

Opérations, comparaisons, expression

Opérateurs arithmétiques sur les nombres

Les opérations arithmétiques usuelles sont effectuées sur des nombres de types `int` ou `float` :

opérateur	notation
addition	<code>a + b</code>
soustraction	<code>a - b</code>
multiplication	<code>a * b</code>
puissance	<code>a**b</code>
divisions décimale	<code>a / b</code>

```
>>> a = 5
>>> b = 2
```

PEP 8

Entourer les opérateurs mathématiques (+, -, /, *) d'un espace avant et d'un espace après.

```
>>> a + b
7
>>> a / b
2.5
>>> a**b
25
```

À noter :

Si `a` et `b` sont deux variables toutes les deux de type `int` alors le résultat d'une opération entre les deux est de type `int`, sauf pour la division qui est toujours de type `float` même si le résultat est un entier :

```
>>> 10 / 5
2.0
```

et si l'un de `a` ou de `b` est de type `float` alors le résultat est toujours de type `float`.

La racine carrée d'un nombre peut s'obtenir avec : `a**0.5` ¹⁰.

L'ordre des priorités mathématiques est respecté.

Il est possible d'affecter une valeur à une variable qui dépend de son ancienne valeur, par exemple l'augmenter d'une quantité donnée (on dit **incrémenter**)¹.

```
>>> a = 3
>>> a = a + 1
>>> a
4
```

PEP 8

Dans ce cas particuliers, on peut omettre les espaces autour de la multiplication (`*`) pour montrer la priorité sur l'addition et améliorer la lisibilité de la formule.

```
>>> a = 2*a + 1
>>> a
9
```

¹

Des raccourcis d'écriture existent pour aller plus vite (mais attention aux erreurs en les utilisant !).

- `a += 1` signifie `a = a + 1` ;
- `a += b` signifie `a = a + b` ; et
- `a *= 2` signifie `a = a * 2` .

Division entière (ou division euclidienne)

L'opérateur de division entière `//` et l'opération modulo `%` utilisés avec des entiers (de type `int`) donnent respectivement le quotient et le reste d'une division euclidienne : si `a` et `b` sont des entiers tels que $a = b \times q + r$, alors `a // b` donne `q` et `a % b` donne `r`².

opérateur	notation
quotient	<code>a // b</code>
reste	<code>a % b</code>

Par exemple, le quotient et le reste de la division entière de 17 par 5 sont 3 et 2 respectivement (car $17 = 3 \times 5 + 2$) :

```
>>> a = 17
>>> b = 5
>>> a // b
3
>>> a % b
2
```

$$\begin{array}{r|l} 17 & 5 \\ -15 & 3 \\ \hline 2 & \end{array}$$

L'opérateur modulo, `%`, qui donne le reste d'une division entière, est très utile pour déterminer si un nombre est divisible par un autre nombre, dans ce cas le reste est égal à zéro :

```
>>> 10 % 5
0
>>> 10 % 3
1
```

10 est divisible par 5 mais pas par 3.

Opérateurs sur les chaînes de caractères

Les textes ou chaînes de caractères, de type `str` (abréviation de *string*), sont définis entre une paire de guillemets (`"`) ou d'apostrophes (`'`)³.

```
>>> chaine1 = 'Hello '
>>> chaine2 = "world"
```

Pour les chaînes de caractères, deux opérations sont possibles, l'addition et la multiplication⁴ :

- L'opérateur d'addition « `+` » **concatène** (assemble) deux chaînes de caractères.

```
>>> chaine1 + chaine2
'Hello world'
```

11

- L'opérateur de multiplication « `*` » entre un nombre entier et une chaîne de caractères **duplique** (répète) plusieurs fois une chaîne de caractères.

```
>>> chaine1 * 3
'Hello Hello Hello '
```

La fonction `len()` donne le nombre de caractère d'une chaîne (y compris les espaces et les signes de ponctuation).

```
>>> ch = 'Hello world'
>>> len(ch)
11
```

Chaque caractère d'une chaîne de caractères `ch` a une position qui va de `0` à `len(ch) - 1`.

- `ch[0]` permet d'accéder au premier caractère en position `0` de la chaîne `ch`,
- `ch[1]` au second caractère en position `1`,
- ...
- `ch[i]` au caractère en `i`^{ième} position,
- ...
- `ch[len(ch) - 1]` au dernier caractère.

⚠ Les positions sont comptées en commençant à la position `0`, le premier caractère est `ch[0]` et non pas `ch[1]` !

```
>>> ch[6]
'w'
```

- De même, en partant de la fin, `ch[-1]` permet d'accéder au dernier caractère, `ch[-2]` à l'avant dernier, etc.

```
>>> ch[-1]
'd'
```

PEP 8

Pas d'espace autour d'un deux-points (:).

- Enfin `ch[i:j]` permet d'obtenir la sous-chaîne de tous les caractères entre les positions `i` (**inclus**) et `j` (**exclus**), appelée une tranche.

```
>>> ch[2:5]
'llo'
```

Les mots-clés `in` et `not in` permettent de vérifier l'appartenance, ou pas, d'une sous-chaîne dans une chaîne :

```
>>> "py" in "python"
True
>>> "Py" not in "python"
True
```

Il existe de nombreuses méthodes⁵ pour traiter les chaînes de caractères, quelques exemples :

fonction	description	exemple
<code>.index('c')</code>	trouve l'index du premier caractère "c" dans une chaîne.	<pre>>>> chaine = 'aaabbbccc' >>> chaine.index('b') 3</pre>
<code>.find('sc')</code>	cherche la position d'une sous-chaîne <code>sc</code> dans la chaîne.	<pre>>>> chaine.find('bc') 5</pre>
<code>.count('sc')</code>	compte le nombre de sous-chaînes <code>sc</code> dans la chaîne.	<pre>>>> chaine.count('bc') 1</pre>
<code>.lower('sc')</code>	onvertit une chaîne en minuscules.	<pre>>>> 'ABCdef'.lower() 'abcdef'</pre>
<code>.upper('sc')</code>	onvertit une chaîne en majuscules.	<pre>>>> 'ABCdef'.upper() 'ABCDEF'</pre>

fonction	description	exemple
<code>.replace('old', 'new')</code>	remplace tous les caractères <code>old</code> par <code>new</code> dans la chaîne.	<pre>>>> 'aaabbbccc'.replace('c', 'e') 'aaabbbeee'</pre>

Opérateurs de comparaison

Les opérations de comparaison usuelles permettent de comparer des valeurs de même type entre elles. Le résultat est toujours un booléen (de type `bool`) égal à `True` ou `False` ⁶.



PEP 8

Entourer les opérateurs de comparaison (`==`, `!=`, `>=`, etc.) d'un espace avant et d'un espace après.

opérateur	notation
<code>=</code>	<code>a == b</code>
<code>≠</code>	<code>a != b</code>
<code><</code>	<code>a < b</code>
<code>≤</code>	<code>a <= b</code>
<code>></code>	<code>a > b</code>
<code>≥</code>	<code>a >= b</code>

7

⚠ Une erreur courante consiste à confondre l'opérateur de comparaison `==` pour vérifier si deux valeurs sont égales avec l'affectation qui utilise le signe `=` !

```
>>> a, b, c = 5, 5, 6
>>> a == b
True
>>> a = c
False
```

Il est possible de combiner les comparaisons, par exemple pour vérifier si `a` est compris entre 2 et 6 :

```
>>> 2 <= a < 6
True
```

entre 7 et 8 :

```
>>> 7 < a < 8
False
```

mais ce n'est pas recommandé car c'est en fait une combinaison de plusieurs comparaisons, ce qui peut donner des hérésies mathématiques :

```
>>> 4 < a > 2
True
```

Les chaînes de caractères, quant à elles, sont comparées en ordre lexicographique, c'est-à-dire caractère par caractère comme l'ordre des mots dans un dictionnaire : on commence par comparer le premier caractère de chaque chaîne, puis en cas d'égalité le deuxième de chaque, et ainsi de suite jusqu'à trouver un caractère qui est différent de l'autre⁸.

```
>>> 'aa' > 'ab'
False
>>> "python" == "python"
True
>>> "python" != "PYTHON"
True
```

⚠ Attention aux majuscules (elles sont "avant" toutes les minuscules) :

```
>>> "java" < "python"
True
>>> "java" > "Python"
True
```

et aux nombres écrits dans des chaînes de caractères :

```
>>> "10" < "2"
True
```

Les nombres de type `int` ou `float` peuvent être comparés entre eux même s'ils sont de types différents :

```
>>> 7 == 7.0
True
>>> 0.0 < 1
True
```

Mais pas les nombres avec les chaînes de caractères :

```
>>> 7 == "7"
False
>>> 7 < '8'
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
TypeError: '<' not supported between instances of 'int' and 'str'
```

⚠ Attention aux égalités entre nombres de type `float` qui ne sont pas toujours encodés de façon exacte⁹ :

```
>>> 0.1 + 0.1 + 0.1 == 0.3
False
```

Opérateurs logiques (ou booléens)

Les opérations logiques peuvent être effectuées sur des booléens (type `bool`). Le résultat est un booléen égal à `True` ou `False`.

opérateur	notation	description	priorité
Négation de <code>a</code>	<code>not a</code>	True si <code>a</code> est False, False sinon	1
<code>a</code> et <code>b</code> (conjonction)	<code>a and b</code>	True si <code>a</code> et <code>b</code> sont True tous les deux, False sinon	2
<code>a</code> ou <code>b</code> (disjonction)	<code>a or b</code>	True si <code>a</code> ou <code>b</code> (ou les deux) est True, False sinon	3

(`a` et `b` sont des booléens).

Comme pour les opérations mathématiques, les opérations logiques suivent des règles de priorité :

1. Négation (`not`),
2. Conjonction (`and`),
3. Disjonction (`or`).

`a or not b and c` est équivalent à `a or ((not b) and c)` mais en pratique les parenthèses sont plus lisibles.

Expressions

Cours

Une **expression** est un calcul d'opérations et de comparaisons dont l'évaluation donne une valeur.

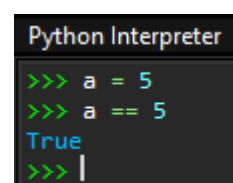
⚠ Attention à ne pas confondre une expression avec une **instruction** dont l'exécution fait quelque chose.

Exemples :

- `2*a + 5` est une expression, elle a une valeur (qui dépend de la valeur de `a`).
- `a == 5` est une expression booléenne, elle vaut `True` ou `False`.
- `a = 5` n'est **pas** une expression, c'est une instruction qui affecte de la valeur 5 à la variable `a`.

À noter:

Quand une affectation est saisie dans la console Python, par exemple `>>> a = 5`, rien n'est affiché par l'interpréteur car ce n'est pas une expression.



```
Python Interpreter
>>> a = 5
>>> a == 5
True
>>> |
```

Quand une expression est saisie dans la console Python, par exemple `>>> a == 5`, elle est évaluée par l'interpréteur et le résultat est affiché en dessous.

Puisqu'elle a une valeur, une expression peut être affectée à une variable : `b = a**2` est une affectation de la valeur de l'expression `a**2` (le carré de `a`) à la variable `b`.

Exercice corrigé

La valeur d'une variable `annee` de type `int` est donnée, par exemple `>>> annee = 2023`.

Ecrire dans l'interpréteur une expression booléenne, qui vaut `True` si `annee` est une année bissextile ou `False` sinon.

« Depuis l'ajustement du calendrier grégorien, l'année sera bissextile (elle aura 366 jours) seulement si elle respecte l'un des deux critères suivants :









1. C1 : l'année est divisible par 4 sans être divisible par 100 (cas des années qui ne sont pas des multiples de 100) ;
2. C2 : l'année est divisible par 400 (cas des années multiples de 100).

Si une année ne respecte ni le critère C1 ni le critère C2, l'année n'est pas bissextile ». Source:

https://fr.wikipedia.org/wiki/Année_bissextile.

Réponse



1. Noter dans cet exemple la différence entre variable informatique et mathématique, et la signification du signe « = ». En mathématique `a = 5` est une équation dont l'inconnue est `a` (elle peut être facilement résolue pour trouver la solution `a = 5`). En informatique, c'est l'affectation du résultat de `2*a + 1` à la variable `a` qui prend une nouvelle valeur (même si `a` était déjà égal à `2*a + 1`). 
2. Vrai pour des entiers positifs. Attention aux surprises avec des nombres relatifs ! Les résultats sont différents entre langages/systèmes informatiques. En Python on peut tester `7 // -5` et `-17 // 5` qui donnent tous les deux `-4` mais `17 % -5` donne `-3` alors que `-17 % 5` donne `3`. 
3. Pouvoir utiliser les apostrophes ou les guillemets offre un énorme avantage : les guillemets permettent d'écrire une chaîne qui contient des apostrophes et vis-versa, par exemple `"J'aime Python"` ou `'Il dit "hello".'` 
4. Attention : les opérateurs `+` et `*` se comportent différemment s'il s'agit d'entiers ou de chaînes de caractères : `2 + 2` est une addition alors que `'2' + '2'` est une concaténation, `2 * 3` est une multiplication alors que `'2' * 3` est une duplication. 
5. Une méthode est un type de fonction particulier propre aux langages orientés objet. Remarquer la construction `nom_variable.nom_methode()` dans ces cas différente de `nom_fonction(nom_variable)` par exemple `len('abc')`. 
6. `True` et `False` (et `None`) sont les rares mots en Python qui s'écrivent avec une majuscule. `TRUE` ou `true` ne sont pas acceptés. 
7. Préférer `is` et `is not` à `==` et `!=` pour comparer à `None`, par exemple `a is not None` plutôt que `a != None`. 
8. Les comparaisons entre chaînes de caractère se font en comparant le point de code Unicode de chaque caractère. Il est donné par la fonction `ord()` (la fonction `chr()` fait l'inverse). Par exemple, `ord('A')` vaut `65` et `ord('a')` vaut `97` donc `'A' < 'a'` est vrai. 
9. Les nombres de type `float` sont encodés par des fractions binaires qui "approchent" leur valeur le plus précisément possible sans être toujours parfaitement exactes. Par exemple le nombre `0.1` est représenté par la valeur `0.1000000000000000055511151231257827021181583404541015625` en Python (`format(0.1, '.55f')` permet d'afficher

toutes les décimales). Une particularité de Python est de ne pas limiter l'encodage des `int` , par exemple comparer `>>> 2*1000` avec `>>> 2.**1000` dans la console. ↩

10. En mathématique $\frac{1}{2}$ $\frac{1}{3}$. ↩

11. « Hello world » (traduit littéralement en français par « Bonjour le monde ») sont les mots traditionnellement écrits par un programme informatique simple dont le but est de faire la démonstration rapide de son exécution sans erreur. Source : https://fr.wikipedia.org/wiki/Hello_world ↩