Python - Operators

Python operators are the constructs which can manipulate the value of operands. These are symbols used for the purpose of logical, arithmetic and various other operations.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called **operands** and + is called **operator**. In this tutorial, we will study different types of Python operators.

Types of Python Operators

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Let us have a quick look on all these operators one by one.

Python Arithmetic Operators

Python arithmetic operators are used to perform mathematical operations on numerical values. These operations are Addition, Subtraction, Multiplication, Division, Modulus, Expoents and Floor Division.

Operator	Name	Example
+	Addition	10 + 20 = 30
-	Subtraction	20 – 10 = 10
*	Multiplication	10 * 20 = 200
/	Division	20 / 10 = 2
%	Modulus	22 % 10 = 2
**	Exponent	4**2 = 16
//	Floor Division	9//2 = 4

Example

Following is an example which shows all the above operations:

Edit & Run

```
a = 21
b = 10

# Addition
print ("a + b : ", a + b)

# Subtraction
print ("a - b : ", a - b)

# Multiplication
print ("a * b : ", a * b)

# Division
print ("a / b : ", a / b)

# Modulus
print ("a % b : ", a % b)

# Exponent
print ("a ** b : ", a ** b)

# Floor Division
print ("a // b : ", a // b)
```

This produce the following result –

```
a + b :  31
a - b :  11
a * b :  210
a / b :  2.1
```

```
a % b :  1
a ** b :  16679880978201
a // b :  2
```

Python Comparison Operators

Python comparison operators compare the values on either sides of them and decide the relation among them. They are also called relational operators. These operators are equal, not equal, greater than, less than, greater than or equal to and less than or equal to.

Operator	Name	Example
==	Equal	4 == 5 is not true.
!=	Not Equal	4 != 5 is true.
>	Greater Than	4 > 5 is not true.
<	Less Than	4 < 5 is true.
>=	Greater than or Equal to	4 >= 5 is not true.
<=	Less than or Equal to	4 <= 5 is true.

Example

Following is an example which shows all the above comparison operations:

Edit & Run

```python
a = 4
b = 5

# Equal
print ("a == b : ", a == b)

# Not Equal
print ("a != b : ", a != b)

# Greater Than
print ("a > b : ", a > b)

# Less Than
print ("a < b : ", a < b)

# Greater Than or Equal to
print ("a >= b : ", a >= b)
```

```
# Less Than or Equal to
print ("a <= b : ", a <= b)
```

This produce the following result –

```
a == b :  False
a != b :  True
a > b :  False
a < b :  True
a >= b :  False
a <= b :  True
```

Python Assignment Operators

Python assignment operators are used to assign values to variables. These operators include simple assignment operator, addition assign, subtraction assign, multiplication assign, division and assign operators etc.

Operator	Name	Example
=	Assignment Operator	a = 10
+=	Addition Assignment	a += 5 (Same as a = a + 5)
-=	Subtraction Assignment	a -= 5 (Same as a = a - 5)
*=	Multiplication Assignment	a *= 5 (Same as a = a * 5)
/=	Division Assignment	a /= 5 (Same as a = a / 5)
%=	Remainder Assignment	a %= 5 (Same as a = a % 5)
**=	Exponent Assignment	a **= 2 (Same as a = a ** 2)
//=	Floor Division Assignment	a //= 3 (Same as a = a // 3)

Example

Following is an example which shows all the above assignment operations:

Edit & Run

```
# Assignment Operator
a = 10

# Addition Assignment
a += 5
print ("a += 5 : ", a)

# Subtraction Assignment
```

```
a -= 5
print ("a -= 5 : ", a)

# Multiplication Assignment
a *= 5
print ("a *= 5 : ", a)

# Division Assignment
a /= 5
print ("a /= 5 : ",a)

# Remainder Assignment
a %= 3
print ("a %= 3 : ", a)

# Exponent Assignment
a **= 2
print ("a **= 2 : ", a)

# Floor Division Assignment
a //= 3
print ("a //= 3 : ", a)
```

This produce the following result –

```
a += 5 :  105
a -= 5 :  100
a *= 5 :  500
a /= 5 :  100.0
a %= 3 :  1.0
a **= 2 :  1.0
a //= 3 :  0.0
```

Python Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and b = 13; Now in the binary format their values will be 0011 1100 and 0000 1101 respectively. Following table lists out the bitwise operators supported by Python language with an example each in those, we use the above two variables (a and b) as operands –

a = 0011 1100

b = 0000 1101

----

a&b = 12 (0000 1100)

a|b = 61 (0011 1101)

a^b = 49 (0011 0001)

~a = -61 (1100 0011)

a << 2 = 240 (1111 0000)

a>>2 = 15 (0000 1111)

There are following Bitwise operators supported by Python language

Operator	Name	Example
&	Binary AND	Sets each bit to 1 if both bits are 1
|	Binary OR	Sets each bit to 1 if one of two bits is 1
^	Binary XOR	Sets each bit to 1 if only one of two bits is 1
~	Binary Ones Complement	Inverts all the bits
<<	Binary Left Shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Binary Right Shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Example

Following is an example which shows all the above bitwise operations:

Edit & Run

```
a = 60            # 60 = 0011 1100
b = 13            # 13 = 0000 1101

# Binary AND
c = a & b        # 12 = 0000 1100
print ("a & b : ", c)

# Binary OR
c = a | b        # 61 = 0011 1101
print ("a | b : ", c)

# Binary XOR
c = a ^ b        # 49 = 0011 0001
print ("a ^ b : ", c)

# Binary Ones Complement
c = ~a;           # -61 = 1100 0011
print ("~a : ", c)
```

```
# Binary Left Shift
c = a << 2;        # 240 = 1111 0000
print ("a << 2 : ", c)

# Binary Right Shift
c = a >> 2;        # 15 = 0000 1111
print ("a >> 2 : ", c)
```

This produce the following result –

```
a & b :  12
a | b :  61
a ^ b :  49
~a :  -61
a >> 2 :  240
a >> 2 :  15
```

Python Logical Operators

There are following logical operators supported by Python language. Assume variable a holds 10 and variable b holds 20 then

[Show Example]

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

Python Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below –

[Show Example]

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Python Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators explained below –

[Show Example]

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

Python Operators Precedence

The following table lists all operators from highest precedence to lowest.

[Show Example]

Sr.No.	Operator & Description
1	****** Exponentiation (raise to the power)
2	**~ + -** Complement, unary plus and minus (method names for the last two are +@ and -@)
3	*** / % //** Multiply, divide, modulo and floor division
4	**+ -** Addition and subtraction
5	**>> <<** Right and left bitwise shift
6	**&** Bitwise 'AND'
7	**^ |** Bitwise exclusive `OR' and regular `OR'
8	**<= < > >=** Comparison operators
9	**<> == !=** Equality operators
10	**= %= /= //= -= += *= **=** Assignment operators
11	**is is not**

	Identity operators
12	**in not in** Membership operators
13	**not or and** Logical operators