

Fiche de cours

Algorithmique et programmation en Python

L'essentiel du chapitre en deux pages

définitions · propriétés · méthodes · à savoir par cœur

1 VARIABLES, AFFECTATION, TYPES

Définition. Une **variable** est une case mémoire désignée par un nom. L'**affectation** $a = 5$ range la valeur 5 dans a . On lit toujours *de droite à gauche* : le membre de droite est d'abord calculé, puis rangé dans la variable de gauche. Ainsi $a = a + 1$ augmente a de 1, et $=$ n'est pas le signe « égal » des mathématiques.

Méthode — échanger deux variables. Passer par une variable auxiliaire : $t = u$ puis $u = v$ puis $v = t$. Écrire $u = v$ puis $v = u$ perd la valeur de u .

Remarque. `type(x)` donne le type de x . Les conversions s'écrivent `int(...)`, `float(...)`, `str(...)` : `int("12")` vaut 12, `int(3.9)` vaut 3 (troncature, pas d'arrondi).

Type	Nom	Exemples
entier	<code>int</code>	-4, 0, 25
flottant	<code>float</code>	2.5, -0.1
booléen	<code>bool</code>	True, False
chaîne	<code>str</code>	"bonjour"

Tableau 1 — Les quatre types de base. Le séparateur décimal est le *point*.

2 OPÉRATEURS

Opérateur	Signification
<code>+</code> <code>-</code> <code>*</code>	somme, différence, produit
<code>/</code>	quotient décimal (toujours <code>float</code>)
<code>//</code> <code>%</code>	quotient et reste de la division euclidienne
<code>**</code>	puissance : $2**5$ vaut 32

Propriété. Les **comparaisons** `==`, `!=`, `<`, `<=`, `>`, `>=` produisent un booléen. On les combine avec `and` (et), `or` (ou), `not` (non). $17 \% 5$ vaut 2 : un entier n est pair si $n \% 2 == 0$, et a est un multiple de b si $a \% b == 0$.

3 INSTRUCTION CONDITIONNELLE

Définition. `if condition:` exécute le bloc **indenté** qui suit lorsque la condition est vraie. `elif` (« sinon si ») teste une nouvelle condition, `else` traite tous les autres cas. Les conditions sont examinées *dans l'ordre* : dès que l'une est vraie, les suivantes ne sont pas testées.

Remarque. L'**indentation** (4 espaces) délimite les blocs : elle fait partie de la syntaxe. Ne pas oublier les deux-points en fin de ligne.

```
if x < 0:
    signe = -1
elif x == 0:
    signe = 0
else:
    signe = 1
```

4 BOUCLES

Définition. La **boucle bornée** `for k in range(a, b, p):` fait prendre à k les valeurs $a, a+p, a+2p, \dots$ en s'arrêtant **avant** b . Avec `range(n)`, k parcourt $0, 1, \dots, n-1$: il y a n tours. On l'emploie quand le nombre de répétitions est *connu à l'avance*.

Définition. La **boucle non bornée** `while condition:` répète le bloc *tant que* la condition est vraie. On l'emploie pour atteindre un **seuil**. Il faut que la condition finisse par devenir fausse, sinon le programme ne s'arrête jamais.

Méthode — compter, accumuler. Initialiser un compteur ou un accumulateur *avant* la boucle ($s = 0, n = 0$), le mettre à jour *dans* la boucle, l'afficher *après*.

```
s = 0
for k in range(1, 11):
    s = s + k
print(s)          # 55
```

```
u = 100
n = 0
while u > 10:
    u = u*0.8
    n = n + 1
print(n)          # 11
```

5 FONCTIONS

Définition. `def f(x)` : définit une **fonction** de paramètre x ; `return` **renvoie** un résultat réutilisable dans un calcul, alors que `print` se contente d'*afficher*. Une fonction peut avoir plusieurs paramètres, séparés par des virgules, et l'appel `f(3)` remplace x par 3.

Remarque. `return` met fin à l'exécution de la fonction. Une fonction sans `return` renvoie la valeur spéciale `None` (« rien ») : `y = f(3)` donne alors `None`.

Notation. Nombres aléatoires : `from random import randint, random` ; `randint(1, 6)` renvoie un entier de 1 à 6 *inclus*, `random()` un flottant de $[0; 1[$. Deux exécutions donnent en général des résultats différents.

```
def moyenne(a, b):
    return (a + b)/2

m = moyenne(12, 15)
print(m)           # 13.5
```

```
from random import randint

de = randint(1, 6)
```

6 LISTES (HORS PROGRAMME — POUR ALLER PLUS LOIN)

Définition. Une **liste** range plusieurs valeurs dans une seule variable. Les cases sont numérotées **à partir de 0** : `notes[0]` est la première. `len(notes)` donne le nombre de cases, `notes.append(v)` ajoute v à la fin, et `[0]*13` crée une liste de 13 zéros. La boucle `for v in notes` parcourt directement les *valeurs*.

Remarque. La dernière case est `notes[len(notes)-1]` : écrire `notes[len(notes)]` provoque une erreur.

```
notes = [12, 8, 15, 9]
s = 0
for v in notes:
    s = s + v
print(s/len(notes)) # 11.0
```

7 SAVOIR-FAIRE DU CHAPITRE

► SAVOIR-FAIRE

- Suivre des affectations successives dans un tableau d'exécution ; échanger deux variables ; reconnaître un type et convertir.
- Utiliser `//`, `%`, `**` et écrire une condition avec `and`, `or`, `not`.
- Écrire et exécuter à la main un `if` / `elif` / `else`, notamment pour un tarif par paliers.
- Choisir entre `for` et `while` ; compter, accumuler, chercher un maximum, atteindre un seuil.
- Écrire une fonction avec `def` et `return`, l'appeler, et simuler une expérience aléatoire pour estimer une fréquence.

8 EXEMPLES TYPES

★ EXEMPLE — exécution à la main

Pour $a = 7$, $b = 3$, $a = a + b$, $b = a - b$, $a = a - b$: on obtient successivement $a = 10$, puis $b = 7$, puis $a = 3$. Ce programme *échange* les deux valeurs.

★ EXEMPLE — seuil avec while

Un capital de 2 000 € augmente de 3 % par an. Avec $c = 2000$, $n = 0$ puis `while c < 2500` : $c = c \cdot 1.03$; $n = n + 1$, on obtient $n = 8$: il faut 8 ans pour dépasser 2 500 €.

★ EXEMPLE — simulation

`from random import randint` puis, sur `for k in range(1000)` : compter les tirages tels que `randint(1, 6) == 6` et diviser par 1000 : on obtient une fréquence proche de $\frac{1}{6} \approx 0,167$.

✗ PIÈGES

- Confondre = (affectation) et `==` (test d'égalité) : `if x = 3` : est une erreur de syntaxe.
- Oublier les deux-points (erreur de syntaxe) ou l'indentation (erreur, ou instruction qui n'est plus rattachée au `if` ou à la boucle).
- Oublier que `range(a, b)` s'arrête *avant* b : `range(1, 11)` donne les entiers de 1 à 10.
- Écrire `print` au lieu de `return` dans une fonction : la valeur ne peut plus être réutilisée dans un calcul.
- Initialiser un compteur *dans* la boucle : il repart à zéro à chaque tour.