Chapitre 1 : Rappels sur la programmation en Python

Python est un langage de programmation de haut niveau utilisé surtout pour les applications scientifiques et le développement des logiciels, dans les sites web (Instagram, Youtube, Google) , l'intelligence artificielle,…etc.

1.1. Introduction : l'informatique est la science de traitement automatique de l'information grâce à un ordinateur.

Un ordinateur est un ensemble de composants électroniques capable de programmer et d'exécuter des instructions. Un ordinateur est constitué de :

- CPU ou unité centrale de traitement (processeur) permet d'exécuter les instructions et comprend une unité de commande (CU) et une unité arithmétique et logique(ALU).

- Mémoire centrale c'est l'endroit où sont temporairement stockés les données et les programmes en cours d'exécution.

- Unités de stockage non volatiles telles que les disques durs et les disques SSD.

- Périphériques ou E/S comme les claviers, souris, Ecran,…etc.

Un logiciel est un ensemble de programmes permettant à l'ordinateur d'exécuter[2] des taches précises. Il existe deux types de logiciel :

- Logiciel système est l'ensemble des programmes permettant le bon fonctionnement de l'ordinateur (système d'exploitation,…).

- Logiciel d'application est l'ensemble des programmes permettant de réaliser des taches précises pour l'utilisateur (Bureautique, Navigation, Programmation,…etc).

1.2. Système d'exploitation :

Un système d'exploitation (SE) ou (OS - Operating System) est un logiciel (ensemble de programmes) qui gère les ressources matérielles d'un ordinateur (processeur, mémoire, disque dur, etc.) et qui rendent ces ressources disponibles à l'usager.

1.3. Réseaux informatiques

Un réseau informatique est un ensemble d'ordinateurs et de périphériques (imprimantes, disque dur,…etc.) connectés entre eux pour échanger des données.

Les types des réseaux informatiques sont : PAN, LAN, MAN et WAN.

PAN (Personal Area Network) ou Réseau personnel : est un petit réseau informatique destiné à connecter des appareils proches (quelques mètres). Ce type de réseau est utilisé pour la communication entre des appareils personnels. Comme exemple : Un ordinateur portable connecté à une imprimante ou une souris sans fil via USB, Bluetooth ou infrarouge.

LAN (local Area Network) ou Réseau local : est un réseau qui permet d'assurer l'interconnexion d'équipements au sein d'un site limité (quelques kilomètres).

MAN (Metropolitan Area Network) ou Réseau métropolitain : est un réseau à l'échelle d'une ville.

WAN (Wide Area Network) ou Réseau personnel : Réseau large ou étendu comme Internet (Inter Networking ou interconnexion de réseaux)

1.**3.1 Adresse IP** (Internet Protocol) : C'est un identifiant unique donné à chaque appareil sur un réseau. Elle permet de trouver et de communiquer avec un autre appareil sur le réseau.

Il existe des adresses IP de version 4 (sur 32 bits, soit 4 octets) et de version 6 (sur 128 bits, soit 16 octets)

1. **IPv4** : format classique → 192.168.0.1
   4 groupes de chiffres entre 0 et 255
2. **IPv6** : plus récente, plus longue →
   2001:0db8:85a3:0000:0000:8a2e:0370:7334

TCP veut dire : Transmission Control Protocole (Langage qui va contrôler le transport des données)

TCP/IP veut dire : transporter les données d'un ordinateur à un autre, en s'appuyant sur les adresses IP des ordinateurs.

1.3.2 DNS (Domain Name System) ou système de noms de domaine : traduit les noms de domaine en adresses IP. Exemple : www.google.com → 142.250.74.132

1.4. ELEMENTS DU LANGAGE :

1.4.1 Variables :

Les variables servent à désigner les différents objets (choses) manipulées par le programme.

Une *variable* est caractérisée par :

– son nom : suite de caractères alphanumériques dont le premier doit être une lettre ou le caractère souligné (_).

– son type précisant la nature de cette variable (nombre entier, réel, caractère, logique).

- son adresse : l'endroit de la mémoire où elle est stockée.

- sa valeur.

Le typage avec Python est dynamique, c'est-à- dire, qu'il n'est pas nécessaire de déclarer une variable avant son utilisation. Il suffit d'attribuer une valeur à un nom de variable pour que celle-ci soit créée avec le type qui correspond à la valeur.

Par exemple :

x = 2 # pour créer un entier initialisé par la valeur 2.

Les types de variables du langage Python sont :

• Les entiers (mot-clé int) ;

• Les réels (mot-clé float) ;

. Les complexes (mot-clé complex) ;

• Les booléens, c'est-à-dire dont la valeur est soit vrai, soit faux (mot-clé bool).

• Les chaines de caractère (mot-clé str) ;

Il existe plusieurs façons d'écrire une chaîne de caractères :

• entre guillemets (" ") ;

• entre triples guillemets (""" """) pour une chaine de caractère s'étalant sur plusieurs lignes ;

• entre apostrophes (' ').

Si la chaine contient des apostrophes, il faut les faire précéder par des anti-lash.

1.4.2 Opérateurs et expressions :

En combinant des variables, des opérateurs et de fonctions, on obtient des expressions.

1.4.2.1 Opérateurs arithmétiques :

+ : addition,

- : soustraction,

* : multiplication,

/ : division réelle (19 / 5 vaut 3.8),

// : division entière (19//5 vaut 3),

% : reste de la division entière de deux entiers (modulo)(19 % 5 vaut 4),

** : puissance (x^y),

Dans les expressions arithmétiques, les règles de priorité sont les règles usuelles mathématiques. Par exemple, l'expression 5 + 3 * 2 a pour valeur 11 et non 16. En cas de doute, il faut mettre des parenthèses.

1.4.2.2 Opérateurs d'affectation :

= (égal)

Forme générale de l'expression d'affectation

variable = expression

Avec Python, on peut assigner une valeur à plusieurs variables simultanément. Exemple :

>>> x = y = 7

>>> x

7

>>> y

7

* **Opérateurs d'affectation élargie :**

Il arrive très souvent de calculer la nouvelle valeur d'une variable en fonction de son ancienne valeur. Python fournit pour cela un jeu d'opérateurs combinés, de la forme :

+=, -=, *=, /=, %=

Forme générale : *variable opérateur= expression*

L'expression est équivalente à :

variable = variable opérateur expression

Par exemple, l'expression i += 3 équivaut à i = i + 3.

1.4.2.3 Opérateurs logiques :

Les principaux opérateurs logiques sont : and, or et not

1.4.2.4 Opérateurs de comparaison :

== : égal,

!= : différent,

< : strictement inférieur,

> : strictement supérieur,

<= : inférieur ou égal,

>= : supérieur ou égal.

1.4.3 Structures de contrôle :

1.4.3.1 Structures de contrôle conditionnelles :

On appelle structures de contrôle conditionnel, les instructions qui permettent de tester si une condition est vraie ou non.

1.4.3.1.1 Structure « if » :

Syntaxe :

if condition :

  instructions

où condition est une expression dont la valeur est booléenne

1.4.3. 1.2. Structure if … else:

L'instruction if précédente ne permet de tester qu'une condition, or la plupart du temps on aimerait pouvoir choisir les instructions à exécuter en cas de non réalisation de la condition. L'expression if ... else permet d'exécuter une autre série d'instructions en cas de non-réalisation de la condition.

Syntaxe :

if condition :

        instructions

else :

        autres instructions

Il est possible d'avoir des instructions if imbriquées (elif).

1.4.3.2 Structures de contrôle itératives (boucles) :

Il existe 2 types de boucles à connaître :

- la boucle for,
- la boucle while.

.

1.4.3.2.1 Boucle while :

while permet d'exécuter des instructions en boucle tant qu'une condition est vraie.

Syntaxe:

while condition :

  instructions

1.4.3.2.2 Boucle for :

Syntaxe:

for variable in range( ):

   instructions

Examples :

```
for i in range(10):          # à partir de 0 (inclu) juqu'à 10 (exclu)
  print(i)
#imprime :
0
1
2
.
.
.
9

for i in range(2, 9):        # à partir de 2 (inclu) juqu'à 9 (exclu)
  print(i)
#imprime :
2
3
4
.
.
.
8

for i in range(2, 9, 2):    # à partir de 2 (inclu)  juqu'à 9 (exclu) avec pas 2
  print(i)
# imprime
2
4
6
8

for i in range(10, 4, -2):  # à partir de 10 (inclu)  juqu'à 4 (exclu) avec pas -2
  print(i)
# imprime
10
```

8
6

1.4.4 Les entrées/sorties en Python :

Les E/S de Python sont réalisées par :

- input ( ) pour lire depuis le clavier.
- print ( ) pour afficher à l'écran.

input ( ) lit toujours une chaine de caractères (str), si on veut un nombre :

nombre=type(input(″donner un nombre″))

1.5. Les fonctions

1.5.1. Généralités

Une fonction est destinée à effectuer une tache précise et renvoi généralement une valeur.

Syntaxe :

```
def nom (paramètres formels) :
  bloc d'instructions
  return résultat
```

Exemple:

```
        # définition de la fonction moyenne
        >>>def moyenne(a,b) :
        …    moy = (a + b)/2
        …    return moy

      x = int(input("x?"))
       y = int(input("y?"))
        # appel de la fonction moyenne

      resultat = moyenne(x,y)
      print(resultat)
```

Les noms des paramètres figurant dans l'en-tête de la fonction se nomment des « arguments formels » ou « paramètres formels ». Leur rôle est de permettre, au sein du corps de la fonction, de décrire ce qu'elle doit faire.

Les paramètres fournis lors de l'appel de la fonction se nomment des « arguments effectifs » ou encore « paramètres effectifs ».

• L'instruction return peut apparaître à plusieurs reprises dans une fonction.

Exemple :

```
def absom ( a, b) :
    s = a + b
    if s>0 :
      return  s
    else :
     return (-s)
```

1.5.2. Cas des fonctions sans valeur de retour ou sans arguments :

- Sans arguments et sans valeur de retour :

```
# définition de la fonction moyenne
def fonc():
    print("hello")
# appel
fonct()
  hello
```

1.5.3. Arguments par défaut :

Python permet d'attribuer des valeurs par défaut à des arguments non fournis lors de l'appel d'une fonction.

Exemple :

```
def fct ( a =0, b=1) : #  définition
   print (a,b)

n=5
```

```
m=10
fct (n, m)  # appel "normal"
fct (n)  #appel avec un seul argument
fct ()  #appel sans argument
```

On aura :

1) 5 10
2) 5 1
3) 0 1

1.5.4. Nombre variable d'arguments :

```
def addition(*nombres):
    return sum(nombres)

print(addition(2, 3))        # 5
print(addition(1, 2, 3, 4))  # 10
```

sum est une fonction prédéfinie qui permet de faire la somme d'un ensemble de valeurs.

1.6.Tableau (liste):

Une variable peut contenir qu'une seule valeur. Un tableau est une collection indicée de variables de même type. Cette structure consiste à réserver un espace de mémoire adjacent dans lequel les éléments du tableau peuvent être rangés.

Exemple:

```
Tab = [1,2,3,4,5]          # tableau ou liste contenant 5 éléments.
Tab=[ ] liste vide
```

Dans une liste de taille N, les cases ne sont pas numérotées de 1 à N mais de 0 à N –1.

On accède au contenu d'une case par l'opérateur []

```
Tab[0]=1;                  # pour affecter 1 à la première case de la liste.
```

Les indices peuvent être négatifs (-1 c'est le dernier élément)

```
print(Tab[-1])          # 5
Tab.append( val)     # ajoute à la fin
```

```
Tab.insert(indice, val)    # insère à l'indice
Tab.remove( val)        # supprime l'élément
del Tab[indice]         # supprime par index
```

* Parcourir une liste :

```
for val in Tab:
    print(val)
```

Avec indice :

```
for i in range(len(Tab)):
    print("Index", i, ":", Tab[i])
```

```
len(Tab)            # longueur
Tab.sort()          # tri
Tab.reverse()       # inverse l'ordre
sum(Tab)            # somme de la liste
2 in Tab            # teste la présence
```

Il est possible de déclarer des listes à deux indices ou liste de liste. Par Exemple :

```
Mat = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
print(Mat[1][2])  # 6
```

1.6.1. Les tuples :

Un tuple est une liste non modifiable. Il est généralement défini par une paire de parenthèses contenant les éléments séparés par des virgules.

Les opérations que l'on peut effectuer sur des tuples sont similaires à celles que l'on effectue sur les listes, sauf que les tuples ne sont pas modifiables.

Exemple :

t = ('A', 'l', 'g', 'é', 'r', 'i', 'e')

t = ('jijel',) + t[1:] ça donne ('jijel', 'A', 'l', 'g', 'é', 'r', 'i', 'e')

1.6.2. Les dictionnaires :

Les données composites telles que les chaînes, listes et tuples sont des séquences, c'est-à-dire des suites ordonnées d'éléments. L'accès à un élément dans de telles données est possible grâce à un indice (son emplacement).

Le dictionnaire est un autre type de données composite modifiable mais pas une séquence. L'accès à un élément dans de telles données est possible grâce à une clé, laquelle pourra être alphabétique, numérique, ou même d'un type composite.

Comme dans une liste, les éléments dans un dictionnaire peuvent être des valeurs numériques, des chaines, des listes, des tuples, des dictionnaires, des fonctions, des classes ou des instances.

Création d'un dictionnaire

```
dict = { "nom": "Ahmed","age": 30, "ville": "jijel" }
print(dict["nom"])    # Ahmed
dict["age"] = 31
```

Quelques méthodes utiles

```
dict.keys()       # Renvoie toutes les clés
dict.values()     # Renvoie toutes les valeurs
dict.items()      # Renvoie toutes les paires (clé, valeur)
dict.clear()      # Vide complètement le dictionnaire
```

1.7. Les fichiers

Il est possible de lire des données ou afficher des résultats à partir des fichiers. Cependant Pour lire ou écrire dans un fichier, celui-ci doit être ouvert. Python utilise la fonction open() pour ouvrir un fichier :

```
fichier = open("fichier.txt", "mode")
```

mode :

- "r" : lecture (read)
- "w" : écriture (write, écrase le fichier s'il existe)
- "a" : ajout (append)
- "x" : création (échoue si le fichier existe déjà)

Toujours fermer le fichier après usage avec fichier.close().

1.7.1. Ecrire dans un fichier :

```
fichier = open("fich.txt", "w")

fichier.write("Python\n")

fichier.close()
```

*** Ajouter dans un fichier :**

```
fichier = open("fich.txt", "a")

fichier.write("Langage de programmation\n")

fichier.close()
```

```
fichier = open("fich.txt", "r")
print(fichier.read() )
fichier.close()
```

1.7.2. Lire un fichier :

```
fichier = open("file.txt", "r")  # mode lecture
f = fichier.read()
print(f)
fichier.close()
```

La méthode read utilisée sans argument, renvoie le contenu complet du fichier sous forme d'une chaîne de caractères.

*** Lire ligne par ligne :**

```
fichier = open("file.txt", "r")
for ligne in fichier:
    print(ligne.strip())
fichier.close()
```

strip() : Nettoie les espaces et sauts de ligne

Avec with (fermeture automatique) :

```
with open("file.txt", "r") as fichier :
    f= fichier.read()
    print(f)
```

Exemple :

Créer un fichier notes.txt contenant des noms et des notes, puis lit et affiche ces notes.

```
# Écriture
with open("notes.txt", "w") as f:
    f.write("Ali 15\n")
    f.write("Ahmed 12\n")
# ouvrir  le fichier notes.txt en mode écriture si le fichier n(existe pas python le
# crée s'il existe déjà , il sera écrasé as f : f est un aléa pour le fichier(notes)
# puis on écrit via f les nom et les notes

with open("notes.txt", "r") as f:
    for ligne in f:
        nom, note = ligne.strip().split()
        print(nom, "à une note de ", note)
```

ligne.strip().split() retourne une liste avec deux éléments : [nom, note] les stoke dans nom et note.

split() : Coupe la ligne en morceau (liste de mots)