Python Operators

In the following tutorial, we will discuss about the operators used in the Python programming language.

An Introduction to Operators in Python

In general, **Operators** are the symbols used to perform a specific operation on different values and variables. These values and variables are considered as the **Operands,** on which the operator is applied. Operators serve as the foundation upon which logic is constructed in a program in a particular programming language. In every programming language, some operators perform several tasks.

Different Types of Operators in Python

Same as other languages, Python also has some operators, and these are given below -

1. **Arithmetic Operators**
2. **Comparison Operators**
3. **Assignment Operators**
4. **Logical Operators**
5. **Bitwise Operators**
6. **Membership Operators**
7. **Identity Operators**

Let us now discuss these operators used in Python in the following sections.

Arithmetic Operators

Python Arithmetic Operators are used on two operands to perform basic mathematical operators like addition, subtraction, multiplication, and division. There are different types of arithmetic operators available in Python including the '+' operator for addition, '-' operator for subtraction, '*' for multiplication, '/' for division, '%' for modulus, '**' for exponent and '//' for floor division.

Let us consider the following table of arithmetic operators for a detailed explanation.

S. No.	Operator	Syntax	Description

1	**+ (Addition)**	r = a + b	This operator is used to add two operands. For example, if a = 15, b = 10 => a + b = 15 + 10 = 25
2	**- (Subtraction)**	r = a - b	This operator 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 = 20 – 5 = 15
3	**/ (divide)**	r = a / b	This operator returns the quotient after dividing the first operand by the second operand. For example, if a = 15, b = 4 => a / b = 15 / 4 = 3.75
4	*** (Multiplication)**	r = a * b	This operator is used to multiply one operand with the other. For

			example, if a = 20, b = 4 => a * b = 20 * 4 = 80
5	**% (reminder)**	r = a % b	This operator returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a % b = 20 % 10 = 0
6	**** (Exponent)**	r = a ** b	As this operator calculates the first operand's power to the second operand, it is an exponent operator. For example, if a = 2, b = 3 => a**b = 2**3 = 2^3 = 2*2*2 = 8
7	**// (Floor division)**	r = a // b	This operator provides the quotient's floor value, which is obtained by dividing the two operands. For example, if a =

15, b = 4 => a // b =
15 // 4 = 3

Program Code:

Now we give code examples of arithmetic operators in Python. The code is given below -

```
a = 46    # Initializing the value of a
b = 4     # Initializing the value of b

print("For a =", a, "and b =", b,"\nCalculate the following:")

# printing different results
print('1. Addition of two numbers: a + b =', a + b)
print('2. Subtraction of two numbers: a - b =', a - b)
print('3. Multiplication of two numbers: a * b =', a * b)
print('4. Division of two numbers: a / b =', a / b)
print('5. Floor division of two numbers: a // b =',a // b)
print('6. Reminder of two numbers: a mod b =', a % b)
print('7. Exponent of two numbers: a ^ b =',a ** b)
```

Output:

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

For a = 46 and b = 4
Calculate the following:
1. Addition of two numbers: a + b = 50
2. Subtraction of two numbers: a - b = 42
*3. Multiplication of two numbers: a * b = 184*
4. Division of two numbers: a / b = 11.5
5. Floor division of two numbers: a // b = 11
6. Reminder of two numbers: a mod b = 2
7. Exponent of two numbers: a ^ b = 4477456

Comparison Operators

Python Comparison operators are mainly used for the purpose of comparing two values or variables (operands) and return a Boolean value as either True or False accordingly. There are various types of comparison operators available in Python including the '==', '!=', '<=', '>=', '<', and '>'.

Let us consider the following table of comparison operators for a detailed explanation.

S. No.	Operator	Syntax	Description
1	==	a == b	Equal to: If the value of two operands is equal, then the condition becomes true.
2	!=	a != b	Not Equal to: If the value of two operands is not equal, then the condition becomes true.
3	<=	a <= b	Less than or Equal to: The condition is met if the first operand is smaller than or equal to the second operand.
4	>=	a >= b	Greater than or Equal to: The condition is met if the first operand is greater than or

			equal to the second operand.
5	>	a > b	Greater than: If the first operand is greater than the second operand, then the condition becomes true.
6	<	a < b	Less than: If the first operand is less than the second operand, then the condition becomes true.

Program Code:

Now we give code examples of Comparison operators in Python. The code is given below -

```
a = 46    # Initializing the value of a
b = 4     # Initializing the value of b

print("For a =", a, "and b =", b,"\nCheck the following:")

# printing different results
print('1. Two numbers are equal or not:', a == b)
print('2. Two numbers are not equal or not:', a != b)
print('3. a is less than or equal to b:', a <= b)
print('4. a is greater than or equal to b:', a >= b)
print('5. a is greater b:', a > b)
print('6. a is less than b:', a < b)
```

Output:

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

For a = 46 and b = 4
Check the following:
1. Two numbers are equal or not: False
2. Two numbers are not equal or not: True
3. a is less than or equal to b: False
4. a is greater than or equal to b: True
5. a is greater b: True
6. a is less than b: False

Assignment Operators

Using the assignment operators, the right expression's value is assigned to the left operand. Python offers different assignment operators to assign values to the variable. These assignment operators include '=', '+=', '-=', '*=', '/=', '%=', '//=', '**=', '&=', '|=', '^=', '>>=', and '<<='.

Let us consider the following table of some commonly used assignment operators for a detailed explanation.

S. No.	Operator	Syntax	Description
1	=	a = b + c	This operator assigns the value of the right expression to the left operand.
2	+=	a += b => a = a + b	Add AND: This operator adds the operand on the right side to the operand on the left side and assigns the resultant value to the left operand.

			For example, if a = 15, b = 20 => a += b will be equal to a = a + b and therefore, a = 15 + 20 => a = 35
3	-=	a -= b => a = a - b	Subtract AND: This operator subtracts the operand on the right side from the operand on the left side and assigns the resultant value to the left operand. For example, if a = 47, b = 32 => a -= b will be equal to a = a - b and therefore, a = 47 - 32 => a = 15
4	*=	a *= b => a = a * b	Multiply AND: This operator multiplies the operand on the right side with the operand on the left side and assigns the resultant value to the left operand. For example, if a = 12, b = 4 => a *= b will be equal to a = a * b and therefore, a = 12 * 4 => a = 48

5	/=	a /= b => a = a / b	Divide AND: This operator divides the operand on the left side with the operand on the right side and assign the resultant value to the left operand. For example, if a = 15, b = 2 => a /= b will be equal to a = a / b and therefore, a = 15 / 2 => a = 7.5
6	%=	a %= b => a = a % b	Modulus AND: This operator calculates the modulus of using the left-side and right-side operands and assigns the resultant value to the left operand. For example, if a = 20, b = 10 => a %= b will be equal to a = a % b and therefore, a = 20 % 10 => a = 0
7	**=	a **= b => a = a ** b	Exponent AND: This operator calculates the exponent value using the operands on both side and

			assign the resultant value to the left operand. For example, if a = 2, b = 3 => a **= b will be equal to a = a ** b and therefore, a = 2 ** 3 = 8
8	//=	a //= b => a = a // b	Divide (floor) AND: This operator divides the operand on the left side with the operand on the right side and assign the resultant floor value to the left operand. For example, if a = 15, b = 2 => a //= b will be equal to a = a // b and therefore, a = 15 // 2 => a = 7

Program Code:

Now we give code examples of Assignment operators in Python. The code is given below -

```
a = 34        # Initialize the value of a
b = 6         # Initialize the value of b

# printing the different results
print('a += b:', a + b)
print('a -= b:', a - b)
print('a *= b:', a * b)
```

```
print('a /= b:', a / b)
print('a %= b:', a % b)
print('a **= b:', a ** b)
print('a //= b:', a // b)
```

Output:

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

```
a += b: 40
a -= b: 28
a *= b: 204
a /= b: 5.666666666666667
a %= b: 4
a **= b: 1544804416
a //= b: 5
```

Bitwise Operators

The two operands' values are processed bit by bit by the bitwise operators. There are various Bitwise operators used in Python, such as bitwise OR (|), bitwise AND (&), bitwise XOR (^), negation (~), Left shift (<<), and Right shift (>>). 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
Let, Binary of x = 0101
     Binary of y = 1000
Bitwise OR = 1101
8 4 2 1
1 1 0 1 = 8 + 4 + 1 = 13
```

15.
16. Bitwise AND = 0000
17. 0000 = 0
18.
19. Bitwise XOR = 1101
20. 8 4 2 1
21. 1 1 0 1 = 8 + 4 + 1 = 13
22. Negation of x = ~x = (-x) - 1 = (-5) - 1 = -6
23. ~x = -6

Let us consider the following table of bitwise operators for a detailed explanation.

S. No.	Operator	Syntax	Description
1	&	a & b	**Bitwise AND:** 1 is copied to the result if both bits in two operands at the same location are 1. If not, 0 is copied.
2	|	a | b	**Bitwise OR:** The resulting bit will be 0 if both the bits are zero; otherwise, the resulting bit will be 1.
3	^	a ^ b	**Bitwise XOR:** If the two bits are different, the outcome bit will be 1, else it will be 0.

4	~	~a	**Bitwise NOT:** The operand's bits are calculated as their negations, so if one bit is 0, the next bit will be 1, and vice versa.
5	<<	a <<	**Bitwise Left Shift:** The number of bits in the right operand is multiplied by the leftward shift of the value of the left operand.
6	>>	a >>	**Bitwise Right Shift:** The left operand is moved right by the number of bits present in the right operand.

Program Code:

Now we give code examples of Bitwise operators in Python. The code is given below -

```
1. a = 7        # initializing the value of a
```

```
b = 8        # initializing the value of b

# printing different results
print('a & b :', a & b)
print('a | b :', a | b)
print('a ^ b :', a ^ b)
print('~a :', ~a)
print('a << b :', a << b)
print('a >> b :', a >> b)
```

Output:

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

```
a & b : 0
a | b : 15
a ^ b : 15
~a : -8
a << b : 1792
a >> b : 0
```

Logical Operators

The assessment of expressions to make decisions typically uses logical operators. Python offers different types of logical operators such as and, or, and not. In the case of the logical AND, if the first one is 0, it does not depend upon the second one. In the case of the logical OR, if the first one is 1, it does not depend on the second one.

Let us consider the following table of the logical operators used in Python for a detailed explanation.

S. No.	Operator	Syntax	Description

1	and	a and b	**Logical 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 both must be true.
2	or	a or b	**Logical OR:** The condition will be true if one of the phrases is true. If a and b are the two expressions, then either a or b must be true to make the condition true.
3	not	not a	**Logical NOT:** If an expression a is true, then not (a) will be false and vice versa.

Program Code:

Now we give code examples of Logical operators in Python. The code is given below -

```
1. a = 7        # initializing the value of a
```

```
2.
3. # printing different results
4. print("For a = 7, checking whether the following conditions are True or F
   alse:")
5. print('\"a > 5 and a < 7\" =>', a > 5 and a < 7)
6. print('\"a > 5 or a < 7\" =>', a > 5 or a < 7)
7. print('\"not (a > 5 and a < 7)\" =>', not(a > 5 and a < 7))
```

Output:

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

For a = 7, checking whether the following conditions are True or False:
"a > 5 and a < 7" => False
"a > 5 or a < 7" => True
"not (a > 5 and a < 7)" => True

Membership Operators

We can verify the membership of a value inside a Python data structure using the Python membership operators. The result is said to be true if the value or variable is in the sequence (list, tuple, or dictionary); otherwise, it returns false.

S. No.	Operator	Description
1	in	If the first operand (value or variable) is present in the second operand (sequence), it is evaluated to be true. Sequence can either be a list, tuple, or dictionary
2	not in	If the first operand (value or variable) is not present in the second operand

(sequence), the evaluation is true. Sequence can either be a list, tuple, or dictionary

Program Code:

Now we give code examples of Membership operators in Python. The code is given below -

```
# initializing a list
myList = [12, 22, 28, 35, 42, 49, 54, 65, 92, 103, 245, 874]

# initializing x and y with some values
x = 31
y = 28

# printing the given list
print("Given List:", myList)

# checking if x is present in the list or not
if (x not in myList):
    print("x =", x,"is NOT present in the given list.")
else:
    print("x =", x,"is present in the given list.")

# checking if y is present in the list or not
if (y in myList):
    print("y =", y,"is present in the given list.")
else:
    print("y =", y,"is NOT present in the given list.")
```

Output:

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

Given List: [12, 22, 28, 35, 42, 49, 54, 65, 92, 103, 245, 874]
x = 31 is NOT present in the given list.

y = 28 is present in the given list.

Identity Operators

Python offers two identity operators as is and is not, that are used to check if two values are located on the same part of the memory. Two variables that are equal do not imply that they are identical.

S. No.	Operator	Description
1	is	If the references on both sides point to the same object, it is determined to be true.
2	is not	If the references on both sides do not point at the same object, it is determined to be true.

Program Code:

Now we give code examples of Identity operators in Python. The code is given below -

```
# initializing two variables a and b
a = ["Rose", "Lotus"]
b = ["Rose", "Lotus"]

# initializing a variable c and storing the value of a in c
c = a

# printing the different results
print("a is c => ", a is c)
print("a is not c => ", a is not c)
print("a is b => ", a is b)
print("a is not b => ", a is not b)
```

```
13.print("a == b => ", a == b)
14.print("a != b => ", a != b)
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

a is c => True
a is not c => False
a is b => False
a is not b => True
a == b => True
a != b => False

Operator Precedence

The order in which the operators are examined is crucial to understand since it tells us which operator needs to be considered first. Below is a list of the Python operators' precedence tables.

S. No.	Operator	Description
1	**	Overall other operators employed in the expression, the exponent operator is given precedence.
2	~, +, -	the minus, unary plus, and negation.

3	*, /, %, //	the division of the floor, the modules, the division, and the multiplication.
4	+, -	Binary plus, and minus
5	>>, <<	Left shift. and right shift
6	&	Binary and.
7	^, |	Binary xor, and or
8	<=, <, >, >=	Comparison operators (less than, less than equal to, greater than, greater then equal to).
9	<>, ==, !=	Equality operators.
10	=, %=, /=, //=, -=, +=, *=, **=	Assignment operators
11	is, is not	Identity operators
12	in, not in	Membership operators

13	not, or, and	Logical operators