

VARIABLES ET AFFECTATION EN PYTHON

Voir le cours en ligne pour plus de détails.

1 Commandes de base

```
# ceci est un commentaire

print('Hello World !') # affichage à l'écran

# pour avoir de l'aide sur la fonction print
help(print)
```

Le texte est encadré par des guillemets (simples ou doubles). Si on ouvre avec un type de guillemets, on ferme avec le même type.

Pour écrire plusieurs éléments à la suite (séparés d'un espace), on les sépare par des virgules en argument de `print`. Par défaut, `print` fait ensuite un retour à la ligne. Ces comportements par défaut peuvent être changés (voir l'aide).

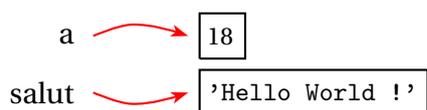
2 Affectation

Première affectation

```
a = 18
print(a)
# la variable a pointe vers la valeur 18

salut = 'Hello World !'
print(salut)
# salut pointe vers le texte 'Hello World !'

print('Python vous dit :', salut)
```



Modification d'une variable

```
a = 'glop'
b = 'pas glop'
a = b
# écrase la variable a pour lui donner la
# valeur 'pas glop'
```

Attention : Le signe égal n'est pas symétrique pour Python. « = » se traduit par « pointe vers ».

Affectations simultanées

```
(a,b,c) = (1,'deux',3)
print('je compte',a,b,c)
```

Échange de deux variables

```
a = 1
b = 5
(a,b) = (b,a) # échange les valeurs
```

3 Opérations sur les nombres

- type **int** : type entier.
- type **float** : type nombre flottant (décimal) on utilise le *point* décimal.

```
2**3      # 8 : la puissance s'écrit **
7//2     # 3 : division entière
7%2      # 1 : reste de division entière

x = 3*2   # x = 6
x = x+1   # x = 7
x += 1    # x = 8
x *= 2    # x = 16
y = x//2  # y = 8

type(x)   # int : type de x
type(x/2) # float
int(5.2)  # 5 : convertit en int
```

4 Tests logiques

- type **bool** : type booléen (vrai ou faux).

Les deux valeurs booléennes s'écrivent **True** et **False**.  ne pas oublier la majuscule.

```
type(False) # bool

x = 2
4 == 5      # False : teste l'égalité
2 == x      # True
2 != x      # False : teste la différence
x < (x+1)   # True
x >= 4      # False
not (x>=4)  # True : contraire
```

Attention : Ne pas confondre l'affectation « = » (non symétrique) avec le test d'égalité « == » (symétrique).

5 Compléments sur les tests logiques

Lorsque l'on met plusieurs opérateurs logiques à la suite, Python étudie leur conjonction : il met un « et » entre eux.

```
3<4<6>4    # True 3<4 et 4<6 et 6>4
```

Remarque : Par défaut, **False** est associé à 0 et **True** associé à 1. Python fait automatiquement la conversion si nécessaire.

```
1 == True   # True
0 == False  # True
1 + True    # 2
```

6 Fonctions

```
def maFonction(arg):  
    suite d'instructions  
    suite d'instructions  
    ...  
    return resultat
```

Attention :

- Ne pas oublier d'indenter le bloc.
- Après return, la fonction s'arrête et sort du bloc (même s'il y a encore des instructions).
- La fonction ne modifie pas la variable globale (sauf mot clef spécifique).

Pour pouvoir réutiliser le résultat renvoyé, il faut l'affecter à une variable (sinon il est perdu...).

```
# les deux fonctions font la même chose  
  
def somme1(a,b):  
    d = a+b  
    return d  
  
def somme2(a,b):  
    return a+b  
  
x = somme1(3,2)      # x = 5
```

7 Erreurs récurrentes

Lorsque Python renvoie une erreur, il indique la ligne et le type d'erreur.

Pour les erreurs de syntaxe, regarder en priorité :

- pas le même nombre de parenthèses ouvrantes et fermantes.
→ l'erreur est indiquée à la ligne qui suit.
- guillemets ouverts et non fermés
→ l'erreur est indiquée à la ligne qui suit.
- simple égal au lieu d'un double pour un test d'égalité.
- manque deux points « : » après def.

Autre erreurs courantes :

- mauvaise indentation.
- variable non définie
(ou utilisation du signe égal dans le mauvais sens).
- ...