Opérateurs en Python

Arithmétiques, comparaison, logiques et affectation

Résumé

Un **opérateur** est un symbole qui effectue une opération sur une ou plusieurs valeurs. Ce cours présente les différentes catégories d'opérateurs en Python : arithmétiques, de comparaison, logiques, d'affectation augmentée et d'appartenance, avec pour fil conducteur un programme qui reconnaît les années bissextiles.

1 Fil conducteur : l'année bissextile

Au fil de ce cours, nous allons découvrir les différentes familles d'opérateurs en Python. Pour donner du sens à chaque notion, nous construirons progressivement un programme qui vérifie si une année est **bissextile**. Rappelons la règle : une année est bissextile si elle est divisible par 4, **sauf** si elle est divisible par 100, **sauf** si elle est divisible par 400. Par exemple, 2024 est bissextile, 1900 ne l'est pas, et 2000 l'est.

Pour résoudre ce problème, nous aurons besoin du modulo (divisibilité), des comparaisons (tester si le reste vaut zéro), des opérateurs logiques (combiner les conditions) et de la priorité des opérations (écrire l'expression correctement). Allons-y.

2 Opérateurs arithmétiques

Python dispose de sept opérateurs pour effectuer des calculs sur les nombres. Les quatre premiers sont ceux de la calculatrice ; les trois suivants sont propres à la programmation.

Opérateur	Nom	Exemple	Résultat
`+`	Addition	`7 + 3`	`10`
`-`	Soustraction	`7 - 3`	`4`
`*`	Multiplication	`7 * 3`	`21`
`/`	Division	`7 / 3`	`2.333...`
`//`	Division entière	`7 // 3`	`2`
`%`	Modulo (reste)	`7 % 3`	`1`
`**`	Puissance	`2 ** 10`	`1024`

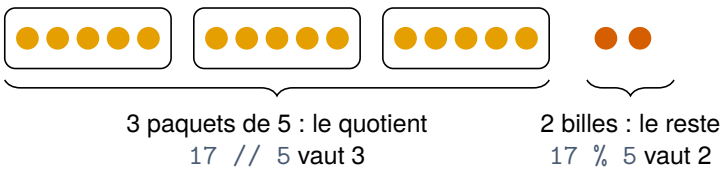
Division décimale et division entière

La **division décimale** `/` renvoie toujours un flottant, même si le résultat est un nombre entier :

```
print(7 / 2)     # 3.5
print(10 / 2)    # 5.0  (flottant, pas entier !)
print(type(10 / 2))  # <class 'float'>
```

Quotient et reste

Prenons 17 billes, à regrouper en paquets de 5. On forme 3 paquets complets, et il reste 2 billes : $17 = 5 \times 3 + 2$. Le nombre 3 est le **quotient**, et 2 est le **reste** de la division euclidienne de 17 par 5.



En Python, la **division entière** `//` donne le quotient, et le **modulo**, écrit avec le symbole `%`, donne le reste.

```
print(17 // 5)
print(17 % 5)
```

Question.
Que répond Python ?

Réponse.

```
3
2
```

La division entière donne le quotient, `3` : le nombre de paquets complets. Le modulo donne le reste, `2` : le nombre de billes qui restent.

On retrouve bien la relation fondamentale : `17 = 5 × 3 + 2`, c'est-à-dire `dividende = diviseur × quotient + reste`.

```
dividende = 17
diviseur = 5
quotient = dividende // diviseur
reste = dividende % diviseur
print(f"{dividende} = {diviseur} × {quotient} + {reste}")
```

```
17 = 5 × 3 + 2
```

Attention avec les flottants : si l'un des opérandes est un flottant, `//` et `%` renvoient un flottant :

```
print(17.0 // 5)   # 3.0  (flottant)
print(17.0 % 5)    # 2.0  (flottant)
```

Applications du modulo

Le modulo est très utile en programmation. Voici quelques exemples courants :

```python
# Tester si un nombre est pair
n = 42
print(n % 2 == 0)  # True (42 est pair)

# Extraire le dernier chiffre d'un nombre
n = 2025
print(n % 10)  # 5

# Convertir des secondes en minutes et secondes
duree = 195  # secondes
minutes = duree // 60
secondes = duree % 60
print(f"{duree} s = {minutes} min {secondes} s")
```

```
True
5
195 s = 3 min 15 s
```

Question.
Que vaut `3 % 5` ?

Réponse.
3. Avec 3 billes et des paquets de 5, on ne forme aucun paquet complet : le quotient `3 // 5` vaut `0`, et il reste les 3 billes. Quand le dividende est plus petit que le diviseur, le reste est le dividende lui-même.

Retour au fil conducteur. Nous savons maintenant tester la divisibilité : `annee % 4 == 0` sera `True` si `annee` est divisible par 4, puisque le reste de la division par 4 vaut alors zéro. Mais nous avons encore besoin des opérateurs de comparaison et logiques pour finaliser notre programme.

3 Priorité des opérations

Python respecte les règles de priorité habituelles des mathématiques :

1. Les parenthèses `()` ;
2. la puissance `**` ;
3. le signe unaire `-` (opposé) ;
4. la multiplication `*`, la division `/`, `//` et `%` ;
5. l'addition `+` et la soustraction `-`.

```python
print(2 + 3 * 4)
```

Question.
Combien vaut ce calcul ?

Réponse.

```
14
```

La multiplication passe avant l'addition : 3 × 4 font 12, puis 2 + 12 font 14. Le résultat n'est donc pas 20.

Pour changer l'ordre, on utilise des parenthèses. Attention au cas particulier de la puissance : plusieurs puissances se calculent **de droite à gauche**.

```
print((2 + 3) * 4)      # 20
print(2 ** 3 ** 2)      # 512  (= 2 ** 9, car 3 ** 2 est calculé d'abord)
```

En cas de doute, ajoutez des parenthèses pour rendre l'expression plus lisible.

Exercice 1 *Division euclidienne et modulo* **(4 points)**

1. Sans exécuter, prédisez le résultat de chaque expression :

   a) `23 // 7`

   b) `23 % 7`

   c) `100 // 3`

   d) `100 % 3`

   Vérifiez à chaque fois la relation : dividende = diviseur × quotient + reste.

2. Écrire un programme qui demande un nombre de minutes et affiche le résultat en heures et minutes.

   Par exemple, pour 200 minutes, le programme affiche : `200 min = 3 h 20 min`.

3. Écrire un programme qui demande un nombre entier à trois chiffres et affiche séparément le chiffre des centaines, des dizaines et des unités. Par exemple, pour 472 : centaines = 4, dizaines = 7, unités = 2.

④ Opérateurs de comparaison

Les opérateurs de comparaison répondent par vrai ou faux, c'est-à-dire par un **booléen** (`True` ou `False`) :

Opérateur	Signification	Exemple	Résultat
`==`	Égal à	`5 == 5`	`True`
`!=`	Différent de	`5 != 3`	`True`
`<`	Strictement inférieur	`3 < 5`	`True`
`>`	Strictement supérieur	`3 > 5`	`False`
`<=`	Inférieur ou égal	`5 <= 5`	`True`
`>=`	Supérieur ou égal	`3 >= 5`	`False`

Important. ne pas confondre `=` (affectation) et `==` (comparaison). Un seul signe égal stocke une valeur dans une variable ; deux signes égal posent une question : les deux valeurs sont-elles égales ?

```
note = 15       # affectation : on donne la valeur 15 à note
note == 15      # comparaison : on teste si note vaut 15 → True
```

```
x = 3
print(x == 4)
print(x)
```

Question.
Qu'affichent ces trois lignes ?

Réponse.

```
False
3
```

`x == 4` pose une question : `x` vaut-il 4 ? La réponse est `False`. Une comparaison ne modifie jamais la variable : `x` vaut toujours `3`.

Les opérateurs de comparaison s'appliquent aussi aux chaînes de caractères. Deux chaînes se comparent caractère par caractère, dans l'ordre alphabétique (on parle d'**ordre lexicographique**) : `"abc" < "abd"` vaut `True`, car les deux chaînes commencent par `"ab"` et `"c"` est avant `"d"`.

Chaînage des comparaisons

Python permet de chaîner les comparaisons, comme en mathématiques :

```
x = 5
print(2 < x <= 9)    # True (équivalent à 2 < x and x <= 9)
print(0 <= x < 3)    # False
```

Exercice 2 *Comparaisons* **(3 points)**

1. Sans exécuter, prédisez le résultat de chaque expression :

   a) `7 > 3`

   b) `7 == 7.0`

   c) `"abc" < "abd"`

   d) `3 < 5 < 2`

2. Que signifie mathématiquement l'expression `0 <= x <= 1` ? Écrire un programme qui demande un nombre `x` et affiche `True` si $x \in [0\ ;\ 1]$, `False` sinon.

5 Opérateurs logiques

Les opérateurs `and`, `or` et `not` permettent de combiner des conditions :

Opérateur	Signification	Exemple	Résultat
`and`	ET logique	`True and False`	`False`
`or`	OU logique	`True or False`	`True`
`not`	NON logique	`not True`	`False`

L'opérateur `and` renvoie `True` uniquement si les **deux** conditions sont vraies. L'opérateur `or` renvoie `True` si **au moins une** des conditions est vraie. L'opérateur `not` inverse une condition. On résume ces règles dans une **table de vérité**, qui donne le résultat pour toutes les valeurs possibles de deux conditions `a` et `b` :

a	b	a and b	a or b
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

```python
age = 17
a_permis = True

print(age >= 18 and a_permis)   # False (17 < 18)
```

```
print(age >= 18 or a_permis)    # True  (a_permis est True)
print(not age >= 18)             # True  (17 n'est pas  18)
```

Un exemple plus proche des conditions qu'on écrit souvent : un élève a-t-il au moins 15 ans et moins de 18 ans ?

```
age = 16
print(age >= 15 and age < 18)
age = 18
print(age >= 15 and age < 18)
```

Question.
Qu'affiche ce programme ?

Réponse.

```
True
False
```

Avec 16 ans, les deux conditions sont vraies, donc le `and` est vrai. Avec 18 ans, la seconde condition, `age < 18`, est fausse : il suffit d'une condition fausse pour que le `and` soit faux.

Évaluation en court-circuit

Python évalue les opérateurs logiques de gauche à droite et **s'arrête dès que le résultat est connu** :

— avec `and` : si la première condition est `False`, Python ne regarde pas la deuxième (le résultat sera forcément `False`) ;

— avec `or` : si la première condition est `True`, Python ne regarde pas la deuxième (le résultat sera forcément `True`).

```
# Ceci ne provoque pas d'erreur, car Python n'évalue pas 1/0
x = 0
print(x == 0 or 1/x > 2)   # True (court-circuit après x == 0)
```

Retour au fil conducteur : le programme complet

Nous avons maintenant tous les outils pour écrire notre programme. Une année est bissextile si elle est divisible par 4, sauf si elle est divisible par 100, sauf si elle est divisible par 400. Divisible par 4 signifie que le reste de la division par 4 vaut zéro : on traduit donc la règle avec le modulo, et les mots-clés `and` et `or`.

```
annee = 2024
bissextile = (annee % 4 == 0) and (annee % 100 != 0 or annee % 400 == 0)
print(f"{annee} est bissextile : {bissextile}")
```

```
2024 est bissextile : True
```

Vérifions avec les cas particuliers : on remplace la valeur de `annee` dans le programme ci-dessus, puis on le relance.

annee	divisible par 4	divisible par 100	divisible par 400	affichage
2024	oui	non	non	`True`
1900	oui	oui	non	`False`
2000	oui	oui	oui	`True`
2025	non	non	non	`False`

Ce programme combine le modulo (`%`), les comparaisons (`==`, `!=`), les opérateurs logiques (`and`, `or`) et les parenthèses pour la priorité. C'est un bon exemple de la puissance des opérateurs combinés.

Exercice 3 *Opérateurs logiques* (4 points)

1. Sans exécuter, prédisez le résultat :

   a) `True and True`

   b) `True and False`

   c) `False or True`

   d) `not False`

   e) `not (True and False)`

2. Avec `age = 15` et `note = 12`, prédisez le résultat :

   a) `age >= 18 and note >= 10`

   b) `age >= 18 or note >= 10`

   c) `not age >= 18`

3. Expliquez pourquoi l'expression suivante ne provoque pas d'erreur de division par zéro :

```
x = 0
resultat = x != 0 and 10 / x > 2
```

⑥ Opérateurs d'affectation augmentée

Il arrive très souvent d'ajouter une valeur à une variable, par exemple pour augmenter un score. Pour modifier la valeur d'une variable en la combinant avec une opération, on utilise les **opérateurs d'affectation augmentée** :

```
score = 0
score += 10    # équivalent à score = score + 10
print(score)   # 10

score *= 2     # équivalent à score = score * 2
print(score)   # 20

score -= 5     # équivalent à score = score - 5
print(score)   # 15
```

On suit la valeur de `score` ligne après ligne :

Instruction	Écriture longue	Valeur de score
`score = 0`		0
`score += 10`	`score = score + 10`	10
`score *= 2`	`score = score * 2`	20
`score -= 5`	`score = score - 5`	15

Les opérateurs disponibles sont `+=`, `-=`, `*=`, `/=`, `//=`, `%=` et `**=`.

⑦ Opérateur d'appartenance `in`

L'opérateur `in` teste si un élément est présent dans une séquence. Pour l'instant, la seule séquence connue est la chaîne de caractères : `in` teste si un caractère, ou une suite de caractères, y figure.

```
voyelles = "aeiouy"
print('a' in voyelles)     # True
print('b' in voyelles)     # False
print('b' not in voyelles) # True

mot = "Python"
print('y' in mot)          # True
print('th' in mot)         # True
```

Exercice 4 *Affectation augmentée et appartenance* **(3 points)**

1. Sans exécuter, prédisez la valeur de `x` après chaque ligne :

```python
x = 10
x += 5
x *= 2
x -= 7
x //= 3
```

2. Écrire un programme qui demande une lettre minuscule et affiche `True` si c'est une voyelle, `False` sinon.

8 Fonctions mathématiques et hasard : les modules `math` et `random`

Python regroupe des fonctions supplémentaires dans des **modules**. On charge un module avec `import`, puis on écrit le nom du module, un point et le nom de la fonction.

Le module `math` fournit les fonctions mathématiques usuelles :

```python
import math

print(math.sqrt(16))    # 4.0 (racine carrée)
print(math.pi)          # 3.141592653589793
print(math.floor(3.7))  # 3 (partie entière)
```

Le module `random` tire des nombres au hasard :

```python
import random

lancer = random.randint(1, 6)   # entier au hasard entre 1 et 6, bornes comprises
x = random.random()             # flottant au hasard entre 0 et 1, 1 exclu
print(f"{lancer} {x}")
```

On peut aussi importer une seule fonction d'un module. On l'utilise alors sans le nom du module devant :

```python
from math import sqrt

print(sqrt(25))   # 5.0
```

Pour découvrir toutes les fonctions d'un module, on l'importe puis on appelle `help()` : `help(math)` ou `help(random)`.

9 À retenir

1. La division entière `//` donne le quotient et le modulo `%` le reste de la division euclidienne : `17 // 5` vaut `3` et `17 % 5` vaut `2`.
2. La division `/` renvoie toujours un flottant.
3. Le modulo sert souvent : `n % 2 == 0` teste si `n` est pair, et `n % 10` donne son dernier chiffre.
4. Python respecte les priorités des mathématiques ; en cas de doute, on ajoute des parenthèses.
5. Les comparaisons renvoient un booléen. Un seul `=` stocke une valeur, deux `==` testent une égalité.
6. `and` est vrai si les deux conditions le sont, `or` si au moins une l'est, et `not` inverse une condition.
7. `x += 1` est l'écriture courte de `x = x + 1`.
8. `in` teste si un caractère, ou une suite de caractères, figure dans une chaîne.

10 Résumé

Catégorie	**Opérateurs**
Arithmétiques	`+`, `-`, `*`, `/`, `//`, `%`, `**`
Comparaison	`==`, `!=`, `<`, `>`, `<=`, `>=`
Logiques	`and`, `or`, `not`
Affectation augmentée	`+=`, `-=`, `*=`, `/=`, `//=`, `%=`, `**=`
Appartenance	`in`, `not in`
Modules	`import math`, `math.sqrt(x)`, `import random`, `random.randint(a, b)`