LYCÉE FRANÇAIS SAINT-EXUPÉRY DE BRAZZAVILLE
LABORATOIRE DE MATHÉMATIQUES

LYCÉE FRANÇAIS
Saint
Exupéry
CONGO BRAZZAVILLE

STAGE DE FORMATION

PARTIE I

Python : des fondamentaux à la programmation

par

Valérien Eberlin

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

aefe
Agence pour l'enseignement français à l'étranger

Année 2023 - 2024

1 Les variables

En mathématiques, la variable apparaît dans des formules comme celle du périmètre d'un cercle ou l'expression symbolique des fonctions : $p = 2\pi R$, $2x - 5 = 8 - 5y$ …

En informatique, on est amené à écrire des instructions comme x = x+1 qui sont d'une nature totalement différente : il ne s'agit pas là d'une égalité, ni d'une équation, mais d'une instruction d'affectation, qui va modifier le contenu de la variable x. On lit cette instruction par « `x` reçoit la valeur `x+1` » et surtout pas « `x` égale `x+1` ».

Une variable est donc en informatique **un conteneur dans lequel tu peux stocker différents types de données**, comme des nombres, des chaînes de caractères ou même des objets plus complexes.

Un modèle rigoureux de la variable informatique consiste à dire qu'une variable est une étiquette collée sur une boîte qui peut contenir différentes valeurs. Le contenu de chaque boîte varie au cours de l'exécution d'un programme (ce qui n'est pas le cas d'une variable mathématique).

Règles pour nommer les variables Python

Il y a quelques règles importantes à suivre lors de la déclaration de variables en Python :

- les noms de variables ne peuvent contenir que des lettres, des chiffres et des underscores ou tiret bas (_) ;
- ils doivent commencer par une lettre (majuscule ou minuscule) ou un underscore, mais ne peuvent pas commencer par un chiffre ;
- les noms de variables sont sensibles à la casse, ce qui signifie que nom et Nom seraient deux variables distinctes ;
- il faut éviter d'utiliser des noms de variables déjà réservés par Python, tels que `or`, `print`, `if`, `else`, etc.

Les différents types de données des **variables** Python

En Python, les types des objets que l'on utilisera le plus fréquemment sont les suivants :

a) `int` : ils représentent les nombres entiers, positifs ou négatifs, comme 42 ou –10
b) `float` : ils représentent les nombres à virgule flottante tels que 3.14
c) `str` (*string* ou chaînes de caractères). Elles représentent du texte et doivent être entourées de guillemets simples ou doubles, par exemple, "Bonjour"
d) `bool` (*boolean*) : booléens (valeur `True` ou `False`).

On peut connaître le type d'un objet à l'aide de la fonction `type()`.

Exemple

```
1 type(5)
```
```
int
```

```
1 type(7.2)
```
```
float
```

```
1 type("Le résultat")
```
```
str
```

```
1 type(1 == 0)
```
```
bool
```

2 Les instructions de base

a) La fonction `print()`

La fonction `print()` permet d'afficher ce qui est inclus entre parenthèses.

Exemples

```
1 print("Hello")
```
```
Hello
```

```
1 var1 = 5
2 var2 = 8
3 print(var1 + var2)
```
```
13
```

```
1 var1 = "Bon"
2 var2 = "jour"
3 print(var1 + var2)
```
```
Bonjour
```

Attention à ne pas confondre `print(a)` et `print("a")`.
`print(a)` affiche la valeur de la variable `a` (si elle existe).
`print("a")` affiche le string `a`.

Exercice 1

```
var3 = "1"
var4="5"
```

Que renvoie `var3 + var4` ?

Exercice 2

1. Quel est le type de 1.0 ?
2. Qu'affiche l'instruction `print(0.1 + 0.2 == 3)` ? Pourquoi ?
3. Que renvoie l'instruction `"andré" < "albert"` ?
4. Qu'affiche l'instruction `print("125"<"2"<"a"<"b")` ? Pourquoi ?

Exercice 3

Qu'affichent les instructions suivantes ?

1) print("1+")
2) print(1+)

b) La fonction `input`

Pour permettre l'interaction du programme avec l'utilisateur, par exemple la saisie de la valeur d'une variable, il faut utiliser l'instruction `input`. L'utilisateur est alors invité à saisir des caractères et à terminer avec la touche « Enter ». On peut avec la fonction `input`, ajouter un message pour l'utilisateur en le mettant entre parenthèse et entre des guillemets.

Exercice 4

```
1 input("Quel est ton nom ?")
Quel est ton nom ?
```

a) Compléter ce programme pour qu'il affiche « `mon nom est Jean` » lorsqu'on complète la fenêtre de saisie.
b) Que se passe-t-il si l'on saisit un entier ou un flottant au lieu d'un string ?

Attention

Par défaut, la commande « input » permet à l'utilisateur de saisir un string.
Si on veut que l'utilisateur saisisse un entier, il faut l'indiquer en faisant précéder `input` de `int`. Par contre, si l'on veut que l'utilisateur saisisse un flottant, il faut l'indiquer en faisant précéder `input` de `float`.

Exercice 5

```
1 int(input("quel est ton âge ?"))
quel est ton âge ?
```

Tester ce programme pour des entiers.
Que se passe-t-il si l'on saisit un flottant au lieu d'un entier ?

Exercice 6

```
1 float(input("Quel est le résultat de 5/2 ?"))
Quel est le résultat de 5/2 ?
```

Tester ce programme pour des flottants.
Que se passe-t-il si l'on saisit un entier au lieu d'un flottant ?

Exercice 7

Que se passe-t-il quand on exécute le programme suivant ?

```
a= input("saisir un entier")
print("l'entier suivant est", a+1)
```

Rectifier le programme si nécessaire.

Exercice 8

Écrire un programme qui demande à l'utilisateur les longueurs (entières) des deux côtés d'un rectangle et affiche son aire.

Exercice 9

Ecrire un programme qui demande à l'utilisateur d'entrer un nombre de secondes et qui l'affiche sous forme d'heures / minutes / secondes.

On pourra utiliser le tableau des symbole en Python ci-après :

Opérateur	Symbole en python	Exemple d'expression	Type de l'expression	Valeur de l'expression
Addition	+	>>> 4 + 5.0	float	9.0
Soustraction	-	>>> 8 - 11	int	-3
Multiplication	*	>>> 7 * 6	int	42
Division	/	>>> 14 / 2	float	7.0
Puissance	**	>>> 2 ** 3	int	8
Quotient de la division	//	>>> 15 // 2	int	7
Reste de la division	%	>>> 15 % 2	int	1
Différent	!=	>>> 15 != 3	bool	True

c) Les commentaires

Le croisillon # permet de faire figurer dans le corps du programme un commentaire qui ne sera pas pris en compte lors de son exécution.
porte sur le reste de la ligne.

d) Les majuscules ou les minuscules

Les instructions Python s'écrivent en minuscules. On peut utiliser des majuscules dans les noms de variables. Par contre, il faudra l'écrire **exactement** de la même façon dans la suite du programme.

3 La fonction `def`

La fonction `def` associe une séquence d'instructions à un nom.

Exemple

```
from turtle import*

def carre():
    for i in range(4):
        forward(100)
        left(90)
    done()
```

Pour la construction de figures géométriques, on importe d'abord le *module Turtle* avec l'instruction `from turtle import*`

Pour exécuter les instructions de la fonction `carre`, on appelle la fonction avec la syntaxe suivante : `carre()` .

L'appel à la fonction `carre()` produit le même effet que si l'on avait exécuté le bloc d'instructions suivant :

```
for i in range(4):
    forward(100)
    left(90)
```

Remarques

Dans la fonction `carre` ci-dessus, la longueur du carré est fixée à 100.
Pour construire des carrés de longueur quelconque, on peut ajouter un argument à notre fonction.

```
from turtle import*

def carre(x):
    for i in range(4):
        forward(x)
        left(90)
    done()
```

Pour construire un carré de coté 150, on appelle la syntaxe : `carre(150)`.

Exercice 10

On considère le programme ci-dessous où la fonction `figure` a pour argument x, y, z.

```
from turtle import*

def figure(x,y,z):
    for i in range(x):
        forward(y)
        left(z)
    done()
```

a) Quels sont les valeurs de x, y et z pour que le programme construise un triangle équilatéral de côté 120.

b) Quel serait les valeurs de x, y et z pour que le programme construise un hexagone régulier de 120 pixels de côté ?

Exercice 11

Tester le programme suivant à l'aide de l'outil EduPython puis, chercher à comprendre en échangeant avec vos collègues, la construction obtenue.

```
from turtle import *
color('red')
begin_fill()
for i in range(4):
    forward(50)
    left(90)
end_fill()
done()
```

Exercice 12

Tester le programme suivant à l'aide de l'outil EduPython puis, chercher à comprendre en échangeant avec vos collègues, la construction obtenue.

```
from turtle import *
color('red', 'yellow')
begin_fill()
while True:
    forward(200)  # avancer de 200 pixels
    left(170)   # tourner à gauche de 170 pixels
    if abs(pos()) < 1:  # si la position de la tortue au point (0 ,0) inférieur
        break
end_fill()
done() # lance la construction

```

Exercice 13

Construire le drapeau de la République du Congo à l'aide du module turtle.
Pour cela, on pourra utiliser quelques instructions du tableau ci-dessous.

`reset()`	effacer tout et recommencer
`goto(x,y)`	aller au point de coordonnées (x,y)
`Shape("turtle")`	faire apparaitre le lutin tortue
`forward(x)`	avancer de x
`backward(x) ou bd(x)`	reculer de x
`up(), down()`	lever ou baisser le crayon
`left(θ) ou lt(θ)`	tourner à gauche de θ degrés
`right(θ) ou rt(θ)`	tourner à droite de θ degrés
`color('red')`	couleur en rouge
`width(x)`	épaisseur du stylo
`circle(R)`	trace un cercle de rayon R
`done() ou mainloop()`	lance la construction
`begin_fill()`	démarre les tracés de remplissage
`end_fill()`	termine et remplissage des tracés

Les fonctions Python et les fonctions mathématiques

Les fonctions Python peuvent aussi représenter des fonctions mathématiques qui calculent des valeurs.

Exemple

```
def f(x):
    return 3*x-4
```

Pour appeler une telle fonction et obtenir son résultat pour x=5, on utilise la syntaxe :
……

4 Instruction conditionnelle

L'instruction conditionnelle `if` permet de soumettre l'exécution d'une instruction ou d'un bloc de codes à une condition.
En plus d'un bloc à n'exécuter que lorsque l'instruction est vérifiée, une instruction `if` peut contenir un bloc alternatif à n'exécuter que dans le cas contraire, introduit avec l'instruction : `else`.

Exemple

```
if n >0:
    print("c'est un nombre postif")
else:
    print("c'est un nombre négatif ou nul")
```

Remarque

Il est possible de proposer trois branches conditionnelles ou plus. Après une première branche conditionnelle `if`, chaque branche suivante peut être ajoutée avec sa propre condition grâce au mot-clé `elif`. L'ensemble pouvant être à nouveau complété d'une ultime branche `else`.

```
if condition1:
    bloc d'instruction 1
elif condition2:
    bloc d'instruction 2
elif condition3:
    bloc d'instruction 3
...
else dernière condition:
    dernière bloc d'instruction
```

Exemple

```
if n < 0:
    print("négatif")
elif n == 0:
    print("nul")
elif n <=2:
    print("petit")
else:
    print("grand")
```

L'ordre dans lequel apparaissent les conditions compte : seule la première branche dont la condition est vraie est exécutée. Python ne considère donc la deuxième branche que si la première n'a pas été sélectionnée, la troisième branche que si ni la première branche, ni la deuxième n'ont été retenues, et ainsi de suite. Ainsi dans le programme précédent, si la variable `n` a la valeur -3, alors la branche introduite par `elif n <=2 :` ne sera pas prise en compte bien que -3 soit inférieur à 2, car la première branche aura déjà été choisie.

Les tests booléens

- a !=0 : la condition « a différent de zéro » s'écrit « a !=0 ».
- a==0 : la condition « a égal à zéro » s'écrit « a==0 », (avec deux fois le symbole =).
- < et > : ces symboles désignent les inégalités strictes.
- <= et >= : ces combinaisons de symboles désignent les inégalités larges.
- and : permet d'effectuer une instruction si deux tests sont vérifiés simultanément.
  On peut sous Python écrire comme condition : `if 2 <= x < 5` : au lieu de `if 2 <= x and x < 5 :`
- or : permet d'effectuer une instruction si au moins un test sur deux est vérifié.

Exercice 14

Ecrire un programme qui demande un entier n à l'utilisateur et affiche tous les diviseurs.

Exercice 15

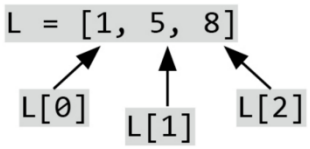
Écrire un programme qui demande trois nombres `a, b` et `c` à l'utilisateur et qui affiche : « plus petit » si `c` est plus petit que les deux autres, « plus grand » si `c` est plus grand que les deux autres, et « entre les deux » si `c` est compris entre les deux autres (sans présager de qui parmi a ou b est le plus petit).

Les listes

Une liste est constituée de termes séparés par une virgule et entourés de crochets.

- Le premier élément de la liste est `L[0]`, le 2e est `L[1]`, ...
- Sa longueur est donnée par `len(L)`
- Si les éléments de la liste sont comparables, le maximum est donné par `max(L)`, le minimum par `min(L)`
- `L=[]` permet de définir une liste vide
- Si L est une liste, l'instruction `L.append(x)` va ajouter l'élément `x` à la liste L.

Exemple



L'instruction `L[2]=14` remplace le troisième élément par la valeur 14. La liste L devient alors `L = [1, 5, 14]`.

Remarque

On peut aussi lire les éléments d'une liste en partant du dernier élément. Dans ce cas : `L[-1]` désigne le dernier élément, `L[-2]` l'avant dernier élément , ...

Définition

Pour créer une liste en compréhension, on utilise la la syntaxe for i in range() suivie éventuellement d'une condition.

Exercice 16

```
L3=[i**2 for i in range(5) if i**2 % 2 ==0]
```

Qu'affiche L3 ?

Exercice 17

Ecrire un programme qui calcule la moyenne d'une liste L :

a) en utilisant la fonction native `sum`

b) sans utiliser la fonction native `sum`

Exercice 18

Écrire un programme qui demande trois entiers à l'utilisateur, puis affiche ces trois entiers dans l'ordre croissant.

Exercice 19

Écrire un programme qui demande une année à l'utilisateur et indique s'il s'agit d'une année bissextile. On rappelle qu'une année est bissextile si elle est multiple de 4 mais pas multiple de 100, ou si elle est multiple de 400.

Exercice 20

Définir une fonction `test_pythagore` qui prend trois entiers a, b et c en arguments et renvoie un booléen indiquant si $a^2 + b^2 = c^2$.

Exercice 21

Ecrire une fonction qui cherche un élément `x` (entier) dans une liste d'entiers. La fonction doit renvoyer le booléen `True` si `x` se trouve dans la liste, `false` sinon.

Exercice 22

Ecrire une fonction minimum qui prend en argument une liste d'entiers et qui renvoie la plus petite valeur contenue dans la liste. Tester cette fonction sur une liste d'entiers.

5 Boucles non bornées

La boucle `while` est une séquence d'instructions qui se répète. Dans certains cas, la boucle `for` ne convient pas car on ne sait pas combien de tours de boucles on doit effectuer. La boucle `while` présente l'avantage par rapport à la boucle `for` d'effectuer une boucle sans en connaître à l'avance le nombre d'itérations : elle est non bornée.

Syntax Python

```
while condition :
        bloc  d'instructions
```

Remarque

Contrairement aux boucles `for` (inconditionnelles), les boucles `while` (conditionnelles) ne s'arrêtent qu'une fois la condition non remplie (test faux). Ainsi, une boucle `while` mal écrite peut engendrer une boucle infinie et il faut alors interrompre l'exécution du programme.

Exercice 23

J'ai planté en 2019 un sapin Noël qui mesure 1,20 m. Il grandit de 30 cm par an.
Je veux le couper quand il dépassera 5 m. En quelle année devrai-je le couper ?

Exercice 24

On donne : $\begin{cases} u_0 = 0 \\ u_{n+1} = 2u_n + 1, \text{pour tout } n \in \mathbb{N} \end{cases}$

On cherche les termes d'une suite (u_n) qui sont strictement inférieurs à 10.