13_python_operators

November 27, 2023

1 Python Operators

Les opérateurs sont des symboles qui effectuent des opérations sur des valeurs et des variables. Par exemple, l'addition est un opérateur qui ajoute deux valeurs, comme 2 + 3 = 5.

2 Arithmetics Operators

Les opérateurs arithmétiques sont utilisés avec des nombres (entiers et flottants) pour effectuer des opérations mathématiques Simple.

Opérateur	Nom	Exemple
+	Addition	5 + 2 = 7
-	Soustraction	7 - 2 = 5
*	Multiplication	7 * 2 = 14
/	Division	7 / 2 = 3.5
//	Floor division	7 // 2 = 3
%	Modulo	7 % 2 = 1
**	Exponentiation	7 ** 2 = 49

[ ]:
```
a = 7
b = 2

# addition
print ('Sum: ', a + b)

# subtraction
print ('Subtraction: ', a - b)

# multiplication
print ('Multiplication: ', a * b)

# division
print ('Division: ', a / b)

# floor division
print ('Floor Division: ', a // b)
```

```
# modulo
print ('Modulo: ', a % b)

# a to the power b
print ('Power: ', a ** b)
```

3 Assignment Operators

Les opérateurs d'affectation sont utilisés pour affecter des valeurs à des variables.

Opérateur	Exemple	Équivalent à
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
//=	x //= 5	x = x // 5
%=	x %= 5	x = x % 5
**=	x **= 5	x = x ** 5

[ ]:
```
# assign 10 to a
a = 10

# assign 5 to b
b = 5

# assign the sum of a and b to a
a *= b       # a = a + b

print(a)
```

4 Comparison Operators

Les opérateurs de comparaison sont utilisés pour comparer deux valeurs. Il renvoie `True` ou `False` selon la condition.

Opérateur	Nom	Exemple
==	Égal	5 == 3 renvoie False
!=	Différent (pas égal)	5 != 3 renvoie True
>	Supérieur	5 > 3 renvoie True
<	Inférieur	5 < 3 renvoie False
>=	Supérieur ou égal	5 >= 3 renvoie True
<=	Inférieur ou égal	5 <= 3 renvoie False

[ ]:
```
a = 5
b = 3

# equal to operator
print('a == b =', a == b)

# not equal to operator
print('a != b =', a != b)

# greater than operator
print('a > b =', a > b)

# less than operator
print('a < b =', a < b)

# greater than or equal to operator
print('a >= b =', a >= b)

# less than or equal to operator
print('a <= b =', a <= b)
```

Note: les opérateurs de comparaison sont utilisé dans les conditions (`if`, `while`).

5 Logical Operators

Les opérateurs logiques sont utilisés pour combiner des conditions.

Opérateur	Nom	Exemple
and	Et	x > 5 and x < 10
or	Ou	x > 5 or x < 10
not	Non	not(x > 5 and x < 10)

[ ]:
```
# logical AND
print(True and True)  # True
print(True and False) # False
print("test", ((2==2) and ((3==5) and (8==8)))) # test False

# logical OR
print(True or False) # True
print(True ^ False) # True

# logical NOT
print(not True) # False
```

6 Python Bitwise Operators

Les opérateurs bit à bit agissent sur les bits binaires des nombres. Dans le tableau suivant: laissez `x = 10` et `y = 4`, nous avons:

Opérateur	Nom	Description	Exemple
&	ET	Définit chaque bit sur 1 si les deux bits sont 1	x & y = 0 (0000 0000)
|	OU	Définit chaque bit sur 1 si l'un des deux bits est 1	x | y = 14 (0000 1110)
^	XOR	Définit chaque bit sur 1 si un seul des deux bits est 1	x ^ y = 14 (0000 1110)
~	NON	Inverse tous les bits	~x = -11 (1111 0101)
«	Décalage à gauche	Décalage des bits vers la gauche, en poussant les bits de remplissage à droite	x « 2 = 40 (0010 1000)
»	Décalage à droite	Décalage des bits vers la droite, en poussant les bits de remplissage à gauche	x » 2 = 2 (0000 0010)

[ ]:
```
x = 10
y = 13

# bitwise AND
print(x & y)

# bitwise OR
print(x | y)

# bitwise NOT
print(~x)

# bitwise XOR
print(x ^ y)

# bitwise right shift
print(x >> 2)

# bitwise left shift
print(x << 1)
```

7 Python Special Operators

Python propose des opérateurs spéciaux comme l'opérateur d'identité, l'opérateur d'appartenance, etc. Ils sont utilisés pour des tâches très spécifiques.

7.1 Identity Operators

Les opérateurs d'indentitle sont utilisés pour vérifier si deux variables sont situées au même endroit de la mémoire. Il renvoie `True` ou `False` selon la condition.

Opérateur	Description	Exemple
is	Renvoie `True` si les variables sont le même objet	x is y
is not	Renvoie `True` si les variables ne sont pas le même objet	x is not y

[ ]:
```
x1 = "antonio"
y1 = "antonio"
y1 = "python"
y12 = "python"
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]
print(x1 is y1)  # prints False
print(y1 == y1)
print(y12 is y2)  # prints True
print(x3 is y3)  # prints False
print(x3 == y3)
```

7.2 Membership Operators

Les opérateurs d'appartenance sont utilisés pour vérifier si une valeur est présente dans une séquence `(string, list, tuple, set et dictionary)`. Il renvoie `True` ou `False` selon la condition.

Opérateur	Description	Exemple
in	Renvoie `True` si une valeur est présente dans la séquence	x in y
not in	Renvoie `True` si une valeur n'est pas présente dans la séquence	x not in y

[ ]:
```
x = 'Hello world'
y = {1:'a', 2:'b'}

# check if 'H' is present in x string
```

```python
print('H' in x)  # prints True

# check if 'hello' is present in x string
print('hello' not in x)  # prints True

# check if '1' key is present in y
print(1 in y)  # prints True

# check if 'a' key is present in y
print('a' in y)  # prints False
```

8 Python Operator Precedence

L'ordre d'exécution des opérateurs est appelé précédence des opérateurs. Les opérateurs avec une précédence plus élevée sont exécutés en premier. Dans le tableau suivant, nous avons classé les opérateurs par ordre de précédence décroissante.

Source

Operator	Description
()	Parentheses
**	Exponentiation (puissance)
~X +X -X	Complement, unary plus and minus (complément, plus et moins unaire)
* / % //	Multiplication, division, modulo and floor division (multiplication, division, modulo et division entière)
+ -	Addition and subtraction (addition et soustraction)
» «	Right and left bitwise shift (décalage bit à droite et à gauche)
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR` and regular `OR` (bitwise exclusive `OR` et `OR` régulier)
<= < > >=	Comparison operators (opérateurs de comparaison)
<> == !=	Equality operators (opérateurs d'égalité)
= %= /= //= -= += *= **=	Assignment operators (opérateurs d'affectation)
is is not	Identity operators (opérateurs d'identité)
in not in	Membership operators (opérateurs d'appartenance)
not or and	Logical operators (opérateurs logiques)

9 Python Operator Overloading

L'opérateur surchargeant signifie donner un sens aux opérateurs (+, /, -, *, etc.) lorsqu'ils sont appliqués à des opérandes de différents types.

Par exemple, ajouter un entier à une chaîne, diviser un entier par un complexe, etc.

Pour effectuer une surcharge d'opérateur, Python fournit certaines fonctions spéciales ou magiques qui sont appelées lorsque ces opérateurs sont appliqués. Par exemple, lorsque vous utilisez l'opérateur `+` pour ajouter deux nombres, l'opérateur `+` est appelé et ajoute les deux nombres. Mais si vous utilisez l'opérateur `+` pour ajouter deux objets, alors derrière les coulisses, la fonction spéciale `__add__` est appelée. Voyons un exemple:

[ ]:
```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    # définit la représentation de l'objet
    def __str__(self):
        return "({0},{1})".format(self.x, self.y)

    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y
        return Point(x, y)


p1 = Point(2, 3)
p2 = Point(-1, 2)
print(p1 + p2)  # affiche (1,5)
```