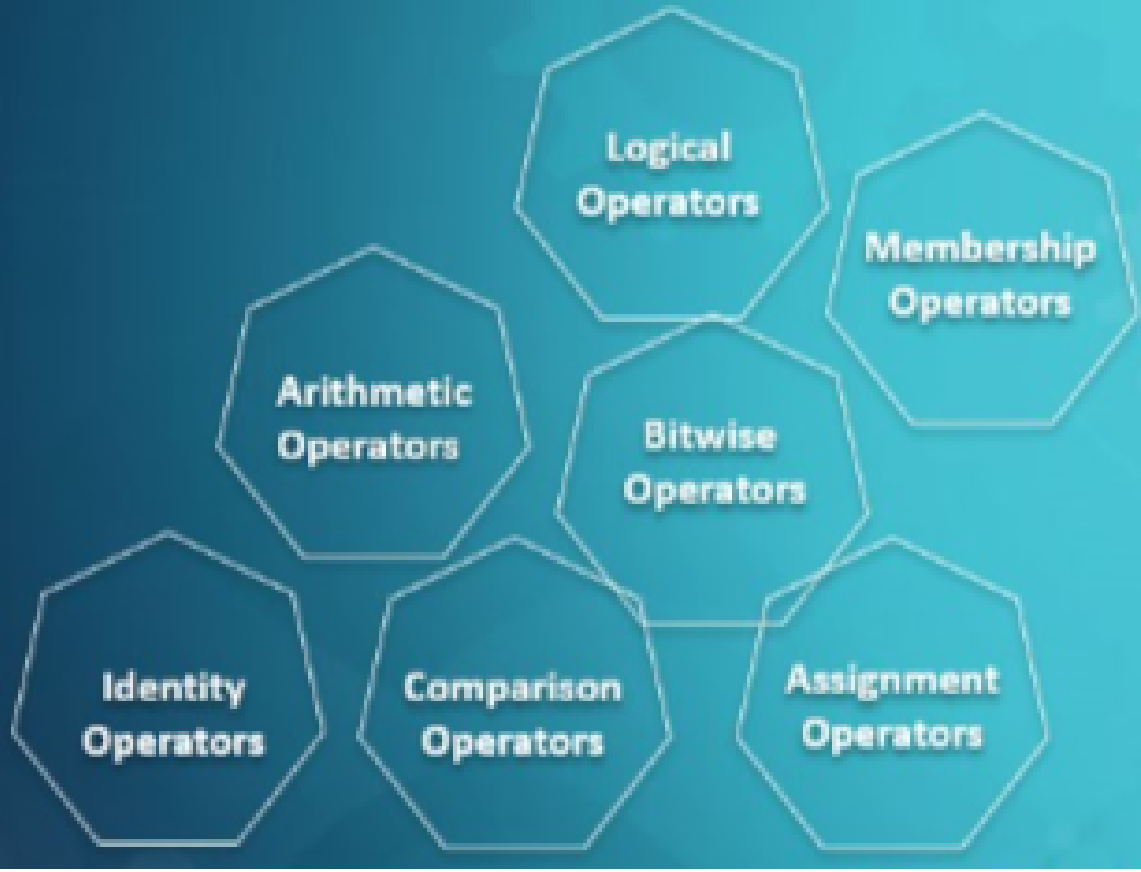
Python

Python Operators



Python Operators

The operator is a symbol that performs a certain operation between two operands, according to one definition. In a particular programming language, operators serve as the foundation upon which logic is constructed in a programme. The different operators that Python offers are listed here.

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic Operators

Arithmetic operations between two operands are carried out using arithmetic operators. It includes the exponent (**) operator as well as the + (addition), - (subtraction), * (multiplication), / (divide), % (reminder), and // (floor division) operators.

Consider the following table for a detailed explanation of arithmetic operators.

Operator	Description
+ (Addition)	It is used to add two operands. For example, if a = 10, b = 10 => a+b = 20
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value results negative. For example, if a = 20, b = 5 => a - b = 15
/ (divide)	It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2.0

* **(Multiplication)**	It is used to multiply one operand with the other. For example, if a = 20, b = 4 => a * b = 80
% (reminder)	It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0
** **(Exponent)**	As it calculates the first operand's power to the second operand, it is an exponent operator.
// **(Floor division)**	It provides the quotient's floor value, which is obtained by dividing the two operands.

Comparison Operators

Comparison operators compare the values of the two operands and return a true or false Boolean value in accordance. The following table lists the comparison operators.

Operator	Description
==	If the value of two operands is equal, then the condition becomes true.
!=	If the value of two operands is not equal, then the condition becomes true.
<=	The condition is met if the first operand is smaller than or equal to the second operand.
>=	The condition is met if the first operand is greater than or equal to the second operand.
>	If the first operand is greater than the second operand, then the condition becomes true.
<	If the first operand is less than the second operand, then the condition becomes true.

Assignment Operators

The right expression's value is assigned to the left operand using the assignment operators. The following table provides a description of the assignment operators.

Operator	Description
=	It assigns the value of the right expression to the left operand.
+=	By multiplying the value of the right operand by the value of the left operand, the left operand receives a changed value. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.
-=	It decreases the value of the left operand by the value of the right operand and assigns the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.

*=	It multiplies the value of the left operand by the value of the right operand and assigns the modified value back to then the left operand. For example, if a = 10, b = 20 => a* = b will be equal to a = a* b and therefore, a = 200.
%=	It divides the value of the left operand by the value of the right operand and assigns the reminder back to the left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0.
**=	a**=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a.
//=	A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a.

Logical Operators

The assessment of expressions to make decisions typically makes use of the logical operators. The following logical operators are supported by Python.

Operator	Description
and	The condition will also be true if the expression is true. If the two expressions a and b are the same, then a and b must both be true.
or	The condition will be true if one of the phrases is true. If a and b are the two expressions, then an or b must be true if and is true and b is false.
not	If an expression **a** is true, then not (a) will be false and vice versa.

Bitwise Operators

The two operands' values are processed bit by bit by the bitwise operators. Consider the case below.

For example,

```
if a = 7
   b = 6
then, binary (a) = 0111
    binary (b) = 0110

hence, a & b = 0011
      a | b = 0111
          a ^ b = 0100
       ~ a = 1000
```

Operator	Description
& (binary and)	A 1 is copied to the result if both bits in two operands at the same location are 1. If not, 0 is copied.
| (binary or)	The resulting bit will be 0 if both the bits are zero; otherwise, the resulting bit will be 1.
^ (binary xor)	If the two bits are different, the outcome bit will be 1, else it will be 0.
~ (negation)	The operand's bits are calculated as their negations, so if one bit is 0, the next bit will be 1, and vice versa.
<< (left shift)	The number of bits in the right operand is multiplied by the leftward shift of the value of the left operand.
>> (right shift)	The left operand is moved right by the number of bits present in the right operand.

Membership Operators

The membership of a value inside a Python data structure can be verified using Python membership operators. The result is true if the value is in the data structure; otherwise, it returns false.

Operator	Description
in	If the first operand cannot be found in the second operand, it is evaluated to be true (list, tuple, or dictionary).
not in	If the first operand is not present in the second operand, the evaluation is true (list, tuple, or dictionary).

Identity Operators

Operator	Description
is	If the references on both sides point to the same object, it is determined to be true.
is not	If the references on both sides do not point at the same object, it is determined to be true.