Jean-Paul Roy

Python

Apprentissage actif

pour l'étudiant et le futur enseignant

ellipses

petite musique japonaise
atmosphère zen…

Introduction

Le langage Python

Les programmeurs ont coutume de dire que les langages de programmation sont *universels*. Chaque langage contient les constructions linguistiques suffisantes pour exprimer n'importe quel calcul, et si les professionnels acceptent de passer par les fourches caudines de langages industriels complexes, efficaces et sécurisés, le choix d'un *petit langage* suffisamment performant à enseigner aux grands débutants s'est posé dès le début. En 1964 naquit le langage **BASIC**[1], comme simplification de son grand aîné **FORTRAN**[2] (1955). BASIC a été le plus populaire en son temps, accompagnant la naissance de la micro-informatique et des premiers ordinateurs personnels. Le langage interactif **LISP**[3] (1958) fut au cœur de l'apparition de l'Intelligence Artificielle, et sa simplification **Logo** fut aussi utilisée à l'école dès la fin des années 1960 dans le cadre de recherches sur le développement cognitif de l'enfant, puis de l'enseignement des bases de la programmation. La maturation des langages industriels continuait d'être mise à profit par des chercheurs souhaitant disposer de langages de script permettant de piloter de gros systèmes logiciels, mais aussi par des universitaires soucieux de langages à vocation pédagogique pour remplacer ce BASIC un peu spartiate.

Le langage ABC développé à l'institut CWI[4] d'Amsterdam visait ces deux objectifs. C'est Guido van Rossum, chercheur entré au CWI en 1983 et influencé par les idées et les types de données présents dans ABC, qui définit le langage **Python** en 1989, un petit langage bientôt appelé à jouer dans la cour des grands.

L'ambition de son créateur (qui se définissait lui-même comme *dictateur bénévole à vie* jusqu'à ce qu'il se retire en 2018 du comité décisionnel de la *Python Software Foundation*) consistait à créer un langage interactif à objets le plus simple possible, basé sur le langage C[5]. Il s'appuyait entre autres sur une phrase

1. **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode, par John Kemeny et Thomas Kurtz.
2. **FOR**mula **TRAN**slator, par John Backus.
3. **LIS**t **P**rocessing, par John McCarthy.
4. **C**entrum **W**iskunde & **I**nformatica, où *Wiskunde* signifie *Mathématiques* en néerlandais.
5. L'implémentation de Python par la maison-mère `www.python.org` se nomme CPython, c'est elle que nous utiliserons majoritairement dans ce livre. Mais nous rencontrerons aussi Jython qui est basé sur Java, ainsi que le compilateur Cython pour les programmeurs C.

d'Albert Einstein : *Les choses doivent être aussi simples que possible, mais pas plus simples.* Le but fut assez remarquablement atteint : la syntaxe est légère, toute valeur est un *objet* (les types sont les classes) que l'on peut ranger dans une variable, passer en argument à une fonction, *etc.*

Certains ont noté une ressemblance avec le langage LISP, mais Guido van Rossum n'a jamais été un grand partisan de la programmation fonctionnelle, bien qu'ayant doté Python des *lambda-fonctions* de LISP (dans une version très limitée) et de primitives comme `map` ou `filter`, voire `reduce` [6]. Ce que van Rossum n'a d'ailleurs jamais voulu reprendre de LISP, et que l'on peut regretter, c'est l'élimination automatique de la récurrence terminale, pour des raisons liées à la mise au point. De son point de vue, la programmation par récurrence doit être évitée. C'est une opinion qui n'engage que lui, mais de fait la programmation en Python utilise majoritairement des itérations et des constructions fonctionnelles comme les *compréhensions de listes.* Les parenthèses de LISP ou les accolades de C/Java, qui structurent les phrases d'un programme, sont remplacées par une *indentation* (distance à la marge) *obligatoire.* Python reste assez solitaire dans cette voie qui s'avère pourtant un bon atout pédagogique d'après mon expérience d'enseignant. Le langage ABC, ou Occam au même moment, tous deux oubliés, en faisaient autant.

LISP (1958)	Python (1989)	Java (1995)

`(if (> x 0)`
`  (print x)`
`  (print y))`
	`if x > 0:`
	`  print(x)`
	`else:`
	`  print(y)`
		`if (x > 0) {`
		`  System.out.println(x);`
		`} else {`
		`  System.out.println(y);`
		`}`

Comme LISP et contrairement à C ou Java, le langage Python est *dynamiquement typé* : les variables sont simplement des noms qui font référence à des objets en mémoire. Comme les autres langages dynamiques, la vérification du typage se fait à l'exécution et non lors d'une phase de compilation.

Les programmes sont organisés en *modules*, permettant de développer ou faire appel à des fonctionnalités distinctes dans des fichiers séparés. Une large bibliothèque de modules est fournie en standard et nous aurons l'occasion d'en explorer plusieurs, ainsi que certains en provenance de sources extérieures.

J'ai choisi la distribution **Anaconda**, car elle contient le langage Python mais aussi un grand nombre de logiciels scientifiques parfois pénibles à installer individuellement à cause de leurs dépendances. Les systèmes Anaconda et Jupyter mériteraient un livre à eux seuls, j'en dirai le minimum pour les rendre vite opérationnels, en privilégiant le fond sur la forme.

6. L'algorithme qui fit la gloire de Google se nomme *MapReduce* en l'honneur des fonctions `map` et `reduce` mises en avant par LISP.

Après quelques hésitations, il m'a paru intéressant de présenter aussi le système *Processing*, destiné aux artistes numériques et organisé autour du langage Java. Depuis peu, il existe un mode Python à l'intérieur de *Processing*, ce dernier prenant alors le nom de **Processing.py**. Nous aurons l'occasion d'y faire bien entendu du graphisme, mais aussi d'entr'ouvrir la porte de la musique électronique (avec la librairie *Beads* de Java). Nous verrons à cette occasion une autre implémentation de Python qui se nomme Jython et qui permet l'accès en Python 2 à des modules programmés en Java. Hélas, à ce jour nous serons en effet quelques rares fois obligés d'utiliser du Python dans son ancienne version 2 et devrons nous adapter. Je n'ai pas voulu masquer la réalité des faits, qui sont têtus car il peut s'avérer nécessaire d'utiliser d'autres langages voire d'autres versions de Python, cela fait partie de la vie du programmeur. Nous pénétrerons aussi dans les grosses bibliothèques mathématiques `numpy` et `sympy` essentielles au scientifique travaillant en Python. Nous n'y passerons pas un temps démesuré car il existe des ouvrages spécialisés pour un travail plus approfondi. J'ai été guidé par un souci constant de vous *mettre le pied à l'étrier* et de vous laisser lors des exercices consulter toutes les ressources du Web, notamment les documentations en ligne.

Guide de lecture

La première partie est là pour éclaircir le langage lui-même, en détaillant ses constructions linguistiques et ses tournures idiomatiques les plus fréquentes, sans vouloir faire un manuel de référence complet que l'on peut trouver par ailleurs sur `https://docs.python.org/fr/3/`. Ayant observé que les étudiants lisaient de moins en moins les livres de A à Z, chapitre après chapitre (moi aussi je l'avoue), j'ai décidé de structurer le texte comme une FAQ sur Internet (*Frequently Asked Questions*) en essayant d'insister sur les points qui cristallisaient la plus forte demande d'éclaircissements tout en gardant une progression suffisamment linéaire, sans m'y plier complètement, pour le lecteur préférant un apprentissage plus traditionnel. Vous téléchargerez les solutions en Python sur la page Web de ce livre aux Editions Ellipses (`www.editions-ellipses.fr`) ainsi que sur :

`http://jean.paul.roy.free.fr`

Qui dit programmation dit algorithmes, mais ce livre traite de la pratique du codage. Ce n'est ni un cours ni un ouvrage d'algorithmique qui explorerait toutes les structures de données. Il vous faudrait pour ces dernières par exemple l'indispensable [COR] [7] qui fait à juste titre abstraction du langage de programmation, ou des ouvrages comme [DUR] s'appuyant explicitement sur Python.

7. Cette notation [COR] renvoie au livre de Cormen & al. cité en bibliographie.

La seconde partie consiste en une sélection de petits exercices ou problèmes de programmation. En plus de la partie algorithmique de base, essentielle, j'ai dû faire des choix et en ai profité pour vous emmener dans des zones moins enseignées comme le graphisme, l'analyse syntaxique, l'intelligence artificielle ou la musique électronique. Je ne saurai trop insister sur l'**apprentissage actif** du lecteur. *Hands on !* comme disent les anglo-saxons, mettez les mains sur le clavier. Mais rappelez-vous : *le plus simple possible, mais pas plus simple*. Ce sera donc un peu délicat par endroits. Lisez et programmez à votre rythme, consultez les références et aides en ligne, cherchez des compléments sur le Web, travaillez en groupe. L'important consiste à passer des obstacles, à avoir le sentiment que l'on a gravi une marche de plus : il vous faut sentir la progression. Si vous résolvez vite tous les problèmes en tapant sur le clavier avec le seul petit doigt de la main gauche, ce livre vous aura été sans doute inutile.

Étudiant en Faculté des Sciences, en Ecole d'Ingénieurs, en IUT, peut-être futur enseignant des lycées préparant le CAPES d'Informatique ou déjà chargé de l'option informatique et utilisant sans doute Python, j'espère que ce livre vous plaira et que la partie *Exercices* vous donnera envie d'aller plus loin. J'ai humblement essayé de faire le livre que j'aurai voulu tenir entre les mains en étant étudiant.

Et si vous êtes simplement un amateur de programmation, sachez que le contenu utilise un peu de mathématiques sans en faire un point de mire. Sa lecture ne requiert jamais plus qu'un niveau de classe terminale scientifique ou exceptionnellement de première année post-Bac. Pour le lecteur à la recherche de sensations plus fortes, il existe des ouvrages de programmation scientifique comme [KAR] ou [LAN], ou vraiment axés sur les mathématiques : [ZIM] ou [BAR] avec le logiciel SAGE souplement basé sur Python.

La bibliographie contient aussi des références en anglais lorsque je n'ai pas trouvé de livres équivalents en français. J'en suis désolé mais à ma décharge, la pratique de l'anglais est devenue indispensable dans toute activité scientifique.

Remerciements

Cet ouvrage est en majeure partie issu de cours donnés en Licence au Département Informatique de l'UFR Sciences de l'Université Nice Sophia-Antipolis. Je remercie les nombreux enseignants et moniteurs ayant collaboré à ces enseignements et qui m'ont fait des retours instructifs. Merci aussi à celles et ceux qui ont relu certaines parties de ce livre. S'il reste des erreurs, il va de soi que j'en suis l'unique responsable.

Table des matières