# Python 101

Ilyas Bambrik

Mars 2018

# Table des matières

# Objectifs

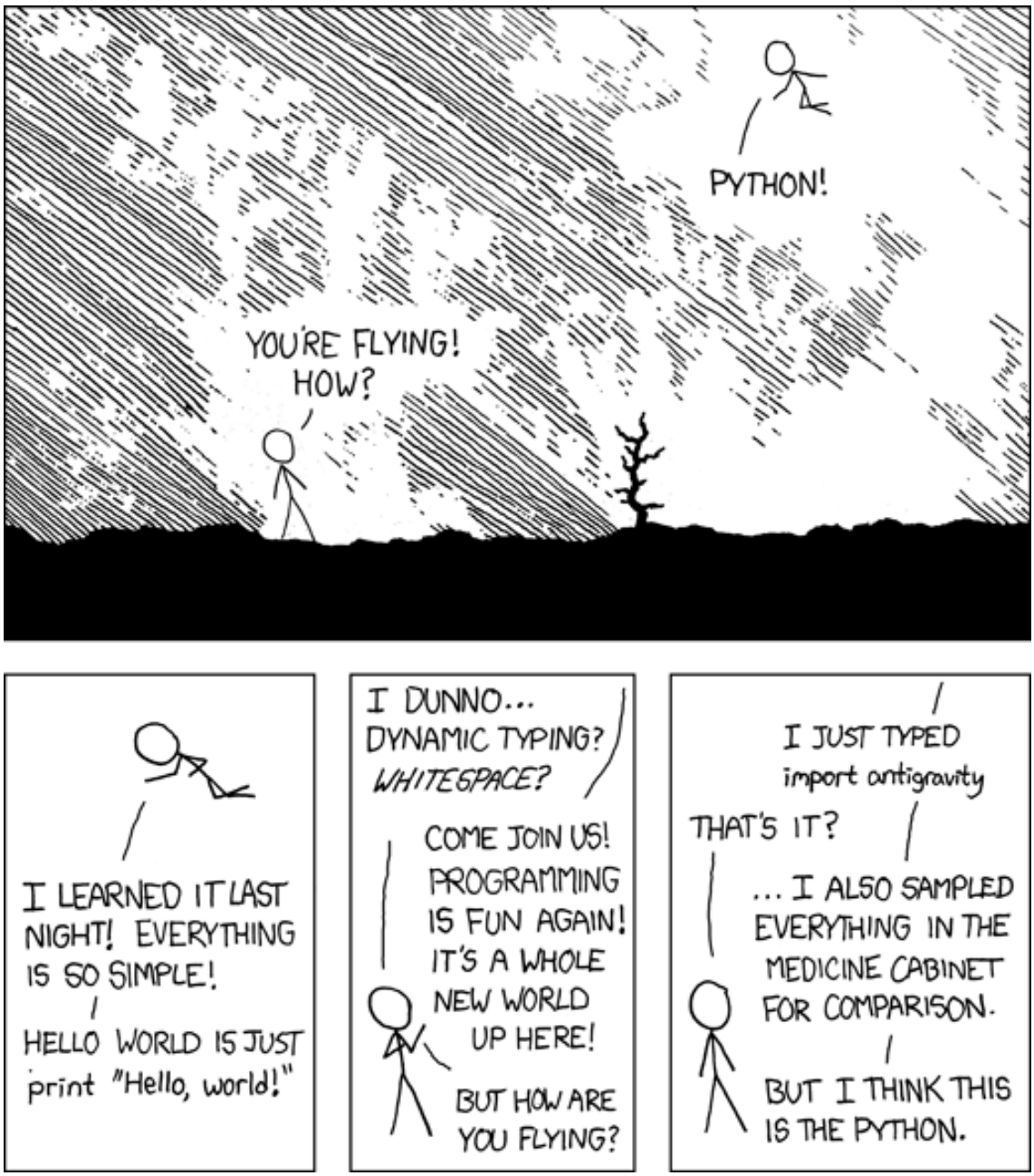
L’objectif de ce court tutoriel c'est de se familiariser avec la syntaxe de base du langage python.

# Introduction


Python est un langage de scripte extrêmement simple syntaxiquement et possédant une bibliothèque riche (regex, manipulation des matrices, combinaison et permutation, programmation web coté serveur,etc). Comme d'autres langages de script ( php et OTCL), python supporte la programmation orientée objet.

Pendant les dernières années, python a gagné en popularité due aux nombreux projets développés en python comme (par exemple OpenStack, Fsociety et MechanicalSoup).

Afin de vérifier que python est correctement installer, il suffit de vérifier que les fichiers sources python (*.py*) sont distingués par l'icône python.


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

# Pourquoi python

Python présente plusieurs avantages :

- Riche bibliothèque ;

# Syntaxe Python


Contrairement aux langages *static-typed* comme C et Java, les types des variables sont *dynamiquement* inférés lors de l'affectation. Ceci facilite énormément la programmation ;

*Listing 1* Simple programme python

```
1 print 'x='
2 x=int(raw_input())
3 print 'p='
4 p=int(raw_input())
5 print x,' puissance ',p,' =',x**p
```

- *print* est la fonction d'affichage sur écran comme *printf* et *System.out.print* (et *println*) ;
- *raw_input* permet la lecture d'une ligne de chaîne de caractères comme *scanf* (voir aussi *gets* en C) et *Scanner.next*;
- Afin de calculer x à la puissance p, l'opérateur ** est utilisé ;

## *Exemple*

*Listing 2* Simple calculatrice en python

```
1 #debut de la procedure 'calculatrice'
2
3 def calculatrice(nbr1, nbr2,operation):
4     if (operation == 'A'):
5         return nbr1+nbr2
6     elif (operation == 'S'):
7         return nbr1-nbr2
8     elif (operation == 'M'):
9         return nbr1*nbr2
10    elif (operation == 'D'):
11        if ( nbr2==0):
12            print "Division par zero"
13            return
14        return nbr1/nbr2
15    print "Operation indefini"
16 #fin de la procedure 'calculatrice'
17
18 print "Entrez la valeur de Nbr1="
19 nbr1=float(raw_input())
20
21 print "Entrez la valeur de Nbr2="
22 nbr2=float(raw_input())
23
24 print "\nEntrez le type d'operation\nA:\tAddition\nS:\tSoustraction\nM:
   \tMultiplication\nD:\tDivision\n"
```

```
25 op=raw_input()
26
27 print "Resultat= ", calculatrice(nbr1, nbr2,op)
28 raw_input()
```

- Il n'existe pas de commentaire multi-lignes en python et le caractère # marque le début d'un commentaire mono-ligne (comme en TCL et Bash/sh);
- La déclaration d'une procédure commence par le mot clé *"def"*. Comme dans le corps du programme, les types des arguments d'une procédure ne sont pas déclarés explicitement (il sont dynamiquement inférés selon le traitement effectué sur les arguments). Par exemple :
  *def NomProcedure* (argment1, argument2,argument3) *:*
- La clause *"switch"* n'existe pas dans la syntaxe python. Cependant, cette clause est remplacée par *"if-elif-else"*, pour le teste de multiple cas. En outre, une condition est formulée de la manière suivante :
  *if* ( *condition*) *:*

Par exemple, le programme présenté dans *Listing 3* illustre comment les conditions sont formulés

*Listing 3* Les conditions et opérateurs logiques

```
1  def Mention(note):
2      if(note <7):
3          return "F"
4      elif (note >=7 and note<10):
5          return "D"
6      elif (note >=10 and note<13):
7          return "C"
8      elif (note>=13 and note<15):
9          return "B"
10
11     return "A"
12
13
14 Note=int(raw_input())
15 print Mention(Note)
```

- Le type "char" n'existe pas en python. De plus les chênes de caractères peuvent être enfermer entre *double quottes* (") ou *single quottes* (') comme en *php*;
- Les opérateurs logiques binaires en python sont *and* et *or* à la place *&&* et // dans les langages C-type. L'opérateur *not* remplace l'opérateur de négation (*!*) . En outre, les valeurs logiques en python sont *True* et *False* (sensible à la case) ;

### *Remarque : Différences entre syntaxe Python 2 et Python 3*

Afin de tester les exemples proposés dans ce cours, il faut installer Pytnon2.7. En outre, il est important de signaler que la syntaxe de Python 2.7 est différente de celle de Python 3.5. Par exemple, la fonction *"raw_input()"* (présente dans python 2.x) est remplacé en python 3.x par la fonction *"input()"*.

## 1. Structure de bloques par tabulations

La principale innovation syntaxique en python est l'utilisation de tabulations. La tabulation dans un programme python indique le niveau du bloque et l'imbrication des instructions. Toute violation de cette règle syntaxique est notifié par un message d'erreur lors de l'interprétation. *Selon le nombre de tabulations séparant l'instruction du début de la ligne, l'interpréteur python infère le bloque où l'instruction est située*. L'imposition de cette convention syntaxique rend la lecture de codes sources facile. Par exemple, voir *Listing 4* :

*Listing 4* Structure de bloque en python

```
1 #Bloque~1
2 instruction1
3 instruction2
4 if (Condition1):
5     #Bloque~2
6     instruction3
7     instruction4
8     if (Condition1):
9         #Bloque~3
10        instruction5
11        instruction6
12    instruction7
13 else :
14    #Bloque~4
15    instruction8
16    instruction9
17 instruction10
18 instruction11
```

## 2. Casting de type en python

Pour effectuer le casting en python, la variable ou la valeur qu'on souhaite convertir est passée en paramètre au type voulu. Le casting en python se fait comme suite :

Variable1=*TypeVariable1*(Varible2D_UnAutreType)

ou bien

Variable1=*TypeVariable1*(ValeurD_UnAutreType)

Cette écriture est équivalente en C / Java à l'instruction suivante :

Variable1= (*TypeVariable1*) Varible2D_UnAutreType ;

Variable1= (*TypeVariable1*) ValeurD_UnAutreType ;

Dans le programme présenté dans *Listing 4* (ligne 28) la valeur lu du clavier (raw_input()), qui est une chaîne de caractères, est convertie en type *float* avant d'être affecté à la variable nbr1.

# Les boucles en python

L'usage de la clause for en python est quelque peut atypique. Au lieu de la déclaration traditionnelle, la boucle en python est déclarée comme une itération d'une variable sur un tableau (comme en Bash). Pour l'exemple présenté dans *Listing 5* (*ligne 4*) la variable "*element*" itère sur les éléments du tableau *tab*. Autrement dit, la variable "*element*" prend la valeur d'un élément de *tab* à chaque itération.

Autres particularités des tableaux en python :

- La déclaration d'un tableau se fait simplement par l'affectation à un tableau vide (*ligne 19*).
- Les éléments sont dynamiquement ajoutés par la méthode *append* (*ligne 22*).

Le programme présenté dans *Listing 5* comporte une procédure qui calcule la somme des valeurs des éléments d'un tableau *(ligne 2*) :

*Listing 5* La boucle *for*

```
1 import sys
2 def Somme (tab):
3     valeur_somme=0
4     for element in tab:
5         valeur_somme=valeur_somme+element
6     return valeur_somme
7
8 # Programme principale
9
10 print "Entrez le nombre d'elements du tableau"
11 Nombre_D_Elements=int(raw_input())
12
13 TableauD_Entier=[]
14 for i in range(Nombre_D_Elements):
15     print "TableauD_Entier[" + i + "]="
16     TableauD_Entier.append(int(raw_input()))
17
18 print "Somme = ",Somme(TableauD_Entier)
```

La fonction Somme suivante est la même fonction présentée dans *Listing 5*, écrite avec la boucle *while (ligne 4):*

*Listing 6* La boucle *while*

```
1 def Somme (tab):
2     valeur_somme=0
3     i=0
4     while(i<len(tab)):
5         valeur_somme=valeur_somme+tab[i]
```

```
6          i=i+1
7      return valeur_somme
```

L'invocation de la fonction *range(Nombre_D_Element)* (ligne 13- *Listing 5*) retourne un tableau de chiffre allant de *0* jusqu'à *Nombre_D_Element - 1*. Pour voir ce résultat, il suffit d'exécuter le programme suivant :

```
1 print range(10)
```

L'affichage de l'exécution précédente sera comme suite :

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

La fonction range() peut avoir de 1 jusqu'à 3 paramètres :

*range(N1,N2)* génère un tableau de nombre allant de *N1* jusqu'à *N2-1*

*range(N1,N2,step)* génère un tableau de nombre allant de *N1* jusqu'à *N2-1* un pas de valeur *step. Step peut être négatif*

*Listing 7* La fonction range()

```
1 print "Tableau de 10 a 20",range(10,20)
2
3 print "Tableau de 50 a 100 avec un pas de 15",range(50,100,15)
```

```
Tableau de 10 a 20 [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

Tableau de 50 a 100 avec un pas de 15 [50, 65, 80, 95]
```

# Les tableaux

IV

Dans le programme suivant, taille du tableau (*TailleTableau*, ligne 1) est lu à partir du clavier.

Ensuite, les éléments lus à partir du clavier sont ajoutés au tableau (*Tableau.append(Element)*, ligne5).

La fonction len() prend en paramètre un tableau et retourne la taille de celui-ci (ligne 8).

*Listing 8* Manipulation des tableaux

```
1 TailleTableau=int(raw_input())
2 Tableau=[]
3 for i in range(TailleTableau):
4     print "Tableau[",i,"]="
5     Tableau.append(int(raw_input()))
6 print "Tableau=",Tableau
```

```
Taille Tableau=
4
Tableau[ 0 ]=
12
Tableau[ 1 ]=
6
Tableau[ 2 ]=
75
Tableau[ 3 ]=
-2
Tableau= [12, 6, 75, -2]
Taille tableau= 4
```

- Comme dans la fonction range() présenté précédemment, l'indice de l'élément tableau supporte jusqu'à 3 paramètres (testez l'exemple présenté dans le listing suivant) :
- Tableau [i] : désigne le i$^{em}$ élément du tableau *si (i>=0)*. Si *i<0* , *Tableau [i] désigne le i$^{em}$ élément avant dernier*.
- Tableau [N1 : N2] : désigne la séquence d'éléments du tableau allant de l'indice *N1* jusqu'à *N2*.
- Tableau [N1 : N2 :step] : désigne la séquence d'éléments du tableau allant de l'indice *N1* jusqu'à *N2* avec un pas = step. Par défaut, Tableau[ : : ] est équivalente à *Tableau[0 :len(Tableau) :1]* .

La fonction *split*() (*équivalente à la fonction split implémenté dans les String en Java*) découpe une chaîne de caractère en morceaux selon le délimiteur (l'espace dans ce cas) et renvoie un tableau des éléments séparés par le *delimiter* :

*Listing 9* Les slicers

```
Tableau=raw_input().split(" ")
print "Tableau = ", Tableau
print "Tableau [-4] = ", Tableau[-4]
print "Tableau [3 : 5] = ", Tableau[3:5]
print "Tableau [5:2:-1]= ", Tableau[5:2:-1]
print "Tableau [::]= ", Tableau[::]
```

```
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150
Tableau =  ['10', '20', '30', '40', '50', '60', '70', '80', '90', '100', '110', '120', '130', '140', '150']
Tableau [-4] =  120
Tableau [3 : 5] =  ['40', '50']
Tableau [5:2:-1]=  ['60', '50', '40']
Tableau [::]=  ['10', '20', '30', '40', '50', '60', '70', '80', '90', '100', '110', '120', '130', '140', '150']
```

*Conseil*

La concaténation des listes se fait par une simple opération + :

`[1,2,3,4,5] + [6,7,8,9] = [1,2,3,4,5,6,7,8,9]`

# 1. Les dictionnaires

Un dictionnaire est une collection qui supporte comme indice le une chaîne de caractère (comme en php) :

```
UnDictionnaireVide={} # initialisation d'un dictionnaire vide
UnDictionnaire ={ "Nom":"Mohammed","Specialite":"Informatique","Age":25}
print UnDictionnaire["Specialite"]
print UnDictionnaire.has_key("Adresse")
print UnDictionnaire.keys()
print UnDictionnaire.values()
```

L'exécution du programme précédant affiche le résultat suivant :

```
Informatique
False
['Nom', 'Age', 'Specialite']
['Mohammed', 25, 'Informatique']
```

# Les classes et objets en python

V

Python supporte de même les concepts de la POO. Le programme présenté dans *Listing III.6* présente une classe TypeJoueur :

*Listing 10* Les classes et objets en python

```
1 class TypeJoueur:
2     def __init__(self):
3         self.PointsDeVie=100
4         self.Score=0
5         self.NomJoueur=''
6 
7     def lireNomJoueur(self):
8         self.NomJoueur = raw_input()
9 
10    def mettreAJourScore(self,scoreRecompense):
11        self.Score=self.Score+scoreRecompense
12 
13    def afficherStatisticJoueur(self):
14        print "NomJoueur=",self.NomJoueur,"\nPoints De Vie=",self.PointsDeVie,
  "\nScore=",self.Score
15 
16 Med=TypeJoueur()
17 
18 Med.lireNomJoueur()
19 Med.mettreAJourScore(10)
20 Med.afficherStatisticJoueur()
```

- La méthode *def __init__(self):* représente le constructeur de la classe en python. En outre, les attributs de la classe sont déclarés à l'intérieur du constructeur __init__.
- Le mot clé *self*, passé en argument dans tous les déclarations de méthode de la classe TypeJoueur, représente l'objet qui a invoqué la méthode ou le constructeur. Cependant, lors de l'appel d'une méthode, cet objet *self* est automatiquement inféré. Par exemple : Med=TypeJoueur()# est équivalente à Med=TypeJoueur(*Med*). Donc l'objet courant (self) prendra la référence de l'objet qui a invoqué la méthode Med.mettreAJourScore(10) # de même cette instruction est équivalente à Med. mettreAJourScore(*Med*,10)
- L'appel du constructeur (*ligne 16*) se fait sans opérateur *new* contrairement à la syntaxe Java.

### *Exemple : Constructeur paramétré*

Si on souhaite ajouter un constructeur avec des arguments *pointsvie*, *score* et *nom* ,il suffit d'ajouter le constructeur suivant :

*Listing 11* Déclaration d'un constructeur paramétré en python

```
1     def __init__(self,pointsvie,score,nom):
2         self.PointsDeVie=pointsvie
3         self.Score=score
4         self.NomJoueur=nom
```