

prise-en-main-complete

September 9, 2026

1 Variables

Définir une variable

```
a = 18
```

Afficher la *valeur* de **a** (**print** est notre ami)

```
[ ]:
```

2 Opérateurs

2.1 Opérations arithmétiques

Tester les différents opérateurs

```
3+2, 8*9.5, -3+2-6.1
```

Comparer 7//3 et 7/3

```
[ ]:
```

```
[ ]:
```

2.2 Opérateurs modulo

On peut déterminer le reste de la division euclidienne.

```
7 % 3 # le reste de la division euclidienne de 7 par 3
```

Déclarer une variable **r** égale au reste de la divion de 10 par 7 auquel on ajoute 1.

```
[ ]:
```

2.3 Comparaisons et opérateurs booléens

Les comparaisons produisent un booléen : **True** ou **False**.

Tester par exemple :

```
3 < 5
```

```
7 == 7
```

```
8 != 2
4 <= 4
```

On peut combiner des expressions booléennes avec les opérateurs **and**, **or** et **not**.

```
3 < 5 and 8 > 2
3 < 5 and 8 < 2
3 < 5 or 8 < 2
not (3 < 5)
```

Tester quelques expressions et prévoir leur résultat avant de les exécuter.

```
[ ]:
```

3 Types

Donner les types des objets suivants

```
8, 9.3, True
(6,8.3)
[6,9.3]
"toto"
```

La fonction **type** est notre amie

```
[ ]:
```

4 Tuples

Déclarer un tuple

```
t = (6,2.5,"toto")
```

Déclarer un tuple à 4 champs ayant avec le nom **x**.

```
[ ]:
```

4.1 Accès aux items

Pour accéder au 1er item du tuple **t** (numéro 0).

```
t[0] # 1er membre de t
t[2] # dernier membre de t
```

Afficher les membres 0,1 et 3 de **t**

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

4.2 Modifier un item

Essayer

```
t[0] = 15
```

[]:

5 Listes

Une liste est une suite d'objets que l'on peut modifier.

5.1 Déclaration

Déclarer une liste :

```
L = [12, 7, 3, 18]
```

On peut également mélanger différents types :

```
L2 = [6, 2.5, "toto", True]
```

Déclarer une liste x contenant cinq nombres.

[]:

5.2 Accès aux éléments

Comme pour les tuples, les indices commencent à 0.

```
L = [12, 7, 3, 18]
```

```
L[0]    # premier élément
```

```
L[2]    # troisième élément
```

```
L[-1]   # dernier élément
```

Afficher le deuxième puis le dernier élément de L.

[]:

5.3 Modifier un élément

Contrairement aux tuples, les listes sont **modifiables**.

```
L = [12, 7, 3, 18]
```

```
L[1] = 100
```

Afficher L avant et après la modification.

[]:

5.4 Ajouter et retirer des éléments

La méthode `append` ajoute un élément à la fin de la liste.

```
L.append(25)
```

La méthode `pop` retire un élément et renvoie sa valeur.

```
x = L.pop()      # retire le dernier élément
y = L.pop(1)     # retire l'élément d'indice 1
```

Tester `append` et `pop` en affichant la liste après chaque opération.

[]:

6 Instructions conditionnelles

Une instruction conditionnelle permet de choisir les instructions à exécuter.

```
x = 7
```

```
if x > 0:
    print("x est positif")
else:
    print("x est négatif ou nul")
```

Attention : en Python, les blocs sont délimités par l'**indentation**.

On peut utiliser `elif` lorsqu'il y a plusieurs cas :

```
if x < 0:
    print("négatif")
elif x == 0:
    print("nul")
else:
    print("positif")
```

Écrire un programme qui affiche si un entier `n` est pair ou impair.

[]:

7 Boucles

7.1 Boucle `for`

Une boucle `for` permet notamment de parcourir une séquence.

```
for x in [4, 7, 2]:
    print(x)
```

La fonction `range` produit une suite d'entiers :

```
for i in range(5):
    print(i)
```

Ici, `i` prend successivement les valeurs 0, 1, 2, 3 et 4.

Afficher les carrés des entiers de 0 à 9.

[]:

7.2 Boucle `while`

Une boucle `while` répète des instructions tant qu'une condition est vraie.

```
i = 0
```

```
while i < 5:
    print(i)
    i = i + 1
```

Écrire une boucle qui affiche les entiers de 10 à 1.

[]:

8 Fonctions

On définit une fonction avec le mot-clé `def`.

```
def carre(x):
    return x * x
```

On peut ensuite l'appeler :

```
carre(5)
```

Définir une fonction `maximum(a, b)` qui renvoie le plus grand des deux nombres.

[]:

8.1 Fonctions avec plusieurs instructions

Une fonction peut naturellement contenir des tests, des boucles et des variables locales.

```
def valeur_absolue(x):
    if x >= 0:
        return x
    else:
        return -x
```

Définir une fonction `somme(L)` qui calcule et renvoie la somme des éléments d'une liste `L`, sans utiliser la fonction `sum`.

[]:

9 Petit exercice récapitulatif

Écrire une fonction

```
def compte_pairs(L):
```

qui prend en argument une liste d'entiers L et renvoie le nombre d'éléments pairs de cette liste.

Par exemple :

```
compte_pairs([3, 8, 12, 5, 7, 4])
```

doit renvoyer 3.

[]: