

Memento PYTHON

Types de variables - fonctions de conversions

- **Type integer → Nombre entier**
`int()` → convertit si possible un décimal ou texte en entier
- **Type float → Nombre décimal**
`float()` → convertit si possible un entier ou texte en décimal
- **Type string → Chaîne de caractères (texte)**
suite de signes définie en la délimitant par des guillemets
`str()` → convertit un nombre en chaîne
- **Type boolean → Logique**
ne prend que deux valeurs : **True** et **False**
- **Affectation =**
`x = ...` → lire « x prend la valeur... »  **astuces**

```
x="12" # x est de type string
y=3    # y est du type integer
z=x+y  # erreur
z=int(x)+y # donne 15
z=x+str(y) # donne "123"
```

```
a=(1+1>2) # a booléen qui vaut False
b=(2*3==6) # b booléen qui vaut True
```

```
a,b=12,15 # équivaut à a=12 et b=15
x+=2      # équivaut à x=x+2
x-=1      # équivaut à x=x-1
```

Entrées, sorties console, opérations numériques

- **Entrée → `input("message")`** : lit un texte saisi au clavier.
 Renvoie donc toujours une chaîne de caractères.
 conversion possible en nombre par `int()` ou `float()`
- **Sortie en console → `print( , , ...)`** : affiche en console les valeurs de tout type en les séparant par une tabulation.
- **Opérations sur les nombres**
`/` → division décimale
`//` → quotient de la division entière
`%` → reste de la division entière
`**` → puissance (remarque : $a^{**}0.5 = \sqrt{a}$)
`abs()` → valeur absolue
`round(x,d)` → arrondi le nombre x à d décimales

```
a=input("Texte?") # a type string
b=float(input("Nombre?")) # b type float
```

```
a=45*2
print("Coût=",a,"€") # affiche "Coût= 90 €"
```

```
d=11/4 # d vaut 2.75
q=11//4 # q vaut 2
r=11%4 # r vaut 3 car 11= 2*4+3
p=4**3 # p vaut 64
r=16**0.5 # r vaut racine(16) soit 4
x=abs(7-10) # x vaut 3
y=round(4/3,4) # y vaut 1,3333
```

Chaînes de caractères

- **Concaténation +** → attache les textes pour n'en former qu'un
- **Caractères d'échappement**
le signe `\` permet de transformer le caractère qui suit
`\n` → saut de ligne (new). `\t` → tabulation
`\"` ou `\'` → guillemet qui ne ferme pas la chaîne
- **longueur d'une chaîne :**
`len()` → renvoie le nombre de caractères d'une chaîne, espaces compris.
- **Indexation** Chaque caractère de la chaîne est indexé (numéroté) **en commençant par 0**
`Chaîne[i]` → renvoie le caractère de rang i
 **astuces :**
`MaChaîne[-1]` → dernier caractère
`MaChaîne[-2]` → avant dernier caractère, etc...
`MaChaîne[i:j]` → caractères indexés de i à j-1
 **Attention : on en peut pas modifier un caractère d'une chaîne par son index, seulement le lire !**
- **Code ASCII**
`chr(x)` → renvoie le caractère de code ASCII x
`ord(char)` → renvoie le code ASCII du caractère char
 `chr(10)` ou `chr(13)` → saut de ligne. `chr(9)` → tabulation

```
T1="Ceci est" # variable type string
T2=" un test." # variable type string
T=T1+T2 # T vaut "Ceci est un test"
```

```
T="ceci est \nun test" # Imprime: ceci est
print(T) # un test
T="Il m'a dit:\\"Attention\\"
print(T) # imprime : Il m'a dit:"Attention"
```

```
T="Ceci est un test"
L=len(T) # entier valant 16
print(T[0]) # écrit 'C'
x=T[2] # x vaut 'c'
y=T[L] # erreur de dépassement
z=T[L-1] # z vaut 't' (dernière lettre)
z=T[-1] # z vaut 't' (dernier index)
z=T[-2] # z vaut 't' (avant dernier index)
z=T[1:6] # z vaut 'eci e' (indexe 1 à 5)
```

```
T="Python"
T[1]="y" # erreur: affectation non autorisée
```

```
x=chr(65) # x vaut 'A'
y=ord('B') # y vaut 66
```

Listes et tuples

- Liste** → suite indexée et modifiable d'éléments de tout type **Attention** : l'indexation commence à 0

NomListe = [élément1, élément2, élément3,...]
NomListe[i] → élément d'index i (lecture ou écriture)
NomListe[0] → premier élément
NomListe[-1] → dernier élément

- Principales fonctions** :

Longueur : **len()** → renvoie le nombre d'éléments
Ajout : NomListe.append(x) → ajoute x en fin de liste
Insertion : NomListe.insert(i, x) → insert x à l'index i
Suppression : NomListe.pop() → supprime le dernier élément
NomListe.pop(i) → supprime élément d'indexe i

- Tuples** → un tuple est une liste non modifiable

NomTuple = (élément1, élément2, élément3,...)
NomTuple[i] → élément d'indexe i en lecture seule

- Liste de listes** → une liste peut contenir des listes !!
NomListe[i][j] désigne l'élément d'index j de la liste d'index i

```
L=[5,8,"Julie"] # Liste de 3 éléments
a=L[0] # lecture: a vaut 5
L[1]=10 # Ecriture: L vaut [5,10,"Julie"]
b=L[-1] # b vaut 'Julie'
x=len(L) # x vaut 3
c=L[3] # erreur dépassement
L.append(3) # L vaut [5,10,'Julie',3]
L.insert(2,"P") # L vaut [5,10,'P','Julie',3]
L.pop() # L vaut [5,10,'P',Julie']
L.pop(2) # L vaut [5,10,'Julie']
```

```
T=(4,8,10)
x=T[0] # x vaut 4
T[0]=5 # erreur, T non modifiable
```

```
Tab=[[4,2,9],[0,7,8]]
x=Tab[0] # x vaut [4,2,1] : index 0 de Tab
y=Tab[0][2] # y vaut 9 : index 2 de Tab[0]
```

Tests

- if** test 1 : # un test est une valeur booléenne (logique)

| bloc si test1 vérifié

elif test 2 : # (facultatif). Sinon si :

| bloc si test 1 non vérifié mais test2 vérifié

.....

else : # (facultatif). Sinon

| bloc si aucun des tests précédent n'est vérifié

suite du programme

```
a=float(input("Donne un nombre"))
if a==0:
    texte="nul"
elif a>=0:
    texte="positif"
else:
    texte="négatif"
print(texte)
```

Boucle « Tant que »

- While** test : # Tant que ...

| Bloc répété tant
que test vérifié
suite du programme

```
# comparateurs
== # égal
!= # différent
> # supérieur
>= # sup ou égal
```

```
a=5
while a<10:
    print(a)
    a=a+2 # affiche 5,7 et 9
```

Boucle « Pour... »

- For** variable in liste : # Pour chaque ... dans... :
| Bloc répété pour chaque valeur de la variable parcourant la liste
suite du programme
- Génération de listes d'entiers**
range(a) → tous les entiers de [0 ; a[
range(a, b) → tous les entiers de [a ; b[
range(a, b, p) → tous les entiers de [a ; b[de p en p

```
#-----
# Pour rompre une boucle
#-----
break # arrête la boucle
continue # passe au tour suivant
```

```
texte=input("12*5=?")
while texte!="60":
    print("C'est faux!")
    texte= input("12*5=?")
print("Correct")
```

```
for k in [4,5,8]:
    x=k*k
    print(x) # affiche 16, 25 et 64

for i in range(3):
    print(i) # affiche 0,1 et 2

for i in range(2,5):
    print(i) # affiche 2,3 et 4

for i in range(2,10,3):
    print(i) # affiche 2,5 et 8
```

Logique : variables booléennes

- Une variable booléenne** ne prend que 2 valeurs **True**, **False**
- Opérateurs booléens**
a or b → vaut True si et seulement si l'un au moins vaut True
a and b → vaut True si et seulement si les deux valent True
not a → contraire de a : True si a False, False si a True
a in Liste → vaut True si et seulement si a élément de Liste

```
a=(1==2) # a booléen valant False
x=3
b=(x>0) # b booléen valant True
c=(a or b) # c vaut True
d=(a and b) # d vaut False
e= not a # e vaut True
L=[2,4,6]
f=(3 in L) # f vaut False
```