

Préface

En tant que professeur ayant pratiqué l'enseignement de la programmation en parallèle avec d'autres disciplines, je crois pouvoir affirmer qu'il s'agit là d'une forme d'apprentissage extrêmement enrichissante pour la formation intellectuelle d'un jeune et dont la valeur formative est au moins égale, sinon supérieure, à celle de branches plus classiques telles que le latin.

Excellente idée donc, que celle de proposer cet apprentissage dans certaines filières, y compris de l'enseignement secondaire. Comprenons-nous bien : il ne s'agit pas de former trop précocement de futurs programmeurs professionnels. Nous sommes simplement convaincus que l'apprentissage de la programmation a sa place dans la formation générale des jeunes (ou au moins d'une partie d'entre eux), car c'est une extraordinaire école de logique, de rigueur, et même de courage.

À l'origine, le présent ouvrage a été rédigé à l'intention des élèves qui suivent le cours *Programmation et langages* de l'option *Sciences & informatique* au 3^e degré de transition de l'enseignement secondaire belge. Il s'agit d'un texte expérimental qui s'inspire largement de plusieurs autres documents publiés sous licence libre sur *Internet*. Il nous a semblé par la suite que ce cours pouvait également très bien convenir à toute personne n'ayant encore jamais programmé, mais souhaitant s'initier à cette discipline en autodidacte.

Nous y proposons une démarche d'apprentissage non linéaire qui est très certainement critiquable. Nous sommes conscients qu'elle apparaîtra un peu chaotique aux yeux de certains puristes, mais nous l'avons voulue ainsi parce que nous sommes convaincus qu'il existe de nombreuses manières d'apprendre (pas seulement la programmation, d'ailleurs), et qu'il faut accepter d'emblée ce fait établi que des individus différents n'assimilent pas les mêmes concepts dans le même ordre. Nous avons donc cherché avant tout à susciter l'intérêt et à ouvrir un maximum de portes, en nous efforçant tout de même de respecter les principes directeurs suivants :

- L'apprentissage que nous visons se veut généraliste : nous souhaitons mettre en évidence les invariants de la programmation et de l'informatique, sans nous laisser entraîner vers une spécialisation quelconque, ni supposer que le lecteur dispose de capacités intellectuelles hors du commun.
- Les outils utilisés au cours de l'apprentissage doivent être modernes et performants, mais il faut aussi que le lecteur puisse se les procurer en toute légalité à très bas prix pour son usage personnel. Notre texte s'adresse en effet en priorité à des étudiants, et toute notre démarche d'apprentissage vise à leur donner la possibilité de mettre en chantier le plus tôt possible des réalisations personnelles qu'ils pourront développer et exploiter à leur guise.
- Nous avons pris le parti d'aborder très tôt la programmation d'une interface graphique, avant même d'avoir présenté l'ensemble des structures de données disponibles, parce que cette programmation présente des défis qui apparaissent plus concrets aux yeux d'un programmeur débutant. D'autre part, nous observons que les jeunes qui arrivent aujourd'hui dans nos classes « baignent » déjà dans une culture informatique à base de fenêtres et autres objets graphiques interactifs. S'ils choisissent d'apprendre la programmation, ils sont forcément impatients de créer par eux-mêmes des applications (peut-être très simples) où l'aspect graphique est déjà bien présent. Nous avons donc choisi cette approche un peu inhabituelle afin de permettre au lecteur de se lancer très tôt

dans de petits projets personnels attrayants, par lesquels ils puisse se sentir valorisé. En revanche, nous laisserons délibérément de côté les environnements de programmation sophistiqués qui écrivent automatiquement de nombreuses lignes de code, parce que nous ne voulons pas non plus masquer la complexité sous-jacente.

Certains nous reprocheront que notre démarche n'est pas suffisamment centrée sur l'algorithmique pure et dure. Nous pensons que celle-ci est moins primordiale que par le passé. Il semble en effet que l'apprentissage de la programmation moderne par objets nécessite plutôt une mise en contact aussi précoce que possible de l'apprenant avec des objets et des bibliothèques de classes préexistants. Ainsi il apprend très tôt à raisonner en termes d'interactions entre objets, plutôt qu'en termes de procédures, et cela l'autorise assez vite à tirer profit de concepts avancés, tels que l'héritage et le polymorphisme.

Nous avons par ailleurs accordé une place assez importante à la manipulation de différents types de structures de données, car nous estimons que c'est la réflexion sur les données qui doit rester la colonne vertébrale de tout développement logiciel.

Choix d'un premier langage de programmation

Il existe un très grand nombre de langages de programmation, chacun avec ses avantages et ses inconvénients. Il faut bien en choisir un. Lorsque nous avons commencé à réfléchir à cette question, durant notre préparation d'un curriculum pour la nouvelle option Sciences & Informatique, nous avons personnellement accumulé une assez longue expérience de la programmation sous *Visual Basic (Microsoft)* et sous *Clarion (Topspeed)*. Nous avons également expérimenté quelque peu sous *Delphi (Borland)*. Il était donc naturel que nous pensions d'abord exploiter l'un ou l'autre de ces langages. Si nous souhaitions les utiliser comme outils de base pour un apprentissage général de la programmation, ces langages présentaient toutefois deux gros inconvénients :

- Ils sont liés à des environnements de programmation (c'est-à-dire des logiciels) propriétaires. Cela signifiait donc, non seulement que l'institution scolaire désireuse de les utiliser devrait acheter une licence de ces logiciels pour chaque poste de travail (ce qui risquait de se révéler assez coûteux), mais surtout que les élèves souhaitant utiliser leurs compétences de programmation ailleurs qu'à l'école seraient implicitement forcés d'acquiescer eux aussi des licences, ce que nous ne pouvions pas accepter.
- Ce sont des langages spécifiquement liés au seul système d'exploitation *Windows*. Ils ne sont pas « portables » sur d'autres systèmes (*Unix, Mac OS, etc.*). Cela ne cadrait pas avec notre projet pédagogique qui ambitionne d'inculquer une formation générale (et donc diversifiée) dans laquelle les invariants de l'informatique seraient autant que possible mis en évidence.

Nous avons alors décidé d'examiner l'offre alternative, c'est-à-dire celle qui est proposée gratuitement dans la mouvance de l'informatique libre¹. Ce que nous avons trouvé nous a enthousiasmés : non seulement il existe dans le monde de l'*Open Source* des interpréteurs et des compilateurs gratuits pour toute une série de langages, mais surtout ces langages sont modernes, performants, portables (c'est-à-dire utilisables sur différents systèmes d'exploitation tels que *Windows, Linux, Mac OS ...*), et fort bien documentés.

Le langage dominant y est sans conteste *C/C++*. Ce langage s'impose comme une référence absolue, et tout informaticien sérieux doit s'y frotter tôt ou tard. Il est malheureusement très rébarbatif et compli-

¹Un logiciel libre (*Free Software*) est avant tout un logiciel dont le code source est accessible à tous (*Open source*). Souvent gratuit (ou presque), copiable et modifiable librement au gré de son acquéreur, il est généralement le produit de la collaboration bénévole de centaines de développeurs enthousiastes dispersés dans le monde entier. Son code source étant « épluché » par de très nombreux spécialistes (étudiants et professeurs universitaires), un logiciel libre se caractérise la plupart du temps par un très haut niveau de qualité technique. Le plus célèbre des logiciels libres est le système d'exploitation **GNU/Linux**, dont la popularité ne cesse de s'accroître de jour en jour.

qué, trop proche de la machine. Sa syntaxe est peu lisible et fort contraignante. La mise au point d'un gros logiciel écrit en C/C++ est longue et pénible. (Les mêmes remarques valent aussi dans une large mesure pour le langage *Java*.)

D'autre part, la pratique moderne de ce langage fait abondamment appel à des générateurs d'applications et autres outils d'assistance très élaborés tels *C++Builder*, *Kdevelop*, etc. Ces environnements de programmation peuvent certainement se révéler très efficaces entre les mains de programmeurs expérimentés, mais ils proposent d'emblée beaucoup trop d'outils complexes, et ils présupposent de la part de l'utilisateur des connaissances qu'un débutant ne maîtrise évidemment pas encore. Ce seront donc aux yeux de celui-ci de véritables « usines à gaz » qui risquent de lui masquer les mécanismes de base du langage lui-même. Nous laisserons donc le C/C++ pour plus tard.

Pour nos débuts dans l'étude de la programmation, il nous semble préférable d'utiliser un langage de plus haut niveau, moins contraignant, à la syntaxe plus lisible. Après avoir successivement examiné et expérimenté quelque peu les langages *Perl* et *Tcl/Tk*, nous avons finalement décidé d'adopter Python, langage très moderne à la popularité grandissante.

Présentation du langage Python

Ce texte de Stéphane Fermigier est extrait d'un article paru dans le magazine *Programmez!* en décembre 1998. Il est également disponible sur <http://www.linux-center.org/articles/9812/python.html>. Stéphane Fermigier est le co-fondateur de l'AFUL (Association Francophone des Utilisateurs de Linux et des logiciels libres).

Python est un langage portable, dynamique, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation. Python est développé depuis 1989 par Guido van Rossum et de nombreux contributeurs bénévoles.

Caractéristiques du langage

Détaillons un peu les principales caractéristiques de Python, plus précisément, du langage et de ses deux implantations actuelles:

- Python est **portable**, non seulement sur les différentes variantes d'*Unix*, mais aussi sur les OS propriétaires : *Mac OS*, *BeOS*, *NeXTStep*, *MS-DOS* et les différentes variantes de *Windows*. Un nouveau compilateur, baptisé *JPython*, est écrit en Java et génère du *bytecode* Java.
- Python est **gratuit**, mais on peut l'utiliser sans restriction dans des projets commerciaux.
- Python convient aussi bien à des **scripts** d'une dizaine de lignes qu'à des **projets complexes** de plusieurs dizaines de milliers de lignes.
- La **syntaxe** de Python est **très simple** et, combinée à des **types de données évolués** (listes, dictionnaires...), conduit à des programmes à la fois très compacts et très lisibles. À fonctionnalités égales, un programme Python (abondamment commenté et présenté selon les canons standards) est souvent de 3 à 5 fois plus court qu'un programme C ou C++ (ou même Java) équivalent, ce qui représente en général un temps de développement de 5 à 10 fois plus court et une facilité de maintenance largement accrue.
- Python gère ses ressources (mémoire, descripteurs de fichiers...) sans intervention du programmeur, par un mécanisme de **comptage de références** (proche, mais différent, d'un *garbage collector*).
- Il n'y a **pas de pointeurs** explicites en Python.
- Python est (optionnellement) **multi-threadé**.

- Python est **orienté-objet**. Il supporte **l'héritage multiple** et **la surcharge des opérateurs**. Dans son modèle objets, et en reprenant la terminologie de C++, toutes les méthodes sont virtuelles.
- Python intègre, comme Java ou les versions récentes de C++, un système **d'exceptions**, qui permettent de simplifier considérablement la gestion des erreurs.
- Python est **dynamique** (l'interpréteur peut évaluer des chaînes de caractères représentant des expressions ou des instructions Python), **orthogonal** (un petit nombre de concepts suffit à engendrer des constructions très riches), **réflectif** (il supporte la métaprogrammation, par exemple la capacité pour un objet de se rajouter ou de s'enlever des attributs ou des méthodes, ou même de changer de classe en cours d'exécution) et **introspectif** (un grand nombre d'outils de développement, comme le *debugger* ou le *profiler*, sont implantés en Python lui-même).
- Comme *Scheme* ou *SmallTalk*, Python est dynamiquement typé. Tout objet manipulable par le programmeur possède un type bien défini à l'exécution, qui n'a pas besoin d'être déclaré à l'avance.
- Python possède actuellement deux implémentations. L'une, **interprétée**, dans laquelle les programmes Python sont compilés en instructions portables, puis exécutés par une machine virtuelle (comme pour Java, avec une différence importante : Java étant statiquement typé, il est beaucoup plus facile d'accélérer l'exécution d'un programme Java que d'un programme Python). L'autre génère directement du *bytecode* Java.
- Python est **extensible** : comme *Tcl* ou *Guile*, on peut facilement l'interfacer avec des bibliothèques C existantes. On peut aussi s'en servir comme d'un langage d'extension pour des systèmes logiciels complexes.
- La **bibliothèque standard** de Python, et les paquetages contribués, donnent accès à une grande variété de services : chaînes de caractères et expressions régulières, services UNIX standards (fichiers, *pipes*, signaux, sockets, threads...), protocoles Internet (Web, News, FTP, CGI, HTML...), persistance et bases de données, interfaces graphiques.
- Python est un langage qui **continue à évoluer**, soutenu par une communauté d'utilisateurs enthousiastes et responsables, dont la plupart sont des supporters du logiciel libre. Parallèlement à l'interpréteur principal, écrit en C et maintenu par le créateur du langage, un deuxième interpréteur, écrit en Java, est en cours de développement.
- Enfin, Python est un langage de choix pour traiter le XML.

Pour le professeur qui souhaite utiliser cet ouvrage comme support de cours

Nous souhaitons avec ces notes ouvrir un maximum de portes. À notre niveau d'études, il nous paraît important de montrer que la programmation d'un ordinateur est un vaste univers de concepts et de méthodes, dans lequel chacun peut trouver son domaine de prédilection. Nous ne pensons pas que tous nos étudiants doivent apprendre exactement les mêmes choses. Nous voudrions plutôt qu'ils arrivent à développer chacun des compétences quelque peu différentes, qui leur permettent de se valoriser à leurs propres yeux ainsi qu'à ceux de leurs condisciples, et également d'apporter leur contribution spécifique lorsqu'on leur proposera de collaborer à des travaux d'envergure.

De toute manière, notre préoccupation primordiale doit être d'arriver à susciter l'intérêt, ce qui est loin d'être acquis d'avance pour un sujet aussi ardu que la programmation d'un ordinateur. Nous ne voulons pas feindre de croire que nos jeunes élèves vont se passionner d'emblée pour la construction de beaux algorithmes. Nous sommes plutôt convaincus qu'un certain intérêt ne pourra durablement s'installer qu'à partir du moment où ils commenceront à réaliser qu'ils sont devenus capables de développer un projet personnel original, dans une certaine autonomie.

Ce sont ces considérations qui nous ont amenés à développer une structure de cours que certains trouveront peut-être un peu chaotique. Le début s'inspire d'un texte américain disponible sous licence libre : « *How to think like a computer scientist* », par Allen Downey, Jeff Elkner et Chris Meyers (voir : <http://greenteapress.com/thinkpython/thinkCSpy/>), mais nous l'avons progressivement éclaté pour y insérer toute une série d'éléments concernant la gestion des entrées/sorties, et en particulier l'interface graphique *Tkinter*. Nous souhaiterions en effet que les élèves puissent déjà réaliser une petite application graphique dès la fin de leur première année d'études.

Très concrètement, cela signifie que nous pensons pouvoir explorer les huit premiers chapitres de ces notes durant la première année de cours. Cela suppose que l'on aborde d'abord toute une série de concepts importants (types de données, variables, instructions de contrôle du flux, fonctions et boucles) d'une manière assez rapide, sans trop se préoccuper de ce que chaque concept soit parfaitement compris avant de passer au suivant, en essayant plutôt d'inculquer le goût de la recherche personnelle et de l'expérimentation. Il sera souvent plus efficace de réexpliquer les notions et les mécanismes essentiels en situation, dans des contextes variés.

Dans notre esprit, c'est surtout en seconde année que l'on cherchera à structurer les connaissances acquises, en les approfondissant. Les algorithmes seront davantage décortiqués et commentés. Les projets, cahiers des charges et méthodes d'analyse seront discutés en concertation. On exigera la tenue régulière d'un cahier de notes et la rédaction de rapports techniques pour certains travaux.

L'objectif ultime sera pour chaque élève de réaliser un projet de programmation original d'une certaine importance. On s'efforcera donc de boucler l'étude théorique des concepts essentiels suffisamment tôt dans l'année scolaire, afin que chacun puisse disposer du temps nécessaire.

Il faut bien comprendre que les nombreuses informations fournies dans ces notes concernant une série de domaines particuliers (gestion des interfaces graphiques, des communications, des bases de données, etc.) sont facultatives. Ce sont seulement une série de suggestions et de repères que nous avons inclus pour aider les étudiants à choisir et à commencer leur projet personnel de fin d'études. Nous ne cherchons en aucune manière à former des spécialistes d'un certain langage ou d'un certain domaine technique : nous voulons simplement donner un petit aperçu des immenses possibilités qui s'offrent à celui qui se donne la peine d'acquérir une compétence de programmeur.

Versions du langage

Python continue à évoluer, mais cette évolution ne vise qu'à améliorer ou perfectionner le produit. Vous n'aurez pas à modifier tous vos programmes afin de les adapter à une nouvelle version qui serait devenue incompatible avec les précédentes. Les exemples de ce livre ont été réalisés les uns après les autres sur une période de temps relativement longue : certains ont été développés sous Python 1.5.2, puis d'autres sous Python 1.6, Python 2.0, Python 2.1, Python 2.2 et enfin Python 2.3.

Tous continuent cependant à fonctionner sans problème sous les versions 2.4 et 2.5 apparues depuis, et ils continueront certainement à fonctionner sans modification majeure sur les versions futures.

Installez donc sur votre système la dernière version disponible, et amusez-vous bien !

Distribution de Python et bibliographie

Les différentes versions de Python (pour *Windows*, *Unix*, etc.), son **tutoriel** original, son **manuel de référence**, la **documentation** des bibliothèques de fonctions, etc. sont disponibles en téléchargement gratuit depuis Internet, à partir du site web officiel : <http://www.python.org>

Il existe également de très bons ouvrages imprimés concernant Python. En langue française, vous pourrez très profitablement consulter les manuels ci-après :

- **Programmation Python**, par Tarek Ziadé, Editions Eyrolles, Paris, 2006, 538 p., ISBN 978-2-212-11677-9. C'est l'un des premiers ouvrages édités directement en langue française sur le langage Python. Excellent. Une mine de renseignements essentielle si vous voulez acquérir les meilleures pratiques et vous démarquer des débutants.
- **Au coeur de Python**, volumes 1 et 2, par Wesley J. Chun, traduction de *Python core programming, 2d edition* (Prentice Hall) par Marie-Cécile Baland, Anne Bohy et Luc Carité, Editions Campus-Press, Paris, 2007, respectivement 645 et 385 p., ISBN 978-2-7440-2148-0 et 978-2-7440-2195-4. C'est un ouvrage de référence indispensable, très bien écrit.

D'autres excellents ouvrages en français étaient proposés par la succursale française de la maison d'éditions O'Reilly, laquelle a malheureusement disparu. En langue anglaise, le choix est évidemment beaucoup plus vaste. Nous apprécions personnellement beaucoup **Python : How to program**, par Deitel, Liperi & Wiedermann, Prentice Hall, Upper Saddle River - NJ 07458, 2002, 1300 p., ISBN 0-13-092361-3, très complet, très clair, agréable à lire et qui utilise une méthodologie éprouvée, et **Learn to program using Python**, par Alan Gauld, Addison-Wesley, Reading, MA, 2001, 270 p., ISBN 0-201-70938-4, qui est un très bon ouvrage pour débutants.

Pour aller plus loin, notamment dans l'utilisation de la bibliothèque graphique Tkinter, on pourra utilement consulter **Python and Tkinter Programming**, par John E. Grayson, Manning publications co., Greenwich (USA), 2000, 658 p., ISBN 1-884777-81-3, et surtout l'incontournable **Programming Python** (second edition) de Mark Lutz, Editions O'Reilly, 2001, 1255 p., ISBN 0-596-00085-5, qui est une extraordinaire mine de renseignements sur de multiples aspects de la programmation moderne (sur tous systèmes).

Si vous savez déjà bien programmer, et que vous souhaitez progresser encore en utilisant les concepts les plus avancés de l'algorithmique Pythonienne, procurez vous **Python cookbook**, par Alex Martelli et David Ascher, Editions O'Reilly, 2002, 575 p., ISBN 0-596-00167-3, dont les recettes sont savoureuses.

Exemples du livre

Le code source des exemples de ce livre peut être téléchargé à partir du site de l'auteur :

<http://www.ulg.ac.be/cifen/inforef/swi/python.htm>

ou bien directement à cette adresse :

http://main.pythomium.net/download/cours_python.zip