Chap 2. Affichage

Fonction print()

- Affiche l'argument passé
- Insère un retour à la ligne (par défaut)
- Si on ne souhaite pas afficher le retour à la ligne: on peut utiliser l'argument par mot-clé **"end"**

In [1]:
```
print ("Bonjour Tout le monde !")
print ("Après 1")
print ("Bonjour Tout le monde !" , end = "")
print("Après 2")
```

```
Bonjour Tout le monde !
Après 1
Bonjour Tout le monde !Après 2
```

In [2]:
```
print ("Hello","world!")
print ("Hello", end =" ") ; print ("world!")
```

```
Hello world!
Hello world!
```

Afficher le contenu d'une variable quel que soit son type.

In [3]:
```
# Exemple ( pour un entier ) :
var = 3
print ( var )
```

```
3
```

Afficher le contenu de plusieurs variables (quel que soit leur type) en les séparant par des virgules :

In [4]:
```
x = 22
nom = "Ahmed"
print (nom , "a", x, "ans")
```

```
Ahmed a 22 ans
```

Afficher sur une seul ligne sans séparateurs:

In [5]:
```
x = 22
nom = "Ahmed"
print (nom , "a", x , "ans" , sep=" ")
print (nom , "a", x , "ans")
print (nom , "a", x , "ans" , sep="*")
```

```
Ahmed a 22 ans
Ahmed a 22 ans
Ahmed*a*22*ans
```

Écriture formatée :

- Depuis Python 3.6 : f-string ,
- comment ?
  - Précéder la chaine de caractère par la lettre f
  - Utilisation des {} pour afficher le contenu des variables (dans l'ordre respectif)

In [6]:
```
prixHT = 200;
prixTTC = 1.2 * prixHT

print (f"Le prix HT est : {prixHT}, et le prix TTC est : {prixTTC}" )
```

```
Le prix HT est : 200, et le prix TTC est : 240.0
```

Une variable peut être utilisée plusieurs fois pour une même f-string :

In [7]:
```
var = "to"
print (f"{ var } et { var } font { var }{ var }")
```

```
to et to font toto
```

On peut mettre entre les accolades des valeurs numériques ou des chaînes de caractères :

In [8]:
```
print (f"Afficher l'entier {10} et le float {2.340876:.2f}")
print (f"Afficher l'entier 10 et le float 2.34")
```

```
Afficher l'entier 10 et le float 2.34
Afficher l'entier 10 et le float 2.34
```

In [9]:
```
# print (f"Afficher la chaine "Classe API"")
# print (f"Afficher la chaine {"Classe API"}")
# print ("Afficher la chaine "Classe API"") # sans formatage
```

In [10]:
```
# print("\"Classe API\" !")

# print("""Afficher "Classe API" !""")

# print (f"Afficher la chaine {'Classe API'}") #guillemets simple à l'intérieur


# ch = "Classe API"
# print (f"Afficher la chaine {ch}") #guillemets simple à l'intérieur des accola
```

Écriture formatée : Spécification de format de l'affichage des variables :

méthode .format()

In [11]:
```
prixHT = 100;
# print(prixHT);
prixTTC = 1.2 * prixHT

ch = "Le prix HT est : {}, et le prix TTC est : {}"
print(ch.format(prixHT , prixTTC))
```

```
Le prix HT est : 100, et le prix TTC est : 120.0
```

In [12]:
```
G = 4500; C = 2575; total = 14800
prop_GC = (4500 + 2575) / 14800
print (" La proportion de GC est ", prop_GC ) # par défaults 16 chiffres après l
```

```
La proportion de GC est  0.4780405405405405
```

Trop de décimales (seize dans le cas présent). Pour un résultat lisible, spécifier le format de l'affichage entre les accolades {}

In [13]:
```
print (f"La proportion de GC est { prop_GC :.2f}")
print (f"La proportion de GC est { prop_GC :.3f}")
```

```
La proportion de GC est 0.48
La proportion de GC est 0.478
```

In [14]:
```
# Formater en entier :d (decimal integer) :
nb = 8
x = nb//4 # division non entière
print (f"le quart de {nb} est  {x :d} ")
```

```
le quart de 8 est  2
```

Alignement et largeur :

- À droite : >
- À gauche : <
- Centré : ^

In [15]:
```
print ('@',10,'@', sep="") ; print ('@',1000,'@', sep="")
print (f"@{10:>6d}@") ; print (f"@{1000:>6d}@")
```

```
@10@
@1000@
@    10@
@  1000@
```

In [16]:
```
print (f"@{10:<6d}@") ; print (f"@{1000:<6d}@")
```

```
@10    @
@1000  @
```

In [17]:
```
print (f"@{10:^6d}@") ; print (f"@{1000:^6d}@")
```

```
@  10  @
@ 1000 @
```

In [18]:
```
print (f"@{10:*^6d}@") ; print (f"@{1000:*^6d}@")
```

```
@**10**@
@*1000*@
```

In [19]:
```
print (f"@{99:0>6d}@") ; print (f"@{7777:0>6d}@")
```

```
@000099@
@007777@
```

Écriture formatée : Largeur et alignement / string:

In [20]:
```
print ("@Filiere API2@") ; print ("@Sections ABC@")
print (f"Filiere @{'API2':>6s}@") ; print (f"Sections @{'ABC':>6s}@") # Largeur
```

```
@Filiere API2@
@Sections ABC@
Filiere @  API2@
Sections @   ABC@
```

Écriture des accollades : il faut les doubler pour éviter mauvaise interprétation

In [21]:
```
# print (f"Reamarque : accolades sans variable {")
# print (f"Reamarque : accolades sans variable }")
# print (f"Reamarque : accolades sans variable \{")
# print (f"Reamarque : accolades sans variable \}")
# print (f"Reamarque : accolades sans variable {}")
```

In [22]:
```
print(f"Afficher accolade Ouvrante : {{")
```

```
Afficher accolade Ouvrante : {
```

In [23]:
```
print(f"Afficher accolade Fermante : }}")
```

```
Afficher accolade Fermante : }
```

In [24]:
```
print(f"Afficher les deux accolades : {{ du texte ici }}")
```

```
Afficher les deux accolades : { du texte ici }
```

Écriture formatée : appel à fonction ou expression dans {}

In [25]:
```
print (f"Le résultat de 5 * 5 vaut {5 * 5}")
print (f"Résultat d'une opération avec des floats : {(4.1 * 6.7)}")
print (f"Le minimum est { min (1, -2, 4)}")
entier = 2
print (f"Le type de { entier }")
```

```
Le résultat de 5 * 5 vaut 25
Résultat d'une opération avec des floats : 27.47
Le minimum est -2
Le type de 2
```

Chap 3. Listes

- Définition :
  - Une liste est une structure de données contenant un ensemble d'éléments pas nécessairement du même type

- Permet une grande flexibilité.
- Une liste est déclarée et initialisée par une série de valeurs, entre crochets [ ], séparées par des virgules.
- "Tableau Dynamique"

Exemples : (L'affichage d'une liste, se fait telle qu'elle a été saisie)

In [26]:
```
animaux = ["girafe", "tigre", "Lion", "Chat"]
Nombres = [5, 2 , 1 , 15]
mixte = ["girafe ", 5, " Chat ", 15]

print(animaux) ; print(Nombres) ; print(mixte)
print(animaux[0])
```

```
['girafe', 'tigre', 'Lion', 'Chat']
[5, 2, 1, 15]
['girafe ', 5, ' Chat ', 15]
girafe
```

- Utilisation : - Pour une liste L de n éléments : indices de 0 à n-1 (n > 0) - Accèder au 1er élément de la liste : L[0] - Accèder au 2ème élément de la liste (s'il existe) : L[1] ... - Accèder au nème élément de la liste : L[n-1] - Attention au débordement : géré par Python (≠ Langage C)

In [27]:
```
print(animaux[0]) ; print(animaux[3]) ; print(animaux[9]) ;
```

```
girafe
Chat
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[27], line 1
----> 1 print(animaux[0]) ; print(animaux[3]) ; print(animaux[9])

IndexError: list index out of range
```

Fonction len() : longueur ou nombre d'éléments de la liste

In [29]:
```
x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(len(x))
```

```
10
```

Listes : Opérateur concaténation +

In [30]:
```
L1 = [" girafe ", " tigre "]
L2 = [" singe ", " souris "]

print((L1 + L2))
```

```
[' girafe ', ' tigre ', ' singe ', ' souris ']
```

Listes : Opérateur de reproduction : multiplication par entier *

In [31]:
```
print((L1 + L2)*2)
print(L1*3)
print(L1 + L2*3)
```

```
[' girafe ', ' tigre ', ' singe ', ' souris ', ' girafe ', ' tigre ', ' singe ',
' souris ']
[' girafe ', ' tigre ', ' girafe ', ' tigre ', ' girafe ', ' tigre ']
[' girafe ', ' tigre ', ' singe ', ' souris ', ' singe ', ' souris ', ' singe ',
' souris ']
```

In [32]:
```
print(L1*L2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[32], line 1
----> 1 print(L1*L2)

TypeError: can't multiply sequence by non-int of type 'list'
```

Listes : Opérations

Opérateur ajout à la fin (queue) : + ou méthode .append()

In [34]:
```
a = []

# ajout par + :
a = a + [15] ; a = a + [ -5]
print(a)
```

```
[15, -5]
```

In [35]:
```
# puis avec la méthode .append() :
a.append (13)
a.append ( -3)
print(a)
```

```
[15, -5, 13, -3]
```

Listes : indice négatif

In [36]:
```
#indice negatif
L = ["A", "B", "C", "D", "E", "F"]
print(L[-1])
print(L[-2])
print(L[-3])
```

```
F
E
D
```

Listes : Tranches : [m : n] :: de l'indice m à l'indice (n-1) # indice n exclu

In [37]:
```
# tranches
L = ["A", "B", "C", "D", "E", "F"]
print(L[0:2])
```

```
print(L[0:3])
print(L[:4])
```

```
['A', 'B']
['A', 'B', 'C']
['A', 'B', 'C', 'D']
```

In [38]:
```
print(L[0:])
print(L[1:])
```

```
['A', 'B', 'C', 'D', 'E', 'F']
['B', 'C', 'D', 'E', 'F']
```

In [39]:
```
print(L[:])
```

```
['A', 'B', 'C', 'D', 'E', 'F']
```

In [40]:
```
print(L[1:-1])
print(L[1:-2])
```

```
['B', 'C', 'D', 'E']
['B', 'C', 'D']
```

In [41]:
```
print(L[-2:])
```

```
['E', 'F']
```

Tranches : parcours avec pas ≠ 1

In [42]:
```
x = [100, 10, 20, 30, 40, 50, 60, 70, 80, 90]

print(x[0:9:1]) # x[deb:fin:pas] avec indice fin exclu
print(x[0:len(x):1]) # x[deb:fin:pas] avec indice fin exclu
```

```
[100, 10, 20, 30, 40, 50, 60, 70, 80]
[100, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

In [43]:
```
print(x[0:9:2])
print("")
print(x[0:9:3])
```

```
[100, 20, 40, 60, 80]

[100, 30, 60]
```

In [44]:
```
print(x[::1])
print(x[::2])
print(x[::3])
```

```
[100, 10, 20, 30, 40, 50, 60, 70, 80, 90]
[100, 20, 40, 60, 80]
[100, 30, 60, 90]
```

Fonction list() : permet de créer une liste

In [45]:
```
print(list('abc'))
```

```
['a', 'b', 'c']
```

Fonction range() : génèrer des nombres entiers compris dans un intervalle

In [46]:
```
# a = []
a = list(range(10))
print(a)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Fonction range() : génèrer des nombres entiers compris dans un intervalle

In [47]:
```
L = list(range(15, 20))
print("\n",L)
```

```
[15, 16, 17, 18, 19]
```

In [48]:
```
# ordre INVERSE
L = list(range(10,0,-1))
print("\n",L)
```

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

In [49]:
```
print(list(range(10,0,-2)))
print(list(range(10,-1,-2)))
```

```
[10, 8, 6, 4, 2]
[10, 8, 6, 4, 2, 0]
```

Autres Exemples :

In [50]:
```
L = list(range(0, 1000, 200)) # pas de 200
print(L)
```

```
[0, 200, 400, 600, 800]
```

In [51]:
```
L = list(range(2, -2, -1)) # ordre inverse : pas = -1
print(L)
```

```
[2, 1, 0, -1]
```

In [52]:
```
#-----------------------------------
#-------------------------- ATTENTION
#-----------------------------------

L = list(range(10,0)) # pas par défaut = 1, d'où résultat vide
print(L)
```

```
[]
```

Liste de Listes

In [53]:
```
#liste de liste
mod_0_3 = [0 , 3, 6, 9]
mod_1_3 = [1, 4, 7, 10]
mod_2_3 = [2, 5, 8, 11]

LL = [mod_0_3 , mod_1_3, mod_2_3]

print(f"{'LL ':10s}:",LL)
print(f"{'LL[1] ':10s}:", LL[1])
print(f"{'LL[2][4]':10s}:",LL[1][3])
```

```
LL        : [[0, 3, 6, 9], [1, 4, 7, 10], [2, 5, 8, 11]]
LL[1]     : [1, 4, 7, 10]
LL[2][4]  : 10
```

Liste : Min, Max, Sum :

In [54]:
```
liste = list(range(10))
print(liste)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(sum(liste))
# 45

print(min(liste))
# 0

print(max(liste))
# 9
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
45
0
9
```

Immuable , Mutable

Type immuable : int, float, bool , str , tuple

- a = 5 est une reference vers le nombre 5 (5 est immuable)
- a += 2 est une reference vers le nombre 7 (7 est immuable)
- L'identifiant en mémoire d'une variable a est id(a)

In [55]:
```
# L'identifiant en mémoire d'une variable a est id(a)
a = 7
print("id (a) :" , id(a))
print("id (7) :" , id(7))
```

```
id (a) : 1372135254512
id (7) : 1372135254512
```

In [56]:
```
b = a
print("id (a) :" , id(a))
print("id (b) :" , id(b))
print("")
```

```
id (a) : 1372135254512
id (b) : 1372135254512
```

In [57]:
```
a += 2
print(" ")
print("id (a) :" , id(a))
print("id (9) :" , id(9))

print("")
print("id (7) :" , id(7))
```

```
id (a) : 1372135254576
id (9) : 1372135254576

id (7) : 1372135254512
```

In [58]:
```python
prenom = "Patrick"

print("prenom :", prenom, " \nid              :",  id(prenom)%1000000)
print("id de \"Patrick\" :",  id("Patrick")%1000000)

# prenom = "Patricke"
# print("prenom Patricke :", id(prenom)%1000000)

# print("")

# prenom += 'e'
# print("id de ", prenom, ":", id(prenom)%1000000)

prenom[0] = "p"
```

```
prenom : Patrick
id              : 699632
id de "Patrick" : 699632
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[58], line 14
      4 print("id de \"Patrick\" :",  id("Patrick")%1000000)
      6 # prenom = "Patricke"
      7 # print("prenom Patricke :", id(prenom)%1000000)
      8
   (...)
     11 # prenom += 'e'
     12 # print("id de ", prenom, ":", id(prenom)%1000000)
---> 14 prenom[0] = "p"

TypeError: 'str' object does not support item assignment
```

Type Mutable : Liste

In [59]:
```python
list_1 = [1, 2]
print(list_1)
# print("id list_1 av :", id(list_1)%1000000)
print("id list_1 av :", id(list_1))


list_1.append(31)
print(list_1)
# print("id list_1 ap :", id(list_1)%1000000)
print("id list_1 ap :", id(list_1))
```

```
[1, 2]
id list_1 av : 1370100930496
[1, 2, 31]
id list_1 ap : 1370100930496
```

Différence entre affectation ' = ' ET ' .copy() '

In [60]:
```
list_1 = [10, 20]
print("id list_1 av :", id(list_1))

list_2 = list_1
print("id list_2 av :", id(list_2))

list_1.append(113)
print("id list_1 ap :", id(list_1))
print("id list_1 ap :", id(list_2))

print(list_1)
print(list_2)
```

```
id list_1 av : 1370100942016
id list_2 av : 1370100942016
id list_1 ap : 1370100942016
id list_1 ap : 1370100942016
[10, 20, 113]
[10, 20, 113]
```

In [61]:
```
list_1 = [10, 20]
list_3 = list_1.copy()
print("L1 Av :",list_1)
print("L3 Av :",list_3)

print("\n- L1.append()\n")
list_1.append(9999)
print("L1 Ap :",list_1)
print("L3 Ap :",list_3)

print("")
print("id list_1 ap :", id(list_1)%1000000)
print("id list_3 ap :", id(list_3)%1000000)
```

```
L1 Av : [10, 20]
L3 Av : [10, 20]

- L1.append()

L1 Ap : [10, 20, 9999]
L3 Ap : [10, 20]

id list_1 ap : 936192
id list_3 ap : 937024
```

Boucles

Boucle for

- itère sur les éléments d'une séquence (liste, chaîne de caractères...),
- dans l'ordre dans lequel ils apparaissent dans la séquence.

Exemple : la séquence est une liste de chaines

In [63]:
```
X = ["A", "B", "C", "D", "E", "F"]
words_API = ['cat', 'window', 'peace', 'University']
for wpk in words_API:
    print(wpk)
print("Fin de la boucle for")
```

```
cat
window
peace
University
Fin de la boucle for
```

Exemple : la séquence est une chaine

In [64]:
```
mot = "XYZ"
for l in mot:
    print(l)
```

```
X
Y
Z
```

Exemple de séquence : liste d'entiers

In [65]:
```
for i in [10, 2, 3]:
    print(i)
```

```
10
2
3
```

Exemple de séquence : Tranche de liste

In [66]:
```
words = ['cat', 'window', 'peace', 'University']
for w in words[1:3]:
    print(w)
```

```
window
peace
```

Exemple : la séquence est une plage d'entiers (range)

- Attention : il faut respecter l'indentation au cas de plusieurs instructions !

In [68]:
```
for i in range(5):
    a = 2*i
    print(a)
```

```
0
2
4
6
8
```

Boucle while

```
- while (condition):
    instuction 1
```

```
        instuction 2
        ...
    - s'exécute tant que la condition est Vrai
```

In [69]:
```python
i = 10
while (i):
    print(i)
    i //= 4
print("Fin Boucle while")
```

```
10
2
Fin Boucle while
```

Instruction break

- Termine la boucle la plus imbriquée, sans compléter l'itération en cours.
- Ne peut apparaître qu'à l'intérieur d'une boucle for ou while,
- mais pas dans une définition de fonction à l'intérieur de cette boucle:

In [70]:
```python
#break

for n in range(61,71):
    print(f"n : {n} ")
    if n % 7 == 0:
        print(f"Enfin, {n} Multiple de 7")
        break
        print("ap Break, Interieur if")
    print(f"Apres break : {n} n'est pas multiple de 7")
```

```
n : 61
Apres break : 61 n'est pas multiple de 7
n : 62
Apres break : 62 n'est pas multiple de 7
n : 63
Enfin, 63 Multiple de 7
```

Instruction continue

- fait passer la boucle à l'itération suivante, en sautant le reste de l'itération actuelle

In [71]:
```python
# continue
for num in range(2, 6):
    print("\niterer : ",num)
    if num % 2 == 0:
        print(f"{num} est un nombre Pair")
        continue
    print(f"Après Continue, num : {num}")
    print("Nombre Impair : ", num)
```

```
iterer :  2
2 est un nombre Pair

iterer :  3
Après Continue, num : 3
Nombre Impair :  3

iterer :  4
4 est un nombre Pair

iterer :  5
Après Continue, num : 5
Nombre Impair :  5
```

Tests

if (else)

In [72]:
```python
a = 5
if a == 5:
    print("égalité")
else:
    print("Difference")
```

```
égalité
```

if (elif) (else)

In [73]:
```python
a = 5
if a == 5:
    print("égalité")
elif a < 5:
    print("Inferieur")
else:
    print("Supérieur")
```

```
égalité
```

In [74]:
```python
# Rappel : double test
x = 10
if 8 < x <= 9:
    print("T")
else:
    print("F")
```

```
F
```

In [75]:
```python
a = 7 ; b = 4

# a = 111
# b = 100

# a = 10; b = False

print("a ^ b : " , a ^ b)
print("a & b : ", a & b)
print("a and b : " , a and b)
```

```
print("a or b : ", a or b)
print("a | b : ", a | b)
```

```
a ^ b :  3
a & b :  4
a and b :  4
a or b :  7
a | b :  7
```

In [76]:
```
import sys
print(sys.version)
```

```
3.9.19 (main, Mar 21 2024, 17:21:27) [MSC v.1916 64 bit (AMD64)]
```

In [77]:
```
# ------------------------------------------------------
# ------------------------------------------------------

# # MARCHE UNIQUEMENT A PARTIR DE Python 3.10

# ------------------------------------------------------
# ------------------------------------------------------

# Print a message, depending on the day of the week
day = "Monday"

match day:
    case "Monday":
        print("It's Monday!")
    case "Tuesday":
        print("It's Tuesday!")
    case "Wednesday":
        print("It's Wednesday!")
    case _:
        print("It's not Monday, Tuesday, or Wednesday.")
```

```
  Cell In[77], line 12
    match day:
          ^
SyntaxError: invalid syntax
```