Chapitre 1

Introduction

Le choix de python

Pourquoi développer en python ? Parce que c'est un langage d'une grande souplesse, qui permet rapidement d'écrire des codes qui fonctionnent. C'est un langage pour lequel il existe une communauté très active de développeurs avec des forums où vous pourrez trouver la solution à de nombreux problèmes et des sites où sont accessibles de nombreux cours et tutoriaux gratuits et bien faits. Python permet de programmer selon plusieurs paradigmes : programmation impérative structurée, programmation orientée objet, programmation fonctionnelle.

C'est un langage Open Source, libre et gratuit. Il est largement utilisé dans la communauté bioinformatique et il existe des bibliothèques BioPython comportant de nombreuses fonctions de traitement des données biologiques. De nombreuses autres bibliothèques Python existent : NumPy ou SciPy pour le calcul numérique, Tkinter pour les interfaces GUI, Matplotlib pour les graphiques 2D, gnuplot-py pour l'interfaçage avec gnuplot). C'est un langage libre et ouvert pour lequel on trouve facilement des supports gratuits et disponibles.

Python est un langage de choix pour de nombreuses tâches : développement de scripts d'administration système, développement de scripts CGI pour les applications web, prototypage rapide d'applications, développement d'applications complexes, langage d'extension d'applications écrites en C, C++, Java, R et autres, développement d'interface graphiques, automatisation de tests unitaires d'applications écrites en C, C++ ou autres, remplacement d'environnement de calcul commerciaux tels que Matlab.

Compilation et interprétation

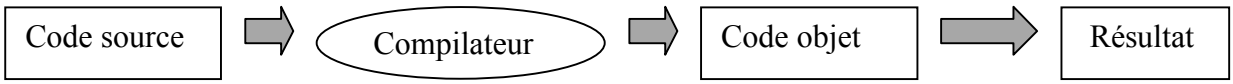
Le code inscrit dans un fichier à l'aide d'un éditeur de texte est appelé programme source. Il existe deux modes de traduction d'un tel programme source en code binaire exécutable par la machine : l'interprétation et la compilation.

Dans l'*interprétation*, le logiciel *interpréteur* doit être utilisé à chaque exécution du programme. Il lit une à une les lignes les analyse et les traduit en code binaire. Les quelques instructions traduites en langage machine sont ensuite exécutées au fur et à mesure. Aucun programme objet (fichier en binaire) n'est généré.

L'interpréteur lit le code source ...

... et le résultat apparaît sur l'écran

La *compilation* consiste à traduire l'intégralité du code source en langage binaire en une seule fois. Un *programme objet* est généré par le logiciel *compilateur*. Le code binaire peut désormais être exécuté indépendamment du compilateur et du code source. C'est un fichier *exécutable* qui peut être conservé (et diffusé) tel quel. Le compilateur n'intervient pas à chaque exécution du programme, il a été compilé une fois pour toutes.



Le compilateur lit le code source ...

... et produit un code objet (binaire).

Le code binaire est directement exécutable et le résultat apparaît à l'écran.

Il existe un compromis entre ces deux techniques qui est utilisé par Python ou Java, c'est de produire un code intermédiaire en *ByteCode*, code compilé similaire à un langage machine, qui sera transmis à l'interpréteur pour l'exécution finale. Le ByteCode étant très facile en interpréter en langage machine l'interprétation sera plus rapide que celle d'un code source, mais l'interpréteur est requis à chaque exécution du programme.

Forces et faiblesses de python

Python est un langage objet (même si nous n'aborderons que très peu cet aspect de la programmation python dans le cadre de ce cours ; en python tout est objet).
Python est un langage interprété, fortement typé à typage dynamique.
Python est un langage de script, clair, compact et portable.
Python est un langage qui permet une grande vitesse de développement.
Python est pourvu d'une importante bibliothèque standard.
Plusieurs paradigmes de programmation sont disponibles.
Plusieurs modes d'exécution d'un code python existent, en particulier l'interpréteur interactif qui est très pratique pour tester la syntaxe d'une instruction.

On ne peut pas générer de binaire compilé ou natif en python. Il n'y a pas ou peu d'optimisations automatiques de la part de l'interpréteur. Il n'y a pas de vérification statique du typage.

Quelques références utiles…

- Apprendre à programmer en Python. Gérard Swinnen. O'Reilly.

Un livre très utile et pédagogique dont je vous recommande la lecture (et qui a fortement inspiré ce cours). Il s'adresse à des débutants en programmation et introduit la programmation par l'exemple en python. Il est accessible sur le Web et est à la bibliothèque. Il contient de nombreux exemples didactiques corrigés à la fin de l'ouvrage.

- Programming Python. Alex Martelli. O'Reilly.

Une « introduction » plus poussée à la programmation Python (1250 pages pour l'édition 2, livrée avec un CD). Nécessite déjà de bonnes bases de programmation. Il existe une version française. Il est à la bibliothèque.

- Python in a Nutshell. Alex Martelli. O'Reilly.

Un condensé de Python, très utile et très bien fait. Très bien.

- Python CookBook. David Ascher. O'Reilly.

Le livre de référence de tout programmeur Python. Très complet mais inutile à votre niveau.

De nombreux cours et tutoriaux sont disponibles en ligne. En voici quelques uns :

- le site officiel de python : http://www.python.org
- l'index des modules disponibles : http://www.python.org/doc/current/modindex.html
- le CookBook: http://aspn.activestate.com/ASPN/Python/CookBook

avec de nombreuses recettes pour résoudre les problèmes courants.

- Apprendre a programmer en Python : version pdf téléchargeable à l'adresse :

http://www.framasoft.net/article1971.html (également téléchargeable au site officiel O'Reilly)

- le tutorial de Guido von Rossum : http://docs.python.org/tut : TRES BIEN
- l'association francophone des développeurs Python : http://www.afpy.org
- liste de discussion francophone : python@aful.org
- forum Usenet francophone : fr.comp.lang.python
- forum Usenet anglophone : comp.lang.python

Chapitre 2

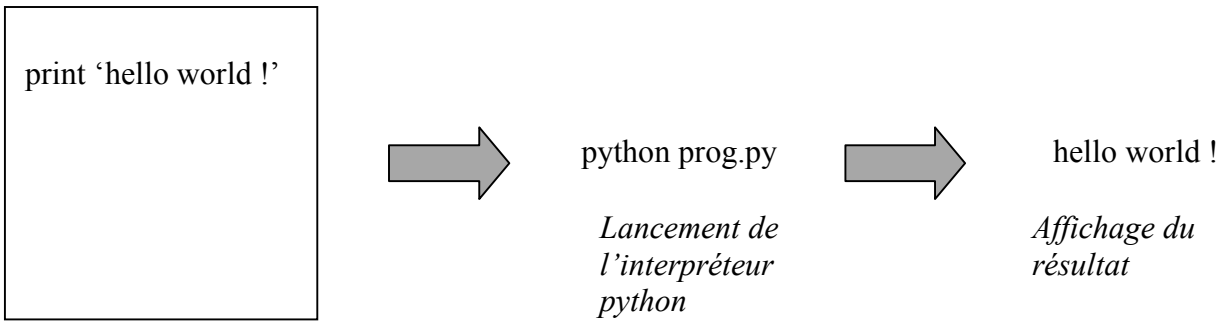
Syntaxe du langage

2.1 Premiers pas en python

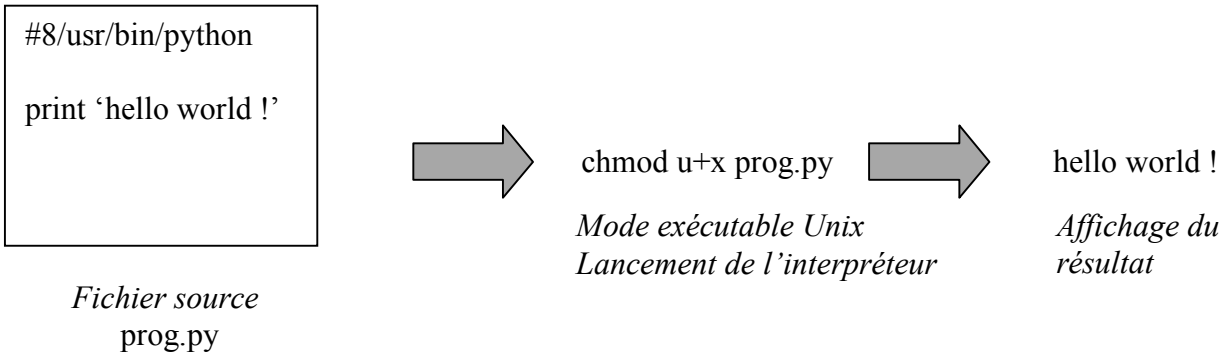
Il existe plusieurs façons d'exécuter un programme python : le mode interactif (interpréteur interactif) ou le mode programme où un code source est fourni à l'interpréteur.

Mode non interactif :

1. Première méthode



2. Deuxième méthode



Mode iteractif :

```
python
>>> print 'hello world !'        ⇐ Instruction python passée à l'interpréteur
hello world !                    ⇐ Résultat affiché
```

2.2 Principes généraux de la syntaxe et du langage

Une seule instruction par ligne en python.
L'indentation est cruciale, elle permet de délimiter les blocs d'instructions (lisibilité accrue).
Les commentaires sont délimités par # et s'étendent jusqu'à la fin de la ligne.
Il n'est pas nécessaire de déclarer les variables (ce qui ne veut pas dire qu'elles n'ont pas de type, ni que le python n'est pas un langage typé…)

On appelle une fonction en utilisant l'opérateur (). Le passage des arguments se fait par référence. Une fonction renvoie toujours une valeur en python (comme en C). Les noms de fonction sont en minuscules, peuvent être composés de plusieurs mots, chacun commençant par une majuscule (convention).

```
>>> y = sin(x)
>>> myFuncScreen()
```

2.3 Les identifiants et l'affectation

En python il n'y a pas de variables à proprement parler mais des identifiants (qui pointent sur des zones mémoire).
Les identifiants servent à référencer les objets et en python tous les éléments manipulables du langage sont des objets.
Les identifiants doivent commencer par une lettre ou un _, ils peuvent contenir des lettres, des chiffres et le caractère _.

- Affectation simple

```
>>> a = 10
>>> txt = 'Quelle belle journée'
```

Les deux identifiants `a` et `txt` sont créés lors de l'affectation. En effet python va évaluer leur type, réserver la place mémoire nécessaire et faire pointer l'identifiant sur cette zone, et ceci de manière complètement transparente pour le programmeur.

- Affectation dynamique

```
>>> a = int(raw_input('Entrez une valeur entière'))
>>> txt = raw_input('Entrez une phrase')
```

Il n'est pas obligatoire de passer un message en argument a `raw_input`. Les valeurs retournées sont de type `str` et donc à convertir en valeurs numériques si besoin.

2.4 Les types de bases

2.4.1 Les opérateurs

Les opérateurs numériques

Expression	*Opération*
-x	Opposé de x
x % y	Reste de la division entière de x par y
x / y	Division de x et y
x // y	Division entière de x par y
x * y	Produit de x et y
x – y	Différence de x et y
x + y	Somme de x et y
x ** y	x puissance y

Les opérateurs de comparaison

Expression	*Opération*
x < y, x <= y	x inférieur (ou égal) à y
x > y, x >= y	x supérieur (ou égal) à y
x == y	x égal à y
x<> y , x != y	x différent de y
x or y, x and y	OU et ET logique
not x	NON logique

2.4.2 Les types de données standard

Les types numériques

Type	*Description*
Entier : `int`	Utilise le type C `long`, minimum 32 bits `32` ou `-10` ou `0xAF21`
Entier long : `long`	Précision illimitée `125L` ou `0x10000000000000L`
Réel : `float`	Utilise le type C double `3.1416` ou `1.48e25`
Complexe : `complex`	`3 + 2j` ou `5.1 + 4j`

Quelques fonctions ou expressions utiles	*Résultat*
`pow(x,y)`	x puissance y
`divmod(x,y)`	(x//y, x%y)
`abs(x)`	norme de x

`complex(re,im)`	re + im * j
`float(x)`	Conversion de x en réel
`int(x)`	Conversion de x en entier
`long(x)`	Conversion de x en entier long

De nombreuses autres fonctions sont disponibles dans le *module math* (fonctions trigonométriques, fonctions exponentielle (`exp`) et logarithme (`log`), fonctions d'arrondis (`ceil` et `floor`), racine carré (`sqrt`) … ainsi que les constantes mathématiques `e`, `pi` …).

Les chaînes de caractères

Pour python, une chaîne de caractères (type string) est une suite de caractères délimitée soit par des apostrophes `'hello world !'` soit par des guillemets `"hello world !"`. En général on utilise des apostrophes sauf si la chaîne contient des apostrophes auquel cas on utilisera des guillemets pour délimiter la chaîne. Les guillemets triple sont aussi autorisés et ils permettent d'utiliser des retours à la ligne dans la chaîne, nous les retrouverons pour la documentation en ligne des code python.

A l'intérieur d'une chaîne on utilisera le caractère \ (*antislash*) pour *échapper* le caractère suivant (c'est à dire que le caractère spécial suivant ne sera pas interprété en tant que caractère spécial).
Exemple :
```
txt3 = 'Quelle belle journée, n\'est-il pas ?'
```
L'apostrophe protégée par \ n'est pas interprétée comme le délimitateur de fin de chaîne.

Les opérations sur les chaînes de caractères

Opération	*Résultat*
x in s	Vrai si x ∈ s, faux sinon (x char ou str)
x not in s	Vrai si x ∉ s, faux sinon (x char ou str)
s + t	Concaténation de s et t (s et t str)
s * n ou n * s	Concaténation de s, n fois (n int)
s[i]	$i^{ième}$ élément de s
s[0]	Premier élément de s
s[i:j]	Tranche située entre i et j dans s (j exclu)
s[i:j:p]	Tranche de i à j par pas de p (i,j,p int)
s[:]	Recopie la séquence en entier
len(s)	La longueur de s
min(s)	Le plus petit élément de s
max(s)	Le plus grand élément de s
<, >, ==, !=	Comparaison alphanumérique

Exemples :
```
>>> txt= """Ceci est un exemple
...de chaine sur
...plusieurs lignes."""
>>> txt
```

```
'Ceci est un exemple\nde chaîne sur\nplusieurs lignes.'
>>>type(txt)
<type 'str'>
>>> a = 'Bonjour '
>>> b = 'Toto'
>>> c = a + b
>>> c
'Bonjour Toto'
>>> len(c)
12
>>> min(c)
' '
>>>max(c)
'u'
>>>d= "Universite d'Evry Val d'Essonne"
>>>len(d)
31
>>>d[ :10]
Universite
>>>d[ :-7]
Universite d'Evry val d'
>>>d[-7:]
Essonne
>>>d[13:17]
Evry
>>>d[::2]           # prend un caractère sur deux de la chaine
"Uiest 'vyVldEsne"
```

Le type booléen

Le type `bool` a été introduit depuis Python 2.3, les constantes `True` et `False` avec python 2.2, avant 0 (faux) et 1 (vrai). Le type booléen dérive donc du type int pour des raisons de compatibilités.

2.4.3 Les autres types de données

Les séquences

Les séquences sont des données composites c'est à dire une collection de données de type de base. En python il existe deux grands types de séquences : les *séquences modifiables* et les *séquences non modifiables*.

- Les chaînes de caractères et les tuples sont des séquences non modifiables.
- Les listes sont des séquences modifiables.
- Toutes les opérations sur les listes non modifiables sont disponibles pour les séquences modifiables.

Nous venons de voir ces méthodes pour les chaînes de caractères (min, max, len, accès aux éléments d'une séquence, concaténation, x in s …), elles sont donc applicables aux autres séquences.

Les tuples

Les tuples sont des séquences non modifiables, ils sont notés entre parenthèses où les valeurs sont séparées par des virgules (les parenthèses sont facultatives mais recommandées).

```
>>> a = (1,2,3)
>>> type (a)
<type 'tuple'>
>>> a
(1, 2, 3)
```

Pour définir un tuple à un élément il faut impérativement mettre la virgule après l'élément.

```
>>> a=(1)           # les parenthèses sont facultatives…
>>> type(a)
<type 'int'>
>>> a = 1,          # … et la virgule est obligatoire
>>> type(a)
<type 'tuple'>
>>> a
(1,)
```

Les listes

Ce sont des séquences modifiables, elles sont notées entre crochets et les valeurs sont séparées par des virgules. Comme pour les tuples on peut mettre une virgule après le dernier élément. On accédera aux éléments d'une liste grâce aux fonctions accessibles aux séquences.

Exemple :

```
>>> l = [1,(2,3),'Toto']
>>> l
[1, (2, 3), 'Toto']
>>> len(l)
3
>>> jour = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi',
'samedi', 'dimanche']
>>> jour[2]
'mercredi'
```

Les opérations sur les listes

Elles sont communes à toutes les séquences modifiables .

Opérations sur séquences modifiables	*Résultat*
`s[i] = x`	Affecte `x` au i^{ième} élément de `s`
`s[i:j] = t`	Remplace la tranche `i:j` par la séquence `t`
`s += t`	Ajoute le contenu de la séquence `t` à fin de `s`
`u = s + t`	Renvoie une nouvelle séquence `u` dont le contenu est la concaténation de `s` et `t`
`del(s[j])`	Enlève le j^{ième} élément de la séquence `s`
`del(s[i:j])`	Enlève la tranche `i:j` de la séquence `s`
`s.append(elt)`	Ajoute `elt` à la fin de la séquence `s`
`range(d,f[,p])`	Crée une liste entiers de d à f (exclus) (pas p)

Exemple :

```
>>> del(jour[6])
>>> jour
['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi']
>>> append.jour('DIMANCHE')
>>> jour
['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi',
'DIMANCHE']
```

2.5 Les structures de contrôle

Point important de la syntaxe python :

- Un bloc démarre par le symbole :
- Toutes les instructions du bloc sont caractérisées par un même niveau d'indentation (il est conseillé d'insérer 4 espaces pour chaque nouveau niveau d'indentation).

2.5.1 Les tests

```
if cond1 :
     #bloc d'instructions à effectuer si cond1 est vraie
elif cond2 :
    #bloc d'instructions à faire si cond2 vraie (cond1 fausse)
else :
    # bloc d'instructions à faire si cond1 et cond2 fausses
```

2.5.2 Les boucles

Il n'existe que deux types de boucle en python : la boucle for et la boucle while (la boucle do while n'existe pas ; pas plus que la boucle do until.

La boucle while

Pour itérer tant qu'une condition est vérifiée

```
while condition :
     # bloc d'instructions corps de la boucle
```

Il existe deux façons de sortir prématurément d'une boucle (quelque soit son type)

`break` : permet de sortir de la boucle,

`Continue` : permet de passer immédiatement à l'itération suivante

La boucle for

Pour itérer sur les valeurs d'une séquence.

```
For variable in sequence :
    # bloc d'instructions corps de la boucle
```

Exercices du TD1