Langage Python 3 – Mémo

I – Entrées, Sorties et Variables

1) Sorties

Pour permettre au programme en cours d'exécution d'afficher un texte ou un nombre on utilise la commande **print**.
Exemples :

```
print("Bonjour !")
```

```
print(2)
```

```
print(1)
print(2)
print(3)
```

Affichage :
1
2
3

```
a = -3
print("Le carré de", a,"est", a * a)
```

```
print(1, end = " ")
print(2, end = " ")
print(3, end = " ")
```

Affichage : 1 2 3

2) Entrées

Afin de pouvoir dialoguer avec un programme en cours d'exécution, il est parfois nécessaire de donner une valeur (en utilisant le clavier) que demande le programme.
Exemples :

```
nom = input("Quel est
    votre nom ?")
```

```
nombre = input("Entrer un nombre")
n = float(nombre)
print("Le carré de", n,"est", n * n)
```

Dans le deuxième exemple, si on tape 4, par exemple, dans toute la suite du programme, la variable n sera égale à 4.
Attention :

- le signe « = » n'est pas le signe égal au sens mathématique : il permet de donner une valeur à une variable. On peut voir $n = 4$ comme $n \leftarrow 4$.
- **input** est une fonction qui renvoie toujours une chaîne de caractères. Pour changer le type d'une variable, on utilise :
  — **str** pour les chaînes de caractères (inutile avec **input**)
  — **int** pour les entiers
  — **float** pour les nombres à virgule flottante.
- Une autre façon de faire pour le deuxième exemple est :

```
n = float(input("Entrer un nombre"))
print("Le carré de", n,"est", n * n)
```

II – Calcul avec Python 3

Les opérations de base

+	addition
–	soustraction
*	multiplication
/	division
**	puissance
//	division entière
%	reste de la division

```
15 // 6
```

Réponse : 2

```
15 % 6
```

Réponse : 3

L'écriture scientifique

Exemple :

```
2.75e3
```

Réponse : 2750.0

III – Boucle for . . . in

```
for n in range(3):
    print(n, end = " ")
```

Résultat : 0 1 2

Remarque : toutes les instructions qui sont indentées "font parties" de la boucle **for**

```
for loop in range(3):
    print("Hello", end = " ")
    print("world")
print("End")
```

Résultat :
Hello world
Hello world
Hello world
End

IV – Test if

1) Instruction if

La commande **if** permet de tester le contenu d'une variable et exécute une série d'instructions si les conditions sont remplies.

```
nombre = float(input("Choisissez un nombre"))
if (nombre>0):
    print("Le nombre choisi est positif")
print("Fin du programme")
```

2) Tester la valeur d'une variable contenant un nombre

Si n désigne une variable contenant un nombre, alors :

Test en français	**Écrit en langage** Python 3
Si n est égal à zéro	`if (n==0):`
Si n est positif	`if (n>0):`
Si n est différent de 34	`if (n!=34):`
Si n est compris strictement entre 0 et 10	`if (n>0) and (n<10):`
Si n est divisible par 5	`if (n%5==0):`

3) Tester plusieurs valeurs d'une variable

Il est parfois utile de tester plusieurs valeurs d'une même variable pour poursuivre l'exécution d'un programme.

```
n = input("Entrer un nombre : ")
if n<0:
    print("Le nombre est négatif")
elif n==0:
    print("Le nombre est égal à zéro")
else:
    print("Le nombre est positif")
```

Si n est négatif, alors on l'affiche

Sinon si n est égal à zéro, alors ...

Sinon forcément n est positif

V – Boucle While

Les deux programmes suivant sont équivalents :

```
a=0
while a<10:
    print("boucle Tant que")
    a=a+1
print("Fin du programme")
```

```
for loop in range(10):
    print("boucle for")
print("Fin du programme")
```

VI – Chaînes de caractères

```
txt='"Bonjour !", dit-elle.\n"Bonjour", répondit-il.'
print(txt)
```

Résultat :
"Bonjour !", dit-elle.
"Bonjour", répondit-il.
Remarques :

- \n insère un retour à la ligne.
- \' insère une apostrophe dans une chaîne délimitée par des apostrophes. De même, \" insère des guillemets dans une chaîne délimitée par des guillemets.

1) Accès aux caractères individuels d'une chaîne

```
ch = "Constance"
print(ch[0], ch[4], ch[8])
```

Résultat : C t e

2) Opérations élémentaires sur les chaînes

```
a = "Un cours"
b = " ça s'apprend !"
c = a + b
print(c)
```

Résultat : Un cours ça s'apprend !

```
ch = "Pierre"
print(len(ch))
```

Donne la longueur de la chaîne de caractère.
Résultat : 6

```
ch = "12"
n = int(ch)
print(n + 8)
```

Conversion d'une chaîne de caractère qui représente un nombre.
Résultat : 20

VII – Les listes

Une liste est une collection d'éléments séparés par des virgules, l'ensemble étant enfermé dans des crochets.
Dans la suite, on considérera cet exemple :

```
cours = ["Déjà vu", "Suites", "Trigonométrie", 3.14, 2718, "Complexes"]
```

```
print(cours[0], cours[2], cours[3])
```

Résultat : Déjà vu Trigonométrie 3.14

```
cours[0]="Révisions"
print(cours)
```

Résultat : ['Révisions', 'Suites', 'Trigonométrie', 3.14, 2718, 'Complexes']

```
print(len(cours))
```

Résultat : 6

```
del(cours[3])
print(cours)
```

Résultat : ['Révisions', 'Suites', 'Trigonométrie', 2718, 'Complexes']

On peut utiliser des *méthodes* de l'*objet* liste. Une méthode est appliquée par un point.

```
cours.append("Exponentielle")
# append signifie "ajouter" en anglais
```

Résultat : ['Révisions', 'Suites', 'Trigonométrie', 2718, 'Complexes', 'Exponentielle']

Il existe d'autres méthodes pour les listes : `sort()` (tri des éléments dans l'ordre croissant), `reverse()` (inverse l'ordre des éléments), `index()` (retrouve l'indice d'un élément), `remove()` (enlève un élément)...

```
print(cours.index("Exponentielle"))
```

Résultat : 5

```
cours.remove(2718)
print(cours)
```

Résultat : ['Révisions', 'Suites', 'Trigonométrie', 'Complexes', 'Exponentielle']

Retour sur l'instruction `range`
`range(5)` est une liste pré-remplie de 5 éléments : [0, 1, 2, 3, 4]
autres syntaxes :
`range(2,5)` ↔ [2, 3, 4]
`range(0,5,2)` ↔ [0, 2, 4] (la syntaxe est `range(start, stop, step)`)

slicing

```
nombres = [2, 45, -7, 19, 183]
```

```
print(nombres[1:3])
```

Résultat : [45, -7]

```
print(nombres[2:3])
```

Résultat : [-7]

```
print(nombres[2:])
```

Résultat : [-7, 19, 183]

```
print(nombres[:2])
```

Résultat : [2, 45]

VIII – Les fonctions

1) Importer des fonctions

On peut ajouter à Python 3 des programmes qui ont été écrits par des tiers. Par exemple, il existe un module `math` qui a été écrit pour apporter des fonctions mathématiques, comme la racine carré :

```
from math import sqrt
print(sqrt(25))
print(sqrt(23565215654734859))
```

Résultat :
5.0
153509659.80919525

Remarques :

- Racine carrée se dit **sq**uare **r**oo**t** en anglais.
- Pour importer toutes les fonctions d'un module, la syntaxe est `from math import *` (en changeant `math` par le module voulu).

2) Procédures

Une procédure est une fonction qui ne renvoie pas de valeur.

A) Sans paramètre

```
def table7():
    n = 1
    while n<11:
        print(n * 7, end = " ")
        n = n + 1
table7()
```

Résultat : 7 14 21 28 35 42 49 56 63 70

B) Avec paramètre

```
def table(base):
    n = 1
    while n<11:
        print(n * base, end = " ")
        n = n + 1
table(8)
```

Résultat : 8 16 24 32 40 48 56 64 72 80

Remarque : on peut définir des fonctions qui acceptent plusieurs paramètres.

3) « Vraies » fonctions

Le résultat des procédures ne peut pas être réutilisé : il n'a pas été stocké. Une « vraie » fonction est une procédure qui permet de réutiliser le résultat. Python 3 utilise la même syntaxe pour les deux, mais on terminera une fonction par un **return**.

```
def cube(x):
    return x**3
u = cube(3) % 10 # le reste de la division
   euclidienne par dix donne le chiffre des unités
print(u)
```

Résultat : 7

4) Variables locales, variables globales

Lorsque des variables sont définies à l'intérieur du corps d'une fonction, ces variables ne sont accessibles qu'à la fonction elle-même. On dit que ces variables sont des variables **locales** à la fonction.
Les variables définies à l'extérieur d'une fonction sont des variables **globales**. Leur contenu est visible de l'intérieur d'une fonction, mais la fonction ne peut pas le modifier.
Cependant, on peut quand même modifier une variable globale par une fonction à l'aide de l'instruction **global** :

```
def plusUn():
    global a
    a = a + 1
    print(a)
a = 15
plusUn()
```

Résultat : 16

```
plusUn()
```

Résultat : 17

```
def plusUn2():
    a = a + 1
    print(a)
a = 15
plusUn2()
```

Résultat : Erreur : UnboundLocalError : local variable 'a' referenced before assignment

IX – Manipulation de fichiers

Dans cette section, on considère un fichier texte `scores.txt` qui se trouve au même endroit que le script Python.

1) Ouvrir un fichier

```
objfichier = open('nomFichier', 'modeLecture')
```

La fonction **open** va créer un *objet-fichier* (`objfichier` ici) auquel on peut appliquer des *méthodes*. Il est ouvert dans le mode de lecture spécifié. Les cas possibles sont :

- `'r'` : ouverture en lecture (**r**ead).
- `'w'` : ouverture en écriture (**w**rite). Le contenu du fichier est écrasé. Si le fichier n'existe pas, il est créé.
- `'a'` : ouverture en écriture en mode ajout (**a**ppend). On écrit à la fin du fichier sans écraser l'ancien contenu du fichier. Si le fichier n'existe pas, il est créé.

```
f = open('scores.txt', 'r')
```

Pour ouvrir le fichier `'scores.txt'` en lecture seule.

De plus, on peut ajouter à ces modes le signe `b` pour ouvrir le fichier en mode *binaire*.

2) Écrire dans un fichier

Cela se fait à l'aide de la méthode **write**. Il ne faut pas oublier de fermer le fichier à l'aide de la méthode **close**.

```
a = 257692
b = 199827
f = open('scores.txt', 'w')
f.write("Scores des joueurs :\n")
f.write("Alice : " + str(a) + "\n")
f.write("Bob : " + str(b) + "\n")
f.close()
```

Résultat : Le fichier `scores.txt` contient maintenant :
Scores des joueurs :
Alice : 257692
Bob : 199827

Remarque : la méthode **write** n'accepte que des chaînes de caractères en paramètre, d'où la conversion à l'aide de **str**.

3) Lire un fichier

Méthode read : elle permet de lire l'intégralité du fichier ou de lire un certain nombre de caractères à partir de la position atteinte.

```
f = open("scores.txt", "r")
t = f.read()
print(t)
f.close()
```

L'intégralité du texte est lu :
Scores des joueurs :
Alice : 257692
Bob : 199827

```
f = open("scores.txt", "r")
t = f.read(8)
print(t)
t = f.read(18)
print(t)
f.close()
```

Résultat :
Scores d
es joueurs :
Alice

Méthode readline : elle ne lit qu'une ligne à la fois.

```
f = open("scores.txt", "r")
f.readline() # on saute la première ligne
t=f.readline()
print(t)
t = f.readline()
print(t)
f.close()
```

Résultat :
Alice : 257692

Bob : 199827

X – Pour en savoir plus...

Selon vos envies ou le projet que vous présenterez au bac vous pourrez avoir besoin d'approfondir certaines notions (interfaces graphiques, manipulation de fichiers, etc.).
Vous pourrez trouver de l'aide en suivant les liens ci-dessous, ou en effectuant une recherche sur internet.

- Le site : france-ioi.org
- Un livre téléchargeable gratuitement (mais on peut l'acheter aussi !) : inforef.be/swi/python.htm
- La documentation officielle : docs.python.org/3/
- Les cours sur openclassrooms (anciennement "site du zéro") : fr.openclassrooms.com/informatique/python/cours