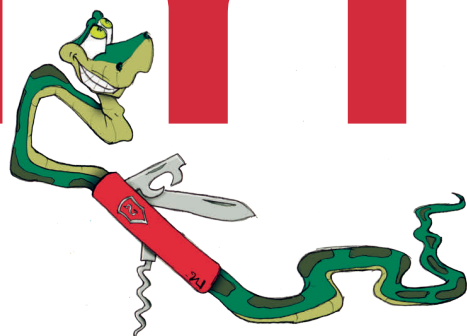


Bob CORDEAU – Laurent POINTAL

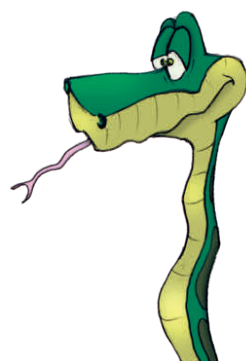
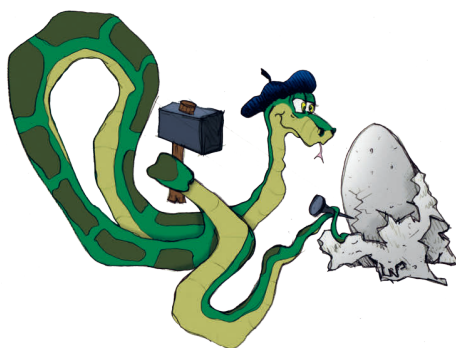
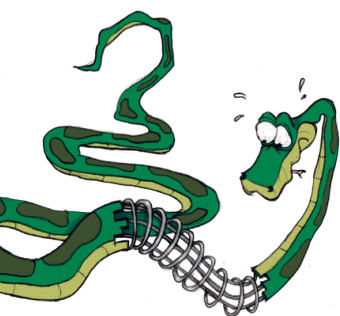
PYTHON



3

Apprendre à programmer
dans l'écosystème Python

2^e édition



DUNOD

Toutes les marques citées dans cet ouvrage
sont des marques déposées par leurs propriétaires respectifs.

Illustrations de couverture et intérieures :
© Hélène Cordeau

© Dunod, 2017, 2018, 2026 pour la présente édition
11, rue Paul Bert, 92240 Malakoff

www.dunod.com

ISBN 978-2-10-089441-3

Préface

Professeur de sciences désormais retraité de l'enseignement secondaire belge, je fus de ces aventuriers qui se lancèrent à la découverte des premiers micro-ordinateurs *grand public* à la fin des années 1970. Il fallait être un peu fou, à cette époque, pour investir des sommes plutôt rondellettes dans ces machines bricolées, au comportement assez capricieux, dont on fantasmait de tirer tôt ou tard des applications extraordinaires, mais souvent sans trop savoir au juste lesquelles, et encore moins comment on pourrait y arriver.

Il n'était évidemment pas question d'Internet en ce temps-là. Trouver de la documentation était une tâche ardue. Les rares documents que l'on parvenait à trouver (*via* les clubs de radio-amateurs, principalement) traitaient davantage d'électronique que de programmation. Et c'était bien nécessaire, il valait mieux être capable de manier le fer à souder ou de trouver l'un ou l'autre copain technicien dans un laboratoire disposant d'un programmeur d'EPROM¹.

C'est dans ce contexte que je découvris en autodidacte (inutile de dire qu'aucune formation n'était encore organisée à l'époque) mes premiers langages de programmation. Sur le TRS-80 de mes débuts, on disposait seulement d'un Basic sommaire et d'un Assembleur. Il fallait s'accrocher. La mise au point d'un tout petit programme pouvait prendre des heures, et même sa sauvegarde (sur cassette à bande magnétique !) pouvait se révéler problématique. Pas question en tout cas d'imaginer une seule seconde enseigner ce genre de choses à mes jeunes élèves.

Mon activité de programmation en ces années-là se focalisa alors sur le développement de simulations expérimentales. Sur le modèle anglo-saxon des années 1960, je souhaitais centrer mon enseignement scientifique sur la découverte et l'investigation personnelle des élèves, et j'organisais donc un maximum de séances de travaux pratiques. La simulation me permettait d'étendre cette méthodologie à des expérimentations cruciales pour la compréhension de principes fondamentaux (en physique ou en biologie, par exemple), mais irréalisables dans le cadre scolaire ordinaire pour des raisons diverses. Avec un programme de simulation d'expérience bien conçu, l'élève peut se trouver plongé dans une situation de travail très proche de celle d'un laboratoire. Je trouvais particulièrement intéressante, sur le plan pédagogique, l'idée qu'en procédant de la sorte j'instaurais pour l'étudiant un véritable droit à l'erreur : en simulation, il peut en effet décider lui-même sa stratégie expérimentale, procéder par tâtonnements, se tromper, recommencer éventuellement un grand nombre de fois ses tentatives, sans qu'il en résulte un coût excessif en temps ou en ressources matérielles.

Je progressais ainsi dans ma connaissance de la programmation, sans aucune intention de l'enseigner un jour, en m'adaptant au fil des années aux évolutions du matériel et des langages, jusqu'à ce jour de 1998 où l'on me demanda de participer à l'élaboration de cursus pour une nouvelle filière d'enseignement secondaire qui serait centrée sur l'apprentissage de l'informatique.

1. La mémoire EPROM (*Erasable Programmable Read-Only Memory*) est un type de mémoire morte reprogrammable. Pour effectuer cette (re)programmation, il faut en général retirer l'EPROM de son support et la placer dans un appareil dédié à cet effet.

Mon expérience et mes contacts m’avaient entre-temps fait prendre conscience de la problématique de la liberté logicielle. J’étais opportunément en train de découvrir l’une des premières distributions *crédibles* de Linux (l’une des premières Red Hat), et je me suis immédiatement persuadé que si l’on voulait effectivement inculquer une saine compréhension de ce que sont l’informatique et ses enjeux à des étudiants aussi jeunes, on se devait de le faire sur la base de logiciels libres.

L’un des cours à mettre en place devait être une initiation à la programmation. J’avais une certaine expérience en la matière, et c’était pour cela qu’on sollicitait mon avis, mais tous les outils que j’avais utilisés personnellement jusque-là étaient des langages propriétaires (Basic, Delphi, Clarion...), et je ne voulais être le démarcheur d’aucun d’entre eux. C’est donc dans cet esprit que je me suis mis à la recherche de ce que je craignais être la quadrature du cercle : un langage de programmation qui soit à la fois libre, multi-plateformes, polyvalent, assez facile à apprendre, avec lequel il soit possible d’aborder un maximum de concepts, tant sur les paradigmes de programmation que sur les structures de données, qui soit surtout de haut niveau et très lisible (je m’imaginais à l’avance le casse-tête que constituerait pour les professeurs le travail de correction d’un programme mal écrit par un élève à l’esprit tordu, dans un langage proche de la machine et à la syntaxe alambiquée...).

Le miracle a eu lieu : j’ai découvert Python. Ses qualités sont décrites dans les pages qui suivent. Restait le problème de l’enseigner à des jeunes de 16-18 ans. En l’occurrence je souhaitais aussi valider autant que possible la stratégie pédagogique d’apprentissage par investigation libre que j’avais développée pour mes cours de sciences, et aucun cours de programmation satisfaisant à mes critères n’existait à l’époque, du moins en français. L’essentiel de la documentation de Python lui-même n’existait d’ailleurs qu’en anglais (on en était à la version 1.5).

Je me suis donc lancé le défi – encore une fois un peu fou – de rédiger mon propre manuel de cours. La suite est connue : bien conscient de mes limitations d’autodidacte, j’ai tout de suite mis mes notes à la disposition de tout le monde sur l’Internet, et j’ai ainsi pu récolter de nombreux avis et conseils, grâce auxquels le texte s’est amélioré au fil du temps et a fini par paraître aussi en version imprimée, distribuée en librairies.

C’est au cours de cette saga que j’ai eu la chance de faire la connaissance de Bob Cordeau, qui m’a gentiment rendu le service de relire mes 430 pages pour y débusquer coquilles et étourderies. Au cours de cet important travail, il a donc eu tout le loisir de constater tous les défauts de mon texte : imprécisions diverses, structuration fantaisiste, concepts omis ou traités de manière triviale...

Il ne m’en a rien dit pour ne pas me faire de la peine, mais il s’est courageusement mis à l’ouvrage pour rédiger son propre texte, que vous aurez le plaisir de découvrir dans les pages qui suivent. Là où je m’étais contenté d’une ébauche brouillonne, Bob et Laurent ont réalisé un vrai travail de « pro » : un des meilleurs textes de référence sur ce merveilleux outil qu’est Python.

Bonne lecture, donc.

Gérard Swinnen¹

1. Auteur d’*Apprendre à programmer avec Python 3*, paru aux éditions Eyrolles, et disponible également en téléchargement libre (<https://infocref.be/swi/python.htm>).

Table des matières

Préface	v
Avant-propos	xiii
1 Programmer en Python	1
1.1 Mais pourquoi donc apprendre à programmer ?	1
1.1.1 Un exemple pratique	2
1.1.2 Et après ?	5
1.2 Mais pourquoi donc apprendre Python ?	6
1.2.1 Principales caractéristiques du langage Python	6
1.2.2 Implémentations de Python	7
1.3 Comment passer du problème au programme	8
1.3.1 Réutiliser	8
1.3.2 Réfléchir à un algorithme	8
1.3.3 Résoudre « à la main »	9
1.3.4 Formaliser	9
1.3.5 Factoriser	9
1.3.6 Passer de l'idée au programme	10
1.4 Techniques de production des programmes	10
1.4.1 Technique de production de Python	11
1.4.2 Construction des programmes	11
1.5 Résumé et thèmes de réflexion	12
2 La calculatrice Python	13
2.1 Modes d'exécution d'un code Python	13
2.2 Identificateurs et mots-clés	14
2.2.1 Identificateurs	14
2.2.2 Mots-clés de Python 3	14
2.2.3 PEP 8 : une affaire de style	14
2.2.4 Nommage des identificateurs	15
2.3 Notion d'expression	15
2.4 Variable et objet	16
2.4.1 Affectation	16
2.4.2 Réaffectation et typage dynamique	17
2.4.3 Attention : affecter n'est pas comparer !	18
2.4.4 Variantes de l'affectation	18

2.4.5	Suppression d'une variable	19
2.4.6	Énumérations	19
2.5	Types de données entiers	21
2.5.1	Type <code>int</code>	21
2.5.2	Type <code>bool</code>	22
2.6	Types de données flottants	23
2.6.1	Type <code>float</code>	24
2.6.2	Type <code>complex</code>	24
2.7	Chaînes de caractères	25
2.7.1	Présentation	25
2.7.2	Séquences d'échappement	25
2.7.3	Opérations	26
2.7.4	Fonctions vs méthodes	26
2.7.5	Méthodes de test de l'état d'une chaîne	27
2.7.6	Méthodes retournant une nouvelle chaîne	27
2.7.7	Méthode retournant un index	27
2.7.8	Indexation simple	28
2.7.9	<i>Slicing</i>	28
2.7.10	Formatage de chaînes	29
2.8	Types binaires	34
2.9	Entrées-sorties de base	35
2.10	Comment trouver une documentation	36
2.11	Résumé et exercices	37
3	Contrôle du flux d'instructions	39
3.1	Indentation significative et instructions composées	39
3.2	Choisir	40
3.2.1	Choisir : <code>if</code> - <code>[elif]</code> - <code>[else]</code>	40
3.2.2	Syntaxe compacte d'une alternative	41
3.3	Boucles	41
3.3.1	Parcourir : <code>for</code>	42
3.3.2	Répéter sous condition : <code>while</code>	42
3.4	Ruptures de séquences	43
3.4.1	Interrompre une boucle : <code>break</code>	43
3.4.2	Court-circuiter une boucle : <code>continue</code>	43
3.4.3	Traitement des erreurs : les exceptions	44
3.5	Résumé et exercices	45
4	Conteneurs standard	49
4.1	Séquences	49
4.2	Listes	50
4.2.1	Définition, syntaxe et exemples	50
4.2.2	Initialisations, longueur de la liste et tests d'appartenance	51
4.2.3	Méthodes modificatrices	51
4.2.4	Manipulation des index et des slices	52
4.3	Tuples	52
4.4	Séquences de séquences	53

4.5	Retour sur les références	53
4.5.1	Les références partagées des objets immutables	54
4.5.2	Les références partagées des objets mutables	54
4.5.3	L'affectation augmentée	55
4.6	Tables de hash	57
4.7	Dictionnaires	59
4.8	Ensembles	60
4.9	Itérer sur les conteneurs	62
4.10	Résumé et exercices	63
5	Fonctions et espaces de nommage	65
5.1	Définition et syntaxe	65
5.2	Passage des arguments	67
5.2.1	Mécanisme général	67
5.2.2	Un ou plusieurs paramètres positionnels, pas de retour	67
5.2.3	Un ou plusieurs paramètres positionnels, un ou plusieurs retours	68
5.2.4	Appel avec des arguments nommés	69
5.2.5	Paramètres avec valeur par défaut	70
5.2.6	Nombre d'arguments arbitraire : passage d'un tuple de valeurs	70
5.2.7	Nombre d'arguments arbitraire : passage d'un dictionnaire	71
5.2.8	Argument mutable	71
5.3	Espaces de nommage	72
5.3.1	Portée des objets	73
5.3.2	Résolution des noms : règle « LEGB »	73
5.4	Résumé et exercices	74
6	Modules et packages	77
6.1	Modules	77
6.1.1	Imports	78
6.1.2	Localisation des fichiers modules	79
6.1.3	Emplois et chargements des modules	80
6.2	Packages	85
6.3	Résumé et exercices	87
7	Accès aux données	89
7.1	Fichiers	89
7.1.1	Gestion des fichiers	90
7.1.2	Ouverture et fermeture des fichiers en mode texte	91
7.1.3	Écriture séquentielle	91
7.1.4	Lecture séquentielle	92
7.1.5	Gestionnaire de contexte with	92
7.1.6	Fichiers binaires	93
7.2	Travailler avec des fichiers et des répertoires	93
7.2.1	Se positionner dans l'arborescence	93
7.2.2	Construction de noms de chemins	94
7.2.3	Opérations sur les noms de chemins	94
7.2.4	Gestion des répertoires	94

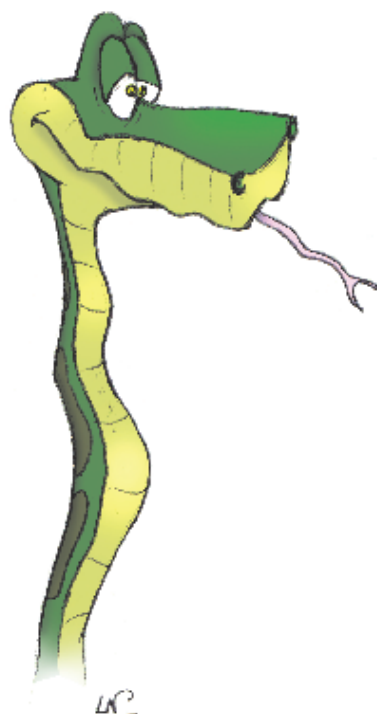
7.3	Sérialisation avec <code>pickle</code> et <code>json</code>	95
7.4	Bases de données relationnelles	96
7.4.1	Comprendre le langage SQL	96
7.4.2	Utiliser SQL en Python avec <code>sqlite3</code>	97
7.5	Micro-serveur web	106
7.5.1	Internet	106
7.5.2	Web	108
7.5.3	Un serveur web en Python	108
7.6	Résumé et exercices	113
8	Programmation orientée objet	115
8.1	Origine et évolution	115
8.2	Terminologie	116
8.3	Définition des classes et des instanciations d'objets	117
8.3.1	Instruction <code>class</code>	117
8.3.2	L'instanciation et ses attributs, le constructeur	118
8.3.3	Retour sur les espaces de noms	120
8.4	Méthodes	121
8.5	Méthodes spéciales	122
8.5.1	Surcharge des opérateurs	123
8.5.2	Exemple de surcharge	123
8.6	Héritage et polymorphisme	124
8.6.1	Formalisme de l'héritage et du polymorphisme	124
8.6.2	Exemple d'héritage et de polymorphisme	126
8.7	Notion de « conception orientée objet »	127
8.7.1	Relation, association	127
8.7.2	Dérivation	128
8.8	Résumé et exercices	129
9	La programmation graphique orientée objet	131
9.1	Programmes pilotés par des événements	131
9.2	Bibliothèque <code>tkinter</code>	131
9.2.1	Présentation	131
9.2.2	Les widgets de <code>tkinter</code>	133
9.2.3	Positionnement des widgets	133
9.3	Deux exemples	134
9.3.1	Une calculatrice	134
9.3.2	<code>tkPhone</code>	134
9.4	Résumé et exercices	141
10	Programmation avancée	143
10.1	Techniques procédurales	143
10.1.1	Pouvoir de l'introspection	143
10.1.2	Utiliser un dictionnaire pour déclencher des fonctions ou des méthodes	145
10.1.3	Listes, dictionnaires et ensembles définis en compréhension	146
10.1.4	Générateurs et expressions génératrices	148
10.1.5	Décorateurs	149

10.2	Techniques objets	151
10.2.1	<i>Functors</i>	151
10.2.2	Accesseurs	152
10.2.3	<i>Duck typing</i>	155
10.2.4	<i>Duck typing</i> ... et annotations de types	157
10.3	Algorithmique	158
10.3.1	Directive <code>lambda</code>	158
10.3.2	Fonctions incluses et fermetures	159
10.3.3	Techniques fonctionnelle : fonctions <code>map</code> , <code>filter</code> et <code>reduce</code>	160
10.3.4	Programmation fonctionnelle <i>pure</i>	162
10.3.5	Applications partielles de fonctions	163
10.3.6	Constructions algorithmiques de base	164
10.3.7	Fonctions récursives	172
10.4	Résumé et exercices	175
11	L'écosystème Python	177
11.1	<i>Batteries included</i>	177
11.1.1	Gestion des chaînes	177
11.1.2	Gestion de la ligne de commande	178
11.1.3	Gestion du temps et des dates	179
11.1.4	Algorithmes et types de données collection	179
11.2	L'écosystème Python scientifique	180
11.2.1	Bibliothèques mathématiques et types numériques	181
11.2.2	IPython, l'interpréteur scientifique	182
11.2.3	Bibliothèques NumPy, Pandas, matplotlib et scikit-image	184
11.3	Bibliothèques tierces	197
11.4	Documentation et tests	197
11.4.1	Documentation	197
11.4.2	Tests	198
11.5	Microcontrôleurs et objets connectés	200
11.6	Résumé et exercices	201
12	Solutions des exercices	203

Annexes

A	Interlude	221
B	Le codage des nombres et des caractères	223
C	Les expressions régulières	229
D	Les messages d'erreur de l'interpréteur	237
E	Résumé de la syntaxe	255

Bibliographie	265
Glossaire et lexique anglais/français	267
Index	281



Avant-propos

*En se partageant, le savoir ne se divise pas,
il se multiplie.*

À qui s'adresse ce livre ?

Issu d'un cours pour les étudiants du département « Mesures physiques » de l'IUT d'Orsay, ce livre s'adresse en premier lieu aux étudiants débutant en programmation, issus des IUT, des BTS, des licences pro et scientifiques, des écoles d'ingénieurs, aux élèves des classes préparatoires scientifiques, aux enseignants du secondaire (et peut-être à leurs élèves les plus motivés) et plus généralement à tout autodidacte désireux d'apprendre Python en tant que premier langage de programmation.

Le langage Python est préconisé pour l'apprentissage de la programmation par l'Éducation nationale pour les classes de lycée. La lecture du thème « Numérique et sciences informatiques » du Conseil supérieur des programmes introduit un vocabulaire et des concepts dont nous avons tenu compte (cf. l'index).

Prenant la programmation à la base, cet ouvrage se veut pédagogique sans cesser d'être pratique. D'une part en fournissant de très nombreux exemples, des exercices corrigés dans le texte et en ligne, et d'autre part en évitant d'être un catalogue exhaustif sur le langage d'apprentissage en offrant d'abord une introduction aux principaux concepts nécessaires à la programmation et en regroupant les éléments plus techniques liés au langage choisi dans les derniers chapitres.

Pour permettre néanmoins d'approfondir ces détails, il propose plusieurs moyens de navigation : une table des matières détaillée en début et, en annexe, des résumés syntaxiques et fonctionnels complets, un glossaire bilingue et un index. De plus, l'ouvrage offre, sur les pages intérieures de la couverture, le mémento Python 3.

Au-delà de l'apprentissage scolaire de la programmation grâce au langage Python, ce livre aborde des aspects souvent négligés, à savoir : le processus de réflexion utilisé dans la phase d'analyse préalable, la façon de passer de l'analyse à l'écriture du programme, de découper proprement celui-ci en fichiers modules réutilisables et ensuite d'utiliser les outils et techniques pour corriger les erreurs.

Nos choix

Cet ouvrage repose sur quelques partis pris :

- la version 3 du langage Python¹;
- le choix de logiciels libres² : la distribution Python scientifique Pyzo, miniconda3, Jupyter Notebook et des outils *open source* de production de documents (X_Y-L^AT_EX);
- une introduction à la programmation, mais aussi à de nombreux à-côtés souvent oubliés et que l'on ne découvre qu'avec l'expérience.

Les exercices

Parmi les exercices proposés à la fin de chaque chapitre (exercices simples ✓, moins simples ✓✓, voire plus difficiles ✓✓✓) ceux marqués du logo 💡 sont corrigés en fin d'ouvrage. Tous les exercices du livre, ainsi que **125 exercices corrigés supplémentaires** au format *notebook*, accompagnent ce cours sur la page web dédiée à l'ouvrage (<https://www.dunod.com/EAN/9782100889112>) et sur le site de GitHub³ (<https://github.com/lpointal/appbclp>).

Présentation des codes

Un code Python *interprété* sera présenté sous la forme :

```
>>> len("abcde")      # Longueur (nombre de caractères dans la chaîne)
5
>>> "abc" + "defg"    # Concaténation (mise bout à bout de deux chaînes ou plus)
'abcdefg'
>>> "Fi! " * 3        # Répétition (avec * entre une chaîne et un entier)
'Fi! Fi! Fi! '
```

Un *script* complet ou un fragment de script Python sera présenté sous la forme :

```
# coding: utf8
""" Jeu de dés """

# Programme principal =====
n = int(input("Entrez un entier [2 .. 12] : "))
while not(n >= 2 and n <= 12):
    n = int(input("Entrez un entier [2 .. 12], s.v.p. : "))
s = 0
for i in range(1, 7):
    for j in range(1, 7):
        if i+j == n:
            s += 1
print(f"=> Il y a {s:d} façon(s) de faire {n:d} avec deux dés.")
```

Les commandes textuelles, résultats d'exécution ou fichiers texte seront présentés sous la forme :

Une commande ou un fichier "texte".

1. Version qui abolit la compatibilité descendante avec les versions antérieures. C'est une grave décision, mûrement réfléchie : « *Un langage qui bouge peu permet une industrie qui bouge beaucoup* » (Bertrand MEYER).

2. Voir la bibliographie (§ p. 266, § E).

3. On y trouvera également un errata.

Mise en lumière des éléments importants

Exemple d’encart de définition :

Définition

 Une **expression** est une portion de code que l’interpréteur Python peut **évaluer** pour obtenir une **valeur**. Les expressions peuvent être simples ou complexes.

Exemple d’encart de remarque :

Remarque

 Une « implémentation » signifie une « mise en œuvre ».

Exemple d’encart de syntaxe :

Syntaxe

 Les méthodes spéciales portent des noms prédéfinis, précédés et suivis de deux caractères de soulignement.

Exemple d’encart d’alerte :

Attention

 Toutes les instructions au même niveau d’indentation appartiennent au même bloc.

À propos de la deuxième édition

Cette deuxième édition présente plusieurs nouveautés :

- elle utilise Python 3.12.4 publié en juin 2024 ;
- à la fin de chaque chapitre le lecteur trouvera :
 - un résumé des nouveaux acquis sous la forme d’un encadré « Ce Qu’il Faut Retenir »,
 - des exercices d’application (pour la plupart corrigés au chapitre 12), incitant le lecteur à mettre en œuvre ces connaissances ;
- le chapitre 2 indique comment trouver une documentation directement dans le *shell* Python ;
- le chapitre 4 montre l’intérêt des tables de *hash* et leur application aux dictionnaires et aux ensembles ;
- le chapitre 6 donne un exemple de définition et d’usage des packages ;
- le chapitre 7, entièrement réécrit, détaille les fichiers binaires, le gestionnaire de contexte `with`, le module `pathlib`. De plus, deux applications sont développées :
 - une introduction au SGBDR et au langage SQL à travers le module `SQLite`,
 - une introduction rapide à Internet et au web à travers une application micro-serveur web ;
- le chapitre 10 décrit les constructions algorithmiques de base : pile, file, liste chaînée, arbre et graphe ;
- le chapitre 11 présente l’écosystème Python scientifique avec notamment `IPython`, `NumPy`, `Pandas` et `matplotlib`. Le domaine du traitement d’image est abordé avec `scikit-image`. La documentation du code est présentée et les tests sont traités avec `pytest`. Enfin nous évoquons les microcontrôleurs et les objets connectés ;
- le propos de l’annexe consacrée à l’encodage de caractères a été étendu au codage des nombres.

Installation de la distribution Python

Afin de suivre l'évolution de Python, nous avons choisi de détailler cette installation en dehors du présent ouvrage, sur le site <https://github.com/lpointal/appbclp/tree/master/installation>.

Le lecteur y trouvera tous les indications nécessaires pour installer Python sur les principales plateformes, notamment en utilisant la distribution `miniconda3`, ainsi que l'installation des environnements de développement utilisés. Une page web annexe présente les bases permettant d'utiliser `conda` et des environnements virtuels.

Les programmes en ligne

L'adresse <https://github.com/lpointal/appbclp> regroupe :

- des documentations générales (abrégé et mémento Python, la documentation Jupyter);
- les principaux programmes utilisés pour illustrer les concepts dans le livre;
- tous les exercices du livre et leur correction;
- les exercices supplémentaires et leur correction.

Remerciements

Les auteurs remercient vivement toutes les personnes qui les ont aidés dans la réalisation de ce projet, notamment :

- Hélène Cordeau pour ses nouvelles illustrations ; les aventures de *Pythoon* enchantent les têtes de paragraphe !;
- Jean-Luc Blanc et Brice Martin, des éditions Dunod, pour leur excellent travail critique de relecture ;
- Cécile Trevian pour son aide professionnelle à la traduction du « Zen de Python » (p. 221, annexe A);
- la liste GUTenberg¹ pour ses conseils avisés à un $\text{\TeX}$ ien reconnaissant.

Nous tenons également à citer :

- Lucile Roussier, dont la confiance a permis à Laurent de sélectionner et ensuite de promouvoir Python comme langage de script au sein du LURE² dès 1995 ;
- nos familles respectives pour avoir supporté nos indisponibilités durant la rédaction de cet ouvrage.

Une pensée spéciale pour Stéphane Barthod : son enseignement didactique auprès des étudiants et son engagement envers ses collègues ne seront pas oubliés.

Enfin il faudrait saluer tous les auteurs butinés sur Internet...

Pour joindre les auteurs

Vous pouvez nous adresser vos remarques aux adresses électroniques suivantes :

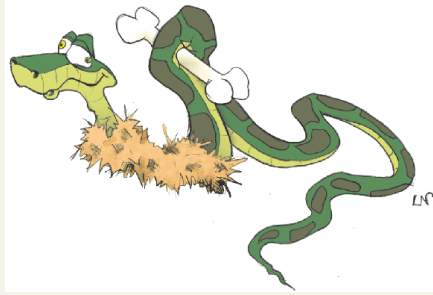
@ pycours@kordeo.eu

@ laurent.pointal@laposte.net

1. Le Groupe francophone des Utilisateurs de $\text{\TeX}$ (gut@ens.fr).

2. Laboratoire pour l'Utilisation du Rayonnement Électromagnétique.

Programmer en Python



Illustré par un exemple cartographique complet, ce premier chapitre expose une méthodologie de développement d'une application.

Il introduit également les grandes caractéristiques du langage Python, le replace dans l'histoire des langages informatiques, donne les particularités de production des programmes, définit la notion si importante d'algorithme et conclut sur les diverses implémentations disponibles.

1.1 Mais pourquoi donc apprendre à programmer ?

Avant de plonger dans les détails de la technique, il est important de se demander pourquoi programmer un ordinateur. L'apprentissage des bases de la programmation de logiciels et, au préalable, l'apprentissage des bases de l'algorithmique, permettent de développer des capacités à comprendre la forme séquentielle des processus divers que l'on peut rencontrer dans la vie de tous les jours¹ et à savoir écrire de tels processus d'une façon formelle. Pour certaines personnes à l'esprit plus analytique, plus mathématique, certains concepts et leur mise en œuvre sont évidents et le passage du problème à la programmation de la solution assez facile. Pour d'autres, à l'esprit plus global (on parle de pensée en arborescence ou analogique), le passage du problème à la forme séquentielle de la solution sera moins aisé, l'apprentissage de la programmation et sa pratique peuvent alors apporter le savoir et des techniques leur permettant cette transition.

L'informatique, ou traitement automatique de l'information, est présente un peu partout dans le monde qui nous entoure, et ce de façon de plus en plus envahissante dans notre quotidien (objets « connectés », smartphones, systèmes de réponse automatique, systèmes auto-pilotés, outils communicants, World Wide Web et autres protocoles de l'Internet...).

Avoir des connaissances en programmation permet de démythifier l'aspect « magique » de l'informatique, d'en comprendre les tenants et aboutissants, les risques liés, les impacts sur la vie privée... et dans une certaine limite de s'approprier cette informatique.

Savoir programmer, donc mettre en œuvre pratiquement des algorithmes existants ou écrire ses propres algorithmes, permet d'automatiser des tâches qui seraient pour le moins fortement chronophages ou répétitives et sources d'erreurs potentielles.

1. Une recette de cuisine, une notice de montage ou de réglage, une aide au remplissage d'un formulaire...