Partie I - Premiers pas avec Python

par Yves Bailly

Date de publication : 01/07/2005

Dernière mise à jour : 20/11/2005

Depuis que vous utilisez un ordinateur, vous avez très certainement entendu parler de cette activité mystérieuse qu'est « programmer », exercée par ces êtres étranges que sont les « programmeurs ». Tout ce que vous voyez sur votre écran est l'expression d'un ou plusieurs programmes. Je vous propose dans cette série d'articles de passer de l'autre coté de l'écran, de découvrir la programmation.

I - Avant propos

Un ordinateur est un outil, comme un tournevis. Mais à la différence du tournevis, l'ordinateur n'est pas conçu pour exécuter une tâche précise : il est au contraire conçu pour être fortement modulable, et adaptable à pratiquement tous les contextes possibles. Pour cela il a la capacité d'exécuter un certain nombre d'ordres, plus communément appelés instructions, chacune correspondant à une tâche simple et élémentaire. Programmer consiste à définir une succession d'instructions élémentaires devant être exécutées par la machine, pour obtenir l'accomplissement d'une tâche complexe. Un aspect essentiel de la programmation est donc la mise en pratique de la méthode cartésienne : décomposer un problème complexe en sous-problèmes simples.

II - Introduction

II-A - Pourquoi programmer ?

Simplement par jeu ! Écrire un programme et le voir s'exécuter, chercher pourquoi il ne fait pas ce que vous attendiez, est une activité ludique en soi.

Plus sérieusement, aussi bien en milieu professionnel que familial, on éprouve fréquement le besoin d'automatiser certaines choses répétitives et laborieuses, d'obtenir certains résultats, de traiter certaines données pour lesquelles on ne dispose pas de logiciel tout prêt - ou alors, ceux qui existent sont mal adaptés ou trop onéreux à acquérir. Dans ce genre de situations, avoir la capacité d'écrire un petit programme peut permettre de gagner beaucoup de temps, de précision et d'efforts. Voici un autre aspect important de la programmation : elle est bien souvent la meilleure réponse à la paresse ! Attention toutefois à ne pas confondre paresse et fainéantise...

II-B - Un peu d'histoire

Les premiers ordinateurs étaient programmés à l'aide de cartes perforées, les instructions étant données sous la forme de longues suites de 0 et de 1 qu'il fallait entrer manuellement. Les programmeurs devaient se confronter directement au *langage machine*, c'est-à-dire au langage brut que comprend la machine - qui n'a rien d'humain. Au milieu des années cinquante apparurent les premiers *langages de programmation*, dont l'objectif premier était de détacher le programmeur du langage machine. Dès lors, les programmes sont écrits dans un dialecte relativement compréhensible, respectant une syntaxe et une grammaire strictes et précises. Commes ces langages furent développés par nos amis anglo-saxons, on pourrait les qualifier de langue anglaise très simplifiée.

Tandis que le langage machine est par nature très lié à l'électronique de l'ordinateur, le développement des langages de programmation a ouvert la voie de l'*abstraction*, principe fondamental qui cherche à éloigner autant que possible le programmeur des contingences matérielles, pour le rapprocher des concepts, de la « chose » concrète ou abstraite qu'il doit traiter. Lorsque l'on veut modéliser informatiquement un moteur de voiture ou la courbe subtile d'une fonction mathématique, autant ne pas trop s'encombrer l'esprit avec l'état moléculaire des transistors du microprocesseur...

C'est ainsi qu'au fil du temps, différents *paradigmes* (ensembles de règles et principes de conception) de programmation se sont développés. Programmation structurée, fonctionnelle, orientée objet... à chaque évolution, l'objectif est toujours le même : permettre la meilleure représentation informatique possible, la plus intuitive possible, de l'idée sur laquelle opère le programme.

Pour un aperçu de l'évolution des langages de programmation, je vous recommande de consulter le site [1] ainsi que les liens qu'il propose en bas de page.

II-C - Le choix du langage

Pour programmer, il nous faut un langage. Il en existe de très nombreux, chacun étant plus ou moins facile à aborder, chacun plus ou moins adapté à tel ou tel domaine. Notre but étant ici l'apprentissage et non la mise sur orbite de satellites, j'ai choisi le langage Python ([2]), pour diverses raisons :

- la syntaxe du langage incite à la clarté, ce qui facilite grandement la relecture des programmes ; il est en effet très pénible de « reprendre » un programme de quelqu'un d'autre, ou que l'on a écrit soi-même longtemps auparavant, dont le simple aspect visuel est embrouillé ;
- c'est un langage orienté objet, actuellement le paradigme de programmation le plus répandu ;
- c'est un langage interprété, donc sa mise en oeuvre est très simple ;
- *l'interpréteur*, le logiciel qui permet à vos programmes de s'exécuter, est un Logiciel Libre, gratuit, et multi-plateforme : un programme écrit sous Linux peut fonctionner sous Windows sans aucune modification ;
- enfin, c'est celui qui me convient le mieux ! La connaissance est un critère de choix, mais il est aussi fréquent qu'un langage soit préféré à un autre pour des raisons subjectives, simplement parce qu'il « plaît » au programmeur, ou qu'un autre lui déplait.

Encore une fois, l'accent sera mis ici sur l'apprentissage de la programmation, pas sur l'apprentissage du langage Python lui-même. Que les connaisseurs du langage me pardonnent donc quelques imprécisions ou non-optimisations.

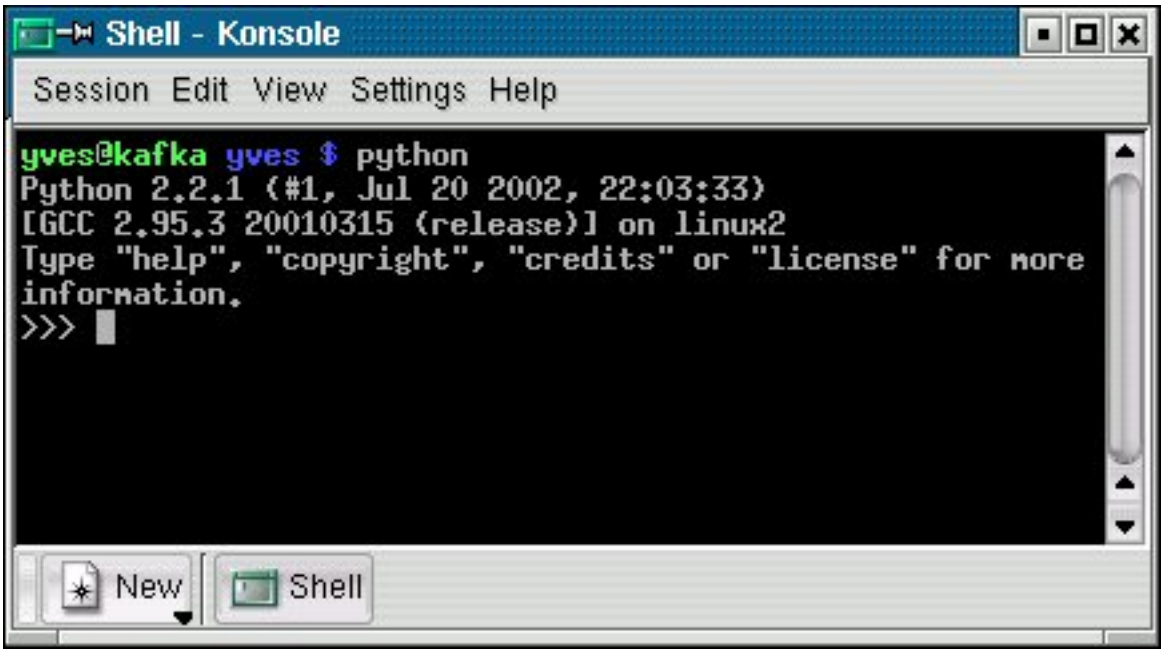
Il est évidemment nécessaire d'installer le langage sur votre ordinateur. La procédure à suivre diffère selon votre distribution Linux, consultez la documentation appropriée, mais il est très probable qu'il soit déjà installé. Si vous avez le malheur de devoir utiliser Windows, consultez le site Web de Python pour télécharger le programme d'installation. Tout ce qui sera fait ici fonctionnera à l'identique sur les deux plate-formes - et mêmes plus, Python étant également disponible pour MacOS ou la plupart des autres systèmes de type Unix.

Mais assez de discours, passons à la pratique !

III - Débuter avec Python

III-A - Premières instructions avec Python

Le coeur d'un langage interprété, c'est l'interpréteur. Pour l'obtenir, si vous êtes sous interface graphique, ouvrez une ligne de commande quelconque **- xterm, konsole**, peu importe. Cela fait, lancez la commande **python**. Voici ce que cela donne chez moi, sous Gentoo Linux, environnement KDE :



Les caractères >>> à gauche sont le *prompt* de l'interpréteur Python, qui vous signale qu'il est prêt à recevoir vos instructions. Sait-il compter ? Essayons, saisissez « 2+3 », puis pressez la touche Entrée :

```
>>> 2+3
5
```

Pas mal. Combien représentent en vieux francs, 5,95 euros ?

```
>>> 5.95 *   6.55957
39.029441499999997
```

Ces exemples simples nous donnent déjà plusieurs informations :

- on peut utiliser l'interpréteur Python comme une calculatrice ;
- pour effectuer une addition, on l'écrit naturellement, en utilisant l'opérateur + ;
- les nombres décimaux sont notés à l'anglaise, en utilisant le point plutôt que notre virgule ;
- la multiplication est effectuée par l'opérateur * (l'étoile, ou astérisque) ;
- remarquez que l'on peut insérer ou non à loisir des espaces entre les valeurs et l'opérateur.

La notion d'opérateur est assez importante en programmation. La soustraction est symbolisée par le tiret -, la division par le *slash* (ou barre oblique) */*. Il en existe de nombreux autres, et leur domaine d'application ne se limite pas aux valeurs numériques. Incidemment, vous venez d'exécutez vos deux premières instructions Python !

Comment afficher du texte ? Très simplement, avec l'instruction **print** :

```
>>> print "Bonjour !"
Bonjour !
```

En langage Python, un texte, plutôt appelé *chaîne de caractères*, est délimité par le caractère guillemet " (ou *double-quote*). Le premier marque le début du texte, le dernier marque la fin. Dans l'exemple précédent, nous avons demandé à Python d'afficher la chaîne de caractères délimitée par la paire de guillemets. Remarquez que ceux-ci ne font pas parti de la chaîne, en effet ils n'apparaissent pas sur l'affichage.

III-B - Les variables

Nous n'avons manipulé dans les exemples précédents que des *valeurs littérales*, c'est-à-dire des données que l'on écrit littéralement, en toutes lettres. Mais l'essence même d'un programme est de travailler sur des données (plus ou moins) quelconques, pour les combiner, les modifier, afin de produire un certain résultat. Il est donc nécessaire de stocker ces données, afin de pouvoir les réutiliser plus tard. Ce stockage s'effectue dans la mémoire de l'ordinateur. Chaque donnée, pour être ultérieurement identifiée, se voit attribuer un nom : c'est ce que l'on désigne usuellement par le terme de *variable*. Au sens strict, une variable est une zone mémoire contenant une donnée susceptible d'être modifiée, le nom n'étant qu'un moyen d'y accéder. Dans la pratique, on identifie le nom à la variable, par abus de langage. Exécutez l'instruction suivante :

```
>>> x = 5
```

Rien ne semble se produire. En fait, cette instruction réserve une zone dans la mémoire, y stocke la valeur **5**, et « étiquette » cette zone avec le nom **x**. Nous venons de créer une variable nommée **x**, en pratique on dit que *on a déclaré la variable* ***x***. On peut schématiser la situation ainsi :

Pour réutiliser cette variable, il suffit d'inscrire son nom, par exemple :

```
>>> 2*x
10
```

Tout se passe comme si le nom **x** avait été remplacé par le contenu de la zone mémoire associée, ici la valeur **5**. Pour changer la valeur de **x** (abus de langage signifiant « *le contenu de la zone mémoire nommée x* »), il suffit d'exécuter une nouvelle affectation (action de stocker une valeur dans une variable, ou plutôt la zone qu'elle représente), par exemple :

```
>>> x = 6
```

Maintenant, **x** contient la valeur 6. La variable peut être elle-même utilisée pour définir sa nouvelle valeur :

```
>>> x = 2*x
```

x contient désormais 12 ! En fait la ligne précédente contient deux instructions enchaînées : *d'abord* multiplier la valeur de **x** par 2, *puis* stocker le résultat dans la variable **x**.

Les variables peuvent naturellement être utilisées pour stocker d'autres choses que des nombres entiers, et les noms ne se limitent pas à une lettre :

```
>>> pi = 3.14159
>>> mon_nom = "Yves Bailly"
```

En langage Python, comme dans la plupart des langages, les noms de variables peuvent être composés des lettres de l'alphabet, des chiffres et du caractères de soulignement _ (*underscore*), avec comme contrainte que le premier caractère ne doit pas être un chiffre. Par ailleurs, Python distingue les majuscules des minuscules : **x** et **X** peuvent être deux noms de variables différentes. D'une manière générale, il est vivement recommandé de donner des noms explicites aux variables. C'est plus long à taper, mais cela facilite grandement la maintenance des programmes. Par exemple, préférez le nom **vitesse** au nom **v**...

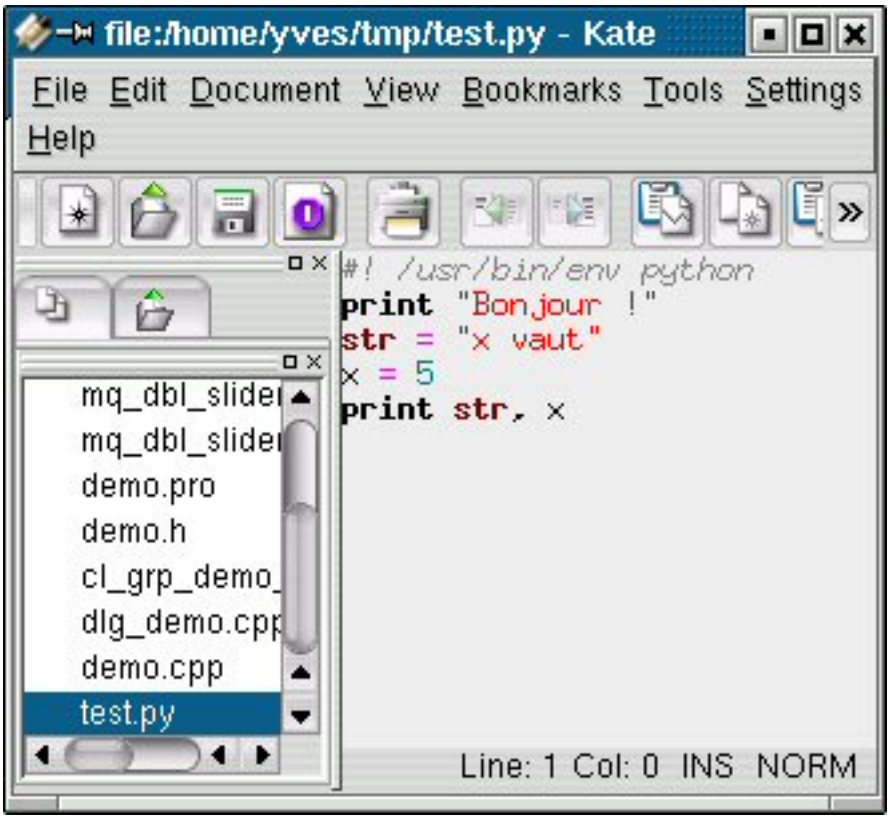
Selon la nature de ce qui est stocké, on parle du *type* d'une variable. Dans les exemples précédents, **x** était une variable de type *entier*, **pi** de type *réel* (ou *virgule flottante*), **mon_nom** de type *chaîne de caractères*. Certains langages, comme Python, déterminent le type d'une variable lors de sa déclaration (qui se trouve être également une affectation). D'autres, comme C/C++, imposent de définir explicitement ce type *avant* toute affectation.

III-C - Votre premier code source

Évidemment, l'utilisation de l'interpréteur telle que nous l'avons fait jusqu'ici ne permet pas de construire un programme complexe, simplement parce qu'il faudrait toujours retaper les instructions. C'est pourquoi les instructions d'un programme sont stockées dans un ou plusieurs fichiers, l'ensemble de ces fichiers constituant ce fameux *code source* : le code tapé par le programmeur pour obtenir le programme désiré.

Pour créer un code source, il suffit d'un éditeur de texte. Attention, il s'agit bien de texte brut, sans aucune mise en forme ! Donc évitez les logiciels tels que OpenOffice, KWord, AbiWord... s'ils sont parfaitement adaptés pour réaliser un document tel que celui que vous lisez actuellement, ils ne conviennent par pour écrire un programme.

Des programmes comme KEdit, xedit ou notepad (sous Windows) conviennent pour écrire un code source. Il est toutefois préférable d'utiliser un éditeur adapté à la programmation, proposant notamment la *mise en évidence syntaxique*, c'est-à-dire faire apparaître dans des couleurs distinctes les variables, les instructions, les types de données, les mots réservés du langage... Il existe de nombreux éditeurs de ce genre, chacun présentant ses avantages et ses inconvénients, les plus connus étant sans doute Emacs et Vim. Personnellement j'utilise actuellement Kate (fourni avec KDE, voyez la capture d'écran plus bas) ou Scite sous Linux, ce dernier étant également disponible sous Windows ([3]). À vous de trouver celui qui vous convient !



Lancez donc un éditeur de texte quelconque, et inscrivez ces quelques lignes (attention, la numérotation des lignes sur la gauche ne doit pas être reproduite, ce n'est qu'une aide pour faire référence plus tard à certaines parties du programme) :

```
1 #!/usr/bin/env python
2 print "Bonjour !"
3 str = "x vaut"
4 x = 5
5 print str, x
```

Sauvegardez le fichier sous un nom quelconque, par exemple **prog.py**. Pour l'exécuter, plusieurs solutions :

- donnez l'attribut exécutable au fichier, soit en ligne de commande avec **chmod +x prog.py**, soit en utilisant les facilités d'un gestionnaire de fichier quelconque ;
- à partir d'une ligne de commande, lancez son interprétation avec **python prog.py** ;
- sous Windows, il suffit en principe de double-cliquer sur le fichier dans l'explorateur.

Le résultat de l'exécution devrait être celui-ci :

```
Bonjour !
x vaut 5
```

Remarquez comme l'instruction **print** est utilisée pour afficher le contenu de deux variables, dont les noms sont donnés à la suite et séparés par une virgule (ligne 5).

La première ligne suit une syntaxe un peu particulière, propre aux systèmes Unix. Elle permet au système de

trouver quel est le logiciel qui doit être utilisé pour interpréter la suite du fichier, ici l'interpréteur Python.

IV - Pour plus tard...

Nous terminerons là-dessus cette première approche. La prochaine fois nous verrons les variables un peu plus en détail, et notamment ce que l'on peut faire avec les chaînes de caractères. Une fois ces bases posées, nous construirons progressivement au fil du temps une application complète, à savoir le jeu des Tours de Hanoï - problème millénaire qui est devenu un grand classique dans le monde de la programmation. La dernière étape étant un programme graphique, utilisant couleurs, boutons et souris !

V - Téléchargement

Vous pouvez télécharger ce cours sous format **pdf**

Langue	Mise à jour	Pages	Taille	Mode FTP	Mode HTTP de secours
	2005.11.20	14	111 Ko	YB01.pdf	YB01.pdf

VI - Références

[1] Histoire des langages de programmation et liens : http://www.levenez.com/lang/

[2] Site web du langage Python : http://www.python.org

[3] L'éditeur Scite : http://www.scintilla.org

VII - Liens