Introduction au langage Python 3

I. Introduction

Le langage de programmation Python est un très bon choix aussi bien pour l'initiation à la programmation que pour la programmation elle-même. C'est un langage de très haut niveau dont la syntaxe encourage à écrire du code clair et de qualité. Dans le domaine de la gestion de la mémoire, nombre de détails de bas niveau propres aux langage comme le C disparaissent.

De plus l'apprentissage de Python est facilité par l'existence d'une interface interactive. Cela dit son intérêt ne se réduit pas à l'apprentissage de la programmation ou de l'algorithmique ; en témoigne sa popularité croissante. Il a été choisi par des acteur majeurs : Google, YouTube, la NASA, etc.

Techniquement parlant, Python est un langage ou l'on peut choisir plusieurs styles de programmation. Il favorise la programmation impérative structurés et la programmation orientés objet ; dans une moindre mesure, il permet de programmer dans un style fonctionnel. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions. C'est un langage multiplateforme, polyvalent, open source et gratuit.

1. *Que peut-on faire avec Python ?*

Beaucoup de choses !

- Du calcul scientifique (librairie NumPy)
- Des graphiques (librairie matplotlib)
- Du traitement du son, de la synthèse vocale (librairie eSpeak)
- Du traitement d'image (librairie PIL), de la vision artificielle par caméra (framework SimpleCV)
- De la bio-informatique (librairie Biopython)
- Des applications avec interface graphique GUI (librairies Tkinter, PyQt, wxPython, PyGTK...)
- Des jeux vidéo en 2D (librairie Pygame)
- Des applications multi-touch (framework kivy pour tablette et smartphone à écran tactile)
- Des applications Web (serveur Web Zope ; frameworks Web Flask, Django)
- Interfacer des systèmes de gestion de base de données (librairie MySQLdb...)
- Des applications réseau (framework Twisted)
- Communiquer avec des ports série RS232 (librairie PySerial), en Bluetooth (librairie pybluez)...
- etc…

2. *Ou peut-on écrire des programmes ?*

Anaconda est un environnement de développement intégré (IDE en anglais : Integrated Development Environment) pour Python.

Anaconda propose un certain nombre d'outils :

- Un éditeur de texte (pour écrire le programme)
- Un interpréteur (pour exécuter le programme)
- Un débogueur (pour tester le programme)

Il existe d'autres IDE pour Python : Eclipse/Pydev, Eric Python IDE, Spyder ...

II. Généralités

1. Expressions

Évaluation des données ou faire les opérations arithmétiques.

⇒ ***Exemple :***

- 1 + 4 * 3
- a+b*4

Opérateurs arithmétiques que nous utiliserons :

Opérateur	description	exemple
+ - * /	addition, soustraction, multiplication, division	A+B ; A*B ; A-B ; A/B
%	modulo, le reste de la division	**4%2 (retourne 0)** **3%2(Retourne une valeur différent de 0**
**	Exponentiation	X**2 (X^2)

** / % ** Ont une priorité plus élevée que + -*

3. Division des entiers

Lorsque nous divisons des entiers avec //, le quotient est également un entier.

⇒ ***Exemples :***

- 35 // 5 =7
- 84 // 10 = 8
- 156 // 100 = 1

L'opérateur % calcule le reste à partir d'une division d'entiers.

4. Fonctions mathématiques

Python possède des fonctions utiles pour effectuer des calculs.

Fonction	Description
`abs(`***value***`)`	La valeur absolue
`ceil(`***value***`)`	Le plus petit entier qui est supérieur ou égal au nombre donné.
`cos(`***value***`)`	Cosinus, en radians
`floor(`***value***`)`	Renvoie le plus grand entier
`log(`***value***`)`	logarithme, base e
`log10(`***value***`)`	logarithme, base 10
`max(`***value1***`, `***value2***`)`	Plus grande de deux valeurs
`min(`***value1***`, `***value2***`)`	Plus petite de deux valeurs
`round(`***value***`)`	donne l'arrondi à la précision demandée
`sin(`***value***`)`	sinus, en radians
`sqrt(`***value***`)`	la racine carrée
e	2.7182818...
pi	3.1415926...

Pour utiliser les fonctions mathématiques il faut inclure dans votre programme la bibliothèque mathématique : ***from math import ****

5. Variables

Une variable est une donnée de votre programme, stockée dans votre ordinateur. C'est un code alphanumérique que vous allez lier à une donnée de votre programme, afin de pouvoir l'utiliser à plusieurs reprises et faire des calculs un peu plus intéressants avec.

5.1. Syntaxe

En Python, pour donner une valeur à une variable, il suffit d'écrire **nom_de_la_variable = valeur**.

```
>>> age=19
```

Une variable doit respecter quelques règles de syntaxe incontournables :

- Le nom de la variable ne peut être composé que de lettres, majuscules ou minuscules, de chiffres et du symbole souligné « _ » (appelé underscore en anglais).
- Le nom de la variable ne peut pas commencer par un chiffre.
- Le langage Python est sensible à la casse, ce qui signifie que des lettres majuscules et minuscules ne constituent pas la même variable (la variable **`AGE`** est différente de **`aGe`**, elle-même différente de **`age`**).

Certains mots-clés de Python sont ***réservés****, c'est-à-dire que vous ne pouvez pas créer des variables portant ce nom.*

And, del, from, none, true, as, elif, global, nonlocal, try, assert, else, if, not, while, break, except, import, or, with, class, false, in, pass, yield, continue, finally, isn raise, def, for, lambda *et* ***return***

5.2. Type de données

Python est un langage dont le typage est automatique. Cela signifie que bien que gérant différents types, lorsqu'une variable est affectée, l'interpréteur trouvera automatiquement son type.

Liste des types	
int	Nombre entier optimisé
long	Nombre entier de taille arbitraire
float	Nombre à virgule flottante
complex	Nombre complexe
str	Chaîne de caractère
unicode	Chaîne de caractère Unicode
tuple	Liste de longueur fixe
list	Liste de longueur variable
dict	dictionnaire
file	Fichier
bool	Booléen
NoneType	Absence de type
NotImplementedType	Absence d'implémentation
function	fonction
module	module

Liste des types prédéfinis en Python

La fonction **type()** permet de connaître le type d'une variable

```
>>> a=4
>>> type(a)
<class 'int'>
```

6. *Fonction print()*

La fonction print permet d'afficher un message, le contenu d'une variable ou bien les deux.

Pour afficher une chaîne de caractères

```
>>> print("bonjour monde")
bonjour monde
```

On peut aussi affecter à une variable une chaîne de caractères.

```
>>> abc = "deux mots"
>>> print(abc)
deux mots
```

Puis un exemple un peu plus complexe utilisant la commande `print`, qui sait gérer différents types de paramètres pour les afficher sur une même ligne si on les sépare avec des virgules.

```
>>> a = 9
>>> print("le carré de ", a, " est ", a**2)
le carré de 9 est 81
```

7. *Fonction input()*

La fonction **input()** fonction provoque une interruption dans le programme courant. L'utilisateur est invité à entrer des caractères au clavier et à terminer avec *<Enter>*. Lorsque cette touche est enfoncée, l'exécution du programme se poursuit, et la fonction fournit en retour une chaîne de caractères correspondant à ce que l'utilisateur a entré. Cette chaîne peut alors être assignée à une variable quelconque, convertie, etc.

On peut invoquer la fonction **input()** en laissant les parenthèses vides. On peut aussi y placer en argument un message explicatif destiné à l'utilisateur. Exemple :

```
prenom = input("Entrez votre prénom : ")
print("Bonjour,", prenom)
```

*La fonction **input()** renvoie toujours une chaîne de caractères(25). Si vous souhaitez que l'utilisateur entre une valeur numérique, vous devrez donc convertir la valeur entrée (qui sera donc de toute façon de type string) en une valeur numérique du type qui vous convient, par l'intermédiaire des fonctions intégrées **int()** (si vous attendez un entier) ou **float()** (si vous attendez un réel). Exemple :*

```
>>> a = input("Entrez une donnée numérique : ")
Entrez une donnée numérique : 52.37
>>> type(a)
<class 'str'>
>>> b = float(a)     # conversion de la chaîne en un nombre réel
>>> type(b)
<class 'float'>
```

8. Structure conditionnelle IF

8.1. Structure if

Exécute un groupe d'instructions uniquement si la condition est vraie. Sinon, les instructions sont ignorées.

⇒ ***Syntaxe :***

```
If condition:
        instructions
```

⇒ ***Exemple :***

```
age= 20
if age> 17:
        print ”bienvenue."
```

8.2. if/else

Exécute un bloc d'instructions si la condition est Vraie et un second bloc d'instructions si elle est fausse.

⇒ ***Syntaxe :***

```
if condition:
        instructions
else:
        instruction
```

⇒ ***Exemple :***

```
note= 12
if note >= 10:
        print ”admis!"
else:
        print ”non admis."
```

Plusieurs conditions peuvent être enchaînées avec elif ("else if"):

```
if condition1:
        instructions
elif condition2:
        instructions
else:
        instructions
```

8.3. Opérateurs logique

Les opérateurs logiques :

Opérateur	description	Exemple	résultat
==	égal	1 + 1 == 2	True
!=	Non égal	3.2 != 2.5	True
<	Inférieur strictement	10 < 5	False
>	Supérieur strictement	10 > 5	True
<=	Inférieur ou égal	126 <= 100	False
>=	Supérieur ou égal	5.0 >= 5.0	True

Opérateurs de comparaison

Les expressions logiques peuvent être combinées avec des opérateurs logiques :

Opérateur	Exemple	Résultat
and	9 != 6 and 2 < 3	True
or	2 == 3 or -1 < 5	True
not	not 7 > 0	False

Opérateurs logiques

9. *Boucle for*

Lorsque l'on souhaite répéter un nombre **donné** de fois la même instruction ou le même bloc d'instructions, la commande `for` est la plus appropriée.

⇒ ***Syntaxe :***

for *nomvariable* **in** *groupedevaleurs*:
    Instructions

Groupedevaleur ça peux être ***Range, List, Tuple, Dictionnaire, une chaine de caractères ou un fichier*** *;*

9.1. Range :

La fonction range spécifier un intervalle de valeurs entiers :

- ***range(fin)***

Liste d'entiers entre 0 et fin(non inclus)

```
>>> for x in range(2):
        print(x)
0
1
```

- ***range(debut, fin)***

Liste d'entiers entre debut (inclus) et fin (non inclus)

```
>>> for x in range(2,4):
        print(x)
2
3
```

- ***range(debut, fin, pas)***

Liste d'entiers entre **debut** (inclus) et **fin** (non inclus) avec le **pas**

```
>>> for x in range(2,7,2):
        print(x)
2
4
8
```

⇒ ***Exemple :***

Puis terminons sur un exemple classique qui est le calcul de la somme des premiers entiers. Disons ici que l'on s'arrête à 30. Autrement dit, on veut calculer $1 + 2 + 3 + \cdots + 30$

```
>>> for i in range(1, 31):   # pour i allant de 1 à 30
...             S = S + i
...    print(S)
...
465
```

9.2. Liste :

Afficher les éléments d'une liste ;

```
>>> L=[6,9,2,10]
>>> for i in L :
            print(i)
...
6
9
2
10
```

9.3. Tuple :

Afficher les éléments d'un Tuple ;

```
>>> L=(6,9,10)
>>> for i in L :
            print(i)
...
6
9
10
```

9.4. Chaine de caractères

Afficher les caractères d'une chaine

```
>>> S= 'salut'
>>> for i in S :
            print(i)
...
s
a
l
u
t
```

9.5. Dictionnaire :

Parcourir les clés d'un dictionnaire :

```
>>> D={'nom' :'essaddouki','penom' :'mostafa'}
>>> for i in D :
        print(i)
...
nom
prenom
```

Pour afficher les valeurs :

```
>>> D={'nom' :'essaddouki','penom' :'mostafa'}
>>> for i in D :
        print(D[i])
...
essaddouki
mostafa
```

9.6. Fichier :

Parcourir le contenu d'un fichier ligne par ligne :

```
>>> f=open('psi.txt')
>>> for ligne in f :
        print(ligne)
```

10. Boucle while

Le principe de la boucle while, c'est d'exécuter un bloc d'instructions tant que (while in english) une condition donnée est vraie.

⇒ ***Exemple :***

```
>>> i = 1
>>> while i <= 5:
...         print(i)
...         i = i + 1
...
1
2
3
4
5
```