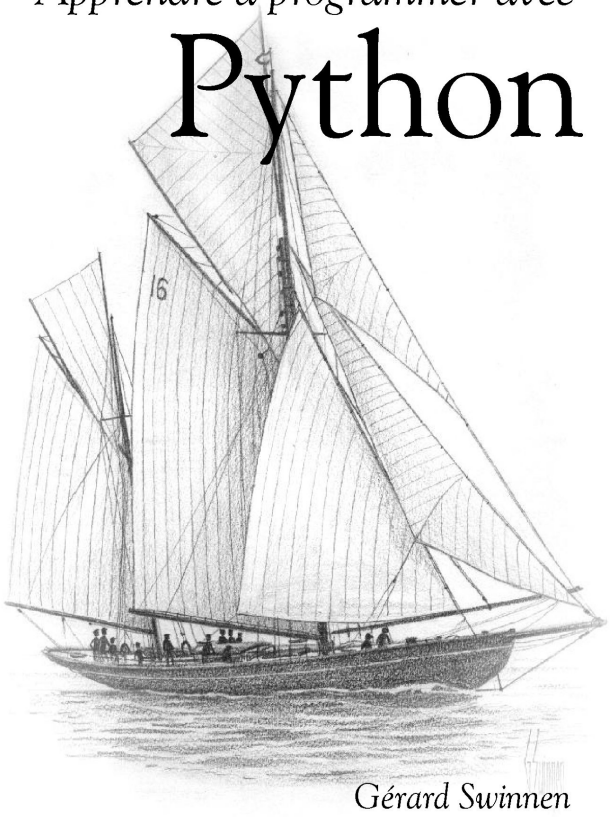
Apprendre à programmer avec Python



Gérard Swinnen

Adaptation libre de "How to think like a computer scientist"
de Allen B. Downey, Jeffrey Elkner & Chris Meyers

Gérard Swinnen est professeur et conseiller pédagogique à l'Institut St-Jean Berchmans à Ste-Marie (Liège). Ce cours d'initiation est essentiellement extrait de son ouvrage « *Apprendre à programmer avec Python* ». Voici ce qu'il nous dit de Python.

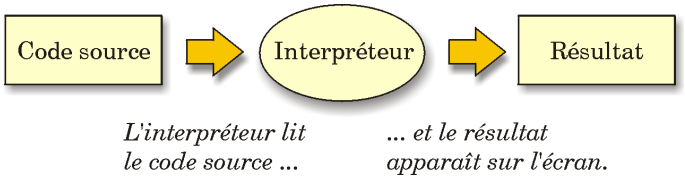
<<Python est un merveilleux langage de programmation. Moderne, portable, puissant, facile à apprendre, il présente en outre l'immense intérêt d'être absolument gratuit !
Estimant que Python constitue à l'heure actuelle le meilleur choix pour commencer un apprentissage de la programmation, nous avons décidé d'apporter notre modeste contribution à la documentation en langue française pour ce langage.
Destinées en premier lieu aux élèves qui suivent le cours "Programmation & langages" de l'option Sciences & Informatique en 5e/6e de l'enseignement technique de transition (enseignement secondaire belge), ces notes vous sont proposées ici en téléchargement gratuit. Leur reproduction et leur distribution restent cependant soumises aux termes de la licence de documentation libre GNU.>>

Grace Hopper, inventeur du compilateur :
« Pour moi, la programmation est plus qu'un art appliqué important. C'est aussi une ambitieuse quête menée dans les tréfonds de la connaissance. »

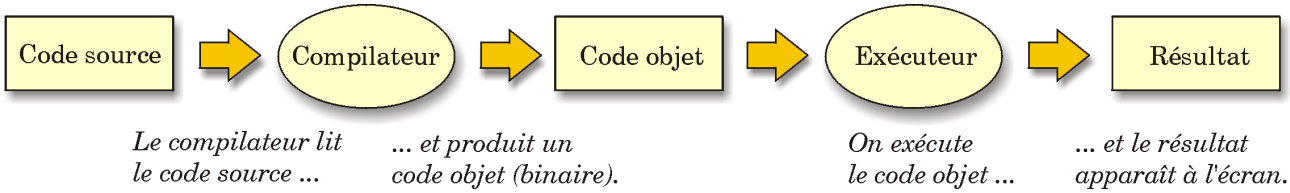
1.1 Compilation et interprétation

Le programme tel que nous l'écrivons à l'aide d'un logiciel éditeur sera appelé désormais ***programme source*** (ou code source). Il existe deux techniques principales pour effectuer la traduction d'un tel programme source en code binaire exécutable par la machine : l'interprétation et la compilation.

- Dans la technique appelée ***interprétation***, le logiciel interpréteur doit être utilisé chaque fois que l'on veut faire fonctionner le programme. Dans cette technique en effet, chaque ligne du programme source analysé est traduite au fur et à mesure en quelques instructions du langage machine, qui sont ensuite directement exécutées. Aucun programme objet n'est généré.



- La ***compilation*** consiste à traduire la totalité du texte source en une fois. Le logiciel compilateur lit toutes les lignes du programme source et produit une nouvelle suite de codes que l'on appelle ***programme objet*** (ou code objet). Celui-ci peut désormais être exécuté indépendamment du compilateur et être conservé tel quel dans un fichier (« fichier exécutable »).



Chacune de ces deux techniques a ses avantages et ses inconvénients :

L'interprétation est idéale lorsque l'on est en phase d'apprentissage du langage, ou en cours d'expérimentation sur un projet. Avec cette technique, on peut en effet tester immédiatement toute modification apportée au programme source, sans passer par une phase de compilation qui demande toujours du temps.

Par contre, lorsqu'un projet comporte des fonctionnalités complexes qui doivent s'exécuter rapidement, la compilation est préférable : il est clair en effet qu'un programme compilé fonctionnera toujours nettement plus vite que son homologue interprété, puisque dans cette technique l'ordinateur n'a plus à (re)traduire chaque instruction en code binaire avant qu'elle puisse être exécutée.

Python est un langage interprété, tout comme java.

Exemple de langage compilé : C ou C++

1.2 Mise au point d'un programme - Recherche des erreurs (« debug »)

La programmation est une démarche très complexe, et comme c'est le cas dans toute activité humaine, on y commet de nombreuses erreurs. Pour des raisons anecdotiques, les erreurs de programmation s'appellent des « *bugs* » (ou « bogues », en France)[1], et l'ensemble des techniques que l'on met en œuvre pour les détecter et les corriger s'appelle « *debug* » (ou « déboguage »).
En fait, il peut exister dans un programme trois types d'erreurs assez différentes, et il convient que vous appreniez à bien les distinguer :

1.2.1 Erreurs de syntaxe

Python ne peut exécuter un programme que si sa ***syntaxe*** est parfaitement correcte. Dans le cas contraire, le processus s'arrête et vous obtenez un message d'erreur. Le terme syntaxe se réfère aux règles que les auteurs du langage ont établies pour la structure du programme.
Tout langage comporte sa syntaxe. Dans la langue française, par exemple, une phrase doit toujours commencer par une majuscule et se terminer par un point. ainsi cette phrase comporte deux erreurs de syntaxe
Dans les textes ordinaires, la présence de quelques petites fautes de syntaxe par-ci par-là n'a généralement pas d'importance. Il peut même arriver (en poésie, par exemple), que des fautes de syntaxe soient commises volontairement. Cela n'empêche pas que l'on puisse comprendre le texte.
Dans un programme d'ordinateur, par contre, la moindre erreur de syntaxe produit invariablement un arrêt de fonctionnement (un « plantage ») ainsi que l'affichage d'un message d'erreur. Au cours des premières semaines de votre carrière de programmeur, vous passerez certainement pas mal de temps à rechercher vos erreurs de syntaxe. Avec de l'expérience, vous en commettrez beaucoup moins.
Gardez à l'esprit que les mots et les symboles utilisés n'ont aucune signification en eux-mêmes : ce ne sont que des suites de codes destinés à être convertis automatiquement en nombres binaires. Par conséquent, il vous faudra être très attentifs à respecter scrupuleusement la syntaxe du langage.
Il est heureux que vous fassiez vos débuts en programmation avec un langage interprété tel que Python. La recherche des erreurs y est facile et rapide. Avec les langages compilés (tel C++), il vous faudrait recompiler l'intégralité du programme après chaque modification, aussi minime soit-elle.

1.2.2 Erreurs sémantiques

Le second type d'erreur est l'erreur ***sémantique*** ou erreur de logique. S'il existe une erreur de ce type dans un de vos programmes, celui-ci s'exécute parfaitement, en ce sens que vous n'obtenez aucun message d'erreur, mais le résultat n'est pas celui que vous attendiez : vous obtenez autre chose.
En réalité, le programme fait exactement ce que vous lui avez dit de faire. Le problème est que ce que vous lui avez dit de faire ne correspond pas à ce que vous vouliez qu'il fasse. La séquence d'instructions de votre programme ne correspond pas à l'objectif poursuivi. La sémantique (la logique) est incorrecte.
Rechercher des fautes de logique peut être une tâche ardue. Il faut analyser ce qui sort de la machine et tâcher de se représenter une par une les opérations qu'elle a effectuées, à la suite de chaque instruction.

---

1 "*bug*" est à l'origine un terme anglais servant à désigner de petits insectes gênants, tels les punaises. Les premiers ordinateurs fonctionnaient à l'aide de "lampes" radios qui nécessitaient des tensions électriques assez élevées. Il est arrivé à plusieurs reprises que des petits insectes s'introduisent dans cette circuiterie complexe et se fassent électrocuter, leurs cadavres calcinés provoquant alors des court-circuits et donc des pannes incompréhensibles.
Le mot français "*bogue*" a été choisi par homonymie approximative. Il désigne la coque épineuse de la châtaigne.

1.2.3 Erreurs à l'exécution

Le troisième type d'erreur est l'erreur en cours d'exécution (*Run-time error*), qui apparaît seulement lorsque votre programme fonctionne déjà, mais que des circonstances particulières se présentent (par exemple, votre programme essaie de lire un fichier qui n'existe plus). Ces erreurs sont également appelées des ***exceptions***, parce qu'elles indiquent généralement que quelque chose d'exceptionnel s'est produit (et qui n'avait pas été prévu). Vous rencontrerez davantage ce type d'erreur lorsque vous programmerez des projets de plus en plus volumineux.

Chapitre 2 : Premiers pas

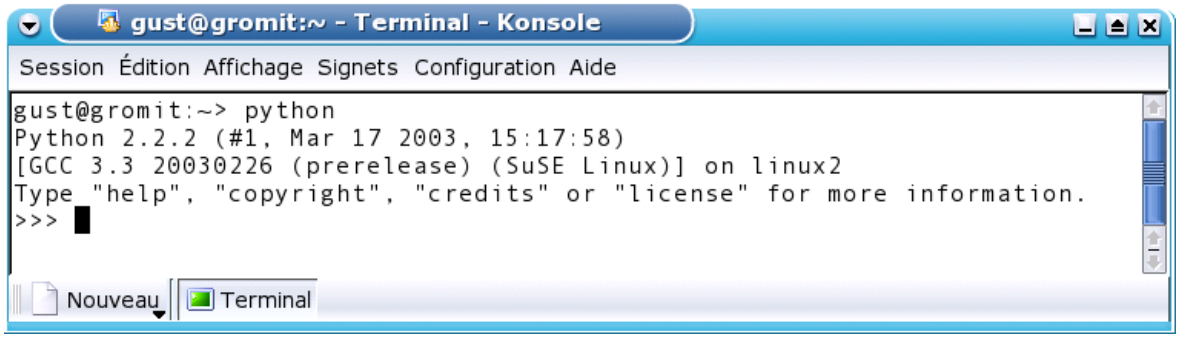
Il est temps de se mettre au travail. Plus exactement, nous allons demander à l'ordinateur de travailler à notre place, en lui donnant, par exemple, l'ordre d'effectuer une addition et d'afficher le résultat.
Pour cela, nous allons devoir lui transmettre des « instructions », et également lui indiquer les « données » auxquelles nous voulons appliquer ces instructions.

2.1 Calculer avec Python

Python présente la particularité de pouvoir être utilisé de plusieurs manières différentes.
Vous allez d'abord l'utiliser ***en mode interactif***, c'est-à-dire d'une manière telle que vous pourrez dialoguer avec lui directement depuis le clavier.

Cela peut se faire dans une console en mode texte, ou « émulateur de terminal » c'est à dire une fenêtre DOS sous windows, ou un « shell » bash sous GNU/Linux (konsole, termx, etc,,,)
Il suffit d'y taper la commande **"python"**

On peut aussi utiliser un logiciel spécialisé, un environnement de travail comme ***IDLE***[2]**.**



IDLE vu sous GNU/Linux, avec le bureau KDE.

Par exemple, vous pouvez tout de suite utiliser l'interpréteur comme une simple calculatrice de bureau. Veuillez donc vous-même tester les commandes ci-dessous :

```
>>> 5+3
>>> 2 - 9                    # les espaces sont optionnels
>>> 7 + 3 * 4                # la hiérarchie des opérations mathématiques est-elle respectée ?
```

2 Sous *Windows*, vous aurez surtout le choix entre l'environnement *IDLE* développé par Guido Van Rossum, inventeur de Python, et *PythonWin*, une interface de développement développée par Mark Hammond.

```
>>> (7+3)*4
```

```
>>> 8./5
```

Si une opération est effectuée avec des arguments de types mélangés (entiers et réels), Python convertit automatiquement les opérandes en réels avant d'effectuer l'opération. Essayez :
```
>>> 4 * 2.5 / 3.3
```

Remarque importante :

dans la plupart des langage de programmation, on doit distinguer les variables réelles $(x \in \mathbb{R})$ (en réalité des nombres décimaux (_____) pour l'ordinateur) des variables entiers naturels $(n \in \mathbb{N})$. Le langage de programmation fait usage de différents ***types de variables***. *(le type 'entier', le type 'réel', le type 'chaîne de caractères', le type 'liste', etc.).*

2.2 Affectation (ou assignation)

En Python comme dans de nombreux autres langages, l'opération d'affectation est représentée par le signe ***égale***[3] :
```
>>> n = 7                                  # donner à n la valeur 7
>>> msg = "Quoi de neuf ?"          # affecter la valeur "Quoi de neuf ?" à msg
>>> pi = 3.14159                          # assigner sa valeur à la variable pi
```

Pour afficher leur valeur à l'écran, il existe deux possibilités. La première consiste à entrer au clavier le nom de la variable, puis <Enter>.
```
>>> n
7
>>> msg
"Quoi de neuf ?"
>>> pi
3.14159
```

Il s'agit cependant là d'une fonctionnalité secondaire de l'interpréteur, qui est destinée à vous faciliter la vie lorsque vous faites de simples exercices à la ligne de commande. A l'intérieur d'un programme, vous utiliserez toujours l'instruction **print** :
```
>>> print msg
Quoi de neuf ?
```

Sous Python, on peut assigner une valeur à plusieurs variables simultanément. Exemple :
```
>>> x = y = 7
```
On peut aussi effectuer des ***affectations parallèles*** à l'aide d'un seul opérateur :

```
>>> a, b = 4, 8.33
```

3 Il faut bien comprendre qu'il ne s'agit en aucune façon d'une égalité, et que l'on aurait très bien pu choisir un autre symbolisme, tel que **n ← 7** par exemple, comme on le fait souvent dans certains pseudo-langages servant à décrire des algorithmes, pour bien montrer qu'il s'agit de relier un contenu (la valeur 7) à un contenant (la variable n).

Exercices

1.1. Décrivez le plus clairement et le plus complètement possible ce qui se passe à chacune des trois lignes de l'exemple ci-dessous :

>>> largeur = 20
>>> hauteur = 5 * 9.3
>>> largeur * hauteur
930

2.2 Calculez le volume de la salle de classe en m^3.

2.3 Opérateurs et expressions

On manipule les valeurs et les variables qui les référencent, en les combinant avec des ***opérateurs*** pour former des ***expressions***. Exemple :

a, b = 7.3, 12
y = 3*a + b/5

La seconde ligne de l'exemple consiste à affecter à une nouvelle variable y le résultat d'une expression qui combine les *opérateurs* * , + et / avec les *opérandes* a, b, 3 et 5.

Signalons au passage la disponibilité de l'opérateur ***modulo***, représenté par le symbole **%**.
Cet opérateur fournit ***le reste de la division entière*** d'un nombre par un autre.
>>> 10 % 3
>>> 10 % 5

Exercice :

1.2. **Testez les lignes d'instructions suivantes. Décrivez dans votre cahier ce qui se passe :**

```
>>>  r , pi = 12, 3.14159
>>>  s = pi * r**2
>>>  print s
>>>  print type(r), type(pi), type(s)
>>>
```

Quelle est, à votre avis, l'utilité de la *fonction* type() ?

Chapitre 3 : Contrôle du flux d'instructions

Nous avons vu que l'activité essentielle d'un analyste-programmeur est la résolution de problèmes. Or, pour résoudre un problème informatique, il faut toujours effectuer une série d'***actions*** dans un certain ***ordre***. La description structurée de ces actions et de l'ordre dans lequel il convient de les effectuer s'appelle un ***algorithme***.
Les *structures de contrôle* sont les groupes d'instructions qui déterminent l'ordre dans lequel les actions sont effectuées. En programmation moderne, il en existe seulement trois :
la *séquence,* la *sélection, ,* et la ***répétition*.**

3.1 Séquence[4] d'instructions

Python exécute normalement les instructions de la première à la dernière, sauf lorsqu'il rencontre une *instruction conditionnelle* comme l'instruction **if .**

3.2 Sélection ou exécution conditionnelle

La plus simple de ces instructions conditionnelles est l'instruction **if**. Veuillez entrer dans votre éditeur Python les deux lignes suivantes :

```
>>> a = 150
>>> if (a > 100):
...     print "a dépasse la centaine"
…
```

Recommencez encore, en ajoutant deux lignes comme indiqué ci-dessous. Veillez bien à ce que la quatrième ligne débute tout à fait à gauche (pas d'indentation), mais que la cinquième soit à nouveau indentée (de préférence avec un retrait identique à celui de la troisième) :

```
>>> a = 20
>>> if (a > 100):
...     print "a dépasse la centaine"
... else:
...     print "a ne dépasse pas cent"
...
```

Frappez <Enter> encore une fois. Le programme s'exécute.

On peut faire mieux encore en utilisant aussi l'instruction **elif** (contraction de « else if ») :

```
>>> a = 0
>>> if a > 0 :
...     print "a est positif"
... elif a < 0 :
...     print "a est négatif"
... else:
...     print "a est nul"
…
```

4 Tel qu'il est utilisé ici, le terme de ***séquence*** désigne donc une série d'instructions qui se suivent. Nous préférerons dans la suite de cet ouvrage réserver ce terme à un concept Python précis, lequel englobe les ***chaînes de caractères***, les ***tuples*** et les ***listes*** (voir plus loin).

3.3 Opérateurs de comparaison

La condition évaluée après l'instruction if peut contenir les ***opérateurs de comparaison*** suivants :

```
x == y          # x est égal à y
x != y          # x est différent de y
x > y           # x est plus grand que y
x < y           # x est plus petit que y
x >= y          # x est plus grand que, ou égal à y
x <= y          # x est plus petit que, ou égal à y
```

Exemple :

```
>>> a = 7
>>> if (a % 2 == 0):
...     print "a est pair"
...     print "parce que le reste de sa division par 2 est nul"
... else:
...     print "a est impair"
…
```

3.4 Répétitions en boucle - l'instruction while

L'une des choses que les machines font le mieux est la répétition sans erreur de tâches identiques. Il existe bien des méthodes pour programmer ces tâches répétitives. Nous allons commencer par l'une des plus fondamentales : la boucle construite à partir de l'instruction **while**.
Veuillez donc entrer les commandes ci-dessous :

```
>>> a = 0
>>> while (a < 7):                  # (n'oubliez pas le double point !)
...    a = a + 1                    # (n'oubliez pas l'indentation !)
...    print a
```

Nous avons ainsi construit notre première ***boucle de programmation***, laquelle répète un certain nombre de fois le bloc d'instructions indentées.

3.5 Élaboration de tables

Recommencez à présent le premier exercice, mais avec la petite modification ci-dessous :

```
>>> a = 0
>>> while a < 12:
...     a = a +1
...     print a , a**2 , a**3
```