# *Chapitre I*

# *Notions de bases de la programmation en Python*

## I.1. Introduction

Le langage Python est développé en 1989 par Guido van Rossum (informaticien Belge né le 31/01/1956), il fait partie des langages de programmation interprétés les plus utilisés dans le monde en raison sa syntaxe très simple et sa bibliothèque qui offre l'accès à une grande variété de services (chaînes de caractères et expressions régulières, protocoles Internet, interfaces graphiques…etc.). Il est orienté objet (c.à.d. il support les notions d'héritage, la surcharge des opérateurs, le polymorphisme ….). De plus, Python est un langage de haut niveau, c.à.d. il ne demande pas beaucoup de connaissances préalables pour être utilisé. Ce chapitre présentera les notions de base de la programmation en Python.

## I.2. Interpréteur Python

Python fait partie des langages interprétés, c'est-à-dire que chaque ligne de code est lue puis interprétée afin d'être exécutée par l'ordinateur.

Pour lancer l'interpréteur Python, ouvrez un Powershell (via le menu de démarre de Windows) puis lancez la commande ' python' comme montre la figure (I.1).

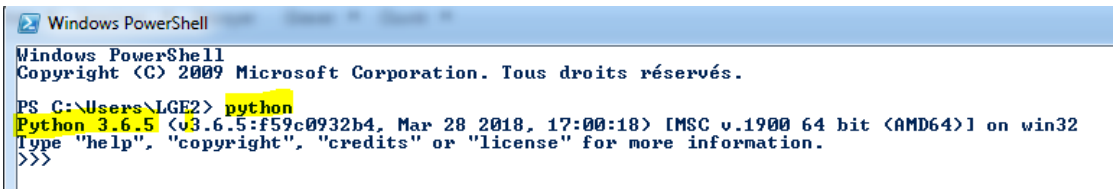
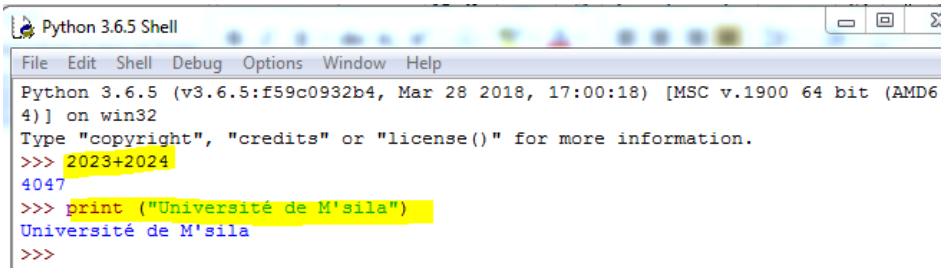



Figure (I.1) Interpréteur de Python.

Ainsi, on peut exécuter quelques commandes comme suit :




Pour quitter l'interrupteur Python il suffit de taper l'instruction ' exit ()' suivi par 'Entrer' ou en pressant simultanément les touches Ctrl+Z suivi par 'Entrer'.

## I.3. Différence entre compilateurs et interpréteur

Le compilateur, traduit l'ensemble du code source en langage binaire utilisable directement par le CPU. Par exemple C, C++, Pascal, Fortran sont des langages compilés.
L'interpréteur lit une partie du code source et exécute directement les instructions binaires qui correspondent et passe à la suite. Parmi les langages interprétés on note Html, Css, Python, Basic.

## I.4. Utilisation des commentaires en Python

Les commentaires sont des phrases ou des paragraphes ajoutés au code du programme pour le but d'expliquer son fonctionnement ou pour ajouter des explications à l'attention du lecteur. Ils n'ont aucun impact sur le fonctionnement du programme (c.à.d. tous ce qui marqué comme commentaire sera ignoré). Pour ajouter un commentaire en Python il suffit de l'écrire devant le caractère dièse ' #'.

*Exemple I.1*

```
>>> a=1
>>> b=5
>>> c=1
>>> delta=b**2-4*a*c # calcul du déterminant
>>> print (delta) # affichage de la valeur de delta
21
```

## I.5. Variables en Python, déclaration et utilisation

Une variable est une partie de mémoire dont sa contenue (nombre, adresse…) peut être modifiée ou initialisée pendant l'exécution du programme. La déclaration d'une variable et son initialisation se font en même temps en Python. Dans l'Exemple I.1, les variables a, b et c sont déclarées et initialisées respectivement par les valeurs 1, 5 et 1.

### I.5.1 Typage dynamique des variables

En Python il n'est pas nécessaire de préciser le type des variables car Python est un langage au typage dynamique, c'est-à-dire qu'il reconnaît le type des variables au moment

de l'exécution, la simple instruction a = 1 est suffi pour définir le type de la variable. L'Exemple I.2 suivant montre cette propriété.

*Exemple I.2*

```
>>> a= 2020
>>> b=3.14
>>> c= "Electronique"
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
>>> type(c)
<class 'str'>
```

Dans l'Exemple I.2 Python a considéré la variable a comme int puisque elle reçoit une valeur entière, la variable b comme float puisque elle reçoit une valeur décimal et la variable c comme string (str) puisque elle reçoit une chaine de caractère. **C'est pour ça on dit que Python est un langage au typage dynamique**. Dans d'autre langage (par exemple en C et C++), il est indispensable de spécifier le type des variables (Python étant un langage dit de haut niveau).

#### I.5.2 Déclaration des chaines des caractères

A noter que les chaînes de caractères doivent placées entre guillemets simple, doublé ou triplé comme montre l'Exemple I.3.

*Exemple I.3*

```
>>> x= "Instrumentation"
>>> b= ' Télécommunication'
>>> c= '''systèmes embarqués'''
>>>
```

*Remarque :*

1. *Le nom de la variable en Python ne doit pas inclure ni un espace, ni des caractères spéciaux -(@, $, #, &...) ni des lettres accentuées (é, ç, à…).*
2. *Le nom de la variable doit commenter toujours par une lettre.*
3. *Il faut absolument éviter d'utiliser les mots clés réservés par Python (par exemple : print, input …etc.).*
4. *Python est sensible à la casse, ce qui signifie que les variables Aa, aA, AA, et aa sont différentes.*

#### I.5.3 Affectation simple et multiple des variables

Sous Python, il est possible d'affecter une valeur à plusieurs variables simultanément, comme il est possible d'effectuer une affectation parallèle. Dans l'Exemple I.4, les

variables a et b sont affectées par la valeur 2023. Les deux variables c et d sont affectées respectivement par les valeurs 2023 et 2024.

*Exemple I.4*

```
>>> a=b=2023
>>> c,d=2023,2024
>>> Dep, Fac= "Electronique","Technologie"
>>> print (a,b)
2023 2023
>>> print(c,d)
2023 2024
>>> print(Dep, Fac)
Electronique Technologie
>>>
```

## I.6. Opérateurs standards en Python

### I.6.1 Opérateurs arithmétiques

Les opérations arithmétiques sont utilisées pour les calculs numériques. Les opérateurs arithmétiques essentiels sont :

+ : Addition
- : Soustraction.
* : Multiplication
/ : Division
** : Puissance ( 2**3 donne 8)
(//) Quotient de la division entière (8//3 donne 2)
(%) Modulo (reste de la division entière, 11 % 3 donne 2)

(**), (//) le quotient de la division entière (, par exemple et le Modulo (reste de la division entière, par exemple 11 % 3 donne 2, 15 % 3 donne 0).

### I.6.2 Opérateurs d'affectation composée

+= (ajouter à)

-= (diminuer de)

*= (multiplier par)

/= (diviser par)

%= (modulo)

*Exemple I.5*

```
1  a=2
2  b=3
3  a += 5   # a vaut 7
4  b -= 2   # b vaut 1
5  a *= 3   # a vaut 21
6  a %= 6   # a vaut 3
```

### I.6.3 Opérateurs de comparaison

Les opérateurs logiques sont utilisés généralement pour faire une condition sur l'exécution ou non d'une ou multiple instruction.

and (et logique), *exemple*, 1 and 0 donne 0, 1 and 1 donne 1.

ou (ou logique), *exemple*, 1 ou 0 donne 0.

**'!'** (négation logique), *exemple* !1 donne 0, !2021 donne 0.

== (est-il égal à ?, *exemple* x= =10, a+1= = 04.

!= (différent de ?), *exemple* x !=10.

<, <=, >,>= (inférieur de, inférieur ou égal de, supérieur de….).

*Exemple* I.6

```
In [20]: True and True
Out[20]: True

In [21]: False and True
Out[21]: False

In [22]: 5==3+2
Out[22]: True

In [23]: 7!=9-2
Out[23]: False

In [24]: True or False
Out[24]: True
```

### I.6.4 Opérateurs sur les chaînes de caractère

Il y a deux opérations possibles sur les chaînes de caractère :

- L'addition (ou la concaténation) qu'il s'agit de rassembler les chaînes de caractère.
- La multiplication qu'il s'agit de dupliquer (répéter) la chaînes de caractère plusieurs fois.

*Exemple* I.7

```
In [3]: Ch1= "Electronique"

In [4]: Ch2= "M'sila"

In [5]: Ch1+Ch2
Out[5]: "ElectroniqueM'sila"

In [6]: Ch1*3
Out[6]: 'ElectroniqueElectroniqueElectronique'
```

Dans l'Exemple I.7 la chaîne de caractère Ch1 (Electronique) est concaténée avec Ch2 (M'sila) ensuite elle dupliquée trois fois (Ch1*3).

## I.7. Conversion de types d'une variable

Il est possible de convertir le type d'une variable à un autre en utilisant les fonction standard *int ()*, *float()*, *str()*…etc. Par exemple, dans le cas de lecture des fichier[1], les nombres sont considérés comme des chaînes de caractère, ainsi il est indispensable de les convertir afin que Python les considère comme des nombres.

*Exemple* I.8

```
In [9]: n1="123"

In [10]: type (n1)
Out[10]: str

In [11]: k=int (n1)

In [12]: k+10
Out[12]: 133

In [13]: type (k)
Out[13]: int
```

## I.8. Affichage et lecture des donnés

### I.8.1 La fonction *print*

Pour afficher une valeur numérique ou chaîne de caractère, on utilise la fonction mot clé '*print*'**.** Cette dernière affiche à l'écran ce qui lui est placé entre ces parenthèses (variable, chaîne de caractères, valeur numérique…etc.).

*Exemple* I.9

```
In [43]: print ("Departement de L'Electronique")
Departement de L'Electronique

In [44]: print (" Electronique ") ; print (" M'sila ")
 Electronique
 M'sila

In [45]: print ( "Electronique" , "de",  " M'sila ", sep =" -")
Electronique -de - M'sila

In [46]: print (" Electronique ", end="") ; print (" M'sila ")
 Electronique  M'sila
```

```
In [18]: a= 2023

In [19]: b=2024

In [20]: c= " l'année universitaire: "

In [21]: print (c,a,'-',b)
 l'année universitaire:  2023 - 2024

In [22]: print (c,a,'-',b,sep="")
 l'année universitaire: 2023-2024
```

[1] Cette notion sera abordée ultérieurement, voire chapitres xx.

### I.8.2 La fonction *f-string*

Les *f-strings (formatted string literals*) permettent une meilleure organisation de l'affichage des variables. Pour utiliser les f-strings, il suffit de placer la lettre 'f' après la première parenthèse de la fonction 'print'. La caractère f va indiquer à Python qu'il s'agit d'une une écriture formatée.

*Exemple* I.10

```
In [46]: a= 2023

In [47]: b=2024

In [48]: c= " l'année universitaire: "

In [49]: print (f"{c} {a} - {b}")
 l'année universitaire:  2023 - 2024

In [50]: print (f"{c}{a}-{b}")
 l'année universitaire: 2023-2024
```

```
In [52]: age =23

In [53]: print(f"Votre age est:{age}")
Votre age est:23

In [54]: print(f"Votre age est:    {age}")
Votre age est:    23
```

### I.8.3 La fonction *input*

Pour lire des donnés entrées au clavier (chaines de caractère, valeur numérique…) on utilise souvent la fonction *input(),* cette dernière provoque une interruption dans le programme courant et invite l'utilisateur à introduire des caractères. Lorsque la touche <Enter> est enfoncée, l'exécution du programme se poursuit et la fonction *input* assignait la chaîne de caractères introduite à la variable spécifiée par l'utilisateur.

*Exemple* I.10

```
In [64]: print(f"Entrer votre age ")
Entrer votre age

In [65]: c= input()
21

In [66]: type (c)
Out[66]: str
```

Pour convertir les nombres entrés au clavier, on utilise les fonction de conversion du type (*int (), float(), bool()*…..).

*Exemple* I.11

```
In [77]: print(f"Entrer votre age ")
Entrer votre age

In [78]: c= input()
21

In [79]: d= int(c)# Convertir la valeur de C au type Int

In [80]: d
Out[80]: 21

In [81]: type(d)
Out[81]: int
```

*Exemple* I.12

```
In [89]: c= int (input(f"Entrer votre age "))
Entrer votre age 21

In [90]: c
Out[90]: 21

In [91]: type (c)
Out[91]: int
```

*Remarque importante*

1. *La fonction input() renvoie toujours une chaîne de caractères, Par conséquent, si il s'agit du lecture des valeurs numériques, il est nécessaire de les convertir à des valeur numérique en utilisant les fonction int(), float()…etc, voir Exemple I.11 et I.12.*

## I.9 Structure de contrôle alternatifs

Les structures de contrôles alternatives permettent de définir la suite dans laquelle les instructions sont exécutées, c.à.d. elles permettent d'imposer des conditions pour l'exécution ou non d'un bloc d'instruction.

### I.9.1 Structure de contrôle alternative if-else

La syntaxe d'utilisation de la structure de contrôle if-else est la suivante :

```
if (Conditions) :
    Bloc d'instructions 1
else :
    Bloc d'instructions 2
```

- Si les conditions (une ou multiples) sont vérifiées, alors le *Bloc d'instructions 1* sera exécuté et le *Bloc d'instructions 2* sera ignoré.
- Si les conditions ne sont pas vérifiées, alors le B*loc d'instructions 2* sera exécuté et le *Bloc d'instructions 1* sera ignoré.

*Exemple* I.13

```
a= int (input("Entrer un entier "))
if (a % 2 == 0):
    print(f"Le nombre {a} est paire ")
else:
    print(f"Le nombre {a} est impaire ")
```

Exemple d'exécution

```
Entrer un entier 14
Le nombre 14 est paire
```

### I.9.2 Structure de contrôle alternative if-elif-…-else

Dans le cas de plusieurs alternatives (conditions), il est possible de combiner plusieurs structures if-else, la syntaxe dans ce cas est comme suit :

```
if (Conditions 1) :
    Bloc d'instructions 1
elif (Conditions 2) :
    Bloc d'instructions 2
        .
        .
elif (Conditions n) :
    Bloc d'instructions n
else :    # Facultative, elle est exécuté lorsqu'aucune condition n'a été vérifiée
    Bloc d'instructions n+1
```

Les conditions Conditions 1, Conditions 2 ... Conditions n sont évaluées l'une après l'autre jusqu'à ce que l'une de celles soit vérifiée, le bloc d'instructions lié à la condition vérifiée est exécuté et le traitement de la commande sera terminée. Le Bloc d'instructions n+1 sera exécuté si seulement si toutes les conditions (1,2…n) ne sont pas vérifiées.

*Exemple* I.14

```
1  a= int (input("Entrer un entier "))
2  if (a> 0):
3      print("Le nombre entré est positif")
4  elif(a==0):
5      print("Le nombre entré est nul")
6  else:
7      print("Le nombre entré est négatif")
```

Exemple d'exécution

```
Entrer un entier -8
Le nombre entré est négatif
```

*Remarque importante*

Le bloc d'instructions est délimité par l'indentation, c.à.d., les lignes de même bloc doivent être décalées vers la droite du même nombre d'espaces, voir Exemple I.14.

## I.10 Liste

Une liste est une structure de données qui contient une série de valeurs de même ou différents types (int, float, char …). Une liste est déclarée par une série de valeurs placées entre deux crochets et séparées entre elles par des virgules

```
a= ["Université", "M'sila", 2023, "Python" ]
```

Les éléments d'une liste sont accessibles utilisant le nom de la liste suivi par l'indice de l'élément (son ordre) comme suit :

```
In [2]: a
Out[2]: ['Université', "M'sila", 2023, 'Python']

In [3]: a[0]
Out[3]: 'Université'

In [4]: a[3]
Out[4]: 'Python'
```

### I.10.1 Opération sur les listes

1- Les liste comme les chaines des caractères, elles supportent la concaténation et la multiplication.

- L'opérateur + pour concaténer deux listes.
- L'opérateur * pour répéter une liste.

Il est possible d'ajouter des nouveaux éléments à une liste déjà crée en utilisant l'opérateur + ou la fonction *append()*.

*Exemple* I.14

```
In [14]: a= ["Université", "M'sila", 2023, "Python" ]

In [15]: a=a+["Electronique"]

In [16]: a
Out[16]: ['Université', "M'sila", 2023, 'Python', 'Electronique']
```

```
In [26]: a= ["Université", "M'sila", 2023, "Python" ]

In [27]: b=["Intstrumentation","Télécomunication"]

In [28]: a+b
Out[28]:
['Université',
 "M'sila",
 2023,
 'Python',
 'Intstrumentation',
 'Télécomunication']
```

Ou en utulisant la fonction append comme suit :

```
In [4]: a= ["Université", "M'sila", 2023, "Python" ]

In [5]: a.append("Algeria/Palestine")

In [6]: a
Out[6]: ['Université', "M'sila", 2023, 'Python', 'Algeria/Palestine']
```

2- il est possible de sélectionner une partie d'une liste en utilisant généralement la syntaxe suivante : [Indice de départ (inclus) : Indice d'arrêt (non inclus) : Pas d'incrémentation].

*Exemple* I.15

```
In [27]: a= ["Université", "M'sila", 2023, "Python", "Algeria/Palestine"]

In [28]: a[0:5:2]
Out[28]: ['Université', 2023, 'Algeria/Palestine']

In [29]: a[0:5:3]
Out[29]: ['Université', 'Python']

In [30]: a[2:5]
Out[30]: [2023, 'Python', 'Algeria/Palestine']

In [31]: a[0:]
Out[31]: ['Université', "M'sila", 2023, 'Python', 'Algeria/Palestine']

In [32]: a[:4]
Out[32]: ['Université', "M'sila", 2023, 'Python']
```

### I.10.2 Fonctions sur les listes

Ci-dessous quelques fonctions sur les listes :

- Fonction *len ( ) permet de calculer la longueur d'une liste ;*

```
In [36]: a= ["Université", "M'sila", 2023, "Python", "Algeria/Palestine"]

In [37]: len(a)
Out[37]: 5
```

- La fonction *range()* permet de générer une liste des nombre entiers.

```
In [39]: list( range(19) )
Out[39]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]

In [40]: list( range(5,-5,-1 ) )
Out[40]: [5, 4, 3, 2, 1, 0, -1, -2, -3, -4]
```

- Fonction *sum()* permet de calculer la somme des éléments d'une liste des nombres.

- Fonctions *min( ), max()* permet de déterminer respectivement le minimum et le maximum d'une liste des nombres

- Fonctions *del( )* permet de supprimer un élément dans une liste en utilisant son indice.

*Exemple* I.16

*# fonctions sum, min, max*

```
In [1]: a=list(range(5))

In [2]: sum(a)
Out[2]: 10

In [3]: min(a)
Out[3]: 0

In [4]: max(a)
Out[4]: 4

In [5]: a
Out[5]: [0, 1, 2, 3, 4]
```

*# fonction del*

```
In [33]: Countries=["Algeria", "Palestine","USA", "Iraq","Qatar"]

In [34]: del(Countries[2])

In [35]: Countries
Out[35]: ['Algeria', 'Palestine', 'Iraq', 'Qatar']
```

- Il est possible de consrure une liste d'une liste comme suit :

```
In [41]: a1=["Rabah Bitat", "Mohamed Boudiaf"]

In [42]: a2=["Krim Belkacem","Larbi Ben M'Hidi"]

In [43]: a3=["Mostefa Ben Boulaïd","Didouche Mourad"]

In [44]: a_all=[a1,a2,a3]
```

```
In [45]: a_all
Out[45]:
[['Rabah Bitat', 'Mohamed Boudiaf'],
 ['Krim Belkacem', "Larbi Ben M'Hidi"],
 ['Mostefa Ben Boulaïd', 'Didouche Mourad']]
```

Ainsi, l'accès aux éléments de la liste globale se fait comme suit :

```
In [52]: a_all[0][0]
Out[52]: 'Rabah Bitat'

In [53]: a_all[0][1]
Out[53]: 'Mohamed Boudiaf'

In [54]: a_all[2][0]
Out[54]: 'Mostefa Ben Boulaïd'
```

Comme il est possible d'accéder d'une sous liste comme suit :

```
In [55]: a_all[0]
Out[55]: ['Rabah Bitat', 'Mohamed Boudiaf']

In [56]: a_all[2]
Out[56]: ['Mostefa Ben Boulaïd', 'Didouche Mourad']
```

## I.11 Structures de contrôle répétitives

Les boucles permettent de répéter l'exécution d'un bloc d'instructions plusieurs fois. Le nombre de répétition soit dépend d'une validation d'une condition logique ou défini par le programmeur.

### I.11.1 Boucle for

La syntaxe de cette la boucle 'for' est la suivante :

*for indice in ( liste):*
*bloc d'instructions*

*Exemple* I.17

```
a= ["Université", "M'sila", 2023, "Python", "Algeria/Palestine"]
for i in [0,1,2,3,4]:
    print (a[i])
```

Résultat d'exécution

```
Université
M'sila
2023
Python
Algeria/Palestine
```

```
a= ["Université", "M'sila", 2023, "Python", "Algeria/Palestine"]
for i in [0,1,2,3,4]:
    print (f"a[{i}]=  {a[i] }")
```

Résultat d'exécution

```
a[0]=  Université
a[1]=  M'sila
a[2]=  2023
a[3]=  Python
a[4]=  Algeria/Palestine
```

```
a= ["Université", "M'sila", 2023, "Python", "Algeria/Palestine"]
for i in range( len (a) ):
    print (f"a[{i}]=  {a[i]*2 }")
```

Résultat d'exécution

```
a[0]=  UniversitéUniversité
a[1]=  M'silaM'sila
a[2]=  4046
a[3]=  PythonPython
a[4]=  Algeria/PalestineAlgeria/Palestine
```

### I.11.2 Boucle while

La syntaxe de cette la 'while' est la suivante :

*while (condition) :*
*bloc d'instructions*

*Exemple I. 18*

```
a= ["Université", "M'sila", 2023, "Python", "Algeria/Palestine"]
i=0
while (i < len(a) ):
    print (f"a[{i}]=  {a[i]}")
    i=i+1
```

Résultat d'exécution

```
a[0]=  Université
a[1]=  M'sila
a[2]=  2023
a[3]=  Python
a[4]=  Algeria/Palestine
```

### I.11.3 Break et continue

Ces deux instructions sont généralement utilisées avec les boucles, l'instruction *break* permet d'arrêter l'exécution tandis que l'instruction *continue* réalise une saute d'écécution d'une ou bloc d'instruction.

*Exemple I. 19*

```
a= [1, 11, 1954, 5, 6,1962]
for  i in range( len(a) )  :
    if (i== 3):
        break
    print (f"a[{i}]=  {a[i]}")
```

Résultat d'exécution

```
a[0]=  1
a[1]=  11
a[2]=  1954
```

*Exemple I. 20*

```
a= [1, 11, 1954, 5, 6,1962]
for  i in range( len(a) )  :
    if (i== 3):
        continue
    print (f"a[{i}]=  {a[i]}")
```

Résultat d'exécution

```
a[0]=  1
a[1]=  11
a[2]=  1954
a[4]=  6
a[5]=  1962
```