Operators in Python

An operator is a symbol that performs an operation. An operator acts on some variables called operands. For example, if we write a+b, the operator '+' is acting on two operands 'a' and 'b'. If an operator acts on a single variable, it is called a unary operator. If an operator acts on two variables, it is called a binary operator. If an operator acts on three variables, then it is called ternary operator. This is one type of classification. We can classify the operators depending upon their nature, as shown below:

1. Arithmetic operators
2. Assignment operators
3. Unary minus operator
4. Relational operators
5. Logical operators
6. Boolean operators
7. Bitwise operators
8. Membership operators
9. Identity operators

Arithmetic operators

These operators are used to perform basic arithmetic operations subtraction, division, etc. There are seven arithmetic operators available in Python. Since these operators act on two operands, they are also called binary operators.

Operator	Meaning	Example	Result
+	Addition operator. Adds two values.	12+5	17
-	Subtraction operator. Subtracts one value from another	12-5	7
*	Multiplication operator. Multiplies values on either side of the operator.	12*5	60
/	Division operator. Divides left operand by the right operand	12/5	2.4
%	Modulus operator. Gives remainder of division.	12%5	2
**	Exponent operator. Calculates exponential power value. ab gives the value of a to the power of b.	12**5	248832

//	Integer division. This is also called Floor division.Performs division and gives only integer quotients.	12//5	2

When there is an expression that contains several arithmetic operators, we should know which operation is done first and which operation is done next. In such cases, the following order of evaluation is used:

1. First parentheses are evaluated.
2. Exponentiation is done next.
3. Multiplication, division, modulus and floor divisions are at equal priority
4. Addition and subtraction are done afterwards
5. Finally, the assignment operation is performed.

Let's take a sample expression: d= (x+y)z**a//b+c. Assume the values of variables as x=1; y=2; z=3; a=2; b=2; c=3. Then, the given expression d=(1+2)*3**2//2+3 will evaluate as:

1. First parentheses are evaluated, d=3*3**2//2-3.
2. Exponentiation is done next. d=3-9//2+3.
3. Multiplication, division, modulus and floor divisions are at equal priority. d=27//2+3 and then d =13+3.
4. Addition and subtraction are done afterwards. d =16.
5. Finally, the assignment is performed. The value 16 is now stored into 'd'. Hence, the total value of the expression becomes 16 which is stored in the variable ‘d’.

Assignment operators

These operators are useful to store the right side value into a left side variable. They can also be used to perform simple arithmetic operations like addition, subtraction, etc., and then store the result into a variable. Here, let a=2.

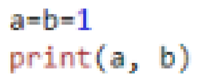
Operator	Meaning	Example	Similar to	Result
=	Assignment operator. Stores right side value into left side variable.	a=2	a=2	a=2
+=	Addition assignment operator. Adds right operand to the left operand and stores the result into left operand.	a+=2	a=a+2	a=4

-=	Subtraction assignment operator. Subtracts right operand from left operand and stores the result into left operand.	a-=2	a=a-2	a=0

*=	Multiplication assignment operator. Multiplies right operand with left operand and stores the result into left operand.	a*=2	a=a*2	a=4
/=	Division assignment operator. Divides left operand with right operand and stores the result into left operand.	a/=2	a=a/2	a=1
%=	Modulus assignment operator. Divides left operand with right operand and stores the remainder into left operand.	a%=2	a=a%2	a=0
//=	Floor division assignment operator. Performs floor division and then stores the result into left operand	a//=2	a=a//2	a=1
**=	Exponentiation assignment operator. Performs power value and then stores the result into the left operand.	a**=2	a=a**2	a=4
&=	Compound bitwise AND assignment operator.It uses the binary representation of both operands, does a bitwise AND operation on them and assigns the result to the variable.	a&=2	a=a&2	a=2
|=	Compound bitwise OR operator.It uses the binary representation of both operands, does a bitwise OR operation on them	a|=2	a=a|2	a=2

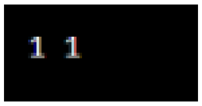
	and assigns the result to the variable.			
^=	Compound bitwise XOR operator.It uses the binary representation of both operands, does a bitwise XOR operation on them and assigns the result to the variable.	a^=2	a=a^2	a=0
>>=	Right shift assignment operator.It moves the specified amount of bits to the right and assigns the result to the variable.	a>>=2	a=a>>2	a=0
<<=	Left shift assignment operator. It moves the specified amount of bits to the left and assigns the result to the variable.	a<<=2	a=<<2	a=8

It is possible to assign the same value to two variables in the same statement as:



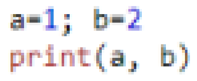
```
a=b=1
print(a, b)
```

Output:



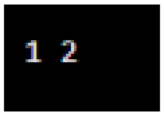
```
1 1
```

Another example is where we can assign different values to two variables as



```
a=1; b=2
print(a, b)
```

Output:



```
1 2
```

The same can be done using the following statement:

```
a,b=1,2
print(a, b)
```

Output:

```
1 2
```

Note: Python does not have increment operator (++) and decrement operator (--) that are available in C and Java.

Unary Minus Operator

The unary minus operator is denoted by the symbol minus (-). When this operator is used before a variable, its value is negated. That means if the variable value is positive, it will be converted into negative and vice versa. For example, consider the following statements:

```
n = 10
print (-n)
num=-10
num =-num
print (num)
```

Output:

```
-10
10
```

Python Comparison Operators:

They are also known as relational operators and are used to compare two quantities. We can understand whether two values are the same or which one is bigger or which one is lesser, etc. For using these operators. These operators will result in True or False depending on the values compared.

Operator	Name	Example
==	Equals operator. If the value of the left operand is equal to the value of the right operand, it gives True else False.	a==b
!=	Not equals operator. If the value of the left operand is not equal to the value of right operand, it returns True else it returns False	a!=b
>	Greater than operator. If the value of left operand is greater than the value of right operand, it gives True else it gives	a>b
<	Less than operator. If the value of the left operand is less than the value of the right operand, it gives True else it gives False.	a<b
>=	Greater than or equal operator. If the value of the left operand is greater or equal than that of the right operand, it gives True else False.	a>=b
<=	Less than or equal operator. If the value of left operand is lesser or equal than that of right operand, it gives True else False.	a<=b

Relational operators are generally used to construct conditions in if statements. For Example,

```
a=1;b=2
if(a>b):
    print("YES")
else:
    print("NO")
```
,

Output:

```
NO
```

Observe the expression (a>b) written after if This is called a condition.When this condition becomes True, if statement will display 'Yes' and if it becomes False, then it will display 'No'. In this case, (1>2) is False and hence ‘No' will be displayed.

Relational operators can be chained. It means, a single expression can hold more than one relational operator. For example,

```
x=15
print(10<x<20)
```

Output:

```
True
```

Here, 10 is less than 15 is True, and then 15 is less than 20 is True. Since both the conditions are evaluated to True, the result will be True.

```
x=15
print(10>=x<20)
```

Output:

```
False
```

Here, 10 is greater than or equal to 15 is False. But 15 is less than 20 is True. Since we get False and True, the result will be False.

```
x=15
print(10<x>20)
```

Output:

```
False
```

Here, 10 is less than 15 is True. But 15 is greater than 20 is False. Since we are getting True and False, the total result will be False. So, the point is this: in the chain of relational operators, if we get all True, then only the final result will be True. If any comparison yields False, then we get False as the final result. Thus,

```
print(1<2<3<4)
```

Output:

```
True
```

```
print(1<2>3<4)
```

Output:

```
False
```

```
print(4>2>=2>1)
```

Output:

```
True
```

Logical Operators

Logical operators are useful to construct compound conditions. A compound condition is a combination of more than one simple condition. Each of the simple conditions is evaluated to True or False and then the decision is taken to know whether the total condition is True or False. We should keep in mind that in case of logical operators, False indicates 0 and True indicates any other number.

Operator	Description	Example	Result
AND	And operator. If x is False, it returns x, otherwise it returns y.	1 and 2	2
OR	Or operator. If x is False, it returns y, otherwise it returns x.	1 or 2	1

NOT	Not operator. If x is False, it returns True, otherwise True.	not(1)	False

Ex:

```
x= 100
y = 200
print(x and y)
print (x or y)
print (not x)
```

Output:

```
200
100
```

Ex:

```
x=1;y=2;z=3
if(x<y and y<z):
    print('Yes')
else:
    print('No')
```

Output:

```
False
Yes
```

In the above statement, observe the compound condition after if. That is x <y and y<z. This is a combination of two simple conditions, x<y and y<z. When 'and' is used, the total condition will become True only if both the conditions are True. Since both the conditions became True, we will get ‘Yes’ as output. Observe another statement below:

```
x= 1;y = 2;z=3
if(x>y or y<z):
    print("Yes")
else:
    print('No')
```

Output:

```
Yes
```

Here, x>y is False but y<z is True. When using 'or' if any one condition is True, it will take the

total compound condition as ‘True' and hence it will display ‘Yes'.

Boolean operators

Operator	Description	Example	Example
and	Boolean and operator. If both x and y are True, then it returns True, otherwise False.	x and y	False
Or	Boolean or operator. If either x or y is True, then it returns True, else False.	x or y	True
not	Boolean not operator. If x is True, it returns False, else True.	not x	False

We know that there are two bool' type literals. They are True and False. Boolean operators act upon "bool" type literals and they provide 'bool' type output. It means the result provided by Boolean operators will be again either True or False.Here, Let x=True, y=False.

Bitwise Operators:

These operators act on individual bits (0 and 1) of the operands. We can use bitwise operators directly on binary numbers or on integers also. When we use these operators on integers, these numbers are converted into bits (binary number system) and then bitwise operators act upon those bits. The results given by these operators are always in the form of integers.

We use the decimal number system in our daily life. This number system consists of 10 digits from 0 to 9. We count all numbers using these 10 digits only. But in the binary number system that is used by computers internally, there are only 2 digits, i.e. 0 and 1 which are called bits (binary digits). All values are represented only using these two bits. It is possible to convert a decimal number into a binary number and vice versa.

There are 6 types of bitwise operators.Here,let x =1, y=2

Operator	Name	Description	Example	Result
&	AND	Sets each bit to 1 if both bits are 1	x&y	0
|	OR	Sets each bit to 1 if one of two bits is 1	x|y	3
^	XOR	Sets each bit to 1 if only one of two bits is 1	x^y	3

~	NOT	Inverts all the bits	~x	-2
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x<<y	4
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x>>y	0

Membership operators:

The membership operators are useful to test for membership in a sequence such as strings, lists, tuples or dictionaries. For example, if an element is found in the sequence or not can be asserted using these operators. There are two membership operators as shown here:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	X in Y
Not in	Returns True if a sequence with the specified value is not present in the object	X not in Y

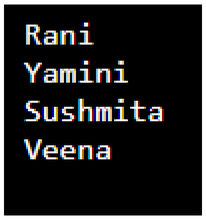
The in Operator:

This operator returns True if an element is found in the specified sequence. If the element is not found in the sequence, then it returns False.

Let's take a group of strings in a list. We want to display the members of the list using a for loop where the ‘in' operator is used. The list of names is given below:
Names= ["Rani", "Yamini", "Sushmita", "Veena"]
Here the list name is 'Names'. It contains a group of names. Suppose, we want to retrieve all the names from this list, we can use a for loop as:

```
Names= ["Rani", "Yamini", "Sushmita", "Veena"]
for name in Names:
    print(name)
```

Output:



```
Rani
Yamini
Sushmita
Veena
```

In the for loop, the variable 'name' is used. This variable will store each and every value of the list names'. It means, 'name' values will be "Rani", "Yamini", etc. The operator ‘in' will be a member of the list and hence, ‘in' will return True. When ‘True' is returned, the print() function in the for loop is executed and that name is displayed. In this way, all the names of the list are displayed.

The not in Operator:

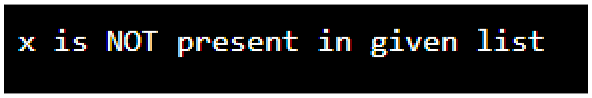
This works in reverse for the 'in' operator. This operator returns True if an element is not found in the sequence. If the element is found, then it returns False.

Ex:

```
x = 24
y = 20
list = [10, 20, 30, 40, 50 ];

if ( x not in list ):
   print("x is NOT present in given list")
else:
   print("x is  present in given list")
```

Output:



```
x is NOT present in given list
```

Identity Operators

These operators compare the memory locations of two objects. Hence, it is possible to know whether the two objects are the same or not. The memory location of an object can be seen using the id() function. This function returns an integer number, called the identity number that internally represents the memory location of the object. For example, id(a) gives the identity number of the object referred to by the name 'a'. See the following statements:

```
a=25
b=25
```

In the first statement, we are assigning the name (or identifier) 'a' to the object 25. In the second statement, we are assigning another name b' to the same object 25. In Python, everything is considered as an object. Here, 25 is the object for which two names are given. If we display an identity number of these two variables, we will get the same numbers. as they refer to the same object.

```
a=25
b=25
print(id(a))
print(id(b))
```

Output:

```
9756992
9756992
```

There are 2 identity operators.

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

The is operator:

The is operator is useful to compare whether two objects are the same or not. It will internally compare the identity number of the objects. If the identity numbers of the objects are the same, it will return True; otherwise, it returns False.

Ex:

```
a=25
b=25
if (a is b):
    print("Both of them have same identity")
else:
    print("Both of them have different identity")
```

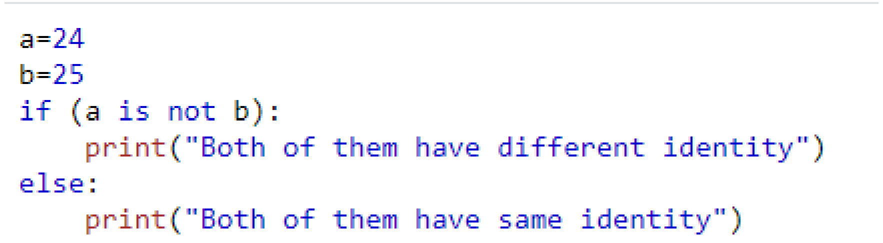
Output:

```
Both of them have same identity
```

The is not operator:

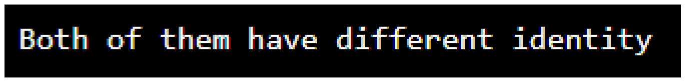
This is not operator returns True, if the identity numbers of two objects being compared are not the same. If they are the same, then it will return False. The ‘is' and 'is not' operators do not compare the values of the objects. They compare the identity numbers or memory locations of the objects. If we want to compare the value of the objects, we should use equality operator (==).

Ex:



```
a=24
b=25
if (a is not b):
    print("Both of them have different identity")
else:
    print("Both of them have same identity")
```

Output:



```
Both of them have different identity
```

Operator Precedence and Associativity

An expression or formula may contain several operators. In such a case, the programmer should know which operator is executed first and which one is executed next. The sequence of execution of the operators is called operator precedence. The following table summarizes the operators according to their precedence. The table shows precedence in descending order. It means the operators which are having highest precedence are listed at the top of the table.

Operator	Name
()	Parenthesis
**	Exponentiation
-,~	Unary minus, Bitwise complement
*,/,//,%	Multiplication, Division, Floor division, Modulus
+,-	Addition, Subtraction
<<,>>	Bitwise left shift, Bitwise right shift
&	Bitwise AND
^	Bitwise XOR

|	Bitwise OR
>,>=,<,<=,==,!=	Relational (comparison) operators
=,%=,/=,//=,-=, +=,*=,**=	Assignment operators
is,is not	Identity operators
in,not in	Membership operators
not	Logical not
or	Logical or
and	Logical and

Precedence represents the priority level of the operator. The precedence will be executed first than of lower precedence.