MON APPRENTISSAGE — SERIES PYTHON

Python

Du zéro à l'expert

Chapitre 3 — Les Opérateurs

Les outils qui donnent vie aux calculs et à la logique

Auteur :
KETOTSA AMÉVI CLAUDE

Date de publication :
1er mars 2026

Publié dans le cadre de mon parcours d'apprentissage Python

Table des matières

Chapitre 3 — Les Opérateurs

À retenir

Les opérateurs sont les **outils fondamentaux** de Python. Ils permettent d'effectuer des calculs, de comparer des valeurs, de combiner des conditions et de modifier des variables. Sans eux, un programme ne peut rien décider, rien calculer, rien produire.

Les opérateurs arithmétiques

Les opérateurs arithmétiques permettent d'effectuer des **calculs mathématiques** de base.

+	-	*	/
Addition	Soustraction	Multiplication	Division
5 + 3 = 8	5 - 3 = 2	5 * 3 = 15	5 / 2 = 2.5

//	%	**
Division entière	Modulo (reste)	Puissance
5 // 2 = 2	5 % 2 = 1	2 ** 8 = 256

Opérateurs arithmétiques en action

```python
a = 17
b = 5

print(a + b)     # Addition              22
print(a - b)     # Soustraction          12
print(a * b)     # Multiplication        85
print(a / b)     # Division reelle       3.4
print(a // b)    # Division entiere      3
print(a % b)     # Modulo (reste)        2
print(a ** b)    # Puissance             1419857

# Exemple concret : calculer le prix TTC
prix_ht = 200
tva = 0.20
prix_ttc = prix_ht * (1 + tva)
print(f"Prix TTC : {prix_ttc}     ")    # Prix TTC : 240.0
```

À retenir

Le **modulo** `%` est très utile en programmation : il permet de savoir si un nombre est pair ou impair, de créer des cycles, ou de limiter des valeurs dans un intervalle. `nombre % 2 == 0` → le nombre est pair.

Les opérateurs de comparaison

Les opérateurs de comparaison **comparent deux valeurs** et retournent toujours un booléen : `True` ou `False`.

Opérateurs de comparaison

```
a = 10
b = 20

print(a == b)    # Egal a                  False
print(a != b)    # Different de            True
print(a < b)     # Inferieur a             True
print(a > b)     # Superieur a             False
print(a <= b)    # Inferieur ou egal        True
print(a >= b)    # Superieur ou egal        False

# Comparer des chaines
prenom1 = "Alice"
prenom2 = "Bob"
print(prenom1 == prenom2)    # False
print(prenom1 != prenom2)    # True

# Exemple concret : verifier l'age
age = 17
est_majeur = age >= 18
print(est_majeur)    # False
```

Erreur fréquente

Ne confondez jamais `=` et `==` ! `=` est une **assignation** (on donne une valeur à une variable). `==` est une **comparaison** (on vérifie si deux valeurs sont égales). C'est l'une des erreurs les plus fréquentes chez les débutants.

Les opérateurs logiques

Les opérateurs logiques permettent de **combiner plusieurs conditions** ensemble.

and

Les DEUX conditions doivent être **True**

True and True → True

True and False → False

False and False → False

or

AU MOINS UNE condition doit être **True**

True or False → True

False or True → True

False or False → False

not

INVERSE la valeur du booléen

not True → False

not False → True

Opérateurs logiques en action

```python
age = 20
a_un_billet = True
est_vip = False

# and      les deux conditions doivent etre vraies
peut_entrer = age >= 18 and a_un_billet
print(peut_entrer)    # True

# or      au moins une condition doit etre vraie
acces_special = est_vip or age >= 21
print(acces_special)    # False

# not      inverse la condition
porte_fermee = False
print(not porte_fermee)    # True (la porte est ouverte)

# Combinaison de plusieurs operateurs
a = 15
resultat = (a > 10) and (a < 20) and (a % 2 != 0)
print(resultat)    # True (15 est entre 10 et 20, et impair)
```

Les opérateurs d'affectation

Les opérateurs d'affectation permettent de **modifier une variable** de façon concise.

Opérateurs d'affectation

```python
score = 100

score += 10     # score = score + 10        110
print(score)

score -= 20     # score = score - 20        90
print(score)

score *= 2      # score = score * 2         180
print(score)
```

```
score //= 3     # score = score // 3       60
print(score)

score **= 2     # score = score ** 2       3600
print(score)

score %= 100    # score = score % 100      0
print(score)

# Exemple concret : compter des points dans un jeu
points = 0
points += 50     # Ennemi vaincu
points += 100    # Niveau termine
points -= 10     # Vie perdue
print(f"Score final : {points}")    # Score final : 140
```

Les opérateurs d'identité et d'appartenance

Python possède deux opérateurs spéciaux très utiles que l'on ne trouve pas dans tous les langages.

</> is, is not, in, not in

```
# is / is not     verifie si deux variables pointent
# vers le MEME objet en memoire
a = None
print(a is None)        # True
print(a is not None)    # False

# in / not in     verifie si une valeur est dans
# une sequence (liste, chaine, etc.)
fruits = ["pomme", "banane", "cerise"]
print("banane" in fruits)       # True
print("mangue" not in fruits)   # True

# Avec les chaines
message = "Bonjour tout le monde"
print("monde" in message)     # True
print("Python" in message)    # False

# Exemple concret : v rifier un acces
admins = ["Alice", "Bob", "Charlie"]
utilisateur = "Alice"
if utilisateur in admins:
    print(f"{utilisateur} a les droits admin.")
```

À retenir

`in` est l'un des opérateurs les plus élégants de Python. Il permet de vérifier l'appartenance à une collection en une seule expression, là où d'autres langages nécessiteraient une boucle entière.

La priorité des opérateurs

Comme en mathématiques, Python applique une **priorité** entre les opérateurs. De la plus haute à la plus basse :

1 — Priorité max	`**`	Puissance
2	`+x, -x, ~x`	Unaires
3	`*, /, //, %`	Multiplication, Division
4	`+, -`	Addition, Soustraction
5	`<, >, <=, >=, ==, !=`	Comparaisons
6	`not`	NON logique
7 — Priorité min	`and, or`	ET / OU logique

Priorité des opérateurs — exemples

```python
# Sans parentheses      Python suit les priorites
resultat = 2 + 3 * 4
print(resultat)    # 14   (pas 20      * avant +)

# Avec parentheses      vous controlez l'ordre
resultat = (2 + 3) * 4
print(resultat)    # 20

# Exemple complexe
x = 10
y = 3
z = 2
resultat = x + y ** z * 2 - 1
# Ordre : y**z=9, 9*2=18, 10+18=28, 28-1=27
print(resultat)    # 27

# Conseil : toujours utiliser des parentheses
# pour rendre le code lisible et eviter les erreurs
resultat_clair = x + ((y ** z) * 2) - 1
print(resultat_clair)    # 27
```

Note importante

En cas de doute sur la priorité, utilisez toujours des **parenthèses**. Elles rendent le code plus lisible et évitent les erreurs de calcul inattendues.

Résumé du Chapitre 3

Ce que j'ai appris dans ce chapitre :

- Les **opérateurs arithmétiques** : `+`, `-`, `*`, `/`, `//`, `%`, `**`
- Les **opérateurs de comparaison** : `==`, `!=`, `<`, `>`, `<=`, `>=`
- Les **opérateurs logiques** : `and`, `or`, `not`
- Les **opérateurs d'affectation** : `+=`, `-=`, `*=`, `/=` ...
- Les **opérateurs spéciaux** : `is`, `is not`, `in`, `not in`
- La **priorité des opérateurs** — et pourquoi les parenthèses sont vos meilleures alliées.
- Ne jamais confondre `=` (assignation) et `==` (comparaison).

"La logique vous mènera d'un point A à un point B. L'imagination vous mènera partout." — Albert Einstein