Python Operators

The operator can be defined as a symbol which is responsible for a particular operation between two operands. Operators are the pillars of a program on which the logic is built in a specific programming language. Python provides a variety of operators, which are described as follows.

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

Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations between two operands. It includes + (addition), - (subtraction), *(multiplication), /(divide), %(reminder), //(floor division), and exponent (**) 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 = 20, b = 10 => a+b = 30
- (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 = 10 => a - b = 10
/ (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 = 10 => a * b = 200
% (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)**	It is an exponent operator represented as it calculates the first operand power to the second operand. x = 2, y = 5 print(x ** y) #same as 2*2*2*2*2

// (Floor division)	It gives the floor value of the quotient produced by dividing the two operands. x = 15, y = 2
	print(x // y)
	#the floor division // rounds the result down to the nearest whole number

Comparison operator

Comparison operators are used to comparing the value of the two operands and returns Boolean true or false accordingly. The comparison operators are described in the following table.

Operator	Description
==	If the value of two operands is equal, then the condition becomes true. x = 5, y = 3, print(x == y) # returns False because 5 is not equal to 3
!=	If the value of two operands is not equal, then the condition becomes true. x = 5, y = 3, print(x != y) # returns True because 5 is not equal to 3
<=	If the first operand is less than or equal to the second operand, then the condition becomes true.

	x = 5, y = 3, print(x <= y) # returns False because 5 is neither less than or equal to 3
>=	If the first operand is greater than or equal to the second operand, then the condition becomes true. x = 5, y = 3, print(x >= y) # returns True because five is greater, or equal, to 3
>	If the first operand is greater than the second operand, then the condition becomes true. x = 5, y = 3, print(x > y) # returns True because 5 is greater than 3
<	If the first operand is less than the second operand, then the condition becomes true. x = 5, y = 3, print(x < y) # returns False because 5 is not less than 3

Assignment Operators

The assignment operators are used to assign the value of the right expression to the left operand. The assignment operators are described in the following table.

Operator	Description
=	It assigns the value of the right expression to the left operand.
+=	It increases 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 = 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 logical operators are used primarily in the expression evaluation to make a decision. Python supports the following logical operators.

Operator	Description
and	If both the expression are true, then the condition will be true. If a and b are the two expressions, a → true, b → true => a and b → true. **x = 5** **print(x > 3 and x < 10)** **# returns True because 5 is greater than 3 AND 5 is less than 10**
or	If one of the expressions is true, then the condition will be true. If a and b are the two expressions, a → true, b → false => a or b → true.

	x = 5 **print(x > 3 or x < 4)** **# returns True because one of the conditions are true (5 is greater than 3, but 5 is not less than 4)**
not	If an expression **a** is true, then not (a) will be false and vice versa. **x = 5** **print(not(x > 3 and x < 10))** **# returns False because not is used to reverse the result**

Membership Operators

Python membership operators are used to check the membership of value inside a Python data structure. If the value is present in the data structure, then the resulting value is true otherwise it returns false.

Operator	Description
in	It is evaluated to be true if the first operand is found in the second operand (list, tuple, or dictionary). **x = ["apple", "banana"]** **print("banana" in x)** **# returns True because a sequence with the value "banana" is in the list**
not in	It is evaluated to be true if the first operand is not found in the second operand (list, tuple, or dictionary). **x = ["apple", "banana"]** **print("pineapple" not in x)** **# returns True because a sequence with the value "pineapple" is not in the list**

Identity Operators

The identity operators are used to decide whether an element certain class or type.

Operator	Description
is	It is evaluated to be true if the reference present at both sides point to the same object. **x = ["apple", "banana"]** **y = ["apple", "banana"]** **z = x** **print(x is z)** **# returns True because z is the same object as x** **print(x is y)** **# returns False because x is not the same object as y, even if they have the same content** **print(x == y)** **# to demonstrate the difference betweeen "is" and "==": this comparison returns True because x is equal to y**
is not	It is evaluated to be true if the reference present at both sides do not point to the same object. **x = ["apple", "banana"]** **y = ["apple", "banana"]** **z = x** **print(x is not z)** **# returns False because z is the same object as x** **print(x is not y)** **# returns True because x is not the same object as y, even if they have the same content** **print(x != y)** **# to demonstrate the difference betweeen "is not" and "!=": this comparison returns False because x is equal to y**

Bitwise Operators

The bitwise operators perform bit by bit operation on the values of the two operands. Consider the following example.

For example,

if a = 7
b = 6
then, binary (a) = 0111
binary (b) = 0011

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

1.

Operator	Description
& (binary and)	If both the bits at the same place in two operands are 1, then 1 is copied to the result. Otherwise, 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)	The resulting bit will be 1 if both the bits are different; otherwise, the resulting bit will be 0.
~ (negation)	It calculates the negation of each bit of the operand, i.e., if the bit is 0, the resulting bit will be 1 and vice versa.
<< (left shift)	The left operand value is moved left by the number of bits present in the right operand.

>> (right shift)	The left operand is moved right by the number of bits present in the right operand.

Operator Precedence

The precedence of the operators is essential to find out since it enables us to know which operator should be evaluated first. The precedence table of the operators in Python is given below.

Operator	Description
**	The exponent operator is given priority over all the others used in the expression.
~ + -	The negation, unary plus, and minus.
* / % //	The multiplication, divide, modules, reminder, and floor division.
+ -	Binary plus, and minus
>> <<	Left shift. and right shift
&	Binary and.
^ |	Binary xor, and or
<= < > >=	Comparison operators (less than, less than equal to, greater than, greater then equal to).
<> == !=	Equality operators.
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators

not or and	Logical operators

Python Operators

Operators are used to perform operations on values and variables. The Python operators are classified into seven different categories:

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

Arithmetic Operators

Arithmetic operators are used to perform simple mathematical operations on numeric values (except complex).

Operator	Meaning	Example
+	Addition	x + y
–	Subtraction	x – y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y

**	Exponentiation	x ** y
//	Floor division	x // y

Here are some examples:

```
x = 6
y = 2

# addition
print(x + y)          # 8

# subtraction
print(x - y)          # 4

# multiplication
print(x * y)          # 12

# division
print(x / y)          # 3

# modulus
print(x % y)          # 0

# exponentiation
print(x ** y)         # 36

# floor division
print(x // y)         # 3
```

For additional numeric operations see the math module.

Assignment Operators

Assignment operators are used to assign new values to variables.

Operator	Meaning	Example	Equivatent to
=	Assignment	x = 3	x = 3
+=	Addition assignment	x += 3	x = x + 3
-=	Subtraction assignment	x -= 3	x = x – 3
*=	Multiplication assignment	x *= 3	x = x * 3
/=	Division assignment	x /= 3	x = x / 3
%=	Modulus assignment	x %= 3	x = x % 3
//=	Floor division assignment	x //= 3	x = x // 3
**=	Exponentiation assignment	x **= 3	x = x ** 3

Comparison Operators

Comparison operators are used to compare two values.

Operator	Meaning	Example
==	Equal to	x == y
!=	Not equal to	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Here are some examples:

```
x = 6
y = 2

# equal to
print(x == y)           # False

# not equal to
print(x != y)           # True

# greater than
print(x > y)            # True

# less than
print(x < y)            # False

# greater than or equal to
print(x >= y)           # True

# less than or equal to
print(x <= y)           # False
```

Logical Operators

Logical operators are used to join two or more conditions.

Operator	Description	Example
and	Returns True if both statements are true	x > 0 and y < 0
or	Returns True if one of the statements is true	x > 0 or y < 0
not	Reverse the result, returns False if the result is true	not(x > 0 and < 0)

Here are some examples:

```
x = 2
y = -2

# and
print(x > 0 and y < 0)        # True

# or
print(x > 0 or y < 0)         # True

# not
print(not(x > 0 and y < 0))    # False
```

Identity Operators

Identity operators are used to check if two objects point to the same object, with the same memory location.

Operator	Description	Example
is	Returns true if both variables are the same object	x is y
is not	Returns true if both variables are not the same object	x is not y

Here are some examples:

```
x = [1, 2, 3]
y = [1, 2, 3]

# is
print(x is y)                 # False

# is not
print(x is not y)      # True
```

Membership Operators

Membership operators are used to check if a specific item is present in a sequence (such as a string, tuple, list, or range) or a collection (such as a dictionary, set, or frozen set).

Operator	Description	Example
in	Returns True if a value is present in the sequence	x in y
not in	Returns True if a value is not present in the sequence	x not in y

Here are some examples:

```
L = ['red', 'green', 'blue']


# in
print('red' in L)         # True


# not in
print('yellow' not in L)    # True
```

Bitwise Operators

Binary operators are used to perform bit-level operations on (binary) numbers.

Operator	Meaning	Example
&	AND	x & y
|	OR	x | y
^	XOR	x ^ y
~	NOT	~x
<<	Left shift	x << 2
>>	Right shift	x >> 2

Here are some examples:

```
x = 0b1100
y = 0b1010

# and
print(bin(x & y))        # 0b1000

# or
print(bin(x | y))        # 0b1110

# xor
print(bin(x ^ y))        # 0b0110

# not
print(bin(~x))                  # -0b1101

# shift 2 bits left
print(bin(x << 2))       # 0b1100

# shift 2 bits right
print(bin(x >> 2))       # 0b0011
```

Operator Precedence (Order of Operations)

In Python, every operator is assigned a precedence. Operator Precedence determines which operations are performed before which other operations.

Operators of highest precedence are performed first. Any operators of equal precedence are performed in left-to-right order.

Precedence	Operator	Description
lowest precedence	or	Boolean OR

	and	Boolean AND
	not	Boolean NOT
	==, ! =, <, <=, >, >=, is, is not	comparisons, identity
	|	bitwise OR
	^	bitwise XOR
	&	bitwise AND
	<<, >>	bit shis
	+, –	addition, subtraction
	*, /, //, %	multiplication, division, floor division, modulo
	+x, -x, ~x	unary positive, unary negation, bitwise negation
highest precedence	**	exponentiation