

Programmation en Python

Cours de maths - 2de

Un programme est une suite d'instructions que l'ordinateur exécute **dans l'ordre**, sans rien deviner. En 2de, tu apprends à écrire en **Python** des programmes qui mémorisent des valeurs (les variables), qui prennent des décisions (les conditions), qui se réutilisent (les fonctions) et qui répètent des calculs (les boucles). Chaque programme de ce chapitre a été réellement exécuté : tu peux le recopier dans ton éditeur et vérifier les affichages.

1. Variables et types de valeurs

Variable Une variable est un **nom** qui désigne une valeur gardée en mémoire. On choisit des noms clairs (*prix*, *age*, *total*), sans espace, sans accent, qui ne commencent pas par un chiffre. Python distingue les majuscules des minuscules : *Prix* et *prix* sont deux variables différentes.

Les quatre types à connaître

- **int** : un nombre entier (7 ; -3 ; 0) ;
- **float** : un nombre « à virgule », appelé flottant (3.5 ; -0.25 ; 2.0). En Python, la virgule décimale est un **point** ;
- **str** : une chaîne de caractères, c'est-à-dire du texte entre guillemets ("robot" ou 'robot') ;
- **bool** : un booléen, qui vaut seulement **True** (vrai) ou **False** (faux), avec une majuscule.

```
x = 5
prix = 3.5
mot = "robot"
fini = True
print(type(x), type(prix), type(mot), type(fini))
```

La commande **type(...)** donne le type d'une valeur. Ici elle affiche :

```
<class 'int'> <class 'float'> <class 'str'> <class 'bool'>
```

Piège : 5, 5.0 et "5" sont trois valeurs différentes 5 est un int, 5.0 est un float et "5" est un str (du texte, pas un nombre). Une division avec « / » donne **toujours un**

Fiche en ligne : maths-facile.eu/programmation-en-python-cours-2de/ · Cours, exercices et contrôles corrigés sur

float : $10 / 5$ vaut 2.0 et non 2. Avec des chaînes, le signe + **colle** les textes : "5" + "3" donne "53", pas 8.

Les opérations sur les nombres Addition +, soustraction −, multiplication *, division /, puissance ** ($3^{**}2$ vaut 9). On peut aussi utiliser % qui donne le **reste de la division euclidienne** : $17 \% 5$ vaut 2 car $17 = 5 \times 3 + 2$. Un nombre n est pair si $n \% 2$ vaut 0. Les priorités sont celles des mathématiques ; on ajoute des parenthèses en cas de doute.

2. Affectation, affichage et saisie

Affectation L'instruction **x = 5** se lit « x prend la valeur 5 ». Ce n'est **pas une égalité mathématique** : l'ordinateur calcule d'abord le membre de droite, puis range le résultat dans la variable de gauche. Ainsi **x = x + 1** a un sens : on augmente x de 1.

Suivre les valeurs pas à pas Voici un programme court.

```
a = 6
b = 4
a = a + b
b = a * 2
print(a, b)
```

Ligne 1 : a vaut 6. Ligne 2 : b vaut 4. Ligne 3 : on calcule $6 + 4 = 10$ et on le range dans a, donc a vaut 10 (l'ancienne valeur 6 est perdue). Ligne 4 : b prend $10 \times 2 = 20$. Ligne 5 : le programme affiche

```
10 20
```

Méthode : tracer un programme

1. Fais un tableau avec une colonne par variable et une ligne par instruction exécutée.
2. Pour chaque ligne, calcule avec les valeurs **actuelles** et ne change que la variable de gauche.
3. Repère les **print** : eux seuls produisent un affichage.

La commande **print** affiche des valeurs séparées par des virgules ; Python met un espace entre elles. Pour mélanger texte et nombre avec +, on convertit le nombre avec **str(...)**.

```
n = 12
```

```
print("Le double de", n, "est", 2 * n)
print("n =" + str(n))
```

Affichage obtenu :

```
Le double de 12 est 24
n =12
```

Pour demander une valeur à l'utilisateur, on utilise **input("message")**. Attention : **input renvoie toujours un str**. Il faut convertir avec **int(...)** ou **float(...)** avant de calculer.

```
age = int(input("Quel est ton age ? "))
print("Dans 10 ans, tu auras", age + 10, "ans")
```

Affichage obtenu :

```
Quel est ton age ? 16
Dans 10 ans, tu auras 26 ans
```

Piège classique Sans conversion, `age + 10` provoque une erreur (on ne peut pas additionner du texte et un nombre), et `input() * 2` répéterait le texte : `"7" * 2` donne `"77"`.

3. Conditions et booléens

Condition Une condition est un test dont le résultat est un **booléen** : True ou False. Les symboles sont : `==` (égal à), `!=` (différent de), `<`, `>`, `<=` (inférieur ou égal), `>=` (supérieur ou égal). On les combine avec **and** (« et » : les deux doivent être vraies), **or** (« ou » : au moins une vraie) et **not** (le contraire).

```
x = 7
print(x > 5, x == 8, x != 7, x >= 3 and x < 7, x < 0 or x > 6)
```

Affichage obtenu :

```
True False False False True
```

Explications pour `x = 7` : `7 > 5` est vrai ; `7 == 8` est faux ; `7 != 7` est faux ; « `x >= 3 and x < 7` » est faux car `7 < 7` est faux ; « `x < 0 or x > 6` » est vrai car la deuxième partie l'est.

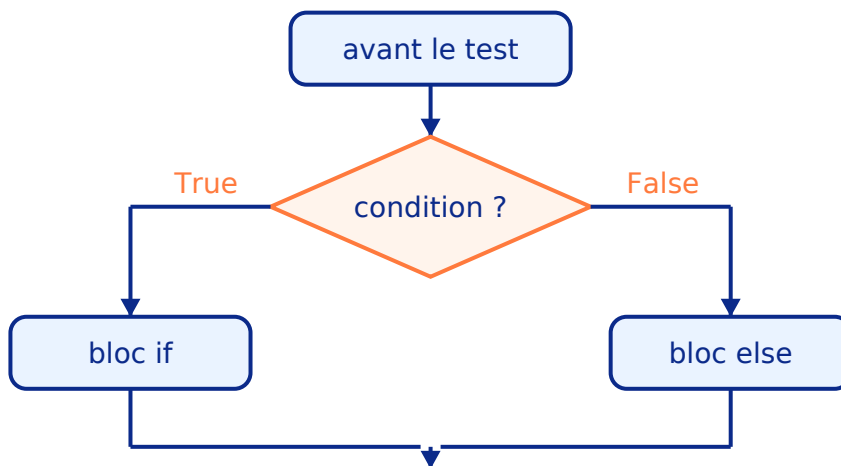
= ou == ? Un seul signe = **affecte** une valeur ; deux signes == **testent** une égalité.

Écrire « if x = 3: » provoque une erreur.

Lien avec les intervalles $x \in [-1 ; 8[$ s'écrit en Python « x >= -1 and x < 8 ». Et $x \notin [2 ; 5]$ s'écrit « x < 2 or x > 5 ».

4. Instructions conditionnelles : if, elif, else

if ... else Après **if condition :**, on écrit un bloc **indenté** (décalé de 4 espaces) exécuté seulement si la condition est vraie. Le bloc qui suit **else :** est exécuté dans le cas contraire. La ligne suivante, **non indentée**, est exécutée dans tous les cas.



```
note = float(input("Note sur 20 : "))
if note >= 10:
    print("Admis")
else:
    print("Refuse")
print("Fin")
```

Affichage obtenu :

```
Note sur 20 : 12.5
Admis
Fin
```

Avec 12.5 la condition est vraie : le programme affiche « Admis » puis « Fin ». Avec 7, il affiche « Refuse » puis « Fin ».

elif pour plusieurs cas elif (« sinon si ») ajoute un test supplémentaire. Python teste les conditions **dans l'ordre** et exécute seulement le bloc de la première qui est vraie ; **else** récupère tout le reste.

```
t = int(input("Temperature en degres : "))
if t < 0:
    print("Gel")
elif t < 20:
    print("Frais")
else:
    print("Doux")
```

Affichage obtenu :

```
Temperature en degres : -4
Gel
```

Méthode : écrire une conditionnelle

1. Liste les cas possibles et vérifie qu'ils couvrent tout, sans se chevaucher.
2. Traduis chaque cas en condition (utilise and/or pour les intervalles).
3. N'oublie ni les « : » ni l'indentation.
4. Teste avec une valeur par cas, **et avec les valeurs aux bords** (ici 0 et 20).

L'indentation compte En Python, c'est le décalage qui délimite un bloc. Une ligne mal décalée ne fait pas la même chose, ou provoque une erreur « IndentationError ».

5. Fonctions

Fonction Une fonction est un bloc d'instructions auquel on donne un nom, pour le réutiliser. On la définit avec **def nom(paramètres)** : suivi d'un bloc indenté. Elle **renvoie** un résultat avec **return**. Rien ne s'exécute tant qu'on ne l'**appelle** pas.



```
def vitesse(distance, temps):
    v = distance / temps
    return v
```

```
print(vitesse(150, 2))
r = vitesse(90, 1.5)
```

```
print(r)
```

Affichage obtenu :

```
75.0  
60.0
```

La fonction *vitesse* a deux paramètres (*distance* et *temps*). L'appel *vitesse(150, 2)* remplace *distance* par 150 et *temps* par 2 : elle renvoie $150 \div 2 = 75.0$. La valeur renvoyée « prend la place » de l'appel, on peut donc la ranger dans une variable (*r*) ou la réutiliser dans un calcul.

return n'est pas print **print** affiche à l'écran mais ne donne rien à la suite du programme. **return** renvoie une valeur utilisable ensuite, et **arrête** immédiatement la fonction. Les variables créées dans une fonction n'existent pas en dehors d'elle.

Une fonction qui renvoie une expression

```
def milieu(a, b):  
    return (a + b) / 2  
  
m = milieu(3, 10)  
print(m)
```

L'appel *milieu(3, 10)* renvoie $(3 + 10) \div 2 = 6.5$. On peut écrire directement le calcul après *return*. Une fonction peut aussi contenir un *if* : dans ce cas on place un *return* dans chaque cas.

Mathbot « Une fonction Python, c'est comme une fonction mathématique : $f(x) = 3x - 7$ devient `def f(x): return 3*x - 7`. Tu peux même compléter un tableau de valeurs avec une boucle ! »

6. La boucle bornée for

Boucle for On l'emploie quand on **connaît à l'avance le nombre de répétitions**.

L'instruction **for i in range(a, b)** : fait prendre à *i* toutes les valeurs entières de **a** à **b - 1** (la valeur *b* n'est jamais atteinte). **range(n)** équivaut à `range(0, n)` : *i* va de 0 à *n - 1*, soit *n* tours. Avec un troisième nombre, **range(a, b, p)** avance par pas de *p*.

```
for i in range(1, 6):
```

```
print(i, "au carre :", i * i)
```

Affichage obtenu :

```
1 au carre : 1
2 au carre : 4
3 au carre : 9
4 au carre : 16
5 au carre : 25
```

Ici `range(1, 6)` donne $i = 1, 2, 3, 4, 5$: cinq tours.

Méthode : calculer une somme avec une boucle

1. Initialise un accumulateur : $s = 0$.
2. Dans la boucle, ajoute : $s = s + \dots$
3. Affiche s **après** la boucle (ligne non indentée).

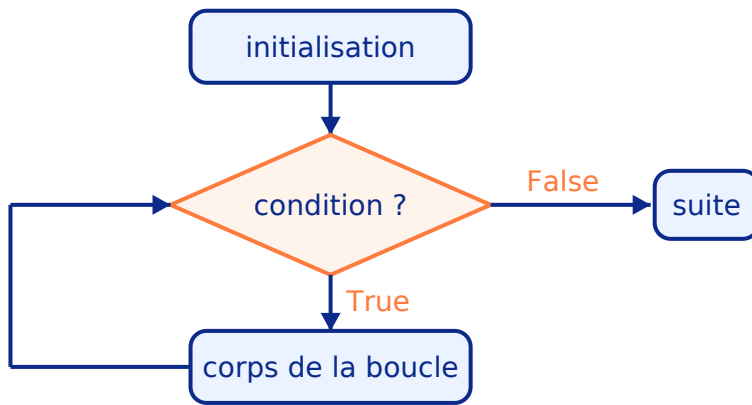
```
s = 0
for k in range(1, 5):
    s = s + k
print(s)
```

On obtient $1 + 2 + 3 + 4 = 10$.

Le piège du dernier nombre Pour parcourir les entiers de 1 à 10 **inclus**, il faut écrire `range(1, 11)`. Pour compter à rebours de 10 à 0, on écrit `range(10, -1, -1)`.

7. La boucle conditionnelle while

Boucle while On l'emploie quand on **ne sait pas à l'avance** combien de tours seront nécessaires. **while condition** : répète le bloc indenté **tant que** la condition est vraie. La condition est testée **avant chaque tour** ; si elle est fausse dès le départ, le bloc n'est jamais exécuté.



Un réservoir qui se vide Un réservoir contient 100 litres. Chaque jour, il perd 20 % de son contenu (il en reste 80 %). On veut savoir au bout de combien de jours il y aura au plus 20 litres.

```

eau = 100
jours = 0
while eau > 20:
    eau = eau * 0.8
    jours = jours + 1
print(jours, eau)
  
```

Le programme affiche « 8 16.7772160000000006 » : au bout de 8 jours, il reste environ 16,78 litres. Voici le suivi :

Tour	eau avant	condition eau > 20	eau après	jours
départ	100			0
1	100	True	80	1
2	80	True	64	2
3	64	True	51,2	3
4	51,2	True	40,96	4
5	40,96	True	32,768	5
6	32,768	True	26,2144	6
7	26,2144	True	20,97152	7
8	20,97152	True	16,777216	8
fin	16,777216	False		8

Méthode : écrire une boucle while

1. **Initialise** les variables avant la boucle.
2. Choisis la condition de **continuation** (le contraire de ce que l'on cherche à

atteindre).

3. Dans le corps, **fais évoluer** une variable de la condition, sinon la boucle ne s'arrête jamais.
4. Après la boucle, affiche le résultat.

Boucle infinie Si la condition reste toujours vraie (variable jamais modifiée, mauvais sens de comparaison), le programme ne s'arrête plus. On l'interrompt avec le bouton « stop » de l'éditeur.

8. Choisir et mettre au point un programme

- **for** si le nombre de tours est connu (parcourir 1 à 20, répéter 5 fois) ; **while** si on répète jusqu'à ce qu'un événement arrive (un seuil franchi).
- Un **if** peut se trouver dans une boucle, une boucle dans une fonction, une fonction dans une autre : on construit les programmes par blocs.
- Les trois erreurs les plus fréquentes : oublier « : », se tromper d'indentation, confondre = et ==.
- Avant d'exécuter, **prévois la sortie à la main** sur un petit exemple ; après, compare. Teste sur des cas simples et sur les cas limites.

Lire une erreur Python indique la ligne fautive. « `SyntaxError` » : problème d'écriture (deux-points, parenthèse oubliés). « `NameError` » : variable jamais définie. « `TypeError` » : mélange de types (texte + nombre).

À retenir

- Quatre types : **int**, **float**, **str**, **bool** ; `type(x)` donne le type ; « / » renvoie un float.
- **x = expression** est une affectation : on calcule à droite, on range à gauche.
- **input** renvoie un str : convertir avec **int** ou **float**.
- Comparaisons : `==`, `!=`, `<`, `>`, `<=`, `>=` ; connecteurs : **and**, **or**, **not**.
- **if / elif / else** : un seul bloc est exécuté ; l'indentation délimite les blocs.
- **def nom(paramètres)** : et **return** ; `return` renvoie, `print` affiche.
- **for i in range(a, b)** : i varie de a à b – 1 ; nombre de tours connu.
- **while condition** : on répète tant que la condition est vraie ; modifier la variable pour éviter la boucle infinie.
- Pour comprendre un programme : tableau de suivi des variables.