < l2 < l1 :   l = l2
        elif    l2 < 0 < l1 :   l = l1
        else :                  l = None

        if l == None : return None

        v = r.origine + r.direction * l
        return v

    def normale (self, p, rayon) :
        """retourne la normale au point de coordonnée p"""
        v = (p - self.centre) / self.rayon
        return v

    def couleur_point (self, p) :
        """retourne la couleur au point de coordonnée p"""
        return self.couleur

    def __str__ (self):
        """affichage"""
        s  = "sphère --- centre : " + str (self.centre)
        s += " rayon : " + str (self.rayon)
        s += " couleur : " + str (self.couleur)
        return s



if __name__ == "__main__" :
    s = sphere (base.vecteur (0,0,0), 5, base.couleur (0,1,0))
    r = base.rayon (    base.vecteur (10,0,0), base.vecteur (1,0,0), \
                        base.pixel (0,0), base.couleur (0,0,0))
    print s
    print r
    p = s.intersection (r)
    print p
```

**Fichier :** image_synthese_scene.py

```
# -*- coding: cp1252 -*-
"""définition d'une scène"""
import image_synthese_base as base
import image_synthese_sphere as obj
import math
import pygame

class scene (object):
    """définit une scène, les axes x,y sont ceux de l'écran,
    z-1 est la distance à l'écran du point (x,y,z)"""

    def __init__ (self, repere, alpha, x,y) :
        """définit la position de l'oeil, l'angle d'ouverture,
        et la taille de l'écran"""
        self.repere = repere
        self.alpha  = float (alpha)
        self.dim    = (int (x), int (y))

    def ajoute_source (self, source):
        """ajoute une source ponctuelle de lumière"""
        if not self.__dict__.has_key ("sources") : self.sources = []
        self.sources.append (source)

    def ajoute_objet (self, objet):
        """ajoute un objet à la scène"""
        if not self.__dict__.has_key ("objets") : self.objets = []
        self.objets.append (objet)

    def __str__ (self) :
        """affichage"""
        s  = "scène ----------------------------\n"
        s += "repère : " + str (self.repere) + "\n"
        s += "angle d'ouverture : " + str (self.alpha) + "\n"
        s += "dimension de l'écran : " + str (self.dim) + "\n"
        if self.__dict__.has_key ("sources") :
            for a in self.sources : s += "   " +str (a) + "\n"
        if self.__dict__.has_key ("objets") :
            for a in self.objets :  s += "   " + str (a) + "\n"
        return s

    def intersection (self, rayon) :
        """calcule le point d'intersection entre un rayon et le plus proche des objets,
        retourne l'objet et le point d'intersection"""
        if not self.__dict__.has_key ("objets") : return None, None
        p       = rayon.origine
        sp,so   = None, None
        for o in self.objets :
            i = o.intersection (rayon)
            if i == None : continue
            if rayon.direction.scalaire (i - p) <= 0 : continue
            if i == rayon.origine : continue
            if sp == None :
                sp  = i
                so  = o
            else :
                v = i - p
                d = sp - p
                if v.norme2 () < d.norme2 () :
                    sp = i
                    so = o
        return so, sp

    def sources_atteintes (self, p) :
        """retourne la liste des sources atteintes depuis une position p de l'espace,
```

```
        vérifie qu'aucun objet ne fait obstacle"""
        res = []
        for s in self.sources:
            r   = base.rayon (s.origine, p - s.origine, base.pixel (0,0), s.couleur)
            o,i = self.intersection (r)
            if i == None : continue
            if (i - p).norme2 () < 1e-10 :   # possible problème d'arrondi
                res.append (s)
                continue
        return res

    def construit_rayon (self, pixel) :
        """construit le rayon correspondant au pixel pixel"""
        x = (pixel.x - self.dim [0] / 2) * math.tan (self.alpha / 2) / min (self.dim)
        y = (pixel.y - self.dim [1] / 2) * math.tan (self.alpha / 2) / min (self.dim)
        v = base.vecteur (x,y,1)
        r = base.rayon (self.repere.origine, self.repere.coordonnees (v), \
                            pixel, base.couleur (1,1,1))
        return r

    def modele_illumination (self, rayon, p, obj, source) :
        """calcule la couleur pour un rayon donné, un point p, un objet obj,
        et une source de lumière source"""
        n   = obj.normale (p, rayon)
        cos = n.cosinus (source.origine - p)
        cl  = obj.couleur_point (p) * cos
        cl  = cl.produit_terme (rayon.couleur)
        return cl

    def couleur_fond (self) :
        """retourne la couleur du fond"""
        return base.couleur (0,0,0)

    def rayon_couleur (self, rayon, ref = True) :
        """retourne la couleur d'un rayon connaissant les objets,
        cette fonction doit être surchargée pour chaque modèle d'illumination,
        si ref == True, on tient compte des rayons réfracté et réfléchi"""

        list_rayon = [ rayon ]
        c = base.couleur (0,0,0)
        b = False
        while len (list_rayon) > 0 :
            r   = list_rayon.pop ()
            o,p = self.intersection (r)

            if p == None : continue

            if ref :
                t   = o.rayon_refracte (r, p)
                if t != None : list_rayon.append (t)
                t   = o.rayon_reflechi (r, p)
                if t != None : list_rayon.append (t)

            sources = self.sources_atteintes (p)
            if len (sources) == 0 : return base.couleur (0,0,0)
            for s in sources :
                cl  = self.modele_illumination (r, p, o, s)
                c  += cl
                b   = True

        if not b : c = self.couleur_fond ()
        else : c.borne ()
        return c

    def construit_image (self, screen):
```

```
        """construit l'image de synthèse où screen est un objet du module pygame"""
        count   = 0
        nbpixel = int (self.dim [0] * self.dim [1] / 100)
        for y in xrange (0, self.dim [1]) :
            for x in xrange (0, self.dim [0]) :
                p = base.pixel (x,y)
                r = self.construit_rayon (p)
                c = self.rayon_couleur (r, True)
                q = (p.x,self.dim [1] - p.y - 1)
                d = (int (c.x * 255), int (c.y * 255), int (c.z * 255))
                pygame.draw.line (screen, d, q,q)
                count += 1
                if count % 150 == 0 : pygame.display.flip ()
                if count % nbpixel == 0 : print "avancement " , count // nbpixel , "%"
        pygame.display.flip ()


def attendre_clic ():
    """dessine une croix sur l'écran et attend la pression d'un clic de souris"""
    reste = True
    while reste:
        for event in pygame.event.get():
            if event.type == pygame.MOUSEBUTTONUP :
                reste = False
                break


if __name__ == "__main__" :
    s = scene (base.repere (), math.pi / 1.5, 400, 300)
    s.ajoute_source ( base.source (base.vecteur (0,10,10), \
                        base.couleur (1,1,1) ) )
    s.ajoute_source ( base.source (base.vecteur (10,10,5), \
                        base.couleur (0.5,0.5,0.5) ) )
    s.ajoute_objet  ( obj.sphere  (base.vecteur (0,0,12), \
                        3, base.couleur (1,0,0) ) )
    s.ajoute_objet  ( obj.sphere  (base.vecteur (0,-400,12), \
                        396, base.couleur (0.5,0.5,0.5) ) )
    print s

    screen  = pygame.display.set_mode (s.dim)
    screen.fill ((255,255,255))
    s.construit_image (screen)

    print "image terminée"
    attendre_clic ()
```

**Fichier :** image_synthese_phong.py

```
# -*- coding: cp1252 -*-
"""implémentation du modèle d'illumination de Phong"""
import image_synthese_scene as scene
import image_synthese_base as base
import image_synthese_sphere as obj
import math
import pygame


class scene_phong (scene.scene):
    """définit une scène et utilise le modèle d'illumination de Phong
    pour construire l'image de synthèse"""

    def __init__ (self, repere, alpha, x,y,
                  ka          = 0.1,
```

```
                    kb          = 0.8,
                    kc          = 0.3,
                    reflet      = 6,
                    fond        = base.couleur (200,200,200)) :
        """définit la position de l'oeil, l'angle d'ouverture,
        et la taille de l'écran"""
        scene.scene.__init__ (self, repere, alpha, x, y)
        self.ka, self.kb, self.kc   = ka,kb,kc
        self.reflet                 = reflet
        self.fond                   = fond
        self.constante              = float (1)

    def __str__ (self) :
        """affichage"""
        s  = scene.scene.__str__ (self)
        s += "ka = %1.3f   kb = %1.3f   kc = %1.3f\n" % (self.ka,self.kb,self.kc)
        s += "reflet " + str (self.reflet) + "\n"
        s += "couleur du fond " + str (self.fond) + "\n"
        return s

    def couleur_fond (self) :
        """retourne la couleur du fond"""
        return self.fond * self.ka

    def modele_illumination (self, rayon, p, obj, source) :
        """calcule la couleur pour un rayon donné, un point p, un objet obj,
        et une source de lumière source"""
        n   = obj.normale (p, rayon).renorme ()
        vr  = rayon.direction.renorme ()
        vr *= float (-1)
        vs  = source.origine - p
        vs  = vs.renorme ()
        bi  = vs + vr
        bi  = bi.renorme ()

        # premier terme
        cos      = n.scalaire (vs)
        couleur  = source.couleur.produit_terme (obj.couleur_point (p)) * (cos * self.kb)

        # second terme : reflet
        cos      = n.scalaire (bi) ** self.reflet
        couleur += source.couleur.produit_terme (source.couleur) * (cos * self.kc)
        couleur  = couleur.produit_terme (rayon.couleur)

        return couleur


if __name__ == "__main__" :
    s = scene_phong (base.repere (), math.pi / 1.5, 400, 300)

    s.ajoute_source ( base.source (base.vecteur (0,10,10), \
                        base.couleur (1,1,1) ) )
    s.ajoute_source ( base.source (base.vecteur (10,10,5), \
                        base.couleur (0.5,0.5,0.5) ) )
    s.ajoute_objet  ( obj.sphere  (base.vecteur (0,0,12), \
                        3, base.couleur (1,0,0) ) )
    s.ajoute_objet  ( obj.sphere  (base.vecteur (0,-400,12), \
                        396, base.couleur (0.5,0.5,0.5) ) )
    print s

    screen  = pygame.display.set_mode (s.dim)
    screen.fill ((255,255,255))
    s.construit_image (screen)

    print "image terminée"
```

```
    scene.attendre_clic ()
```

**Fichier :** `image_synthese_facette.py`

```
# -*- coding: cp1252 -*-
"""définition d'une sphère"""
import image_synthese_base as base
import image_synthese_sphere as obj
import image_synthese_phong as scene
import image_synthese_scene as scene_p
import pygame
import math

class facette (base.objet):
    """définit un triangle dans l'espace"""

    def __init__ (self, a,b,c, couleur):
        """initialisation"""
        self.a, self.b, self.c = a,b,c
        ab = b - a
        ac = c - a
        self.vnorm = ab.vectoriel (ac)
        self.vnorm = self.vnorm.renorme ()
        self.couleur = couleur

    def intersection_plan (self, r) :
        """retourne le point d'intersection entre le plan et le rayon r"""
        if r.direction.scalaire (self.vnorm) == 0 :
            return None
        oa      = self.a - r.origine
        l       = self.vnorm.scalaire (oa) / self.vnorm.scalaire (r.direction)
        p       = r.origine + r.direction * l
        return p

    def point_interieur (self, p) :
        """dit si un point appartient à l'intérieur du triangle"""
        pa      = self.a - p
        pb      = self.b - p
        pc      = self.c - p
        theta   = pa.angle (pb, self.vnorm)
        theta  += pb.angle (pc, self.vnorm)
        theta  += pc.angle (pa, self.vnorm)
        theta   = abs (theta)
        if theta >= math.pi * 0.9 : return True
        else : return False

    def intersection (self, r) :
        """retourne le point d'intersection avec le rayon r,
        retourne None s'il n'y pas d'intersection"""
        p = self.intersection_plan (r)
        if p == None : return None
        if self.point_interieur (p) : return p
        else : return None

    def normale (self, p, rayon) :
        """retourne la normale au point de coordonnée p et connaissant le rayon"""
        if rayon.direction.scalaire (self.vnorm) < 0 :
            return self.vnorm
        else :
            return - self.vnorm

    def couleur_point (self, p) :
        """retourne la couleur au point de coordonnée p"""
```

```
        return self.couleur

    def __str__ (self):
        """affichage"""
        s  = "facette --- a : " + str (self.a)
        s += " b : " + str (self.b)
        s += " c : " + str (self.c)
        s += " couleur : " + str (self.couleur)
        return s


class rectangle (facette):
    """définit un rectangle dans l'espace"""

    def __init__ (self, a,b,c,d, couleur):
        """initialisation, si d == None, d est calculé comme étant
        le symétrique de b par rapport au milieu du segment [ac]"""
        facette.__init__(self, a,b,c, couleur)
        if d != None : self.d = d
        else :
            i       = (a + c) / 2
            self.d  = b + (i-b) * 2

    def point_interieur (self, p) :
        """dit si un point appartient à l'intérieur du triangle"""
        pa      = self.a - p
        pb      = self.b - p
        pc      = self.c - p
        pd      = self.d - p
        theta   = pa.angle (pb, self.vnorm)
        theta  += pb.angle (pc, self.vnorm)
        theta  += pc.angle (pd, self.vnorm)
        theta  += pd.angle (pa, self.vnorm)
        theta   = abs (theta)
        if theta >= math.pi * 0.9 : return True
        else : return False

    def __str__ (self):
        """affichage"""
        s  = "rectangle --- a : " + str (self.a)
        s += " b : " + str (self.b)
        s += " c : " + str (self.c)
        s += " d : " + str (self.d)
        s += " couleur : " + str (self.couleur)
        return s




if __name__ == "__main__" :

    s = scene.scene_phong (base.repere (), math.pi / 1.5, 400, 300)

    s.ajoute_source ( base.source (base.vecteur (0,8,8), \
                        base.couleur (0.6,0.6,0.6) ) )
    s.ajoute_source ( base.source (base.vecteur (10,0,0), \
                        base.couleur (0.6,0.6,0.6) ) )
    s.ajoute_source ( base.source (base.vecteur (8,8,4.5), \
                        base.couleur (0.6,0.6,0.6) ) )
    s.ajoute_objet  ( obj.sphere  (base.vecteur (1,0,5), \
                        1, base.couleur (1,0,0) ) )
    s.ajoute_objet  ( obj.sphere  (base.vecteur (0,-400,12), \
                        396, base.couleur (0.5,0.5,0.5) ) )
    s.ajoute_objet (facette ( base.vecteur (0,-2.5,6),   \
```

```
                            base.vecteur (-2,-2.5,3), \
                            base.vecteur (1,-3.5,4.5), \
                            base.couleur (0.2,0.8,0)))
    s.ajoute_objet (rectangle ( base.vecteur (0,-2.5,6),   \
                            base.vecteur (-2,-2.5,3), \
                            base.vecteur (-2,2.8,3.5), \
                            None, \
                            base.couleur (0,0,1)))
    print s

    screen  = pygame.display.set_mode (s.dim)
    screen.fill ((255,255,255))
    s.construit_image (screen)

    print "image terminée"
    scene_p.attendre_clic ()
```

### 29.3.2 Correction

Le fichier `image_synthese_facette_image.py` contient la réponse aux deux questions de l'énoncé de ce TD. La scène qui y est incluse en exemple contient à la fois une facette rectangulaire dont la surface est une image et une sphère en partie réfléchissante puisqu'elle réfléchit un rayon d'une intensité diminué de moitié par rapport à celle du rayon incident.

**1)** La classe `rectangle_image` est un nouvel objet qui hérite de la classe `rectangle` du fichier `image_synthese_facette.py`. Le comportement de ce nouvel objet est identique à celui d'un rectangle, seule la couleur n'est plus uniforme le long de la surface (méthode `couleur_point`) puisqu'elle est obtenue depuis une image. La phase d'initialisation change aussi pour charger l'image.

Le fonction `couleur_point` considère que les vecteurs $\overrightarrow{AB}$ et $\overrightarrow{AD}$ forment le repère dans lequel sera affichée l'image. Cette fonction calcule la correspondance entre les coordonnées du point d'intersection avec un rayon de lumière et le pixel de l'image qui doit y être affiché.

**2)** La classe `sphere_reflet` redéfinit la méthode `rayon_reflechi` afin de retourner un rayon réfléchi défini par l'expression (29.2) (page 776). Le paramètre `reflet` présent dans le constructeur `__init__` de la classe `sphere_reflet` permet de définir l'intensité lumineuse réfléchie par la surface. Pour l'exemple de scène incluse dans ce programme, l'intensité du rayon réfléchi est la moitié de celle du rayon incident.

**Fichier :** `image_synthese_facette_image.py`

```
# -*- coding: cp1252 -*-
"""définition d'une sphère"""
import image_synthese_base as base
import image_synthese_sphere as obj
import image_synthese_phong as scene
import image_synthese_scene as scene_p
import image_synthese_facette as facette
import pygame
import math
import psyco



class rectangle_image (facette.rectangle):
    """définit un rectangle contenant un portrait"""
```

```
    def __init__(self, a,b,c,d, nom_image, invertx = False):
        """initialisation, si d == None, d est calculé comme étant
        le symétrique de b par rapport au milieu du segment [ac],
        la texture est une image,
        si invertx == True, inverse l'image selon l'axe des x"""
        facette.rectangle.__init__(self, a,b,c,d, base.couleur (0,0,0))
        self.image      = pygame.image.load (nom_image)
        self.nom_image  = nom_image
        self.invertx    = invertx

    def __str__ (self):
        """affichage"""
        s  = "rectangle image --- a : " + str (self.a)
        s += " b : " + str (self.b)
        s += " c : " + str (self.c)
        s += " d : " + str (self.d)
        s += " image : " + self.nom_image
        return s

    def couleur_point (self, p) :
        """retourne la couleur au point de coordonnée p"""
        ap  = p - self.a
        ab  = self.b - self.a
        ad  = self.d - self.a
        abn = ab.norme2 ()
        adn = ad.norme2 ()
        x   = ab.scalaire (ap) / abn
        y   = ad.scalaire (ap) / adn
        sx,sy = self.image.get_size ()
        k,l   = int (x * sx), int (y * sy)
        k   = min (k, sx-1)
        l   = min (l, sy-1)
        l   = sy - l - 1
        if not self.invertx :
            c   = self.image.get_at ((k,l))
        else :
            c   = self.image.get_at ((sx-k-1,l))
        cl  = base.couleur (float (c [0]) / 255, float (c [1]) / 255, float (c [2]) / 255)
        return cl

class sphere_reflet (obj.sphere) :
    """implémente une sphère avec un reflet"""

    def __init__ (self, centre, rayon, couleur, reflet):
        """initialisation, reflet est un coefficient de réflexion"""
        obj.sphere.__init__ (self, centre, rayon, couleur)
        self.reflet = reflet

    def __str__ (self):
        """affichage"""
        s  = "sphère reflet --- centre : " + str (self.centre)
        s += " rayon : " + str (self.rayon)
        s += " couleur : " + str (self.couleur)
        return s

    def rayon_reflechi (self, rayon, p) :
        """retourne le rayon réfléchi au point p de la surface,
        si aucune, retourne None"""
        if p == rayon.origine : return None
        n = self.normale (p, rayon)
        n = n.renorme ()
        y = n.scalaire (rayon.direction)
        d = rayon.direction - n * y * 2
        r = base.rayon (p, d, rayon.pixel, rayon.couleur * self.reflet)
        return r
```

```
if __name__ == "__main__" :

    psyco.full ()

    s = scene.scene_phong (base.repere (), math.pi / 1.5, 400, 200)

    s.ajoute_source ( base.source (base.vecteur (0,8,8), \
                        base.couleur (0.4,0.4,0.4) ) )
    s.ajoute_source ( base.source (base.vecteur (10,0,0), \
                        base.couleur (0.4,0.4,0.4) ) )
    s.ajoute_source ( base.source (base.vecteur (8,8,4.5), \
                        base.couleur (0.4,0.4,0.4) ) )
    s.ajoute_objet  ( obj.sphere  (base.vecteur (3,-4,7), \
                        1, base.couleur (1,0,0) ) )
    s.ajoute_objet  ( sphere_reflet  (base.vecteur (0,-400,12), \
                        396, base.couleur (0.5,0.5,0.5), 0.5 ) )

    s.ajoute_objet (rectangle_image ( base.vecteur (8,-3.5,9),   \
                                base.vecteur (2,-3.5,8), \
                                base.vecteur (2,3.8,8), \
                                None, \
                                "bette_davis.png",
                                invertx = True))

    s.ajoute_source ( base.source (base.vecteur (7,2,8), \
                        base.couleur (0.2,0.2,0.2) ) )
    s.ajoute_source ( base.source (base.vecteur (12.5,3,5), \
                        base.couleur (0.2,0.2,0.2) ) )

    s.ajoute_source ( base.source (base.vecteur (-12.5,1,6), \
                        base.couleur (0.2,0.2,0.2) ) )

    s.ajoute_objet (facette.rectangle ( base.vecteur (-12.4,0.99,5.9),   \
                                base.vecteur (-12.6,0.99,5.9), \
                                base.vecteur (-12.6,0.99,6.1), \
                                None, \
                                base.couleur (0,0,0)))

    print s

    screen  = pygame.display.set_mode (s.dim)
    screen.fill ((255,255,255))
    s.construit_image (screen)

    print "sauvegarde de l'image"
    im = pygame.display.get_surface()
    print "image size : ", im.get_size ()
    pygame.image.save (im, "c:\\temp\\image.jpg")
    pygame.image.save (im, "c:\\temp\\image.tif")
    pygame.image.save (im, "c:\\temp\\image.bmp")

    print "image terminée"
    scene_p.attendre_clic ()
```

**fin correction TD 29.3.1** □

# Chapitre 30

# Exercices plutôt informatiques

## 30.1 Conserver ses fichiers sur une clé USB avec une interface graphique

### 30.1.1 Enoncé

L'objectif de ce TD est de reprendre le programme du TD 21.1.1 et de lui adjoindre une fenêtre graphique pour le rendre plus facilement utilisable comme celle de la figure 30.1.

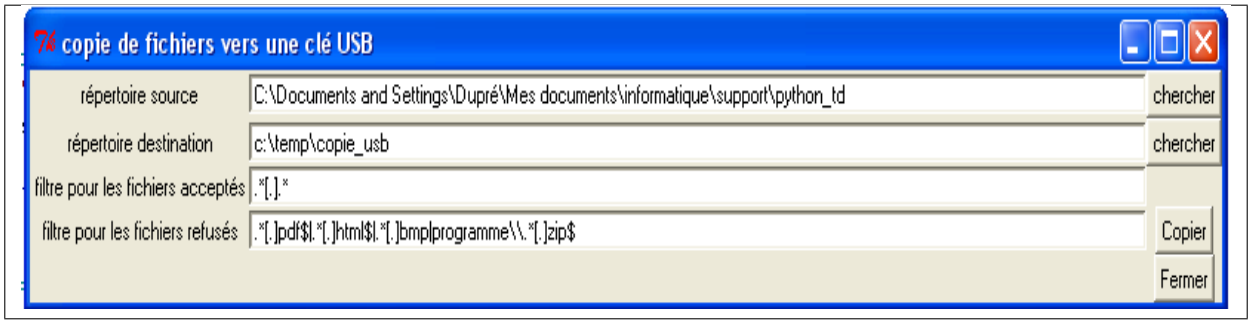



**Figure 30.1 :** *Exemple de fenêtre demandée pour le TD 30.1.1. Elle permet de modier les répertoires source et destination, les deux filtres, définis au TD 21.1.1, puis de lancer la copie.*

**1)** Il s'agit dans cette première question de dessiner la fenêtre graphique. Trois types d'objet issus du module `Tkinter` ou *widgets* interviennent :

| | |
|---|---|
| `Label` | Cet objet permet d'afficher du texte, des légendes, permettant d'indiquer ce qu'il faut saisir dans les objets de saisie voisins. |
| `Entry` | Cet objet permet de saisir du texte, un nom de répertoire, ou un filtre par exemple. |
| `Button` | Insère un bouton dans une fenêtre graphique. Relié à une fonction, la pression de ce bouton exécute cette fonction. |

L'exemple de code suivant permet de créer un objet pour chacun de ces types et de les placer en diagonale. Cet exemple permet d'obtenir la fenêtre de la figure 30.2. Ces objets doivent être précédés du préfixe `Tkinter` ou `Tk` dans l'exemple qui suit car ils appartiennent au module `Tkinter`.

```
import Tkinter as Tk              # import du module Tkinter

root = Tk.Tk ()                   # création de la fenêtre principale,
                                  # c'est toujours la première ligne

b = Tk.Button (text = "bouton")   # création d'un bouton
e = Tk.Entry ()                   # création d'une zone de saisie
l = Tk.Label (text = "légende")   # création d'une légende

b.grid (row = 0, column = 0)      # positionnement de b dans la case (0,0)
e.grid (row = 1, column = 1)      # positionnement de e dans la case (1,1)
l.grid (row = 2, column = 2)      # positionnement de e dans la case (2,2)
```

```
root.mainloop ()                    # on lance le programme et l'affichage de la fenêtre
```

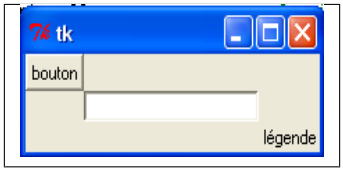



**Figure 30.2 :** *Fenêtre simple comprenant une instance des objets* `Button`, `Entry` *et* `Label`.

Il est possible d'allonger la taille d'un objet `Button`, `Label`, ou `Entry`. Il suffit d'ajouter quelques lignes entre la création des objets et le lancement de la fenêtre par l'intermédiaire de l'instruction `root.mainloop`(). Le résultat est celui de la figure 30.3.

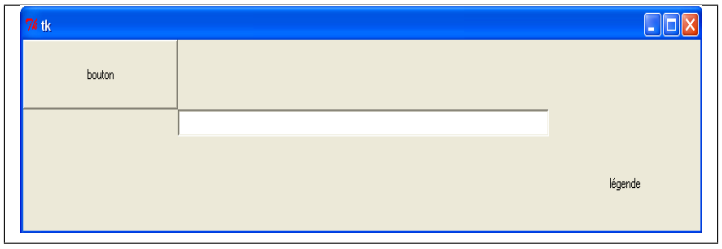



**Figure 30.3 :** *Même fenêtre que celle de la figure 30.2, les dimensions des objets sont changé.*

```
import Tkinter as Tk                # import du module Tkinter

root = Tk.Tk ()                     # création de la fenêtre principale,
                                    # c'est toujours la première ligne

b = Tk.Button (text = "bouton")     # création d'un bouton
e = Tk.Entry ()                     # création d'une zone de saisie
l = Tk.Label (text = "légende")     # création d'une légende

b.grid (row = 0, column = 0)        # positionnement de b dans la case (0,0)
e.grid (row = 1, column = 1)        # positionnement de e dans la case (1,1)
l.grid (row = 2, column = 2)        # positionnement de e dans la case (2,2)

# spécifications
b.config (width = 30, height = 4)       # largeur de 30, hauteur de 4
e.config (width = 50, font = "arial")   # largeur de 50, police arial
l.config (width = 30, height = 6)       # largeur de 30, hauteur de 6

root.mainloop ()                    # on lance le programme et l'affichage de la fenêtre
```

On désire maintenant insérer du texte dans la zone de saisie. On utilise pour cela la méthode `insert`. On obtient la figure 30.4.




**Figure 30.4 :** *On reprend le modèle de la figure 30.2, la zone de saisie contient maintenant du texte.*

```
import Tkinter as Tk              # import du module Tkinter

root = Tk.Tk ()                   # création de la fenêtre principale,
                                  # c'est toujours la première ligne

b = Tk.Button (text = "bouton")   # création d'un bouton
e = Tk.Entry ()                   # création d'une zone de saisie
l = Tk.Label (text = "légende")   # création d'une légende

b.grid (row = 0, column = 0)      # positionnement de b dans la case (0,0)
e.grid (row = 1, column = 1)      # positionnement de e dans la case (1,1)
l.grid (row = 2, column = 2)      # positionnement de e dans la case (2,2)

e.insert (0, "texte par défaut")  # on insère une chaîne de caractères dans la zone de saisie

root.mainloop ()                  # on lance le programme et l'affichage de la fenêtre
```

Ces trois exemples permettent de dessiner une fenêtre graphique semblable à celle de la figure 30.1 qui contient quatre zones de saisie (`Entry`) et quatre zones de textes (`Label`). Elle contient aussi quatre boutons (`Button`). Dans un premier temps, on se contentera des boutons intitulés `Fermer` et `Copier`. Les deux autres seront utilisés dans la dernière partie de cet énoncé. Pour réaliser ce programme, on peut par exemple créer une nouvelle classe qui héritera de celle créée lors du TD 21.1.1.

```
class copie_usb_window (copie_usb.copie_usb):

    def fenetre (self) :
        # ici, création de la fenêtre graphique
        # ...
```

Pour afficher la fenêtre graphique et lancer la copie, on utilise une liste d'instructions semblable à celle utilisé dans le TD 21.1.1.

```
# programme principal
ch1             = "C:\\Documents and Settings\\Dupré\\" \
                    "Mes documents\\informatique\\support\\python_td"
ch2             = "c:\\temp\\copie_usb"
filtre_accept   = ".*[.].*"
filtre_refuse   = ".*[.]pdf$|.*[.]html$|.*[.]bmp|programme\\\\.*[.]zip$"

c = copie_usb_window (ch1, ch2, filtre_accept, filtre_refuse)
c.fenetre ()   # affiche la fenêtre graphique
```

**2)** Il faut maintenant faire en sorte que la pression d'un bouton entraîne à une action. On reprend le premier exemple et on affecte au bouton `b` une fonction qui écrire une chaîne de caractères. Ceci mène au programme suivant :

```
import Tkinter as Tk              # import du module Tkinter

root = Tk.Tk ()                   # création de la fenêtre principale,
                                  # c'est toujours la première ligne

b = Tk.Button (text = "bouton")   # création d'un bouton
e = Tk.Entry ()                   # création d'une zone de saisie
l = Tk.Label (text = "légende")   # création d'une légende

b.grid (row = 0, column = 0)      # positionnement de b dans la case (0,0)
e.grid (row = 1, column = 1)      # positionnement de e dans la case (1,1)
```

```
l.grid (row = 2, column = 2)      # positionnement de e dans la case (2,2)

# fonction à exécuter si le bouton b est pressé
def affichage_bouton ():
    print "chaîne affichée si le bouton est pressé"

# on affecte la fonction affichage_bouton au bouton b
b.config (command = affichage_bouton)

root.mainloop ()                  # on lance le programme et l'affichage de la fenêtre
```

On peut aussi vouloir que la fenêtre se ferme si le bouton `b` est pressé. Dans ce cas, on remplace l'avant dernière ligne par la ligne suivante :

```
b.config (command = root.destroy)
```

Lors de la pression du bouton `b`, la fenêtre disparaît et le programme se termine. Il reste à savoir comment manipuler une ligne de saisie (classe `Entry`). Pour cet exercice, les trois méthodes suivantes seront utiles.

| **Fonctions utiles :** | |
|---|---|
| `get()` | Permet d'obtenir le contenu d'une zone de saisie. |
| `delete(i,j)` | Détruit les caractères de la zone de saisie compris entre `i` et `j` exclu. |
| `insert(i,s)` | Insère dans la zone de saisie le texte `s` à la position `i`. |

On peut par exemple utiliser ces trois méthodes pour récupérer le contenu d'une zone de saisie, le mettre en majuscules, puis modifier le contenu de la zone de saisie. Dans l'exemple qui suit, on suppose que la variable `e` désigne une zone de saisie.

```
s = e.get ()    # on récupère le contenu de la zone de saisie
i = len (s)     # i est égal à la longueur de la chaîne s
e.delete (0, i) # on vide la zone de saisie
s = s.upper ()  # mise en majuscules
e.insert (0,s)  # on remplace le contenu de la zone de saisie
```

Il ne reste plus qu'à programmer les fonctions qui seront actionnées par la pression des boutons de la fenêtre graphique décrite dans la première question.

**Travail à faire :**

L'interface fenêtrée impose que les répertoires source et destination soient saisis à la main. L'interface de la figure 30.1 propose aussi que ces répertoires puissent être aussi saisis à l'aide d'une boîte de dialogue. Il suffit d'ajouter deux autresboutons (`Button`) intitulés `Chercher`, ils font appel à une fonction qui utilise le code suivant :

```
import selection_file
fs = selection_file.FileSelection (titre, "c:\\temp", file = False)
r  = fs.run ()
print r  # chaîne de caractères ou None
```

Ce code crée une seconde boîte de dialogue (celle de la figure 30.5) qui permet de sélectionner un répertoire à l'aide de la souris sans avoir à le saisir au clavier. La méthode `run` de la classe `selection_file.FileSelection` retourne la valeur `None` si le bouton `Annuler` a été pressé et un répertoire si le bouton `Ok` a été pressé. Cette boîte de dialogue est définie dans le fichier `selection_file.py`.

**Fichier** `selection_file.py`

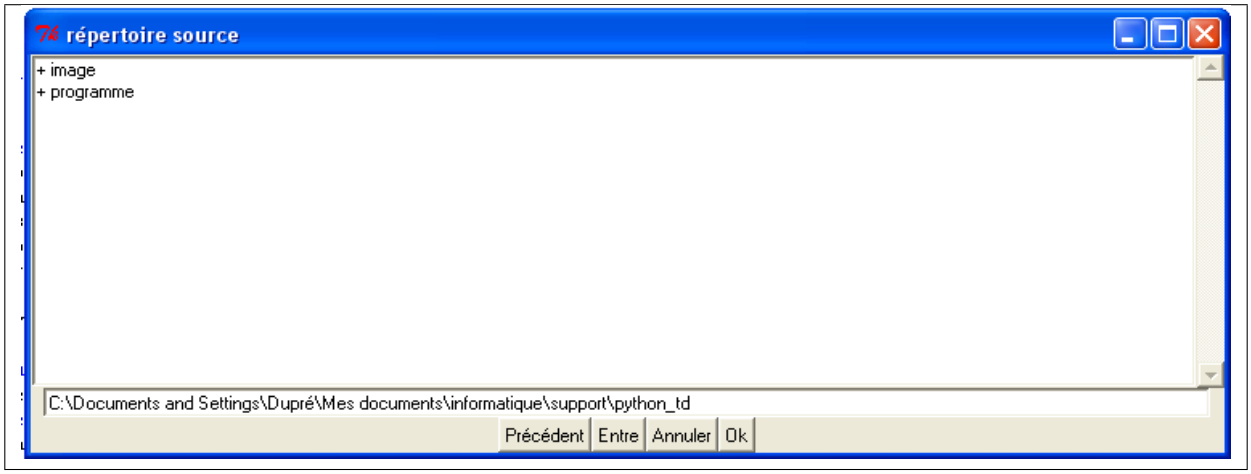



**Figure 30.5 :** *Boîte de dialogue implémentée dans le fichier* `selection_file.py` *et permettant de sélectionner un répertoire ou un fichier. Les noms de répertoires sont précédés du symbole +.*

```
# -*- coding: cp1252 -*-
"""module contenant une boîte de dialogue permettant
de sélectionner un fichier ou un répertoire,
il utilise l'interface Tkinter"""
import Tkinter as Tk
import os.path
import os

class FileSelection (object) :
    """classe permettant de sélectionner un fichier
    ou un répertoire à travers une boîte de dialogue"""

    def __init__ (self, titre = "Sélection de fichier", \
                  chemin = None, file = True, exist= True) :
        """initialise la classe
        @param      titre           titre de la fenêtre
        @param      chemin          fichier ou répertoire par défaut
        @param      file            True : fichier, False : répertoire
        @param      exist           True : le répertoire ou le fichier
                                           sélectionné doit exister"""
        self.titre  = titre
        self.chemin = chemin
        self.file   = file
        self.exist  = exist

        if self.chemin == None :  self.chemin = os.getcwd ()

    def get_list (self) :
        """retourne la liste des fichiers et des répertoires (2 listes),
        répertoires seulement et [] si self.file == False"""
        if os.path.isdir (self.chemin) :
            list    = os.listdir (self.chemin)
        else :
            ch,fi   = os.path.split (self.chemin)
            list    = os.listdir (ch)

        lifile  = []
        lidir   = []
        for l in list :
            if os.path.isdir (self.chemin + "\\" + l) :
                lidir.append (l)
            elif self.file :
```

```
            lifile.append (l)

    lidir.sort ()
    lifile.sort ()
    return lidir, lifile

def run (self) :
    """lance la boîte de dialogue et retourne la chaîne sélectionnée"""
    top         = Tk.Toplevel ()
    top.wm_title (self.titre)

    fli     = Tk.Frame (top)
    scrollbar = Tk.Scrollbar (fli)
    li      = Tk.Listbox (fli, width = 120, height = 15, \
                          yscrollcommand = scrollbar.set)
    scrollbar.config (command = li.yview)
    ch      = Tk.Entry (top, width = 120)
    f       = Tk.Frame (top)
    prec    = Tk.Button (f, text = "Précédent")
    suiv    = Tk.Button (f, text = "Entre")
    annul   = Tk.Button (f, text = "Annuler")
    ok      = Tk.Button (f, text = "Ok")

    prec.grid (column = 0, row = 0)
    suiv.grid (column = 1, row = 0)
    annul.grid (column = 3, row = 0)
    ok.grid (column = 4, row = 0)
    li.pack (side = Tk.LEFT)
    scrollbar.pack(side = Tk.RIGHT, fill = Tk.Y)
    fli.pack ()
    ch.pack ()
    f.pack ()

    def update_chemin () :
        """mise à jour du chemin dans la boîte de dialogue"""
        s = ch.get ()
        ch.delete (0, len (s))
        ch.insert (0, self.chemin)

    def update_list () :
        """mise à jour de la liste des fichiers et répertoires
        à partir de la chaîne dans la boîte de dialogue"""
        self.chemin     = ch.get ()
        lidir, lifile   = self.get_list ()
        li.delete (0, Tk.END)
        if len (lidir) > 0 :
            for l in lidir : li.insert (Tk.END, "+ "+ l)
        if len (lifile) > 0 :
            for l in lifile : li.insert (Tk.END, "  "+ l)

    def precedent () :
        """passe au répertoire précédent"""
        if os.path.isdir (self.chemin) :
            ch, last    = os.path.split (self.chemin)
            self.chemin = ch
        else :
            ch, last    = os.path.split (self.chemin)
            ch, last    = os.path.split (ch)
            self.chemin = ch
        update_chemin ()
        update_list ()

    def suivant () :
        """rentre dans un répertoire"""
        sel = ch.get ()
```

```
            if os.path.isdir (sel) :
                self.chemin = sel
                update_chemin ()
                update_list ()

        def update_sel () :
            """mise à jour de la chaîne de caractères
            dans la boîte de dialogue à partir de la ligne
            sélectionnée dans la liste"""
            li.after (200, update_sel)
            sel = li.curselection ()
            if len (sel) == 1 :
                t = li.get (sel [0])
                c = self.chemin + "\\" +  t [2:len (t)]
                c = c.replace ("\\\\", "\\")
                s = ch.get ()
                ch.delete (0, len (s))
                ch.insert (0, c)

        def annuler () :
            """annule la recherche"""
            self.resultat = False
            top.destroy ()
            top.quit ()

        def accepter () :
            """accepte le résultat"""
            self.resultat    = True
            self.chemin = ch.get ()
            top.destroy ()
            top.quit ()

        prec.config (command = precedent)
        suiv.config (command = suivant)
        annul.config (command = annuler)
        ok.config (command = accepter)

        update_chemin ()
        update_list ()
        update_sel ()
        ch.focus_set ()
        top.mainloop ()

        if self.resultat : return self.chemin
        else : return None


if __name__ == "__main__" :
    root = Tk.Tk ()

    def run () :
        r = FileSelection ("sélection d'un fichier", "c:\\")
        s = r.run ()
        print "fichier sélectionné ", s

    Tk.Button (text = "fenêtre", command = run).pack ()
    Tk.Button (text = "fermer", command = root.destroy).pack ()
    root.mainloop ()
```

### 30.1.2 Correction

Le programme qui suit répond aux deux questions de l'énoncé de ce TD ainsi qu'à la question subsidiaire. La classe `copie_usb_window` hérite de la classe `copie_usb` définie lors de la correction du TD 21.1.1 (page 21.1.2). Il ajoute deux méthodes, la première `chercher` permet de sélectionner un répertoire à l'aide d'une fenêtre (voir figure 30.5, page 796) implémentée dans le fichier `selection_file.py`. La seconde méthode `fenetre` crée une fenêtre (voir figure 30.1, page 792) qui permet de modifier les répertoires source et destination, les deux filtres puis de lancer la copie. Cette présentation est celle suggérée par la première question.

```
# -*- coding: cp1252 -*-
"""copie de fichiers sur une clé USB, interface fenêtrée"""

import Tkinter as Tk    # pour la fenêtre
import selection_file   # pour rechercher un répertoire
import copie_usb        # version sans fenêtre
import re               # pour les expressions régulières

class copie_usb_window (copie_usb.copie_usb):
    """recopie des fichiers sur une clé USB avec une fenêtre graphique"""

    def chercher (self, s, titre) :
        """modifie un répertoire"""
        fs = selection_file.FileSelection (titre, chemin = s, file = False)
        r  = fs.run ()
        if r != None : return r
        else : return s

    def fenetre (self) :
        """change les paramètres de la classe par l'intermédiaire d'une fenêtre"""

        # définition de la fenêtre
        root            = Tk.Tk ()

        source          = Tk.Entry (width = 100)
        source_label    = Tk.Label (text = "répertoire source")
        source_label.grid (column = 0, row = 0)
        source.grid (column = 1, row = 0)

        dest            = Tk.Entry (width = 100)
        dest_label      = Tk.Label (text = "répertoire destination")
        dest_label.grid (column = 0, row = 1)
        dest.grid (column = 1, row = 1)

        filtre_accept   = Tk.Entry (width = 100)
        accept_label    = Tk.Label (text = "filtre pour les fichiers acceptés")
        filtre_accept.grid (column = 1, row = 2)
        accept_label.grid (column = 0, row = 2)

        filtre_refuse   = Tk.Entry (width = 100)
        refuse_label    = Tk.Label (text = "filtre pour les fichiers refusés")
        filtre_refuse.grid (column = 1, row = 3)
        refuse_label.grid (column = 0, row = 3)

        def chercher_source () :
            s = source.get ()
            i = len (s)
            s = self.chercher (s, "répertoire source")
            source.delete (0, i)
            source.insert (0,s)

        def chercher_dest () :
            s = dest.get ()
```

```
            i = len (s)
            s = self.chercher (s, "répertoire source")
            dest.delete (0, i)
            dest.insert (0,s)

        def copier () :
            self.ch1    = source.get ()
            self.ch2    = dest.get ()
            self.accept = re.compile (filtre_accept.get ())
            self.refuse = re.compile (filtre_refuse.get ())
            self.copie ()

        brepsource      = Tk.Button (text = "chercher", command = chercher_source)
        brepdest        = Tk.Button (text = "chercher", command = chercher_dest)
        brepsource.grid (column = 2, row = 0)
        brepdest.grid (column = 2, row = 1)

        bfermer         = Tk.Button (text = "Fermer", command = root.destroy)
        bfermer.grid (column = 2, row = 4)
        bcopier         = Tk.Button (text = "Copier", command = copier)
        bcopier.grid (column = 2, row = 3)

        root.wm_title ("copie de fichiers vers une clé USB")

        source.insert (0, self.ch1)
        dest.insert (0, self.ch2)
        filtre_accept.insert (0, self.accept.pattern)
        filtre_refuse.insert (0, self.refuse.pattern)

        root.mainloop ()


if __name__ == "__main__" :
    print "copie de fichiers vers une clé USB"
    ch1             = "C:\\Documents and Settings\\Dupré\\" \
                        "Mes documents\\informatique\\support\\python_td"
    ch2             = "c:\\temp\\copie_usb"
    filtre_accept   = ".*[.].*"
    filtre_refuse   = ".*[.]pdf$|.*[.]html$|.*[.]bmp|programme\\\\.*[.]zip$"

    # filtre_accept accepte tout type de fichier
    # filtre_refuse refuse tous les fichiers dont l'extension est pdf, html ou
    # inclus dans le répertoire programme et ayant l'extension zip

    c = copie_usb_window (ch1, ch2, filtre_accept, filtre_refuse)
    c.fenetre ()
```

**fin correction TD 30.1.1** □

## 30.2 Conserver ses fichiers sur une clé USB avec une interface graphique (2)

### 30.2.1 Enoncé

L'objectif de ce TD est de reprendre le programme du TD 30.1.1 et d'enregister les paramètres utilisés lors de la dernière sauvegarde afin de les retrouver lors de la prochaine utilisation. Comme pour le TD 30.1.1,

l'idée est de reprendre la classe `copie_usb_window`, de la dériver en une classe qui permet de lire et d'écrire les paramètres utilisés pour la copie de fichiers vers une clé USB.

**1)** La première tâche est d'écrire une méthode qui écrit dans un fichier texte les quatre paramètres de la classe qui sont les deux répertoires source et destination, les deux filtres utilisés pour sélectionner les fichiers à copier. On pourra s'inspirer du petit programme suivant.

```
# ouverture fichier essai.txt en mode écriture
f = open ("essai.txt", "w")
# écriture d'une chaîne de caractères dans le fichier
f.write ("première ligne")
# passage à la ligne
f.write ("\n")
# fermeture du fichier
f.close ()
```

**2)** La seconde tâche est d'écrire une méthode qui lit dans un fichier texte les quatre paramètres de la classe écrit par la méthode implémentée lors de la question précédente. On pourra aussi s'inspirer du petit programme suivant.

```
# ouverture fichier essai.txt en mode lecture
f = open ("essai.txt", "r")
# lecture de la première ligne du fichier
s = f.readline ()
# on enlève de la chaîne de caractères s le symbole de fin de ligne
s = s.replace ("\n", "")
# affichage de s
print s
# fermeture du fichier
f.close ()
```

**3)** On peut maintenant surcharger la méthode `fenetre` de la classe `copie_usb_window` définie au TD 30.1.1 de sorte qu'elle lise ses paramètres dans un fichier et qu'elle les écrive une fois la copie terminée.

### 30.2.2 Correction

Le programme qui suit répond aux deux questions de l'énoncé de ce TD. La classe `copie_usb_window_save` hérite de la classe `copie_usb_window` définie lors de la correction du TD 30.1.1 (page 30.1.2). Il ajoute deux méthodes, la première `ecrire_parametre` permet d'écrire les quatre paramètres de la classe `copie_usb` définie au TD 21.1.1 (correction page 459). La méthode `lire_parametre` permet de les relire. Enfin, la fonction `fenetre` de la classe `copie_usb_window` est surchargée dans la classe `copie_usb_window_save` d'appeler les deux méthodes précédentes avant et après l'affichage de l'interface graphique.

```
# -*- coding: cp1252 -*-
"""copie de fichiers sur une clé USB, interface fenêtrée,
enregistrement des paramètres précédents"""

import copie_usb_window   # version avec fenêtre                                    5
import os.path            # pour détecter l'existence d'un fichier
import re                 # pour les expressions régulières

class copie_usb_window_save (copie_usb_window.copie_usb_window):
    """recopie des fichiers sur une clé USB avec une fenêtre graphique,            10
    et enregistrement des précédents paramètres"""
```

```
    def ecrire_parametre (self, txt) :
        """écriture des paramètres dans le fichier txt"""
        # ouverture du fichier en mode écriture
        f = open (txt, "w")
        f.write (self.ch1)
        f.write ("\n")
        f.write (self.ch2)
        f.write ("\n")
        f.write (self.accept.pattern)
        f.write ("\n")
        f.write (self.refuse.pattern)
        f.write ("\n")
        f.close ()

    def lire_parametre (self, txt) :
        """relecture des paramètres écrits dans le fichier txt s'il existe"""
        if os.path.exists (txt) :
            f           = open (txt, "r")

            s           = f.readline ()
            s = s.replace ("\n", "")
            self.ch1    = s

            s           = f.readline ()
            s = s.replace ("\n", "")
            self.ch2    = s

            s           = f.readline ()
            s = s.replace ("\n", "")
            self.accept = re.compile (s)

            s           = f.readline ()
            s = s.replace ("\n", "")
            self.refuse = re.compile (s)

            f.close ()

    def fenetre (self) :
        """méthode fenêtre surchargée pour lire les derniers paramètres
        et enregistrer les nouveaux"""
        # première étape, lire les précédents paramètres
        self.lire_parametre ("copie_usb_window_save.txt")
        # seconde étape, appel de la méthode précédente
        copie_usb_window.copie_usb_window.fenetre (self)
        # troisième étape, écriture des paramètres
        self.ecrire_parametre ("copie_usb_window_save.txt")


if __name__ == "__main__" :

    print "copie de fichiers vers une clé USB"
    ch1             = "C:\\Documents and Settings\\Dupré\\" \
                        "Mes documents\\informatique\\support\\python_td"
    ch2             = "c:\\temp\\copie_usb"
    filtre_accept   = ".*[.].*"
    filtre_refuse   = ".*[.]pdf$|.*[.]html$|.*[.]bmp|programme\\\\.*[.]zip$"

    # filtre_accept accepte tout type de fichier
    # filtre_refuse refuse tous les fichiers dont l'extension est pdf, html ou
    # inclus dans le répertoire programme et ayant l'extension zip

    c = copie_usb_window_save (ch1, ch2, filtre_accept, filtre_refuse)
    c.fenetre ()
```

**fin correction TD 30.2.1** □

Neuvième partie

# BIBLIOGRAPHIE, INDEX, REMERCIEMENTS

# Bibliographie


[Martelli2004] Alex Martelli, *Python en concentré*, O'Reilly, édition française (2004)

[Swinnen2004] Gérard Swinnen, *Apprendre à programmer avec Python, voir à l'adresse http://www.cifen.ulg.ac.be/inforef/swi/python.htm*, O'Reilly, édition française (2004)

[Pilgrim2004] Mark Pilgrim, *Dive Into Python, voir à l'adresse http://diveintopython.org, titre français Plongez au coeur de Python*, APress (2004)

[Ziadé2006] Tériak Ziadé, *Programmation Python*, Eyrolles (2006)

[Dayley2007] Brad Dayley, *Python - L'essentiel du code et des commandes*, Campuss Press (2007)

[Brucher2008] Matthieu Brucher, *Les Fondamentaux du langage - La Programmation pour les scientifiques*, Editions ENI (2008)

[Bailly2008] Yves Bailly, *Initiation à la programmation avec Python et C++*, Editions Le programmeur (2008)

[Gouriéroux1983] Gouriéroux, Monfort, *Analyse des séries temporelles*, Economic (1983)

[Saporta2006] Gilbert Saporta, *Probabilités, analyse des données et statistique*, Technip (2006)

[Riva2005] Fabrice Riva, *Applications Financières sous Excel en Visual Basic*, Economica (2005)

[Berchtold1996] S. Berchtold, D. A. Keim, H. P. Kriegel, *The X-Tree : An index structure for high dimension data*, Proceedings of the 22nd Internation Conference on Very Large Databases, Bombay, India (1996)

[Beckmann1990] N. Beckmann, H. P. Kriegel, P. Schneider, B. Seeger, *The $R^*$-tree : an efficient and robust access method for points and rectangles*, Proceedings of SIGMOD conference, Atlantic City (1990), pp 322-331

[Bottou1991] L. Bottou, *Une approche théorique de l'apprentissage connexionniste, Application à la reconnaissance de la parole*, Thèse de l'Université de Paris Sud, Centre d'Orsay ( 1991)

[Bracewell1986] Ronald Bracewell, *The Hartley Transform*, Oxford University Press (1986)

[Bresenham1965] J. E. Bresenham, *Algorithm for computer control of a digital plotter*, IBM System Journal 4(1) (1965), pp 25-30

[Bresenham1977] J. E. Bresenham, *A linear algorithm for incremental digital display of circuler arcs*, CGIP 20(2) (1977)

[Broyden1967] C.G. Broyden, *Quasi-Newton methods and their application to function minimization*, Math. Comput 21 (1967), pp 368-

[Davidon1959] C. W. Davidon, *Variable metric method for minimization*, A.E.C. Research and Development Report, ANL-5990 (1959)

[Dempster1977] A. P. Dempster, N. M. Laird, D. B. Rubin, *Maximum-Likelihood from incomplete data via the EM algorithm*, Journal of Royal Statistical Society B 39 (1977), pp 1-38

[Driancourt1996] X. Driancourt, *Optimisation par descente de gradient stochastique de systèmes modulaires combinant réseaux de neurones et programmation dynamique, Application à la reconnaissance de la parole*, Thèse de l'Université de Paris Sud, Centre d'Orsay (1996)

[Duhamel1990] P. Duhamel, M. Vetterli, *Fast Fourier Transform : a tutorial review and a state of the art.*, Signal Processing 19(4) (1990), pp 259-299

[Faragó1993] A. Faragó, T. Linder, G. Lugosi, *Fast Nearest-Neighbor Search in Dissimilarity Spaces*, IEEE Transactions on Pattern Analysis and Machine Intelligence 15(9) (1993), pp 957-962

[Faure2000] Robert Faure, Bernard Lemaire, Christophe Picouleau, *Précis de recherche opérationnelle, 5$^i$me édition*, Dunod (2000)

[Fletcher1963] R. Fletcher, M. J. D. Powell, *A rapidly convergent descent method for minimization*, , Computer Journal (1963), pp 6-163 168

[Fletcher1993] R. Fletcher, *An overview of Unconstrained Optimization*, Numerical Analysis Report NA/149 (1993)

[Guttman1984] A. Guttman, *R-Trees : A Dynamic Index Structure for Spatial Searching*, Proceedings ACM SIGMOD (1984), pp 47-57

[Helsgaun2000] K. Helsgaun, *An effective implementation of the Lin-Kernighan traveling salesman heuristic*, European Journal of Operational Research 126 (2000), pp 106-130

[Koivisto1999] L. Koivisto, *Quinquennal report of Neural Networks Research Center and Laboratory of Computer Science and Information Science, Helsinki, University of Technology*, http ://www.cis.hut.fi/research/reports/quinquennial/ 126 (1999), pp 106-130

[Mallat2000] Stéphane Mallat, *Une exploration des signaux en ondelettes*, Editions de l'Ecole Polytechnique (2000)

[Math2000] auteurs divers, *Les mathématiciens*, Belin pour la science (2000)

[Moré1977] J. J. Moré, *The Levenberg-Marquardt algorithm : Implementation and theory*, Proceedings of the 1977 Dundee Conference on Numerical Analysis, G. A. Watson, ed., Lecture Notes in Mathematics, vol. 630, Springer-Verlag, Berlin (1977), pp 105-116

[Moreno2003] Francisco Moreno-Seco, Luisa Mico, Jose Oncina, *A modification of the LAESA algorithm for approximated k-NN classification*, Pattern Recognition Letters 24 (2003), pp 47-53

[Nussbaumer1982] H. J. Nussbaumer, *Fast Fourier Transform and Convolutions Algorithms*, Springer-Verlag, Berlin (1982)

[Rico-Juan2003] J. R. Rico-Juan, L. Mico, *Comparison of AESA and LAESA search algorithms using string and tree-edit-distances*, Pattern Recognition Letters 24 (2003), pp 1417-1426

[Robert1996] Christian Robert, *Méthodes de Monte-Carlo par chaînes de Markov*, Economica (1996)

[Sellis1987] T. Sellis, N. Roussopoulos, C. Faloutos, *The R+tree - a Dynamic Index for Multi-Dimensional Objects*, Proceedings of the 13th VLDB conference (1987), pp 507-518

[Vesanto2000] Juha Vesanto, Esa Alhoniemi, *Clustering of the Self-Organizing Map*, IEEE Transactions on Neural Networks 11(3) (2000), pp 586-600

[Jacquillat2008] Bertrand Jacquillat, *Hedge funds, private equity, marchés financiers les frères ennemis ?*, PUF (2008)

[Henry2008] Gérard-Marie Henry, *Hedge funds*, Eyrolles (2008)

[Hull2007] John Hull, *Options futures et autres actifs dérivés*, Pearson Education (2007)

[Potters2005] Marc Potters, Jean-Philippe Bouchaud, *Trend followers lose more often than they gain*, Référencé depuis le site http://econpapers.repec.org/, accessible à l'adresse http://www.cfm.fr/papers/0508104.pdf (2005)

[RocchiChristiaens2007] Jean-Michel Rocchi, Arnaud Christiaens, *Hedge Funds, tome 1, Histoire de la gestion alternative et de ses techniques*, Séfi Editions (2007)

[Wilmott2008] Paul Willmott, *La finance quantitative en 50 questions*, Eyrolles, édition française (2008)

# Remerciements

De nombreuses personnes m'ont aidé à réaliser ce livre dont la rédaction s'est étalée sur plusieurs années. La décision de choisir le langage *Python* à l'ENSAE a été prise grâce à une initiative de David Sibaï qui m'a aussi apporté ses conseils lors de la conception des examens à l'ENSAE. Emmanuel Guérin m'a largement aidé à comprendre les subitilités du langage *Python* notamment son interfaçage avec le langage *C++*. Christopher Kermorvant m'a souvent fait découvrir des utilisations intéressantes du langage grâce à sa curiosité [1]. Je remercie également Jérémie Jakubowicz, un autre curieux, qui enseigne également. Je remercie Georges Pavlov et Emmanuel Héry qui ont toujours très bien organisé mes enseignements à l'ENSAE. Je remercie aussi Clémence Dupré dont l'aide a été précieuse lors de la rédaction. Il ne faut pas oublier tous les élèves de l'ENSAE qui m'ont eu comme professeur dont les questions nombreuses et pertinentes m'ont aidé à concevoir les exercices que j'ai insérés dans ce livre. Je remercie enfin l'ENSAE, école dont je suis sorti et qui m'a permis de revenir pour enseigner. Sans elle, il n'y aurait rien eu de tout cela.

1. Curiosité dont un synonyme est *veille technologique*.

# Index


`False` . . . 32
`None` . . . 30, 56
`pythonw` . . . 444
`True` . . . 32
`__builtins__` . . . 49
`__class__` . . . 117
`__debugbreak` . . . 150
`__doc__` . . . 49
`__file__` . . . 49
`__init__.py` . . . 137, 149
`__main__` . . . 135
`__name__` . . . 49
symboles *Python*
    ''' . . . 33
    ' . . . 33
    () . . . 37
    ** . . . 31
    *= . . . 31
    * . . . 31, 37, 73
    += . . . 31
    + . . . 31, 34, 37
    , . . . 44, 50
    -= . . . 31
    - . . . 31
    . . . 39, 84–86
    /= . . . 31
    / . . . 31
    : . . . 34, 39, 44, 53, 57, 58, 83, 84, 124, 125
    << . . . 31
    <= . . . 32, 54
    < . . . 32, 54
    == . . . 32, 54
    = . . . 50, 101, 102
    >= . . . 32, 54
    >> . . . 31
    > . . . 32, 54
    [] . . . 34, 37, 39, 45, 93
    % . . . 31
    & . . . 31
     . . . 57, 64
    _ . . . 92
    {} . . . 44
    \ . . . 30, 33, 76

**A**

accélération . . . 712
accent . . . 21, 173
Ackermann . . . 310
ACP . . . 528
acquisition . . . 480
ADN . . . 9
affectation . . . 104
    multiple . . . 50, 60
aide . . . 67, 89
    HTML . . . 90
Algorithme
    1-PPV ou plus proche voisin . . . 557
    algorithme BFGS . . . 581
    algorithme de Kruskal . . . 532
    algorithme du plus court chemin (Bellman) 519
    algorithme du plus court chemin (Ford) . 520
    carte de Kohonen . . . 530
    construction d'un circuit eulérien . . . 534
    construction d'un circuit hamiltonien . . . 535
    descente de gradient de Newton . . . 578
    fusion de deux listes triées . . . 389
    insertion d'un objet dans un R-tree . . . 561
    k-PPV ou k-plus proches voisins . . . 558
    LAESA . . . 562
    LAESA : sélection des pivots . . . 562
    lancer de rayon . . . 731
    mouvement d'une corde . . . 713
    optimisation stochastique . . . 583
    plus court chemin à partir d'un arbre de poids minimum . . . 531
    plus proche voisin d'après [Faragó1993] . . 564
    portrait . . . 758
    recherche dans un R-tree . . . 560
    recherche dichotomique . . . 391, 714
    simplification du circuit hamiltonien . . . . 536
    simulation d'une variable selon un mélange de deux lois normales . . . 593
    tracé d'une ellipse (Bresenham) . . . . 703, 753
    tracé d'une ligne (Bresenham) . . . . . 702, 750
    tri d'entiers . . . 394
    tri par fusion . . . 390
    tri par insertion dichotomique . . . 391
    tri par sélection . . . 389
algorithme . . . 11
    coût . . . 389–391, 520, 719
    quicksort . . . 391
    tri rapide . . . 391

aliasing ..... 720
ampoule ..... 692
analyse en composantes principales (ACP) .. 227
ancêtre ..... 112
antialiasing ..... 732
appelable ..... 74
apprentissage
    second ordre ..... 580
arborescence ..... 158, 457, 792, 800
arbre ..... 485
arbre de poids minimum ..... 532, 538
arc ..... 234
architecture ..... 150
arrondi ..... 31, 279, 280, 289
ascii ..... 173
asynchrone ..... 202
attente ..... 602
attribut
    __bases__ ..... 89, 112
    __class__ ..... 89
    __dict__ ..... 88, 89, 107, 113
    __doc__ ..... 89
    __module__ ..... 89
    __name__ ..... 89
    __slots__ ..... 108
    statique ..... 97, 108
attributs croissants ..... 114

**B**

B+ tree ..... 558
backup ..... 342
balise ..... 462
barre d'outils ..... 402
barycentre ..... 733
base de données ..... 367
    relationnelle ..... 368
base de données relationnelle ..... 378
Bellman ..... 520
BFGS ..... 581
boîte
    englobante ..... 559
    fenêtre ..... 559
    objet ..... 559
*Boost Python*
    boost :: python :: bases ..... 147
    boost :: python :: class_ ..... 147
    boost :: python :: def ..... 146
    boost :: python :: init ..... 147
    BOOST_PYTHON_MODULE ..... 146
    def_readwrite ..... 148
    def ..... 148
    return_internal_reference ..... 149
    __debugbreak ..... 150
boucle ..... 41, 57, 410
    break ..... 61
    continue ..... 61
    fin normale d'une boucle ..... 62
    infinie ..... 57
    itération suivante ..... 61
    sortie prématurée ..... 61
bouton ..... 402, 793
Branch and bound ..... 561
Bresenham ..... 702, 703, 750, 753, 758
brique ..... 482
Broch, Henri ..... 692
byte ..... 161
bytecode ..... 13, 134

**C**

*C, C++* ..... 140
calcul
    distribué ..... 202
    parallèle ..... 202
    précision ..... 268, 292, 307
call stack ..... 122, 276
carré magique ..... 228
carte de Kohonen ..... 527
cartes de Kohonen ..... 528
Cauchy ..... 580
CD ..... 719
centre d'une sphère ..... 727
CGI ..... 358, 362
    CGI ..... 363
    html ..... 363
chaîne de caractères ..... 33
    concaténation ..... 34, 42
    conversion ..... 34, 48
    encodage ..... 173
    formatage ..... 35
    manipulation ..... 34
    préfixe r ..... 33
    préfixe u ..... 173
chaînes de Markov ..... 521
chambre noire ..... 725
Charpak, Georges ..... 692
chemin relatif, absolu ..... 135, 159, 160
chiffres manuscrits ..... 564
circuit
    déplacement ..... 536, 537, 543
    eulérien ..... 531, 532
    hamiltonien ..... 531, 534
    retournement ..... 536, 537, 543
classe ..... 83
    datetime ..... 173
    dict ..... 44
    float ..... 31

int . . . 31
list . . . 38
object . . . 112
set . . . 50
str . . . 33, 174
Thread . . . 202
timedelta . . . 173
tuple . . . 36
unicode . . . 174
ZipFile . . . 157
ancêtre . . . 112, 115
attribut . . . 83, 86, 113, 407
attribut implicite . . . 89
classe incluse . . . 91
close . . . 801
commentaire . . . 89
compilation . . . 117
constructeur . . . 87, 102
copie . . . 101, 102, 105
copie profonde . . . 105
dériver . . . 110
destructeur . . . 88, 102
file . . . 801
FileSelection . . . 795
héritage . . . 108, 267
héritage multiple . . . 115
HTMLParser . . . 461, 463, 466
instance . . . 84
itérateur . . . 94
liste des attributs . . . 88
méthode . . . 83, 84, 407
opérateur . . . 91, 92
propriété . . . 99
sous-classe . . . 91
surcharge . . . 93, 111, 239
classes . . . 13
classification . . . 557
clé . . . 44
collage de séquences . . . 42
coloriage des cartes . . . 521
commande . . . 334, 336
commentaire . . . 30, 67, 89, 404
compact disque . . . 719
compilation . . . 117
complexité . . . 484
composante connexe . . . 531, 729
compression . . . 719, 720
condition d'arrêt . . . 578
confusion
chaîne = tableau . . . 326
guillemet . . . 326
initialisation d'une boucle . . . 327
range . . . 326
conique . . . 707
connexité
4 . . . 701, 750
8 . . . 702, 703, 707, 750, 753
consanguinité . . . 699
constructeur . . . 87, 408
conteneur . . . 38
contrainte . . . 537
contrôle . . . 402
contrôle (graphique) . . . 176
convergence . . . 538, 580
vitesse . . . 580
conversion . . . 31, 34, 48, 259
Cooley . . . 721
copie . . . 104, 105, 457, 792, 800
copie profonde . . . 105
correction
TD * . . . 471
TD 10 . . . 801
TD 12 . . . 463
TD 2 . . . 685
TD 5 . . . 753
TD 6 . . . 760
TD 7 . . . 789
TD 8 . . . 459
TD 9 . . . 799
cosinus hyperbolique . . . 711, 712
couleur . . . 725
courbe de niveau . . . 706
Courrier International . . . 699
coût . . . 388, 531
algorithme . . . 520
polynômial . . . 521
coût d'un algorithme . . . 222, 265, 266
crible d'Eratosthène . . . 61
critère
Cauchy . . . 580
CSV . . . 156


**D**


dames . . . 217
database . . . 372
datamining . . . 24
date de naissance . . . 173
dates . . . 173
debug . . . 150, 266
débugger . . . 19, 150
Définition
algorithme . . . 11
arbre de poids minimum . . . 532
attribut . . . 86
attribut statique . . . 97
B+ tree . . . 558

boucle ... 57
chaîne de caractères ... 33
circuit eulérien ... 531
circuit hamiltonien ... 531
classe ... 83
compilateur et compilation ... 12
composante connexe ... 531
constante ... 29
coût d'un algorithme ... 520
cycle dans un graphe ... 520
dictionnaire ... 44
exception ... 122
fonction ... 64
fonction récursive ... 69
héritage ... 111
instance ... 84
instruction ... 12
langage interprété ... 12
liste ... 38
méthode ... 84
méthode statique ... 96
mot-clé ... 12
point d'entrée du programme ... 133
portée d'une variable ... 70
problème NP-complet ... 528
programme ... 12
rayon ... 727
surcharge ... 111
T-uple ... 36
test ... 53
type immuable (ou immutable) ... 30
type modifiable (ou mutable) ... 38
variable ... 29
variable globale ... 71
variable locale ... 70
degré de liberté ... 537
densité ... 563, 685
exponentielle ... 602
Poisson ... 600
déplacement ... 536, 537
deprecated ... 52
dériver ... 110
descente
gradient ... 577
Design Patterns ... 120
destructeur ... 408
Devenez sorciers, devenez savants ... 692
DFP ... 582
diagonalisable ... 447
dichotomie ... 78, 242
dictionnaire
clé ... 44
clé modifiable ... 47
copie ... 46
exemple ... 45
valeur ... 44
diffraction ... 721
Dijstra ... 520
discriminant ... 728
distance
édition ... 525
Kullback-Leiber ... 587
Levenstein ... 525
distribution
queue ... 586
diviser pour régner ... 389
division entière ... 32
droite
tracé ... 701, 750
Dynamic Link Library (DLL) ... 145

**E**

écart-type ... 584
échantillon ... 585
éditeur
*Boa Constructor* ... 20
*DrPython* ... 20
*Eclipse* ... 19
*Emacs* ... 20
*Python Scripter* ... 19
*SciTe* ... 17
*TextWrangler* ... 20
*The Eric Python IDE* ... 20
efficacité ... 218
Einstein ... 494
ellipse ... 758
EM ... 589
emv ... 584
encodage ... 173
énénement rare ... 586
énigme ... 494
énoncé
apprentissage ... 214, 217, 220, 228, 234
écrit ... 259, 271, 279, 289, 295, 305
équation différentielle ... 643
pratique ... 242, 244, 247
ENSAE ... 8, 259
équation d'une shère ... 727
équation différentielle ... 711
Eratosthène ... 61
espace
métrique ... 558
vectoriel ... 558
espérance ... 584
estimateur du maximum de vraisemblance ... 584, 588

événement ..... 175
brut ..... 189
Excel ..... 471, 684, 685
exception ..... 40, 122, 123, 150, 221, 316
ArithmeticError ..... 126
AttributeError ..... 126
Exception ..... 126
FloatingPointError ..... 126
ImportError ..... 126
IndentationError ..... 126
IndexError ..... 121, 126, 264, 291
IOError ..... 126
KeyError ..... 126
NameError ..... 126
OverflowError ..... 126
StopIteration ..... 126, 130
TypeError ..... 126
UnicodeError ..... 126
ValueError ..... 126
ZeroDivisionError ..... 126
héritage ..... 127
imbrication ..... 128
message ..... 126, 127
piège ..... 131
exercice
arrondi ..... 279, 289
boucles ..... 279
calcul ..... 260, 272
classe ..... 276, 277
comparaison ..... 290
comptage ..... 273, 283, 293, 306
copie ..... 263, 277
coût d'un algorithme ..... 265
dictionnaire ..... 262, 273, 293
division ..... 290
écriture en base 3 ..... 276
Fibonacci ..... 261, 280, 289
grenouille ..... 261
héritage ..... 267, 284
liste ..... 281
logarithme ..... 271, 307
matrice ..... 268
moyenne ..... 265, 277
parcours ..... 271, 279, 291, 306
précision des calculs ..... 268, 292, 307
probabilité ..... 274, 287
somme de chiffres ..... 259
suite récurrente ..... 260, 261, 280, 289
suppression dans une boucle ..... 264
tri ..... 259, 265
tri d'entiers ..... 285
tri, position ..... 305
valeurs propres ..... 268
variance ..... 265, 277
Expectation-Maximisation ..... 589
expression régulière ..... 458, 461, 463
expressions régulières ..... 167
assemblage ..... 169
caractères spéciaux ..... 168
ensemble de caractères ..... 168
exemple ..... 169
groupe ..... 167, 169
multiplicateurs ..... 168
extension
bat ..... 335
chm ..... 345
hhc ..... 345
hhk ..... 345
hhp ..... 345
iss ..... 349
extension de fichier
bat ..... 165
chm ..... 26
dot ..... 240
html ..... 239
PDF ..... 23
pdf ..... 240
png ..... 240
py ..... 20, 134
pyc ..... 134
rar ..... 157
tex ..... 240
zip ..... 157
extra-caractère ..... 34
\ ..... 34
\' ..... 34
\n ..... 34
\r ..... 34
\t ..... 34
\% ..... 34
\\ ..... 34

**F**

facette ..... 728, 733, 745, 775
factorielle ..... 393, 484
FAQ ..... 18
Fast Fourier Transform ..... 719
FastCGI ..... 358
fenêtre ..... 792, 800
fenêtre graphique ..... 792
Fermat ..... 518
FFT ..... 719
Fibonacci ..... 280, 281, 700
fichier ..... 153, 800
print ..... 154
arborescence ..... 158

binaire . . . 161
buffer . . . 155
chemin absolu . . . 159
copie . . . 159, 457, 792, 800
écriture . . . 153
écriture, ajout . . . 155
fermeture . . . 153, 155
fin de ligne . . . 156, 322
HTML . . . 322
lecture . . . 155
logs . . . 155
ouverture . . . 153, 155
pdf . . . 378
rar . . . 157
sérialisation . . . 163
suppression . . . 159
tampon . . . 155
trace . . . 155
XML . . . 322
zip . . . 157
fil d'exécution . . . 202
file d'attente . . . 692
filtrage . . . 720
filtre . . . 458
passe-bas . . . 720
passe-haut . . . 720
finance . . . 22, 607
action . . . 608
analyse technique . . . 614
backtest . . . 620
bandes de Bollinger . . . 611
bid-offer spread . . . 609
Brent Crude Oil . . . 613
broker . . . 607
calcul stochastique . . . 613
call . . . 613
capitalisation . . . 608
carnet d'ordres . . . 609
Close . . . 628
commodity . . . 612
Contrarian . . . 615
corrélation . . . 610
coût de transaction . . . 622
daily . . . 609
deal . . . 625
devises . . . 608
drawdown . . . 623
FED . . . 611
Future . . . 608, 612
gestion alternative . . . 611
gestion de portefeuille . . . 618
graphe . . . 628
growth . . . 617
hedge . . . 613
Hedge Funds . . . 607, 611
High . . . 628
indicateur
combinaison . . . 618
DPR . . . 617
EBIDTA . . . 617
EPS . . . 617
P/E . . . 617
P/S . . . 617
Information Ratio . . . 623
intraday . . . 608
large cap . . . 608
levier . . . 611
liquidité . . . 608
long . . . 614
Low . . . 628
MACD . . . 612
Markowitz . . . 618
maturité . . . 608, 612
Mean Reversing . . . 615
mid cap . . . 608
moving average . . . 611, 612
moyenne mobile . . . 611
moyenne mobile exponentielle . . . 612
NAV . . . 621
New-York Merchantile Exchange . . . 613
NYMEX . . . 613
OHLC . . . 628
Open . . . 628
option . . . 613
pair trading . . . 615
portefeuille . . . 618
put . . . 613
ratio de Sharpe . . . 623
rebalancing . . . 618
rendement . . . 609
rendement roulant . . . 623
roll . . . 613
RSI . . . 612
short . . . 614
slippage . . . 622
small cap . . . 608
sous-jacent . . . 612
stratégie
Commodity Trading Advisor . . . 612
CTA . . . 612
Emerging Markets . . . 613
Equity Market Neutral . . . 612, 619
Event Driven . . . 612
Fixed Income Arbitrage . . . 613
Global Macro . . . 613
Long / Short Equity . . . 612

stratégie automatique . . . 607
style . . . 617
subprime . . . 611
taux d'intérêt . . . 608
tick . . . 622
tick by tick . . . 609
track record . . . 613
trend following . . . 614
True Range . . . 612
value . . . 617
VAR . . . 623
volatilité . . . 610
Finlande . . . 699
firewall . . . 341
flot maximal . . . 521
fonction . . . 64
abspath . . . 159
abs . . . 77
callable . . . 74, 77
chdir . . . 160
chr . . . 77
close . . . 163
cmp . . . 77, 234, 235, 291
commonprefix . . . 159
compile . . . 75, 117, 126, 167, 169, 170
complex . . . 77
copy . . . 43, 47, 101, 103, 160, 278, 283
cos . . . 138
decode . . . 174
deepcopy . . . 43, 105
dirname . . . 159
dir . . . 48, 77, 90
divmod . . . 50
dump . . . 163
encode . . . 174
eval . . . 48, 49, 74, 75
exec . . . 117
exists . . . 159
exp . . . 138
fflush . . . 155
getatime . . . 159
getctime . . . 159
getcwd . . . 160
getmtime . . . 159
getsize . . . 159
help . . . 67, 77, 89, 90, 112, 115
hex . . . 48
id . . . 47, 77, 235, 237, 239
isabs . . . 159
isdir . . . 159
isfile . . . 159
isinstance . . . 77, 84, 117, 124
iskeyword . . . 52
issubclass . . . 77, 112, 116
iter . . . 94
join . . . 159
len . . . 34, 37, 39, 45, 77
listdir . . . 159
load_dynamic . . . 149
load . . . 163
log . . . 138
makedirs . . . 160
map . . . 76, 79
max . . . 37, 39, 45, 77, 78
min . . . 37, 39, 45, 77, 78
mkdir . . . 160
open . . . 154, 163, 174
ord . . . 77
pack . . . 162
Popen . . . 336
print . . . 93
range . . . 41, 58, 77, 232
raw_input . . . 55, 150
reload . . . 77, 134
remove . . . 160
rename . . . 160
replace . . . 214
repr . . . 48, 77
rmdir . . . 160
round . . . 77
sin . . . 138
sleep . . . 204
splitext . . . 159
split . . . 136, 159, 214, 216
str . . . 34, 48, 174
sum . . . 39, 77, 79
system . . . 166, 336, 343
type . . . 31, 49, 77
unicode . . . 174
unpack . . . 162
walk . . . 160
xrange . . . 41, 58, 77, 232
zip . . . 42, 60, 76, 77
__import__ . . . 135, 136
Ackermann . . . 310
aide . . . 67
compile . . . 458, 463
copy . . . 459
cos . . . 387
exists . . . 459
fft . . . 721
gauss . . . 684
générateurs . . . 73
imbriquée . . . 71
InputBox . . . 408
isdir . . . 459

isfile . . . 459
join . . . 462
listdir . . . 459
locale . . . 71
log . . . 698
makedirs . . . 459
match . . . 458, 463
mkdir . . . 459
MsgBox . . . 408
portée . . . 71
randint . . . 685
random . . . 698
récursive . . . 69, 160
round . . . 684
spéciale . . . 121
sqrt . . . 685
surcharge . . . 67
urlopen . . . 462
usuelle . . . 76
xrange . . . 387, 750
fonction de répartition . . . 286, 698
fonction quadratique . . . 584
fond d'écran . . . 471
Ford . . . 520
Ford-Fulkerson . . . 521
format
binaire . . . 161
CSV . . . 156
texte . . . 153
zip . . . 157
formulaire . . . 408
formule
Stirling . . . 393
forum de discussion . . . 175
Fourier . . . 446, 719

**G**

garbage collector . . . 104, 131, 140, 141, 144, 148
gêne . . . 699
générateurs . . . 73
gestion
centralisée . . . 342
décentralisée . . . 342
Gibbs . . . 721
girafe . . . 479
*Google Code* . . . 13
Gouraud . . . 733
GPL . . . 13
gradient . . . 577, 706
conjugué . . . 580
descente . . . 577, 578
stochastique . . . 582
gradient projeté . . . 619
graphe . . . 234, 380, 485
Bellman . . . 520
chemin, poids . . . 520
circuit eulérien . . . 531, 532
circuit hamiltonien . . . 531, 534
composante connexe . . . 531
cycle . . . 520
Dijkstra . . . 520
flot . . . 521
Ford . . . 520
matrice d'adjacence . . . 514
non-orienté . . . 513
ordonnancement . . . 521
orienté . . . 513
plus court chemin . . . 513, 518
poids minimum . . . 532, 538
successeur . . . 514
théorie . . . 517
graphe XY . . . 684
graphisme . . . 380
gravitation . . . 710

**H**

Hartley . . . 721
héritage . . . 108
multiple . . . 115
sens . . . 114
histogramme . . . 286, 585, 604, 684
HTML . . . 23, 239, 346, 363, 461, 463
< /UL > . . . 346
< a > . . . 364
< b > . . . 364
< br > . . . 364
< del > . . . 364
< h1 > . . . 364
< h2 > . . . 364
< hr > . . . 364
< i > . . . 364
< img > . . . 364
< ins > . . . 364
< p > . . . 364
< table > . . . 364
< td > . . . 364
< tr > . . . 364
< UL > . . . 346
border = . . . 364
href = . . . 364
name = . . . 364
src = . . . 364
attribut . . . 364
balise . . . 364
formulaire . . . 366
lecture . . . 322

syntaxe . . . 364
HTMLParser . . . 466
hyperlien . . . 461, 462, 464

**I**

identificateur . . . 29, 69
illumination . . . 726, 733
image
gif . . . 177
png . . . 240
image de synthèse . . . 724, 775
immuable . . . 30
indentation . . . 13, 20, 53, 57, 58, 76, 126
indépendance . . . 585
inégalité de Jensen . . . 588
instance
création . . . 84
suppression . . . 101
intégrale . . . 387
intensité lumineuse . . . 789
interface graphique . . . 175, 800, 801
Internet
recherche . . . 461
interpolation . . . 733
interprété . . . 13
intersection . . . 50, 727
itérateur . . . 59, 94, 130

**J**

Java . . . 140
Jensen . . . 588
jeu de caractères . . . 173
accents . . . 21
ascii . . . 173
cp1252 . . . 21, 173
latin-1 . . . 21, 173
unicode . . . 173
utf-8 . . . 21, 173, 174
jeu de dames . . . 217

**K**

k-ppv . . . 557
Kelvin . . . 721
Kohonen . . . 527
kPPV . . . 557
Kruskal . . . 532
Kullback-Leiber . . . 587

**L**

LAESA . . . 561, 562
Lagrange, multiplicateurs . . . 619
lancer de rayon . . . 724, 725, 730, 775
langage
haut niveau . . . 13
indentation . . . 13
interprété . . . 13
modulaire . . . 13
orienté objet . . . 13
typage dynamique . . . 13
langage de programmation
*C++* . . . 141
*C, C++* . . . 13, 140
*Java* . . . 13, 140
*Ruby* . . . 356
*R* . . . 24
*SQL* . . . 367
*VBA* . . . 150
Latex . . . 23, 235, 240, 471
Léonard de Pise . . . 700
Levenstein . . . 525
lien hypeertexte . . . 462, 464
ligne . . . 758
ligne de commande . . . 165, 334, 336
ligne trop longue . . . 404
likelihood . . . 585
liste
chaînée . . . 38
copie . . . 42
insertion . . . 38
suppression . . . 38
tri . . . 38
valeur par défaut . . . 294
liste de listes . . . 81
log-vraisemblance . . . 585, 594
logiciel
*Excel* . . . 9
*R* . . . 449
logs . . . 150, 155, 334
loi
Bernouilli . . . 683
binomiale . . . 683
exponentielle . . . 602, 692
gamma . . . 602
mélange . . . 602
normale . . . 684
Poisson . . . 599, 600, 692, 697
lumière ambiente . . . 734

**M**

machine learning . . . 24
macro . . . 399
mail . . . 158
envoi automatique . . . 158
Markov . . . 521
matrice . . . 79, 81, 115, 268
creuse . . . 220
écriture . . . 154

lecture . . . 156
représentation . . . 217
Strassen . . . 119
maximum . . . 78
mesure . . . 563
métaprogrammation . . . 144
méthode
append . . . 40, 41
clear . . . 45
close . . . 157
copy . . . 45
count . . . 35, 40
extend . . . 40
findall . . . 167, 170
find . . . 35
get_nowait . . . 210
get . . . 45, 210
groups . . . 170
group . . . 170
has_key . . . 45
index . . . 40, 77
insert . . . 40
isalpha . . . 35
isdigit . . . 35
isSet . . . 205
items . . . 45
iteritems . . . 45, 46
iterkeys . . . 46
itervalues . . . 45, 46
join . . . 35, 42
keys . . . 45
lower . . . 35
match . . . 169, 170
next . . . 94
pack . . . 162
popitem . . . 45
pop . . . 40
put_nowait . . . 210
put . . . 210
qsize . . . 210
read . . . 157, 162
remove . . . 40
replace . . . 35
reverse . . . 35, 40
run . . . 202
search . . . 169, 170
setdefault . . . 45
sort . . . 40
span . . . 170
split . . . 35, 170
start . . . 202
strip . . . 35
sub . . . 169, 170
unpack . . . 162
update . . . 45
upper . . . 35
values . . . 45
wait . . . 205
write . . . 157, 162
ajout . . . 98
commentaire . . . 89
config . . . 793
constructeur . . . 87
delete . . . 795
destructeur . . . 88
get . . . 795
grid . . . 793
insert . . . 795
mainloop . . . 793
non-paramétrique . . . 586
noyaux . . . 586
paramétrique . . . 586
second ordre . . . 580
sort . . . 388
statique . . . 96
surcharge . . . 111
write . . . 801
méthode des potentiels . . . 521
méthode des rectangles . . . 387
*Microsoft* . . . 375
minimum . . . 78
mise en production . . . 332
MNIST . . . 567
modèle
illumination . . . 726, 733, 776
modifiable . . . 30, 37
modulation de fréquence . . . 720
module . . . 133, 404
import . . . 133
__init__.py . . . 137, 149
arborescence . . . 137
biggles . . . 689, 721
bresenham ligne . . . 543
chemin relatif . . . 135
classe . . . 407
copy . . . 538, 544, 721
emplacement . . . 135
externe . . . 139
FFT . . . 721
file_selection . . . 795
gplt . . . 688
HTMLParser . . . 461, 463, 466
IaGraph . . . 687
identificateur . . . 134
math . 387, 515, 538, 544, 594, 685, 698, 714, 721

Matrix . . . 515
nom . . . 133, 135
numarray . . . 721
Numeric . . . 515, 590, 687, 721
os . . . 459
os.path . . . 459
paquetage . . . 137
PIL . . . 515
PIL.Image . . . 515
PIL.ImageDraw . . . 515
PIL.random . . . 515
psyco . . . 544, 590, 597, 759
pygame . . . 538, 544, 714, 721, 750, 759
random . . . 538, 544, 594, 684, 685, 698
re . . . 458, 462, 463
répertoire . . . 135
ScientificPython . . . 687
SciPy . . . 688
scipy . . . 388, 721
shutil . . . 459
string . . . 721
Tkinter . . . 792
urllib . . . 462, 463
urlparse . . . 462, 463
UserArray . . . 515

module externe
biggles . . . 23
BioPython . . . 9
ChartDirector . . . 23
cx − oracle . . . 377
doxypy . . . 349
elixir . . . 24
epydoc . . . 349
GMPy . . . 271
HalPython . . . 620
matplotlib . . . 22, 224, 241, 381, 628
mdp . . . 24, 227
mutagen . . . 158, 169
MySQLdb . . . 373, 376
mysqldb . . . 27
numpy . . . 23, 446, 449
orange . . . 24
pdftools . . . 378
PIL . . . 24
Pmw . . . 195
psyco . . . 25
pydot . . . 23, 234
PyGame . . . 25
pygoogledesktop . . . 140
pyGTK . . . 24
pymedia . . . 158
pyparsing . . . 23, 234
pypdf . . . 153
pyQt . . . 24
pyr . . . 449
pysvn . . . 342
pytesser . . . 380
pywin32 . . . 373
reportLab . . . 378
rpyc . . . 27
Rpy . . . 24
rpy . . . 451
scipy . . . 9, 23, 443, 446
SQLAlchemy . . . 24
SQLObject . . . 24, 378
win32com.client . . . 443
win32com . . . 153, 377
wxPython . . . 24, 175

module interne
calendar . . . 138, 173
cgi . . . 138
cmath . . . 138
codecs . . . 138, 174
copy 43, 47, 101–103, 105, 107, 138, 263, 278, 283, 295
csv . . . 138
datetime . . . 138, 173
doctest . . . 330
gc . . . 138
getopt . . . 138, 166, 335
glob . . . 138, 159, 165, 334
hashlib . . . 138
htmllib . . . 138
HTMLParser . . . 156
imp . . . 149
keyword . . . 52
logging . . . 334
math . . . 138
odbc . . . 373
os.path . . . 136, 138, 159
os . . . 138, 159, 160, 166, 336, 343
pickle . . . 138, 163
profile . . . 138, 328
pstats . . . 328
pydoc . . . 342, 343
Queue . . . 209
random . . . 138, 274
re . . . 138, 167, 216
shutil . . . 138, 159, 160, 165, 334
sqlite3 . . . 138, 368
string . . . 138, 214
struct . . . 161, 162
subprocess . . . 336
sys . . . 136, 138, 165, 335
threading . . . 138, 202, 206
time . . . 138, 204

Tix ..... 182, 195, 196
Tkinter ..... 24, 138, 175
traceback ..... 363
unittest ..... 138, 331
urllib ..... 138, 139, 221
xml.dom ..... 138
xml.sax ..... 138, 156
zipfile ..... 138, 157
montant numérique, littéral ..... 214
Monte Carlo ..... 139
mot-clé ..... 407, 461
and ..... 32, 54
assert ..... 332
as ..... 134
break ..... 61
classmethod ..... 98
class ..... 83
continue ..... 61
def ..... 64, 65, 72, 84, 85
del ..... 39, 45, 101, 102, 265
elif ..... 53
else ..... 53, 62, 123–125
except ..... 123–125
exec ..... 75
for ..... 41, 42, 46, 57, 58, 73, 94
from ..... 134
global ..... 71
if ..... 42, 53
import ..... 133, 134
in ..... 32, 37, 39, 41, 45, 54, 57
is ..... 32, 43, 54
lambda ..... 73
not ..... 32, 37, 39, 45, 54
or ..... 32, 54
pass ..... 56, 84
print ..... 21, 48, 93, 154, 229
property ..... 99
raise ..... 126, 127
return ..... 64, 229
self ..... 84, 86
staticmethod ..... 96
try ..... 123–125
while ..... 57
yield ..... 73, 95
And ..... 405
As ..... 404
Case ..... 409
Dim ..... 404
Do ..... 410
Else ..... 409
ElseIf ..... 409
Error ..... 412
Exit ..... 410
For ..... 410
Function ..... 411
Goto ..... 412
If ..... 409
Loop ..... 410
Mod ..... 405
Next ..... 410
On ..... 412
Or ..... 405
Select ..... 409
Set ..... 412
Static ..... 411
Sub ..... 411
Then ..... 409
Type ..... 406
Wend ..... 410
While ..... 410
With ..... 406
moyenne ..... 584
MP3 ..... 169, 719
multicœurs ..... 202

**N**

nœud ..... 234
nearest neighbors ..... 557
Newton ..... 307, 577
nombre
entier ..... 31
premier ..... 61
réel ..... 31
normale ..... 706
normale d'une shpère ..... 728
noyau ..... 586

**O**

Object Database Connectivity ..... 373
objet ..... 13, 83, 176
Button ..... 792
Entry ..... 792, 795
Label ..... 792
Tk ..... 793
obsolète ..... 52
OCR ..... 380
octet ..... 161
ODBC ..... 373
login ..... 375
mot de passe ..... 375
password ..... 375
ondelette ..... 446
open source ..... 342
opérateur ..... 405
__abs__ ..... 121
__add__ ..... 92, 121
__cmp__ ..... 121

__complex__ ..... 121
__contains__ ..... 121
__copy__ ..... 103
__deepcopy__ ..... 105
__delitem__ ..... 121
__del__ ..... 88, 102
__div__ ..... 121
__float__ ..... 121
__getitem__ ..... 121
__getstate__ ..... 164
__iadd__ ..... 92, 121
__idiv__ ..... 121
__imul__ ..... 121
__init__ ..... 87
__int__ ..... 121
__ipow__ ..... 121
__isub__ ..... 121
__iter__ ..... 94
__len__ ..... 121
__mul__ ..... 121
__name__ ..... 135
__new__ ..... 102
__pow__ ..... 121
__setitem__ ..... 121
__setstate__ ..... 164
__slots__ ..... 38
__str__ ..... 93, 121, 129
__sub__ ..... 121
comparaison ..... 32
priorité ..... 31, 54
opérations élémentaires ..... 51
optimisation ..... 577, 599
avec contrainte ..... 528, 531
BFGS ..... 581
combinatoire ..... 528
DFP ..... 582
Newton ..... 577
stochastique ..... 583
valeurs initiale ..... 593
ordinateur ..... 11
ordonnancement ..... 521
ordre
méthode du second ordre ..... 580
orienté objet ..... 13
outil
*Adobe Reader* ..... 240
*Apache* ..... 26, 356, 357, 362, 375, 383–385, 451
*BeyondCompare* ..... 26
*Boost Python* ..... 141, 144
*Boost* ..... 144
*BuildBot* ..... 27
*CherryPy* ..... 27
*django* ..... 27
*Doxygen* ..... 349
*doxygen* ..... 26
*doxypy* ..... 26
*EasyPHP* ..... 371, 383
*epydoc* ..... 26
*Excel* ..... 154, 156, 322
*GCC* ..... 25
*gcc* ..... 141
*GNUPlot* ..... 380
*Graphviz* ..... 23, 234, 349
*HTML Help Workshop* ..... 26, 342
*InnoSetup* ..... 27, 349
*Jython* ..... 140
*Latex* ..... 23, 240, 349
*Matlab* ..... 22
*MatPlotLib* ..... 381
*Mercurial* ..... 27
*Microsoft Excel* ..... 138
*Microsoft SQL Server* ..... 371, 375
*Microsoft Visual Studio C++* ..... 145
*Miktex* ..... 240, 349
*MinGW* ..... 141
*MoinMoin* ..... 26, 356
*Mozilla Firefox* ..... 239
*MySQL* ..... 27, 371
*Octave* ..... 22
*Oracle* ..... 371
*PostGreSQL* ..... 371
*PyPy* ..... 26
*Pyrex* ..... 25
*pythonw* ..... 356, 444
*R/Apache* ..... 451
*RPyC* ..... 27
*Rweb* ..... 451
*R* ..... 449
*SciTe* ..... 336
*Sphinx* ..... 26
*SQLite3* ..... 138
*SQLite* ..... 27, 368
*SQL* ..... 367
*Standard Template Library (STL)* ..... 144
*SVN* ..... 26, 337–340, 342
*SWIG* ..... 141
*Sybase* ..... 371
*Tesseract* ..... 380
*TeXnicCenter* ..... 241
*Tortoise SVN* ..... 336
*TortoiseSVN* ..... 26, 337
*TurboGears* ..... 28
*Wiki* ..... 328
*wiki* ..... 26, 356
*WinRar* ..... 157
*WinZip* ..... 157

*Zope* . . . . . 28
Miktex . . . . . 421
outils
Ghostscript . . . . . 415
GSview . . . . . 415

**P**

papilon . . . . . 529
paquetage . . . . . 137
parallélisation . . . . . 202
paramètre . . . . . 64
immuable . . . . . 68
modifiable . . . . . 68
nombre variable . . . . . 72
ordre . . . . . 66
passage . . . . . 411
passage par référence . . . . . 68
passage par valeur . . . . . 68
valeur par défaut . . . . . 65, 412
valeur par défaut, modifiable . . . . . 66, 294
pare-feu . . . . . 341
passage
par adresse . . . . . 411
par valeur . . . . . 411
patron de classe . . . . . 144
PDF . . . . . 240
pdf . . . . . 378
écrire . . . . . 378
lire . . . . . 378
pentium . . . . . 760, 761
permutation . . . . . 484, 528
permutations . . . . . 233
pesanteur . . . . . 710
Phong . . . . . 733, 776
pièce
normale . . . . . 108
très truquée . . . . . 111
truquée . . . . . 109
pièce jointe . . . . . 158, 362
pile d'appels . . . . . 122, 276
pivot . . . . . 561
sélection . . . . . 562
pixel . . . . . 183, 701, 730
plan . . . . . 728
plannification de tâches . . . . . 527
plus proches voisins . . . . . 557
poids . . . . . 710
point d'entrée . . . . . 133, 135, 175
pointeur . . . . . 149
pointeur d'arrêt . . . . . 150
poisson . . . . . 588
polygone . . . . . 729
port . . . . . 202, 383
portable . . . . . 144
portée
fonctions . . . . . 71
variables . . . . . 70
portrait . . . . . 758, 775
Postscript . . . . . 417
poulet . . . . . 599
PPV
1-PPV . . . . . 557
k-PPV . . . . . 557
précision des calculs . . . . . 268, 292, 307
prédécesseur . . . . . 234
principe d'optimalité . . . . . 518
priorité des opérateurs . . . . . 31
probabilité . . . . . 275
Problème
distance de Kullback-Leiber . . . . . 587
problème jouet . . . . . 521
procédure . . . . . 399
processeur . . . . . 202
processus
Poisson . . . . . 602, 697
programmation dynamique . . . . . 518
programmation objet . . . . . 83
programmation séquentielle quadratique . . . . . 619
programmes *VBA*
normalisation d'une matrice 4x4 . . . . . 400
programmes, exemples
`biggles` . . . . . 689
`Button` . . . . . 177
`Canvas` . . . . . 183
`CheckButton` . . . . . 179
`compile` . . . . . 75, 117
`copy` . . . . . 263, 282, 283, 295
`DISABLED` . . . . . 176
`doctest` . . . . . 330
`Entry` . . . . . 178
`eval` . . . . . 49, 74
`Frame` . . . . . 187
`grid` . . . . . 186
`IaGraph` . . . . . 687
`id` . . . . . 47, 48
`Label` . . . . . 176
`Listbox` . . . . . 180
`Listbox` réagissant au curseur . . . . . 197
`mainloop` . . . . . 188
`map` . . . . . 76
`Menu` . . . . . 192
`pack` . . . . . 185
`ping` . . . . . 341
`psyco` . . . . . 597
`RadioButton` . . . . . 180
`reload` . . . . . 134

`scipy` ..... 688
`set` ..... 50
`subprocess` ..... 336
`sys.path` ..... 135
`Text` ..... 178
`yield` ..... 73, 95
`zip` ..... 76
`__name__` ..... 135
`__slots__` ..... 108
*Apache* ..... 375, 384
*Apache*, CGI ..... 362–366
*Boost Python* ..... 152
*Boost Python* , mafonction.cpp ..... 152
*Boost Python* , mafonction.h ..... 152
*Boost Python* , main.cpp ..... 152
*Boost Python* , test.py ..... 152
*InnoSetup* ..... 349–351, 353–356
*MoinMoin* ..... 357–362, 384
*SQL* ..... 367–370, 372, 374, 375, 377
ACP ..... 227
algorithme BFGS ..... 590
algorithme de Kohonen ..... 538
algorithme de Kruskal ..... 761
arrondi ..... 279
attendre un clic ..... 654
attendre un thread (1) ..... 204
attendre un thread (2) ..... 205
attendre une touche ..... 655
barre de défilement ..... 181
barycentre ..... 91
BFGS, scipy ..... 590, 592
boucle ..... 260
boucle au sein d'une classe (1) ..... 302
boucle au sein d'une classe (2) ..... 303
boucle au sein d'une classe (3) ..... 303
bouton image ..... 177
calcul d'un quantile ..... 443
calcul d'une intégrale ..... 387
calcul de table de mortalité ..... 672
calcul de taux de fécondité ..... 675
calcul des valeurs propres ..... 447
carré magique (1) ..... 229
carré magique (2) ..... 231
carré magique (3) ..... 231
carré magique et compteur (4) ..... 232
carré magique et permutations (5) ..... 233
circuit hamiltonien ..... 759
clés croissantes ..... 58
code de Vigenère ..... 661
comparaison ..... 291
comptage ..... 80, 274
compteur ..... 280
concaténation ..... 35
connexion ODBC, ajout de données ..... 374
connexion ODBC, création d'une table ..... 373
constructeur ..... 116
construction d'un installateur ..... 351
construction d'un installateur de mise à jour 353
coordonnées polaires ..... 64
copie ..... 43, 44, 47, 66, 101, 103, 105, 106
copie de fichiers avec interface graphique 799
copie de fichiers vers une clé USB ..... 459
copie de fichiers, sauvergarde des paramètres 801
création d'une table avec `sqlite3` ..... 369
damier (1) ..... 218
damier (2) ..... 219
damier (3) ..... 219
damier (4) ..... 219
date de naissance ..... 173
dates ..... 173
décomposition en base 3 ..... 277
décryptage du code de Vigenère ..... 662
dessin du cours d'une action avec *Excel* 453
destructeur ..... 101
déterminer la taille des poissons ..... 597
deux boucles au sein d'une classe (1) ..... 303
deux boucles au sein d'une classe (2) ..... 304
deux boucles au sein d'une classe (3) ..... 304
deux boucles au sein d'une classe (4) ..... 305
deux threads secondaires ..... 203
dictionnaire ..... 46
distance d'édition ..... 526
écrire un rapport au format HTML ..... 564
écriture condensée ..... 42
énigme de Harry Potter, tome 1 (avec des classes) ..... 500
énigme de Harry Potter, tome 1 (avec des fonctions) ..... 499
ensemble des permutations ..... 484
envoi d'un mail automatiquement ..... 158
Eratosthène ..... 61
estimation du nombre $\pi$ ..... 138
exception ..... 123–125, 128, 129
exécution de tests unitaires répartis sur plusieurs fichiers ..... 333
exemple de fichier *hhk* ..... 346
exemple de fichier *hpp* pour *HTML Help Workshop* ..... 345
exemple de module ..... 133
exemple de test avec `doctest` ..... 330
exemple de test plus complet avec `doctest` 331
exercice pour s'évaluer , correction 2006 ..... 243
exercice pour s'évaluer , correction 2007 ..... 245

exercice pour s'évaluer , correction 2008 . 248
exercice pour s'évaluer , correction 2009 251, 256
existence d'un module . . . . . 136
expressions régulières (1) . . . . . 170
expressions régulières (2) . . . . . 171
extraction de liens Internet . . . . . 226
extraction du texte d'un fichier *pdf* . . . . . 378
factorielle . . . . . 69
Fibonacci . . . . . 280, 281, 289
fichier , écriture . . . . . 155
fichier , lecture . . . . . 155, 156
fichier ouvert et exception . . . . . 131
fichier zip . . . . . 157
fichiers binaires (1) . . . . . 162
fichiers binaires (2) . . . . . 163
filtrage d'un signal sonore . . . . . 721
fonction `isinstance` . . . . . 117
fonctions usuelles pour la communication entre *Python* et *Excel* . . . . . 447
fond d'écran . . . . . 471
formatage de chaîne de caractères . . . . . 36
génération automatique d'un fichier *chm* 346
génération automatique de l'aide . . 328, 343
goûts en commun . . . . . 682
graphe avec matplotlib . . . . . 224
graphe d'une série financière . . . . . 629
graphiques avec GNUPlot . . . . . 380
graphiques avec MatPlotLib . . . . . 381
grenouille . . . . . 261, 262
héritage . . . . . 118
héritage multiple . . . . . 116
histogramme . . . . . 603, 604, 685, 689
hypercube . . . . . 293–295
image de synthèse, base . . . . . 735, 776
image de synthèse, complet . . . . . 744, 785
image de synthèse, facette . . . . . 745, 787
image de synthèse, facette image . . . . . 789
image de synthèse, scène . . . . . 741, 782
image de synthèse, sphère . . . . . 739, 781
import d'une DLL . . . . . 149
import de module (1) . . . . . 133
import de module (2) . . . . . 134
import de module (3) . . . . . 134
import de module (4) . . . . . 135
import dynamique de module . . . . . 136
importer un module en C++ . . . . . 142
insertion de valeurs dans une table `sqlite3` 369
installation d'un module . . . . . 140
intégrale de Monte Carlo . . . . . 139
intégrale, scipy . . . . . 388
interface . . . . . 176, 792–794
interface graphique avec Tkinter . . . 647, 648
itérateur . . . . . 59, 94, 130
jeu avec des girafes . . . . . 656
jeu de caractères . . . . . 173, 174
jeu de caractères , fichier . . . . . 174
jeu du pendu avec Tkinter . . . . . 652
joueurs asynchrones . . . . . 210
lancer un navigateur en ligne de commande 166
lancer un programme *Python* en ligne de commande . . . . . 165, 335
lecture d'un fichier texte . . . . . 221
lecture d'une page HTML . . . . . 139
ligne de commande (1) . . . . . 165
ligne de commande (2) . . . . . 166
ligne de commande *SciTe* . . . . . 336
liste `ComboBox` . . . . . 182
liste avec barre de défilement . . . . . 182
liste de fichiers (1) . . . . . 159
liste de fichiers (2) . . . . . 160
liste de fichiers (3) . . . . . 165
liste de tests avec `unittest` . . . . . 332
liste des modules . . . . . 136
liste récursive (1) . . . . . 296
liste récursive (2) . . . . . 296
logarithme . . . . . 307
loi exponentielle . . . . . 603, 698
matrice . . . . . 79, 81, 119
matrice , écriture . . . . . 154, 156
matrice , lecture . . . . . 156
matrice, lignes nulles . . . . . 275
maximum . . . . . 78
maximum avec indices . . . . . 78
maximum avec la fonction `max` . . . . . 78
maximum sur un sous-ensemble . . . . . 78
mélange de lois Poisson . . . . . 604
méthode paramètre d'une fonction . . . . . 119
méthode statique . . . . . 96, 97
module `pickle` et classes (1) . . . . . 164
module `pickle` et classes (2) . . . . . 164
module à tester . . . . . 332
moyenne mobile centrée (1) . . . . . 298
moyenne mobile centrée (2) . . . . . 299
moyenne, variance . . . . . 265
occurrences dans une liste . . . . . 632
OHLC . . . . . 619
opérateur . . . . . 93
opérateur addition . . . . . 92
optimisation en fonction d'une donnée aléatoire . . . . . 601
optimisation en fonction d'une donnée aléatoire selon un mélange de lois . . 605
partager des données . . . . . 206

pièce ..... 109–111
plus court chemin dans un graphe . 515, 522
plus proches voisins ..... 570
plus proches voisins LAESA ..... 574
plus proches voisins LAESA sur des images 575
plus proches voisins sur des images ..... 571
popularité avec des moteurs de recherche 636
premier thread ..... 203
propriété ..... 99
pygame, son ..... 655
pygame, texte, police ..... 655
racine carrée ..... 307
recherche ..... 77
recherche de mot-clés sur Internet (1) .. 464
recherche de mot-clés sur Internet (2) .. 466
recherche dichotomique ..... 78
récupération d'un texte sur Internet .... 221
récupération de chiffre de la base MNIST 567
référencement d'objets ..... 102
résolution d'un puzzle ..... 487
résolution d'un sudoku (classes) ..... 476
résolution d'un sudoku (fonctions) ..... 474
résolution de l'énigme d'Einstein ..... 494
sélection d'un fichier ..... 195, 795
séquence aléatoire ..... 88
sérialisation ..... 163
simulation d'un pendule ..... 644
simulation d'une corde en chute libre ... 714
simulation de longévité des ampoules ... 698
simulation de tailles de poissons ..... 594
simulation génétique ..... 668
somme ..... 29, 41, 51, 79
somme d'éléments aléatoires ..... 297
sortie HTML ..... 239
sortie PDF ..... 240
star ..... 634
suppression ..... 62
suppression dans une liste ..... 264
table équilibrée ..... 508
table harmonie ..... 502
test ..... 55
thread et interface graphique ..... 208
Tkinter avec des classes ..... 198
Tkinter, bouton ..... 188
Tkinter, compte à rebours ..... 196
Tkinter, fonction `bind` ..... 190
Tkinter, menu ..... 193
Tkinter, séquence d'événements ... ..... 199
tombola ..... 287
tracé d'une ellipse ..... 756
tracé d'une ligne ..... 708, 750, 754
tri ..... 39, 80
tri avec positions initiales ..... 80
tri classique ..... 641
tri d'entiers ..... 285, 396
tri d'un sous-ensemble d'un tableau .... 300
tri fusion ..... 395, 641
tri par insertion ..... 396
tri par sélection ..... 394
tri rapide (1) ..... 236
tri rapide (2) ..... 237
tri, position ..... 305
tri, position initiale ..... 80
trois objets Tkinter ..... 650
trois objets Tkinter (2) ..... 651
trois objets Tkinter (2) (correction) .... 652
un module en C++ ..... 141
utilisation de messages personnalisés ... 200
variable locale ..... 296
VBA ..... 404–413
VBA Python ..... 444–448
VBA, chercher des fichier ..... 428
VBA, chercher des fichiers ..... 427, 428
VBA, conversion ps pdf ..... 417, 421
VBA, document Word ..... 416
VBA, envoyer un mail ..... 417
VBA, exception ..... 417
VBA, présentation d'images ..... 429
VBA, Shell ..... 417
VBA, sudoku ..... 437
vecteur ..... 81
version graphique de la fonction `raw_input` 55
voyageur de commerce et algorithme de Kruskal ..... 544
projection ..... 528
propagation de la chaleur ..... 721
propriété ..... 99, 100
propriétés croissantes ..... 114
psychokinèse ..... 692
puzzle ..... 479, 481
py ..... 20
pyc ..... 134
Python
dictionnaire ..... 684
*Python Scripter* ..... 19

## Q

quadrilatère ..... 728, 745, 775
quadrillage ..... 532, 537
quasi-Newton ..... 580
quicksort ..... 234, 558

**R**

*R* . . . . . 24
R
    `data.frame` . . . . . 450
    `dev.off` . . . . . 452
    `png` . . . . . 452
    `read.table` . . . . . 450
    `tseries` . . . . . 449
R* tree . . . . . 561
R+ Tree . . . . . 561
R-tree . . . . . 559
racine carrée . . . . . 307
radiosité . . . . . 734
Rappel
    loi de Poisson et loi exponentielle . . . . . 693
rapport . . . . . 23
rayon . . . . . 724, 727
    réfléchi . . . . . 776
rayon incident . . . . . 789
recherche dichotomique . . . . . 242, 391
recherche par dichotomie . . . . . 712
reconnaissance
    manuscrite . . . . . 564
récurrence . . . . . 271, 484, 485
récursivité . . . . . 69, 260, 261, 289
    condition d'arrêt . . . . . 69
référence circulaire . . . . . 107, 164
références
    Bailly2008 . . . . . 8, 805
    Beckmann1990 . . . . . 561, 805
    Berchtold1996 . . . . . 561, 805
    Bottou1991 . . . . . 580, 583, 805
    Bracewell1986 . . . . . 721, 805
    Bresenham1965 . . . . . 702, 805
    Bresenham1977 . . . . . 703, 707, 805
    Broyden1967 . . . . . 581, 805
    Brucher2008 . . . . . 8, 805
    Davidon1959 . . . . . 582, 805
    Dayley2007 . . . . . 8, 805
    Dempster1977 . . . . . 589, 805
    Driancourt1996 . . . . . 580, 805
    Duhamel1990 . . . . . 720, 805
    Faragó1993 . . . . . 563, 805
    Faure2000 . . . . . 696, 806
    Fletcher1963 . . . . . 582, 806
    Fletcher1993 . . . . . 581, 806
    Gouriéroux1983 . . . . . 607, 805
    Guttman1984 . . . . . 559, 806
    Helsgaun2000 . . . . . 528, 535–537, 806
    Henry2008 . . . . . 608, 806
    Hull2007 . . . . . 607, 806
    Jacquillat2008 . . . . . 607, 611, 806
    Koivisto1999 . . . . . 528, 806
    Mallat2000 . . . . . 720, 806
    Martelli2004 . . . . . 8, 805
    Math2000 . . . . . 721, 806
    Moré1977 . . . . . 580, 806
    Moreno2003 . . . . . 562, 806
    Nussbaumer1982 . . . . . 720, 806
    Pilgrim2004 . . . . . 8, 805
    Potters2005 . . . . . 625, 806
    Rico-Juan2003 . . . . . 561, 806
    Riva2005 . . . . . 402, 805
    Robert1996 . . . . . 593, 806
    RocchiChristiaens2007 . . . . . 607, 611, 806
    Saporta2006 . . . . . 607, 805
    Sellis1987 . . . . . 559, 561, 806
    Swinnen2004 . . . . . 8, 805
    Vesanto2000 . . . . . 528, 529, 806
    Wilmott2008 . . . . . 607, 806
    Ziadé2006 . . . . . 8, 805
reflet . . . . . 745
réflexion . . . . . 726
réfraction . . . . . 726
release . . . . . 266
remarque
    *Apache* écoute un autre port que le port 80 341
    *Apache*, comprendre une erreur . . . . . 385
    affectation et copie . . . . . 104
    Algorithme Expectation-Maximisation . . 589
    ambiguïté . . . . . 98
    associer un événement à tous les objets . 191
    attendre l'exécution d'un script sous *Excel* 448
    blocage d'un programme . . . . . 207
    caractères spéciaux et expressions régulières 168
    chaîne de caractères, caractères spéciaux 406
    changement de port pour *Apache* . . . . . 383
    code de fin de ligne . . . . . 156, 322
    commentaires . . . . . 30
    connexion sur *Apache* impossible depuis une autre machine du réseau . . . . . 341
    constructeur et fonction . . . . . 88
    convergence de l'algorithme 23.16 . . . . . 538
    conversion en nombre entier . . . . . 31
    coût d'un algorithme . . . . . 266
    coût dépendant du contenu des données . 521
    déclaration d'une méthode statique . . . . . 96
    désactiver un événement . . . . . 191
    dictionnaire et clé de type tuple ... . . . . 220
    division entière . . . . . 31
    écriture différée . . . . . 154
    événement associé à une méthode ... . . . . 189
    événement spécifique . . . . . 192

fenêtre intempestive . . . 16
fichier de traces . . . 155
fichier formaté . . . 322
fichiers ouverts et non fermés . . . 154
focus . . . 191
héritage multiple et constructeur ... . . . 116
indentation . . . 53
instruction sur plusieurs lignes . . . 30
instructions d'installation et de configuration . . . 328
les accents dans un script CGI . . . 363
liste et barre de défilement . . . 181
majuscules et minuscules . . . 158
matrice et dictionnaire . . . 220
mauvais événement . . . 191
Method resolution order . . . 112
méthodes et fonctions . . . 85
modification d'un dictionnaire dans une boucle . . . 46
nombres français et anglais . . . 157
paramètres contenant des espaces ... 166, 335
passage par adresse . . . 68
plusieurs erreurs . . . 124
plusieurs Listbox . . . 182
plusieurs ordres passés dans la même journée . . . 621
portefeuille d'actions, de stratégies . . . 619
portefeuille d'actions, position vendeuse . 619
pour aller plus loin . . . 358
préfixe r, chaîne de caractères . . . 33
priorité des opérations . . . 54
problème d'exécution . . . 652
problèmes de compilation . . . 348
propriété et héritage . . . 100
recharger un module . . . 134
réduction des accès à quelques threads . . 207
retournement, déplacement, ... . . . 537
SGBD et les dates . . . 375
sortie texte et graphique . . . 23
suppression de variables associées à la même instance . . . 101
surcharge d'attributs . . . 113
syntaxe et espaces . . . 52
T-uple, opérateur [] . . . 37
template *C++* . . . 147
type d'une instance . . . 84
utilisation d'un logiciel de suivi de source 144
utilité de l'arbre de poids minimum . . . 537
valeurs par défaut de type modifiable . . . . 66
variables globales . . . 652
version du langage HTML . . . 463
repère . . . 732, 775
répertoire . . . 159, 457, 792, 800
repository . . . 336
représentation . . . 480
représentation des données . . . 217
représentation paramétrique . . . 727
ressort . . . 712
résultat d'une fonction . . . 64
retournement . . . 536, 537

**S**

saison . . . 606
saut . . . 51
scène . . . 724
Schwartz . . . 721
*SciTe* . . . 17
script
CGI . . . 356, 362
Self Organized Map . . . 528
séquence . . . 51, 60
collage . . . 42
sérialisation . . . 107, 138
service
Windows . . . 375
SGBD . . . 371
shebang . . . 385
simulation . . . 712
loi
exponentielle . . . 602
Poisson . . . 602
mélange de lois normales . . . 592
simulation numérique . . . 692, 700
SOM . . . 528, 529
somme . . . 79
son . . . 719
sortie graphique . . . 23
sous-programme . . . 64
spammeur . . . 158
sphère . . . 775
SQL . . . 367
`AVG` . . . 370
`CREATE` . . . 369
`FROM` . . . 370
`GROUP BY` . . . 370
`HAVING` . . . 370
`INSERT` . . . 370
`INTO` . . . 370
`primary key` . . . 369
`SELECT` . . . 370
`TABLE` . . . 369
`UPDATE` . . . 370
`VALUES` . . . 370
curseur . . . 369
requête . . . 367
table . . . 367

unicode . . . 370
Standard Template Library . . . 145
stationnaire . . . 693
Stirling . . . 393
STL . . . 145
stochastique . . . 582, 583
Strassen . . . 119
strétégie . . . 607
successeur . . . 234
sudoku . . . 432, 474
supermarché . . . 599
surcharge . . . 67, 93, 111
attribut . . . 113
fonction . . . 111, 112
opérateur . . . 93
synchronisation . . . 202, 204, 206
syntaxe . . . 51
`copy` . . . 101
`deepcopy` . . . 105
`def` . . . 64, 65, 72
`for` . . . 58, 59
`if elif else` . . . 53, 55
`if else` . . . 53
`issubclass` . . . 116
`lambda` . . . 73
`range` . . . 41
`staticmethod` . . . 96
`while` . . . 57
`__add__` . . . 92
`__copy__` . . . 103
`__deepcopy__` . . . 105
`__iadd__` . . . 92
`__init__` . . . 87
`__str__` . . . 93
ajout de méthode . . . 98
appel à une méthode de l'ancêtre . . . 113
attribut figé . . . 108
attribut statique . . . 97
chaîne de caractères, formatage . . . 36
chaîne de caractères, formatage des nombres 36
classe, appel d'une méthode de l'ancêtre 113
classe, attribut statique . . . 97
classe, attribut, définition . . . 86
classe, constructeur . . . 87
classe, création d'une variable de type objet 84
classe, définition . . . 83
classe, héritage . . . 112
classe, instance . . . 87
classe, méthode statique . . . 96
classe, méthode, appel . . . 85
classe, méthode, définition . . . 85
classe, propriété . . . 99
constructeur . . . 87
destructeur . . . 88
espace . . . 52
exception . . . 124
exception d'un type donné . . . 125
exception, lancement . . . 127
fonction, appel . . . 64
fonction, définition . . . 64
fonction, nombre variable de paramètres . . 72
fonction, nombre variable de paramètres (appel) . . . 72
fonction, ordre des paramètres . . . 66
fonction, valeur par défaut . . . 65
fonctions et chaînes de caractères . . . 34
formatage d'une chaîne de caractères . . . 36
formatage des nombres . . . 36
héritage . . . 112
instance . . . 87
méthode . . . 34
méthode statique . . . 96
propriété . . . 99
test . . . 53
test condensé . . . 55
test enchaîné . . . 53


## T


table . . . 367
tabulations . . . 20
taille mémoire . . . 162, 163
TD
TD * . . . 471
TD 10 . . . 800
TD 12 . . . 461
TD 2 . . . 683
TD 5 . . . 750
TD 6 . . . 757
TD 7 . . . 775
TD 8 . . . 457
TD 9 . . . 792
template . . . 144
temps
exécution . . . 328
mesure . . . 328
tension . . . 710
test . . . 51, 53, 56
`if 0` . . . 56
`if None` . . . 56
systématique . . . 329
unitaire . . . 331
test de non-régression . . . 27
The Independent . . . 699
Théorème

[Faragó1993][1] ..... 564
[Faragó1993][2] ..... 564
convergence de la méthode de Newton (Bottou1991) ..... 579
distance de Kullback-Leiber ..... 587
simulation d'une loi de Poisson ..... 602
simulation d'une loi quelconque ..... 602
somme de loi exponentielle iid ..... 602
théorème central limite ..... 683
thread ..... 202
acquire ..... 207
Condition ..... 205
event_generate ..... 207
Event ..... 205
isSet ..... 205
Lock ..... 206, 207
release ..... 207
run ..... 202
Semaphore ..... 207
sleep ..... 204
start ..... 202
threading ..... 202
Thread ..... 202
wait ..... 205
fil d'exécution ..... 202
verrou ..... 206, 207
Tkinter
< Button − i > ..... 190
< ButtonRelease − i > ..... 190
< Double − Button − i > ..... 190
< Enter > ..... 190
< Key > ..... 190
< Leave > ..... 190
< Motion > ..... 190
add_cascade ..... 192
add_command ..... 192
after_cancel ..... 197
after ..... 197
background ..... 185
bind_all ..... 191
bind ..... 189, 190, 200
borderwidth ..... 185
BOTTOM ..... 186
Button ..... 177
Canvas ..... 183
Checkbutton ..... 179
columnspan ..... 186
column ..... 186
ComboBox ..... 182
config ..... 179, 181, 183, 184, 192
create_line ..... 183
create_rectangle ..... 183
create_text ..... 183
curselection ..... 181
deiconify ..... 194
delete ..... 178, 179, 181
destroy ..... 194
DISABLED ..... 184
Entry ..... 178
event_generate ..... 200, 207, 209
Event ..... 189, 191, 200, 209
exportselection ..... 182
focus_set ..... 188, 191
font ..... 185
Frame ..... 181, 187
geometry ..... 194
get ..... 178–181
grid_forget ..... 186
grid ..... 186
help ..... 184
iconify ..... 194
insert ..... 178, 181
IntVar ..... 179, 180
itemconfigure ..... 182, 197
justify ..... 185
Label ..... 176, 177
LEFT ..... 185–187
Listbox ..... 180, 197
mainloop ..... 188, 195
Menu ..... 192
nearest ..... 197
NORMAL ..... 184
pack_forget ..... 186
pack ..... 185
PhotoImage ..... 177
place_forget ..... 187
place ..... 187
Radiobutton ..... 180
resizable ..... 194
RIGHT ..... 185–187
rowspan ..... 186
row ..... 186
Scrollbar ..... 181, 182
select_clear ..... 181
select_set ..... 181
select ..... 179
set ..... 180
size ..... 181
sticky ..... 186
takefocus ..... 185
Text ..... 178
title ..... 194
Tk ..... 188
Toplevel ..... 194
TOP ..... 186
unbind_all ..... 192

`unbind` ... 192
`withdraw` ... 194
barre de défilement ... 182
boucle de message ... 188
bouton ... 177
bouton radio ... 180
canevas ... 183
case à cocher ... 179
case ronde ... 180
compte à rebours ... 196
contrôle ... 176
événement, association ... 189
événement, désactivation ... 192
fenêtre ... 194
focus ... 185, 188, 191
liste ... 180
menu ... 192
objets ... 176
positionnement d'objet ... 185
réactivité ... 202
sous-fenêtre ... 187
tabulation ... 188
thread ... 202, 207
touche tabulation ... 185
touches muettes ... 190
widget ... 176
zone de saisie ... 178
zone de saisie à plusieurs lignes ... 178
zone de texte ... 176
toy problem ... 521
tracé d'une ellipse ... 703, 753
tracé d'une ligne ... 702, 750
tracé d'une ligne (Bresenham) ... 750
traces ... 150, 155
traction ... 712
trading ... 607
transformée de Fourier
discrète ... 719
rapide ... 719
transformée de Hartley ... 721
travelling salesman problem ... 527
tri ... 80
classique ... 641
comparaisons successives ... 393
entiers ... 285, 393
fusion ... 388, 389, 641
insertion ... 390
insertion dichotomique ... 388
liste ... 38
position initiale ... 80
quicksort ... 234, 388, 521, 558
rapide ... 388
sélection ... 388
triangle ... 745
trigonométrique ... 481
TSP ... 527, 757
typage dynamique ... 13
type ... 29
`bool` ... 32
`complex` ... 38
`dict` ... 44, 104, 273
`Exception` ... 124
`False` ... 32
`float` ... 31
`frozenset` ... 50
`int` ... 31
`list` ... 38, 59, 104
`None` ... 30, 64
`set` ... 50
`string` ... 33
`True` ... 32
`tuple` ... 36
Boolean ... 405
Byte ... 405
chaîne de caractères ... 33
complexe ... 38
Date ... 405
dictionnaire ... 44, 104, 218
Double ... 405
ensemble ... 50
entier ... 31
fonction ... 81
immuable ... 30, 68
immutable ... 30
Integer ... 405
liste ... 38, 59, 104, 218
Long ... 405
modifiable ... 30, 38, 68, 83
mutable ... 30
objet ... 84
réel ... 31
rien ... 30
séquence ... 42
String ... 405
T-uple ... 538
Variant ... 405
types fondamentaux ... 29

**U**

unicode ... 173, 370
union ... 50
USB ... 457, 792, 800
utf-8 ... 21, 173

**V**

vague ... 586
valeur ... 44

propre . . . 443
valeur par défaut . . . 65
objet modifiable . . . 295
valeur par défaut modifiable . . . 294
Van Rossum, Guido . . . 12
variable
déclaration . . . 404
globale . . . 71, 84, 86, 407
identificateur . . . 29, 70
locale . . . 70
portée . . . 70
statique . . . 204, 206
type . . . 29, 404
variable aléatoire . . . 683
variables
indépendantes . . . 585
variance . . . 584
variations saisonnières . . . 606
VBA . . . 399
`ShellWait` . . . 448, 452
`Shell` . . . 445, 448
vecteur . . . 37, 81, 115
propre . . . 443
vecteur normal . . . 706
version, debug, release . . . 149, 150, 266
vitesse d'exécution . . . 140
voyageur de commerce . . . 527, 757–759
vraisemblance . . . 584, 594

**W**

widget . . . 176, 792
wiki . . . 26
Wikipedia . . . 356
WikiPython . . . 356
Windows
service . . . 375

**X**

XML
lecture . . . 322

**Y**

Yahoo Finance . . . 22, 449

**Liens**

zone . . . 537
zone de saisie . . . 793
zone de texte . . . 793
numéros
Algorithme
16.1 . . . 389
16.2 . . . 389
16.3 . . . 390
16.4 . . . 391
16.5 . . . 391
16.6 . . . 394
23.1 . . . 519
23.10 . . . 531
23.13 . . . 532
23.14 . . . 534
23.15 . . . 535
23.16 . . . 536
23.20 . . . 557
23.21 . . . 558
23.23 . . . 560
23.24 . . . 561
23.25 . . . 562
23.26 . . . 562
23.27 . . . 564
23.3 . . . 520
23.30 . . . 578
23.32 . . . 581
23.33 . . . 583
23.37 . . . 593
23.7 . . . 530
27.2 . . . 702
27.3 . . . 703
27.4 . . . 713
27.5 . . . 714
28.2 . . . 731
29.1 . . . 750
29.2 . . . 753
29.3 . . . 758
Définition
2.1 . . . 11
2.2 . . . 12
2.3 . . . 12
2.4 . . . 12
2.5 . . . 12
2.6 . . . 12
23.11 . . . 531
23.12 . . . 532
23.2 . . . 520
23.22 . . . 558
23.4 . . . 520
23.6 . . . 528
23.8 . . . 531
23.9 . . . 531
28.1 . . . 727
3.1 . . . 29
3.13 . . . 36
3.15 . . . 38
3.16 . . . 38
3.19 . . . 44
3.2 . . . 29
3.5 . . . 30
3.8 . . . 33

4.12 . . . . . 64
4.19 . . . . . 69
4.2 . . . . . 53
4.20 . . . . . 70
4.21 . . . . . 70
4.22 . . . . . 71
4.8 . . . . . 57
5.1 . . . . . 83
5.10 . . . . . 86
5.18 . . . . . 96
5.21 . . . . . 97
5.34 . . . . . 111
5.35 . . . . . 111
5.4 . . . . . 84
5.6 . . . . . 84
6.1 . . . . . 122
7.1 . . . . . 133
démonstration
théorème 23.31 . . . . . 579
théorème 23.38 . . . . . 602
théorème 23.40 . . . . . 602
théorème 23.39 . . . . . 603
Problème
23.34 . . . . . 587
Rappel
27.1 . . . . . 693
syntaxe
3.10 . . . . . 34
3.11 . . . . . 36
3.12 . . . . . 36
3.17 . . . . . 41
4.10 . . . . . 58
4.11 . . . . . 59
4.13 . . . . . 64
4.14 . . . . . 64
4.15 . . . . . 65
4.17 . . . . . 66
4.23 . . . . . 72
4.24 . . . . . 72
4.25 . . . . . 73
4.3 . . . . . 53
4.9 . . . . . 57
5.11 . . . . . 86
5.12 . . . . . 87
5.13 . . . . . 87
5.15 . . . . . 92
5.16 . . . . . 92
5.17 . . . . . 93
5.19 . . . . . 96
5.2 . . . . . 83
5.22 . . . . . 97
5.24 . . . . . 98
5.25 . . . . . 99
5.27 . . . . . 101
5.29 . . . . . 103
5.3 . . . . . 84
5.31 . . . . . 105
5.32 . . . . . 105
5.33 . . . . . 108
5.36 . . . . . 112
5.38 . . . . . 113
5.41 . . . . . 116
5.7 . . . . . 85
5.8 . . . . . 85
6.2 . . . . . 124
6.4 . . . . . 125
6.5 . . . . . 127
9.10 . . . . . 192
9.5 . . . . . 190
Théorème
23.28 . . . . . 564
23.29 . . . . . 564
23.31 . . . . . 579
23.35 . . . . . 587
23.38 . . . . . 602
23.39 . . . . . 602
23.40 . . . . . 602
C :/Python25 . . . . . 14
C :/Python25/python.exe . . . . . 135
http :// addictedtor. free. fr/ graphiques/ . . . . . 451
http :// addictedtor. free. fr/ graphiques/ allgraph. php 443, 449
http :// biggles. sourceforge. net/ . . . . . 22
http :// biostat. mc. vanderbilt. edu/ twiki/ bin/ view/ Main/ RApacheProject . . . . . 451
http :// boa-constructor. sourceforge. net/ . . . . . 20
http :// code. foosel. org/ doxypy . . . . . 26, 90, 349
http :// code. google. com/ appengine/ . . . . . 13
http :// code. google. com/ p/ gmpy/ . . . . . 271
http :// code. google. com/ p/ pydot/ . . . . . 23, 235
http :// code. google. com/ p/ pytesser/ . . . . . 380
http :// code. google. com/ p/ pythongoogledesktop/ 140
http :// code. google. com/ p/ quodlibet/ wiki/ Development/ Mutagen . . . . . 158, 169
http :// codespeak. net/ pypy/ dist/ pypy/ doc/ home. html . . . . . 26
http :// coverage. livinglogic. de/ index. html . . . . . 182
http :// cran. r-project. org/ doc/ manuals/ R-intro. pdf 449
http :// cx-oracle. sourceforge. net/ . . . . . 377
http :// dev. mysql. com/ doc/ refman/ 5. 0/ fr/ myodbc-windows-binary-installation. html . . 372
http :// dev. mysql. com/ downloads/ mysql/ 5. 0. html#downloads . . . . . 371
http :// docs. python. org/ c-api/ intro. html#objects-types-and-reference-counts . . . 144
http :// docs. python. org/ download. html . . . . . 14
http :// docs. python. org/ howto/ regex. html#regex-howto . . . . . 168
http :// docs. python. org/ library/ codecs. html#id3174
http :// docs. python. org/ library/ datetime. html . . 173
http :// docs. python. org/ library/ exceptions. html . 126

http :// docs. python. org/ library/ operator. html#mapping-operators-to-functions . . . . . 121
http :// docs. python. org/ library/ os. html . . . . . 160
http :// docs. python. org/ library/ os. path. html . . . 159
http :// docs. python. org/ library/ re. html . . . . . 170
http :// docs. python. org/ library/ re. html#regular-expression-syntax . . . . . 168
http :// docs. python. org/ library/ shutil. html . . . . . 160
http :// docs. python. org/ library/ stdtypes. html#id4 35, 174
http :// docs. python. org/ library/ stdtypes. html#set50
http :// docs. python. org/ library/ stdtypes. html#string-formatting-operations . . . . . 37
http :// docs. python. org/ library/ struct. html . . . . . 162
http :// docs. python. org/ library/ tix. html . . . . . 182
http :// docs. python. org/ modindex. html . . . . . 138
http :// docs. python. org/ reference/ datamodel. html#special-method-names . . . . . 121
http :// docs. python. org/ reference/ expressions. html#operator-summary . . . . . 31
http :// drpython. sourceforge. net/ . . . . . 20
http :// effbot. org/ tkinterbook/ . . . . . 175
http :// elixir. ematia. de/ trac/ . . . . . 24
http :// epydoc. sourceforge. net/ . . . . . 26, 90, 349
http :// fr. wikipedia. org/ wiki/ HTML . . . . . 364
http :// gnuprog. info/ prog/ python/ pwidget. php . . 175
http :// matplotlib. sourceforge. net/ . . . . . 22, 224, 241
http :// mdp-toolkit. sourceforge. net/ . . . . . 24, 227
http :// moinmo. in/ HelpOnConfiguration . . . . . 357
http :// moinmo. in/ HelpOnInstalling/ ApacheOnWin32 356
http :// moinmo. in/ HelpOnMoinWikiSyntax . . . . . 360
http :// moinmo. in/ MoinDev/ ThemeDevelopment . 360
http :// moinmoin. wikiwikiweb. de/ ThemeMarket . . 360
http :// msdn. microsoft. com/ en-us/ library/ ms669985. aspx . . . . . 342
http :// pages. cs. wisc. edu/ ghost/ doc/ AFPL/ index. htm . . . . . 415
http :// pydev. sourceforge. net/ . . . . . 19
http :// pydoc. org/ 2. 1/ Tix. html . . . . . 182
http :// pyopengl. sourceforge. net/ . . . . . 25
http :// pyparsing. wikispaces. com/ . . . . . 23, 234
http :// python. net/ crew/ mhammond/ win32/ Downloads. html . . . . . 25, 153, 377
http :// rpy. sourceforge. net/ index. html . . . . . 449
http :// scintilla. sourceforge. net/ SciTEDownload. html 17
http :// scintilla. sourceforge. net/ SciTEFAQ. html . . 18
http :// sebsauvage. net/ python/ snyppets/ index. html 328
http :// sourceforge. net/ projects/ directpython/ ... . 25
http :// sourceforge. net/ projects/ mysql-python27, 373, 376
http :// subversion. tigris. org/ . . . . . 26
http :// subversion. tigris. org/ project_packages.html 340
http :// tortoisesvn. tigris. org/ . . . . . 26, 145, 336
http :// wiki. python. org/ moin/ IntegratedDevelopmentEnvironments . . . . . 20
http :// wiki. python. org/ moin/ PythonEditors . . . . . 20
http :// wiki. python. org/ moin/ TkInter . . . . . 176
http :// wikipython. flibuste. net/ . . . . . 8, 22, 356
http :// wikipython. flibuste. net/ CodeWindows# ExempleavecExcel . . . . . 25
http :// www. abcbourse. com/ marches/ stats_eco.aspx 608
http :// www. adobe. com/ fr/ products/ acrobat/ readstep2. html . . . . . 240
http :// www. barebones. com/ products/ textwrangler/ 20
http :// www. boddie. org. uk/ david/ Projects/ Python/ pdftools/ . . . . . 378
http :// www. boost. org/ doc/ libs/ 1_36_0/ libs/ python/ doc/ index.html . . . . . 141
http :// www. boost. org/ libs/ python/ doc/ . . . . . 150
http :// www. boost. org/ libs/ python/ doc/ tutorial/ doc/ html/ index. html . . . . . 145
http :// www. cfm. fr/ papers/ 0508104. pdf . . . . . 806
http :// www. cifen. ulg. ac. be/ inforef/ swi/ python. htm . . . . . 805
http :// www. cosc. canterbury. ac. nz/ greg. ewing/ python/ Pyrex/ . . . . . 25
http :// www. die-offenbachs. de/ eric/ index. html . . . 20
http :// www. dma. utc. fr/ polytex/ cours. pdf . . . . . 619
http :// www. gnu. org/ copyleft/ gpl. html . . . . . 13
http :// www. gnu. org/ software/ emacs/ windows/ . 20
http :// www. gnu. org/ software/ plotutils/ . . . . . 23
http :// www. jrsoftware. org/ isinfo. php . . . . . 349
http :// www. jython. org/ Project/ . . . . . 140
http :// www. lsp. ups-tlse. fr/ Besse/ pub/ R/ rdebuts_fr.pdf . . . . . 449
http :// www. math. montana. edu/ Rweb/ . . . . . 451
http :// www. microsoft. com/ downloads/ . . . . . 26
http :// www. python. org/ download/ . . . . . 14
http :// www. python. org/ download/ linux/ . . . . . 15
http :// www. python. org/ download/ mac/ . . . . . 15
http :// www. python. org/ psf/ license/ . . . . . 12
http :// www. pythonware. com/ products/ pil/ . . . . . 24
http :// www. pythonweb. org/ projects/ webmodules/ doc/ 0. 5. 3/ html_multipage/lib/node48.html 34
http :// www. riverbankcomputing. co. uk/ software/ pyqt/ intro . . . . . 24
http :// www. scintilla. org/ SciTE. html . . . . . 17
http :// www. scootersoftware. com/ . . . . . 26
http :// www. selenic. com/ mercurial/ wiki/ . . . . . 27
http :// www. stack. nl/ dimitri/ doxygen/ . . . . . 349
http :// www. tsp. gatech. edu/ index. html . . . . . 528
http :// www. vex. net/ parnassus/ . . . . . 22
http :// www. xavierdupre. fr/ enseignement/ initiation/ logpdf. txt . . . . . 247
http :// www. xavierdupre. fr/ enseignement/ initiation/ td_note_texte.txt . . . . . 242
http :// www. xavierdupre. fr/ hal_python/hal_python_help_html/index.html 620
http :// www. xavierdupre. fr/ hal_python/hal_python_help_html/sample_test_python_s 620
http :// www. xavierdupre. fr/ hal_python/hal_python_help_html/sample_test_python_s 620
http :// www. xavierdupre. fr/ hal_python/hal_python_help_html/sample_test_python_t 620
http :// www. xlstat. com/ fr/ home/ . . . . . 443
http ://127.0.0.1/svn/ . . . . . 341
http ://buildbot.net/trac . . . . . 27
http ://code.google.com/ . . . . . 13

http ://diveintopython.org . . . . . 805
http ://docs.python.org/ . . . . . 26
http ://econpapers.repec.org/ . . . . . 806
http ://gcc.gnu.org/ . . . . . 25
http ://httpd.apache.org/ . . . . . 26, 383
http ://localhost/ . . . . . 383
http ://localhost/mywiki . . . . . 384
http ://localhost/svn/ . . . . . 341, 384
http ://localhost :81/ . . . . . 383
http ://miktex.org/ . . . . . 240, 349, 421
http ://mmm-experts.com/ . . . . . 19
http ://moinmo.in/ . . . . . 26, 357
http ://numpy.scipy.org/ . . . . . 23, 446, 449
http ://pages.cs.wisc.edu/ghost/gsview/ . . . . . 415
http ://pmw.sourceforge.net/ . . . . . 195
http ://pybrary.net/pyPdf/ . . . . . 25, 153, 379
http ://pydoc.org/ . . . . . 90
http ://pymedia.org/ . . . . . 158
http ://pypi.python.org/pypi . . . . . 22
http ://pysvn.tigris.org/ . . . . . 342
http ://pyx.sourceforge.net/ . . . . . 23
http ://rpy.sourceforge.net/ . . . . . 24
http ://rpyc.wikispaces.com/ . . . . . 27
http ://sphinx.pocoo.org/ . . . . . 26
http ://turbogears.org/ . . . . . 28
http ://www.advsofteng.com/ . . . . . 23
http ://www.afpy.org/ . . . . . 22
http ://www.ailab.si/orange/ . . . . . 24
http ://www.boost.org/ . . . . . 144
http ://www.cherrypy.org/ . . . . . 27
http ://www.django-fr.org/ . . . . . 27
http ://www.easyphp.org/ . . . . . 371, 383
http ://www.eclipse.org/ . . . . . 19
http ://www.gnuplot.info/ . . . . . 23, 380
http ://www.graphviz.org/ . . . . . 23, 26, 234, 349
http ://www.gutenberg.net/ . . . . . 220, 223
http ://www.innosetup.com/ . . . . . 27
http ://www.lemonde.fr/ . . . . . 225
http ://www.mingw.org/ . . . . . 25, 141
http ://www.mysql.fr/ . . . . . 27
http ://www.pybots.org/ . . . . . 27
http ://www.pygame.org/ . . . . . 25
http ://www.pygresql.org/ . . . . . 373
http ://www.pygtk.org/ . . . . . 24
http ://www.python.org/ . . . . . 8
http ://www.python.org/doc/ . . . . . 14
http ://www.r-project.org/ . . . . . 24, 449, 451
http ://www.rarlab.com/ . . . . . 157
http ://www.reportlab.org/rl_toolkit.html . . . . . 378
http ://www.scipy.org/ . . . . . 23, 25, 388, 446
http ://www.scipy.org/Weave . . . . . 25
http ://www.sqlalchemy.org/ . . . . . 24
http ://www.sqlite.org/ . . . . . 27, 368
http ://www.sqlobject.org/ . . . . . 24, 378
http ://www. stack. nl/ ~dimitri/ doxygen/ . . . . . 26, 90
http ://www.swig.org/ . . . . . 141
http ://www.toolscenter.org/ . . . . . 241
http ://www.wxpython.org/ . . . . . 24
http ://www.xavierdupre.fr . . . . . 8
http ://www.xavierdupre.fr/ . . . . . 145, 148, 152, 247
http ://www.zope.org/ . . . . . 28
https ://sourceforge.net/projects/pywin32/ . . . . . 373

# Table des matières

*1. Avant-propos* . . . . . 8

*Neuvième partie I Le langage Python* 10

*2. Introduction* . . . . . 11
2.1 Ordinateur et langages . . . . . 11
2.1.1 L'ordinateur . . . . . 11
2.1.2 Termes informatiques . . . . . 11
2.2 Présentation du langage *Python* . . . . . 12
2.2.1 Histoire résumée . . . . . 12
2.2.2 Le langage en quelques points . . . . . 13
2.2.3 Avantages et inconvénients du langage *Python* . . . . . 13
2.3 Installation du langage *Python* . . . . . 14
2.3.1 Installation du langage *Python* . . . . . 14
2.3.2 Particularité de *Mac OS X* et *Linux* . . . . . 15
2.3.3 Utilisation de l'éditeur de texte . . . . . 15
2.4 Installation d'un éditeur de texte . . . . . 17
2.4.1 Editeur SciTe . . . . . 17
2.4.2 Python Scripter . . . . . 19
2.4.3 Eclipse et PyDev . . . . . 19
2.4.4 Côté *Mac OS X* : *TextWrangler* . . . . . 20
2.4.5 Autres éditeurs . . . . . 20
2.4.6 Tabulations . . . . . 20
2.5 Premier programme . . . . . 20
2.5.1 Afficher le résultat d'un calcul . . . . . 20
2.5.2 Les accents . . . . . 21
2.6 Installation d'extensions (ou modules externes) . . . . . 22
2.6.1 Graphisme . . . . . 22
2.6.2 Calculs scientifiques . . . . . 23
2.6.3 Images . . . . . 24
2.6.4 Base de données . . . . . 24
2.6.5 Interfaces graphiques . . . . . 24
2.6.6 *Microsoft Windows*, *Microsoft Excel*, PDF . . . . . 25
2.6.7 Accélération de *Python* . . . . . 25

2.7 Outils connexes . . . 26
2.7.1 Outils quotidiens . . . 26
2.7.2 Outils occasionnels . . . 26
2.7.3 Outils pour des besoins spécifiques . . . 27
2.7.4 Outils Internet . . . 27

*3. Types et variables du langage Python* . . . 29
3.1 Variables . . . 29
3.2 Types immuables (ou *immutable*) . . . 30
3.2.1 Type "rien" . . . 30
3.2.2 Nombres réels et entiers . . . 31
3.2.3 Booléen . . . 32
3.2.4 Chaîne de caractères . . . 33
3.2.4.1 Création d'une chaîne de caractères . . . 33
3.2.4.2 Manipulation d'une chaîne de caractères . . . 34
3.2.4.3 Formatage d'une chaîne de caractères . . . 35
3.2.5 T-uple . . . 36
3.2.6 Autres types . . . 38
3.3 Types modifiables (ou *mutable*) . . . 38
3.3.1 Liste . . . 38
3.3.1.1 Définition et fonctions . . . 38
3.3.1.2 Exemples . . . 39
3.3.1.3 Fonctions `range`, `xrange` . . . 41
3.3.1.4 Boucles et listes . . . 41
3.3.1.5 Collage de séquences, fonction `zip` . . . 42
3.3.1.6 Concaténation de chaîne de caractères . . . 42
3.3.2 Copie . . . 42
3.3.3 Dictionnaire . . . 44
3.3.3.1 Définition et fonctions . . . 44
3.3.3.2 Exemples . . . 45
3.3.3.3 Copie . . . 46
3.3.3.4 Clés de type modifiable . . . 47
3.4 Extensions . . . 48
3.4.1 Mot-clé `print`, `repr`, conversion en chaîne de caractères . . . 48
3.4.2 Informations fournies par *Python* . . . 48
3.4.3 Affectations multiples . . . 50
3.4.4 Ensemble . . . 50

*4. Syntaxe du langage Python (boucles, tests, fonctions)* . . . 51
4.1 Les trois concepts des algorithmes . . . 51
4.2 Tests . . . 53
4.2.1 Définition et syntaxe . . . 53
4.2.2 Comparaisons possibles . . . 54

4.2.3 Opérateurs logiques . . . 54
4.2.4 Ecriture condensée . . . 55
4.2.5 Exemple . . . 55
4.2.6 Passer, instruction `pass` . . . 56
4.3 Boucles . . . 57
4.3.1 Boucle `while` . . . 57
4.3.1.1 Syntaxe et exemples . . . 57
4.3.1.2 Condition . . . 57
4.3.2 Boucle `for` . . . 58
4.3.2.1 Syntaxe et exemples . . . 58
4.3.2.2 Listes et boucle `for` . . . 59
4.3.2.3 Itérateurs . . . 59
4.3.2.4 Plusieurs variables de boucles . . . 60
4.3.3 Ecriture condensée . . . 60
4.3.4 Pilotage d'une boucle . . . 61
4.3.4.1 Passer à l'itération suivante : `continue` . . . 61
4.3.4.2 Sortie prématurée : `break` . . . 61
4.3.4.3 Fin normale d'une boucle : `else` . . . 62
4.3.4.4 Suppression ou ajout d'éléments lors d'une boucle . . . 62
4.4 Fonctions . . . 63
4.4.1 Définition, syntaxe . . . 64
4.4.2 Exemple . . . 64
4.4.3 Paramètres avec des valeurs par défaut . . . 65
4.4.4 Ordre des paramètres . . . 66
4.4.5 Surcharge de fonction . . . 67
4.4.6 Commentaires . . . 67
4.4.7 Paramètres modifiables . . . 68
4.4.8 Fonction récursive . . . 69
4.4.9 Portée . . . 70
4.4.9.1 Portée des variables, des paramètres . . . 70
4.4.9.2 Portée des fonctions . . . 71
4.4.10 Nombre de paramètres variable . . . 72
4.4.11 Ecriture simplifiée pour des fonctions simples . . . 73
4.4.12 Fonctions générateur . . . 73
4.4.13 Identificateur appelable . . . 74
4.4.14 Fonctions ajoutées lors de l'exécution du programme . . . 74
4.4.14.1 Fonction `eval` . . . 74
4.4.14.2 Fonctions `compile`, `exec` . . . 75
4.5 Indentation . . . 76
4.6 Fonctions usuelles . . . 76
4.7 Constructions classiques . . . 77
4.7.1 Recherche d'un élément . . . 77

4.7.2 Maximum, minimum . . . 78
4.7.3 Recherche dichotomique . . . 78
4.7.4 Décomposition en matrice . . . 79
4.7.5 Somme . . . 79
4.7.6 Tri . . . 80
4.7.7 Tri et position initiale . . . 80
4.7.8 Comptage . . . 80
4.7.9 Matrice en un vecteur . . . 81
4.7.10 Fonctions et variables . . . 81

*5. Classes* . . . 83
5.1 Présentation des classes : méthodes et attributs . . . 83
5.1.1 Définition, déclaration . . . 83
5.1.2 Méthodes . . . 84
5.1.3 Attributs . . . 86
5.2 Constructeur . . . 87
5.3 Apport du langage *Python* . . . 88
5.3.1 Liste des attributs . . . 88
5.3.2 Attributs implicites . . . 89
5.3.3 Commentaires, aide . . . 89
5.3.4 Classe incluse . . . 91
5.4 Opérateurs, itérateurs . . . 91
5.4.1 Opérateurs . . . 91
5.4.2 Itérateurs . . . 94
5.5 Méthodes, attributs statiques et ajout de méthodes . . . 96
5.5.1 Méthode statique . . . 96
5.5.2 Attributs statiques . . . 97
5.5.3 Ajout de méthodes . . . 98
5.5.4 Propriétés . . . 99
5.6 Copie d'instances . . . 101
5.6.1 Copie d'instance de classe simple . . . 101
5.6.2 Copie d'instance de classes incluant d'autres classes . . . 102
5.6.3 Listes et dictionnaires . . . 104
5.6.4 `copy`et `deepcopy` . . . 105
5.7 Attributs figés . . . 107
5.8 Héritage . . . 108
5.8.1 Exemple autour de pièces de monnaie . . . 108
5.8.2 Syntaxe . . . 112
5.8.3 Sens de l'héritage . . . 114
5.8.4 Héritage multiple . . . 115
5.8.5 Fonctions `issubclass`et `isinstance` . . . 116
5.9 Compilation de classes . . . 117

5.10 Constructions classiques . . . 118
5.10.1 Héritage . . . 118
5.10.2 Deux lignées d'héritages . . . 119

*6. Exceptions* . . . 122
6.1 Principe des exceptions . . . 122
6.1.1 Attraper toutes les erreurs . . . 122
6.1.2 Obtenir le type d'erreur, attraper un type d'exception . . . 124
6.1.3 Lancer une exception . . . 126
6.1.4 Héritage et exception . . . 127
6.1.5 Instructions `try`, `except`imbriquées . . . 128
6.2 Définir ses propres exceptions . . . 129
6.2.1 Dériver une classe d'exception . . . 129
6.2.2 Personnalisation d'une classe d'exception . . . 129
6.3 Exemples d'utilisation des exceptions . . . 130
6.3.1 Les itérateurs . . . 130
6.3.2 Exception ou valeur aberrante . . . 131
6.3.3 Le piège des exceptions . . . 131

*7. Modules* . . . 133
7.1 Modules et fichiers . . . 133
7.1.1 Exemple . . . 133
7.1.2 Autres syntaxes . . . 134
7.1.3 Nom d'un module . . . 135
7.1.4 Emplacement d'un module . . . 135
7.1.5 Ajouter un module en cours d'exécution . . . 136
7.1.6 Liste des modules importés . . . 136
7.1.7 Attributs communs à tout module . . . 137
7.1.8 Arborescence de modules, paquetage . . . 137
7.2 Modules internes . . . 138
7.3 Modules externes . . . 139
7.4 *Python* et les autres langages . . . 140
7.4.1 Langage *Java* . . . 140
7.4.2 Langage *C* . . . 140
7.5 Ecrire un module en langage *C++* . . . 141
7.6 *Boost Python* . . . 144
7.6.1 Exemple . . . 145
7.6.2 Les grandes lignes . . . 145
7.6.2.1 Initialiser le module . . . 146
7.6.2.2 Ajout d'une fonction . . . 146
7.6.2.3 Ajout d'une classe . . . 147
7.6.2.4 Ajout d'une méthode . . . 147
7.6.2.5 Ajout d'un attribut . . . 148

7.6.3 Exemple concret : ajout d'une fonction . . . 148
7.6.4 Garbage collector et pointeur . . . 148
7.6.5 Utilisation, installation . . . 149
7.6.6 Détails sur l'exemple du paragraphe 7.6.1 . . . 149
7.6.7 Debuggage . . . 150
7.6.8 Pour aller plus loin . . . 150

*8. Fichiers, expressions régulières, dates* . . . 153
8.1 Format texte . . . 153
8.1.1 Ecriture . . . 153
8.1.2 Ecriture en mode "ajout" . . . 155
8.1.3 Lecture . . . 155
8.2 Fichiers zip . . . 157
8.2.1 Lecture . . . 157
8.2.2 Ecriture . . . 157
8.3 Manipulation de fichiers . . . 158
8.3.1 Gestion des noms de chemins . . . 159
8.3.2 Copie, suppression . . . 159
8.3.3 Liste de fichiers . . . 159
8.4 Format binaire . . . 161
8.4.1 Ecriture dans un fichier binaire . . . 161
8.4.2 Lecture d'un fichier binaire . . . 162
8.4.3 Exemple complet . . . 162
8.4.4 Objets plus complexes . . . 163
8.5 Paramètres en ligne de commande . . . 165
8.6 Expressions régulières . . . 167
8.6.1 Premiers pas . . . 167
8.6.2 Syntaxe . . . 168
8.6.2.1 Les ensembles de caractères . . . 168
8.6.2.2 Les multiplicateurs . . . 168
8.6.2.3 Groupes . . . 169
8.6.2.4 Assembler les caractères . . . 169
8.6.3 Fonctions . . . 169
8.6.4 Exemple plus complet . . . 169
8.7 Dates . . . 173
8.8 Problème de jeux de caractères . . . 173

*9. Interface graphique* . . . 175
9.1 Introduction . . . 175
9.2 Les objets . . . 176
9.2.1 Zone de texte . . . 176
9.2.2 Bouton . . . 177
9.2.3 Zone de saisie . . . 178

9.2.4 Zone de saisie à plusieurs lignes . . . 178
9.2.5 Case à cocher . . . 179
9.2.6 Case ronde ou bouton radio . . . 180
9.2.7 Liste . . . 180
9.2.8 Liste avec barre de défilement, Combobox . . . 182
9.2.9 Canevas . . . 183
9.2.10 Menus . . . 184
9.2.11 Méthodes communes . . . 184
9.3 Disposition des objets dans une fenêtre . . . 185
9.3.1 Emplacements . . . 185
9.3.1.1 Méthode `pack` . . . 185
9.3.1.2 Méthode `grid` . . . 186
9.3.1.3 Méthode `place` . . . 187
9.3.2 Sous-fenêtre . . . 187
9.4 Evénements . . . 188
9.4.1 Fenêtre principale . . . 188
9.4.2 Focus . . . 188
9.4.3 Lancer une fonction lorsqu'un bouton est pressé . . . 188
9.4.4 Associer n'importe quel événement à un objet . . . 189
9.4.5 Menu . . . 192
9.4.6 Fonctions prédéfinies . . . 194
9.5 D'autres fenêtres . . . 194
9.5.1 Créer une seconde boîte de dialogues . . . 194
9.5.2 Fenêtres standard . . . 195
9.6 Constructions classiques . . . 196
9.6.1 Compte à rebours . . . 196
9.6.2 Contrôles personnalisés : utilisation des classes . . . 197
9.6.3 Fenêtres personnalisées : utilisation des classes . . . 198
9.6.4 Séquence d'événements . . . 199
9.6.5 Communiquer un résultat par message . . . 200

*10. Threads* . . . 202
10.1 Premier thread . . . 202
10.2 Synchronisation . . . 204
10.2.1 Attente . . . 204
10.2.2 Partage d'informations . . . 206
10.3 Interface graphique . . . 207
10.4 Files de messages . . . 209

*Neuvième partie II Enoncés pratiques, exercices* 213

*11. Exercices pratiques pour s'entraîner* . . . 214

11.1 Montant numérique, montant littéral . . . 214
11.1.1 Enoncé . . . 214
11.1.2 Correction . . . 215
11.2 Représentation des données, partie de dames . . . 217
11.2.1 Enoncé . . . 217
11.2.2 Correction . . . 218
11.3 Reconnaître la langue d'un texte . . . 220
11.3.1 Enoncé . . . 220
11.3.2 Correction . . . 221
11.4 Carrés magiques . . . 228
11.4.1 Enoncé . . . 228
11.4.2 Correction . . . 229
11.5 Tri rapide ou quicksort . . . 234
11.5.1 Enoncé . . . 234
11.5.2 Correction . . . 235

*12. Exercices pratiques pour s'évaluer* . . . 242
12.1 Recherche dichotomique (2005-2006) . . . 242
12.2 Ajouter un jour férié (2006-2007) . . . 244
12.2.1 Avec des fonctions uniquement . . . 244
12.2.2 Programme équivalent avec des classes . . . 245
12.3 Fréquentation d'un site Internet (2007-2008) . . . 247
12.4 Ajout d'une pièce de monnaie (2008-2009) . . . 250
12.5 Fusion de fichiers (2008-2009) . . . 255

*13. Exercices écrits* . . . 259
13.1 Premier énoncé (2005-2006) . . . 259
13.1.1 Programme à deviner * . . . 259
13.1.2 Somme des chiffres d'un nombre * . . . 259
13.1.3 Calculer le résultat d'un programme ** . . . 260
13.1.4 Suite récurrente (Fibonacci) * . . . 261
13.1.5 Comprendre une erreur d'exécution * . . . 262
13.1.6 Copie de variables ** . . . 263
13.1.7 Comprendre une erreur d'exécution ** . . . 264
13.1.8 Comprendre une erreur de logique ** . . . 265
13.1.9 Coût d'un algorithme ** . . . 265
13.1.10 Héritage ** . . . 267
13.1.11 Précision des calculs *** . . . 268
13.2 Second énoncé (2006-2007) . . . 271
13.2.1 Logarithme en base deux ** . . . 271
13.2.2 Calculer le résultat d'un programme ** . . . 271
13.2.3 Calculer le résultat d'un programme ** . . . 272
13.2.4 Comprendre une erreur d'exécution * . . . 273

13.2.5 Comprendre une erreur de logique *** . . . 274
13.2.6 Récursivité * . . . 276
13.2.7 Compléter un programme ** . . . 276
13.2.8 Comprendre une erreur de logique *** . . . 277
13.3 Troisème énoncé (2007-2008) . . . 279
13.3.1 Calcul d'un arrondi * . . . 279
13.3.2 Même programme avec 1,2,3 boucles ** . . . 279
13.3.3 Suite récurrente (Fibonacci) ** . . . 280
13.3.4 Calculer le résultat d'un programme * . . . 281
13.3.5 Comprendre une erreur de logique ** . . . 283
13.3.6 Héritage ** . . . 284
13.3.7 Analyser un programme *** . . . 285
13.3.8 Comprendre une erreur de logique *** . . . 287
13.4 Quatrième énoncé (2007-2008) . . . 289
13.4.1 Suite récurrente (Fibonacci) * . . . 289
13.4.2 Calculer le résultat d'un programme * . . . 289
13.4.3 Calculer le résultat d'un programme * . . . 290
13.4.4 Ecrire un programme à partir d'un algorithme ** . . . 290
13.4.5 Comprendre une erreur d'exécution * . . . 291
13.4.6 Précision des calculs ** . . . 292
13.4.7 Analyser un programme ** . . . 293
13.5 Cinquième énoncé (2008-2009) . . . 295
13.5.1 Portée des variables . . . 295
13.5.2 Somme d'éléments aléatoires . . . 297
13.5.3 Coût d'une moyenne mobile centrée . . . 298
13.5.4 Deviner l'objectif d'une fonction . . . 300
13.5.5 Calcul du plus court chemin dans un graphe . . . 301
13.5.6 Héritage . . . 302
13.6 Exercices supplémentaires . . . 305
13.6.1 Position initiale d'un élément après un tri ** . . . 305
13.6.2 Comprendre une erreur de logique * . . . 306
13.6.3 Comprendre une erreur d'exécution * . . . 306
13.6.4 Précision des calculs *** . . . 307

*Neuvième partie III Résumé partique, outils d'aide, algorithmes à connaître* 309

*14. Instructions fréquentes en Python* . . . 311
14.1 Le langage . . . 311
14.2 Les variables . . . 311
14.2.1 Les types immuables . . . 312
14.2.2 Les types modifiables . . . 313
14.2.2.1 Liste . . . 314

14.2.2.2 Dictionnaire . . . 314
14.2.2.3 Tableaux . . . 315
14.3 Tests et boucles . . . 315
14.3.1 Tests . . . 315
14.3.2 Boucles . . . 316
14.4 Fonctions . . . 318
14.5 Classes . . . 318
14.5.1 Déclaration d'une classe . . . 319
14.5.2 Attributs et méthodes . . . 319
14.5.3 Opérateurs . . . 320
14.5.4 Copie d'instances . . . 320
14.5.5 Héritage . . . 321
14.6 Fichiers . . . 321
14.6.1 Ecriture dans un fichier texte . . . 322
14.6.2 Lecture dans un fichier texte . . . 322
14.7 Modules . . . 323
14.8 Exceptions . . . 324
14.9 Erreurs, confusions fréquentes . . . 326
14.9.1 Variables . . . 326
14.9.1.1 Chaîne de caractères = tableau de caractères . . . 326
14.9.1.2 Guillemets ou pas . . . 326
14.9.2 Boucles . . . 326
14.9.2.1 `range`ou pas `range` . . . 326
14.9.2.2 Initialisation . . . 327
14.9.3 Fonctions . . . 327
14.9.3.1 Différence entre `print`et `return` . . . 327

*15. Outils d'aide au développement* . . . 328
15.1 Modules d'aide au développement . . . 328
15.1.1 Temps d'exécution . . . 328
15.1.2 Tests systématiques avec `doctest` . . . 329
15.1.3 Tests unitaires avec `unittest` . . . 331
15.1.4 Ecrire des fichiers de log . . . 334
15.1.5 Lancer un programme *Python* en ligne de commande . . . 334
15.1.6 Lancer un programme *Python* en ligne de commande depuis un programme *Python* 336
15.2 Tortoise SVN : suivi de version . . . 336
15.2.1 Utilisation personnelle . . . 336
15.2.1.1 Installation et création d'un repository . . . 337
15.2.1.2 Utilisation . . . 337
15.2.1.3 Configuration . . . 338
15.2.1.4 Plusieurs copies du repository . . . 339
15.2.2 Utilisation à plusieurs . . . 340

15.2.2.1 Installation sur la machine contenant le repository . . . 340
15.2.2.2 Installation sur les autres machines . . . 341
15.2.2.3 Backup . . . 342
15.2.2.4 Via internet . . . 342
15.2.2.5 Limitations . . . 342
15.2.2.6 Module `pysvn` . . . 342
15.3 HTML Help Workshop : construction d'un fichier d'aide . . . 342
15.3.1 Création automatique d'une documentation avec `pydoc` . . . 343
15.3.2 Compilation des pages HTML avec *HTML Help Workshop* . . . 345
15.3.3 Prolongements . . . 349
15.4 InnoSetup : installateur . . . 349
15.4.1 Sections *InnoSetup* . . . 349
15.4.1.1 section [`Setup`] . . . 349
15.4.1.2 section [`Files`] . . . 350
15.4.1.3 section [`Run`] . . . 350
15.4.1.4 section [`Icons`] . . . 350
15.4.1.5 section [`Registry`] . . . 351
15.4.1.6 section [`Code`] . . . 351
15.4.2 Examples . . . 351
15.4.2.1 Exemple 1 : installateur pour un module *Python* . . . 351
15.4.2.2 Exemple 2 : mise à jour du module . . . 353
15.4.3 Quelques trucs en plus . . . 354
15.4.3.1 Créer un raccourci . . . 354
15.4.3.2 Executer des tâches plus compliquées . . . 355
15.5 MoinMoin : wiki . . . 356
15.5.1 Installation . . . 356
15.5.2 Configuration d'*Apache* . . . 357
15.5.3 Configuration de *MoinMoin* . . . 357
15.5.3.1 Fichier `moin.cgi` . . . 357
15.5.3.2 Fichier `wikiconfig.py` . . . 358
15.5.3.3 Sécurité, création d'utilisateurs . . . 359
15.5.3.4 Apparence . . . 360
15.5.4 Utilisation sur un réseau d'entreprise . . . 360
15.5.5 Ecrire des pages . . . 360
15.5.5.1 Création d'une page . . . 360
15.5.5.2 Ajouter des liens . . . 361
15.5.5.3 Ajouter des pièces jointes . . . 362
15.5.5.4 Droits spécifiques à une page . . . 362
15.5.6 Script CGI . . . 362
15.5.6.1 Script simple . . . 362
15.5.6.2 Script HTML . . . 363
15.5.6.3 Attraper les exceptions . . . 363

15.5.6.4 Résumé de la syntaxe HTML . . . 364
15.5.6.5 Appeler une page avec des paramètres . . . 364
15.5.6.6 Formulaires . . . 366
15.6 SQL . . . 367
15.6.1 Les bases de données . . . 367
15.6.2 Utiliser une base de données . . . 368
15.6.3 Utiliser un serveur de base de données . . . 371
15.6.3.1 Installer un serveur de base de données sous *Windows* . . . 371
15.6.3.2 Configurer une connexion ODBC sous *Windows* . . . 373
15.6.3.3 Utilisation d'un serveur de base de données . . . 373
15.6.3.4 ODBC et script CGI . . . 375
15.6.3.5 Sans ODBC . . . 376
15.7 Modules divers . . . 378
15.7.1 PDF . . . 378
15.7.2 Tesseract . . . 380
15.8 Graphisme . . . 380
15.8.1 GNUPlot . . . 380
15.8.2 MatPlotLib . . . 381
15.9 Apache : serveur . . . 383
15.9.1 Installation . . . 383
15.9.2 Configuration de *Apache* pour *MoinMoin* . . . 384
15.9.3 Configuration pour des applications CGI . . . 386

*16. Algorithmes usuels* . . . 387
16.1 Calcul d'une intégrale . . . 387
16.1.1 Description de l'objectif . . . 387
16.1.2 Différents programmes . . . 387
16.2 Tri simple et tri fusion . . . 388
16.2.1 Trier une liste avec *Python* . . . 388
16.2.2 Tri par sélection . . . 389
16.2.3 Tri par fusion . . . 389
16.2.4 Tri par insertion . . . 390
16.2.5 Démonstration de l'algorithme de tri par fusion . . . 391
16.2.6 A propos de coût d'algorithme . . . 392
16.2.7 Coût minimal d'un tri . . . 393
16.2.8 Tri d'entiers . . . 393
16.2.9 Programme du tri par sélection . . . 394
16.2.10 Programme du tri par fusion . . . 395
16.2.11 Programme du tri par insertion dichotomique . . . 396
16.2.12 Programme du tri d'entiers . . . 396

*Neuvième partie IV Visual Basic For Applications (VBA)* 398

*17. Introduction* . . . 399
17.1 Visual Basic et macros . . . 399
17.2 Première procédure . . . 399
17.3 Macro liée à une feuille Excel . . . 400
17.4 Erreurs . . . 401
17.5 Autres possibillités . . . 402

*18. Le langage VBA* . . . 404
18.1 Préliminaires . . . 404
18.2 Variables et opérateurs . . . 404
18.2.1 Types prédéfinis . . . 404
18.2.2 Opérateur . . . 405
18.2.3 Conversion . . . 406
18.2.4 Types définis par l'utilisation . . . 406
18.2.5 Classes . . . 407
18.3 Affichage . . . 408
18.4 Tests . . . 409
18.4.1 Tests avec `If` . . . 409
18.4.2 Tests avec `Select Case` . . . 409
18.5 Boucles . . . 410
18.5.1 Boucle `For` . . . 410
18.5.2 Boucle `While` . . . 410
18.5.3 Boucle `Loop` . . . 410
18.6 Fonctions et procédures . . . 411
18.6.1 Procédures . . . 411
18.6.2 Fonctions . . . 411
18.6.3 Variables statiques . . . 411
18.6.4 Passages des paramètres . . . 411
18.6.5 Surnom : `Set` . . . 412
18.6.6 Fonctions couramment utilisées . . . 412
18.7 Erreurs . . . 412
18.8 Fichiers . . . 413
18.8.1 Ecriture . . . 413
18.8.2 Lecture . . . 413

*19. Exercices autour de l'automatisation de tâches quotidiennes* . . . 415
19.1 Création de PDF puis envoi automatique de mails . . . 415
19.1.1 Enoncé . . . 415
19.1.2 Correction . . . 420
19.2 Création d'une présentation à partir de photos de vacances . . . 424
19.2.1 Enoncé . . . 424

19.2.2 Correction . . . . . 429
19.3 Résolution d'un sudoku . . . . . 432
19.3.1 Enoncé . . . . . 432
19.3.2 Correction . . . . . 437

*20. VBA et Python* . . . . . 443
20.1 *Excel* et *Python* . . . . . 443
20.1.1 Version simple . . . . . 443
20.1.2 Version améliorée . . . . . 444
20.2 *Python*, valeurs propres . . . . . 446
20.2.1 Sans *Excel* . . . . . 446
20.2.2 Avec *Excel* . . . . . 446
20.3 Des graphes complexes avec *R* sous *Excel* . . . . . 449
20.3.1 Graphes avec *R* seul . . . . . 449
20.3.2 *R* avec *Python* . . . . . 451
20.3.3 *R* avec *Python* et *Excel* . . . . . 452
20.3.4 *R* et *Excel* . . . . . 452

*Neuvième partie V Exemples variés* 455

*21. Périphériques* . . . . . 457
21.1 Copie de fichiers vers une clé USB . . . . . 457
21.1.1 Enoncé . . . . . 457
21.1.2 Correction . . . . . 459
21.2 Recherche sur Internet . . . . . 461
21.2.1 Enoncé . . . . . 461
21.2.2 Correction . . . . . 463
21.3 Python, Excel, Latex . . . . . 471
21.3.1 Enoncé . . . . . 471
21.3.2 Correction . . . . . 471
21.4 Fond d'écran . . . . . 471
21.4.1 Enoncé . . . . . 471
21.4.2 Correction . . . . . 471

*22. Logique et jeux* . . . . . 474
22.1 Résolution d'un sudoku . . . . . 474
22.1.1 Enoncé . . . . . 474
22.1.2 Correction . . . . . 474
22.2 Résolution d'un puzzle . . . . . 479
22.2.1 Description des données . . . . . 480
22.2.1.1 classe `Bord` . . . . . 480
22.2.1.2 classe `Piece` . . . . . 481
22.2.1.3 classe `Puzzle` . . . . . 482

22.2.2 Résolution . . . 484
22.2.2.1 Permutations . . . 484
22.2.2.2 Application au puzzle . . . 485
22.2.3 Prolongements . . . 486
22.2.4 Programme final . . . 486
22.3 Résolution d'une énigme . . . 494
22.3.1 Enoncé . . . 494
22.3.2 Correction . . . 494
22.3.3 Enoncé . . . 498
22.3.4 Correction . . . 499
22.4 Simulation informatique . . . 501
22.4.1 Enoncé . . . 501
22.4.2 Correction . . . 502
22.4.3 Comment observer un problème à l'aide d'un programme informatique . . . 502
22.4.4 Chaînes de Markov . . . 506
22.4.5 Le programme . . . 508

*23. Mathématique, recherche opérationnelle* . . . 513
23.1 Plus court chemin dans un graphe . . . 513
23.1.1 Enoncé . . . 513
23.1.2 Correction . . . 517
23.1.3 Théorie des graphes, un peu d'histoire . . . 517
23.1.4 Principe d'optimalité . . . 518
23.1.5 Plus court chemin dans un graphe . . . 518
23.1.6 Coût de l'algorithme . . . 520
23.1.7 A propos des graphes . . . 521
23.1.8 Programme . . . 521
23.1.9 Distance d'édition . . . 525
23.2 Problème du voyageur de commerce . . . 527
23.2.1 Enoncé . . . 527
23.2.2 Cartes de Kohonen . . . 528
23.2.3 Arbre de poids minimal et circuit hamiltonien . . . 531
23.2.4 Arbre de poids minimum . . . 531
23.2.5 Circuit eulérien . . . 532
23.2.6 Circuit hamiltonien . . . 534
23.2.7 Simplification du circuit hamiltonien . . . 535
23.2.8 Correction . . . 538
23.2.9 Carte de Kohonen . . . 538
23.2.10 Construction à partir d'un arbre de poids minimum . . . 543
23.3 Classification à l'aide de plus proches voisins . . . 557
23.3.1 Enoncé . . . 557
23.3.2 Principe . . . 557

23.3.3 B+ tree . . . 558
23.3.4 R-tree ou Rectangular Tree . . . 559
23.3.5 LAESA . . . 561
23.3.6 Résultats théoriques . . . 563
23.3.7 Correction . . . 564
23.3.8 Programmes annexes . . . 564
23.3.9 Programme de classification . . . 570
23.3.10 Optimisation . . . 574
23.4 Optimisation numérique du second ordre . . . 577
23.4.1 Enoncé . . . 577
23.4.2 Algorithme et convergence . . . 577
23.4.3 Méthode du second ordre . . . 580
23.4.4 Minimisation avec gradient stochastique . . . 582
23.4.5 Premier exemple : optimisation d'une fonction quadratique . . . 584
23.4.6 Court rappel sur l'estimateur du maximum de vraisemblance . . . 584
23.4.7 Second exemple : proportion de mâles chez les poissons . . . 588
23.4.8 Correction . . . 589
23.4.9 Premier exemple, fonction quadratique . . . 589
23.4.10 Second exemple, proportion de mâles et de femelles chez les poissons . . . 592
23.4.11 Simulation d'une série $(X_i)_i$ . . . 592
23.4.12 Estimation des paramètres avec une méthode du premier degré . . . 593
23.4.13 Estimation des paramètres avec une méthode du second degré . . . 597
23.5 Optimiser de coûts en environnement aléatoire . . . 599
23.5.1 Un problème simple . . . 599
23.5.2 Modélisation de la demande . . . 602
23.5.3 Variations saisonnières . . . 606
23.5.4 Prolongations . . . 607
23.6 Stratégie automatique de trading en finance . . . 607
23.6.1 Domaines financiers . . . 608
23.6.1.1 Marchés liquides . . . 608
23.6.1.2 Description d'une série financière . . . 608
23.6.1.3 Rendements, Volatilité, Corrélation . . . 609
23.6.1.4 Moyenne mobile, bandes de Bollinger . . . 611
23.6.1.5 Achats, ventes, levier . . . 611
23.6.1.6 Hedge Funds . . . 611
23.6.2 Familles de stratégies . . . 614
23.6.2.1 Trend Following . . . 614
23.6.2.2 Mean Reversing . . . 615
23.6.2.3 Pair Trading . . . 615
23.6.2.4 Styles, Value, Growth . . . 617
23.6.2.5 Gestion de portefeuille . . . 618
23.6.2.6 Horizon de trading, intraday, daily . . . 619

23.6.3 Analyse d'une stratégie . . . 620
23.6.3.1 Backtest . . . 620
23.6.3.2 Modélisation . . . 622
23.6.3.3 Résultats standards . . . 623
23.6.3.4 Variation de paramètres . . . 623
23.6.3.5 Autres séries financières de même nature . . . 624
23.6.3.6 Décomposition en deals . . . 625
23.6.3.7 Pour aller plus loin . . . 626
23.6.4 Diversification . . . 627
23.6.5 Conclusion . . . 628
23.6.6 Morceaux informatiques . . . 628
23.6.6.1 Dessin d'une série financière . . . 628

*Neuvième partie VI Exercices variés* 631

*24. Sujets informatiques* . . . 632
24.1 Dictionnaires (1) . . . 632
24.1.1 Enoncé . . . 632
24.1.2 Correction . . . 632
24.2 Dictionnaires (2) . . . 634
24.2.1 Enoncé . . . 634
24.2.2 Correction . . . 634
24.3 Popularité selon différents moteurs de recherce . . . 635
24.3.1 Enoncé . . . 635
24.3.2 Correction . . . 636
24.4 Tri simple et tri fusion . . . 641
24.4.1 Enoncé . . . 641
24.4.2 Correction . . . 641
24.5 Equation différentielle . . . 643
24.5.1 Enoncé . . . 643
24.5.2 Correction . . . 644

*25. Sujets autour des interfaces graphiques* . . . 647
25.1 Interface graphique . . . 647
25.1.1 Enoncé . . . 647
25.1.2 Correction . . . 648
25.1.3 Enoncé . . . 650
25.1.4 Correction . . . 652
25.2 Jeu avec *PyGame* . . . 653
25.2.1 Enoncé . . . 653
25.2.2 Correction . . . 656

*26. Sujets mathématiques, statistiques* . . . 660

26.1 Casser le code de Vigenère . . . 660
26.1.1 Enoncé . . . 660
26.1.2 Correction . . . 662
26.2 Disparition des gens aux yeux bleus . . . 666
26.2.1 Enoncé . . . 666
26.2.2 Correction . . . 668
26.3 Calculer des espérances de vie . . . 670
26.3.1 Enoncé . . . 670
26.3.2 Correction . . . 672
26.4 Trouver une personne ayant les mêmes goûts que vous . . . 679
26.4.1 Enoncé . . . 679
26.4.2 Correction . . . 682
26.5 Calcul d'un histogramme . . . 683
26.5.1 Enoncé . . . 683
26.5.2 Correction . . . 685

*Neuvième partie VII Domaine hors sujets* 691

*27. Sujets mathématiques* . . . 692
27.1 File d'attente . . . 692
27.1.1 Enoncé . . . 692
27.1.2 Correction . . . 692
27.1.2.1 File d'attente, cas M/M/1 . . . 693
27.1.2.2 File d'attente, cas M/M/S . . . 695
27.1.2.3 Retour aux ampoules . . . 696
27.1.2.4 Programme informatique . . . 698
27.2 Simulation génétique . . . 699
27.2.1 Enoncé . . . 699
27.2.2 Correction . . . 700
27.3 Tracer une ligne sans multiplication . . . 701
27.3.1 Enoncé . . . 701
27.3.1.1 L'algorithme de Bresenham . . . 701
27.3.1.2 Extension aux ellipses . . . 703
27.3.1.3 Extension aux coniques . . . 707
27.3.2 Correction . . . 708
27.4 Simulation d'une corde . . . 710
27.4.1 Enoncé . . . 710
27.4.1.1 Equation théorique de la corde en équilibre . . . 710
27.4.1.2 Solution dans un cas particulier . . . 712
27.4.1.3 Simulation informatique . . . 712
27.4.2 Correction . . . 714

*28. Inclassables* . . . 719
28.1 Filtrage du signal sonore . . . 719
28.1.1 Enoncé . . . 719
28.1.1.1 Transformée de Fourier discrète . . . 719
28.1.1.2 Transformée de Fourier rapide . . . 719
28.1.1.3 Filtrage . . . 720
28.1.1.4 Horizons . . . 721
28.1.2 Correction . . . 721
28.2 Images de synthèse, modèle d'illumination . . . 724
28.2.1 Enoncé . . . 724
28.2.1.1 La scène . . . 724
28.2.1.2 Modèle d'illumination . . . 726
28.2.1.3 Géométrie et sphère . . . 727
28.2.1.4 Géométrie et facettes . . . 728
28.2.1.5 Lancer de rayon . . . 730
28.2.1.6 Antialiasing . . . 732
28.2.1.7 Modèle d'illumination de Phong . . . 733
28.2.1.8 Interpolation de Gouraud . . . 733
28.2.1.9 Horizons . . . 734
28.2.2 Correction . . . 735
28.2.2.1 Fichier : `image_synthese_base.py` . . . 735
28.2.2.2 Fichier : `image_synthese_sphere.py` . . . 739
28.2.2.3 Fichier : `image_synthese_scene.py` . . . 741
28.2.2.4 Fichier : `image_synthese_phong.py` . . . 743
28.2.2.5 Fichier : `image_synthese_facette.py` . . . 745

*Neuvième partie VIII Exercices autour de sujets déjà abordés* 749

*29. Exercices plutôt mathématiques* . . . 750
29.1 Tracé d'une ligne sans multiplication . . . 750
29.1.1 Enoncé . . . 750
29.1.2 Correction . . . 753
29.2 Un portrait lointain à l'aide du TSP . . . 757
29.2.1 Enoncé . . . 757
29.2.2 Correction . . . 760
29.3 Image de synthèse . . . 775
29.3.1 Enoncé . . . 775
29.3.2 Correction . . . 789

*30. Exercices plutôt informatiques* . . . 792
30.1 Conserver ses fichiers sur une clé USB avec une interface graphique . . . 792
30.1.1 Enoncé . . . 792

30.1.2 Correction . . . . . 799
30.2 Conserver ses fichiers sur une clé USB avec une interface graphique (2) . . . . . 800
30.2.1 Enoncé . . . . . 800
30.2.2 Correction . . . . . 801

*Neuvième partie IX Bibliographie, index, remerciements* 804

*Index* 808