Operators in Python

What we will learn?

- Classification of Operators
- Operator Precedance and Associativity
- Mathematical Functions

Operator: It is a symbol that performs an operation.

Operand: The variables on which operator acts

Classification

On the basis of Number of Operands

1. **Unary Operator:** Operator acts on single variable
2. **Binary Operator:** Operator acts on two variables
3. **Ternary Operator:** Operator acts on three variables

On the basis of type of operation

1. Arithmetic Operators
2. Assignment Operators
3. Unary Minus Operators
4. Relational Operators
5. Logical Operators
6. Boolean Operators
7. Bitwise Operators
8. Membership Operators
9. Identity Operators

Arithmetic Operators

There are seven operators under this category

1. Addition (+) : Adds two values
2. Subtraction (-) : Subtracts one value from other
3. Multiplication (*) : Multiplies values on either side of the operator
4. Division (/) : Divides left operand by right operand

5. Modulus (%): Gives remainder of the division
6. Exponent (**): Calculate exponential value. a ** b gives the value of a to the power b
7. Floor Division (//) : Performs division and gives only integer quotient

Order of Evaluation is as follows

> Step 1: First parentheses are evaluated

> Step 2: Exponentiation is next

> Step 3: Multiplication, Division, Modulus and Floor division are at equal priority

> Step 4: Addition and subtraction done afterwards

> Step 5: Finally, assignment operation is performed

```
13 + 5
```

```
18
```

```
13 - 5
```

```
8
```

```
13 / 5
```

```
2.6
```

```
13 % 5
```

```
3
```

```
13 ** 5
```

```
371293
```

```
13 // 5
```

```
2
```

```
d = (1 + 2) * 3 ** 2 // 2 + 3
print(d)
```

```
16
```

Assignment Operators

These operators are useful to store the right side value into left side variable.

They can also be used to perform simple arithmetic operation and then store the result into variable

1. Assignment (=): Stores right side value into left side variable
2. Addition Assignment (+=): Adds right operand to the left operand and store the results into left operand
3. Subtraction Assignment (-=): Subtract right operand from left operand and stores the result into left operand
4. Multiplication Assignment (*=): Multiply right operand with left operand and stores the result into left operand
5. Division Assignment (/=): Divides left operand with right operand and stores the result into left operand
6. Modulus Assignment (%=): Divides left operand with right operand and stores the remainder into left operand
7. Exponential Assignment (**=): Performs power value and store the result into left operand
8. Floor Division Assignment (//=) : Performs floor division and then stores result into left operand

```
x = 20
y = 10
z = 5

z = x + y
print (z)

z += x
print("z=",z)

z -= x
print("z=",z)

z *= x
print("z=",z)

z /= x
print("z=",z)

z %= x
print("z=",z)

z = 17
```

```
z = 17
x = 3

z //= x
print("z=",z)

z **= x
print("z=",z)
```

```
30
z= 50
z= 30
z= 600
z= 30.0
z= 10.0
z= 5
z= 125
```

Multiple Assignment

There are different ways to assign values to variables

```
a = b = 1  # Assign same values to two variables in single statement
print("a=", a)
print("b=", b)
```

```
a= 1
b= 1
```

```
a = 1; b = 2  # Assign different values to two variables in single statement
print("a=", a)
print("b=", b)
```

```
a= 1
b= 2
```

```
a, b, c, d = 1, 2, 3, 4  # Another way of assigning different values to two variables in sing
print("a=", a)
print("b=", b)
print("c=", c)
print("d=", d)
```

```
a= 1
b= 2
c= 3
d= 4
```

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.

If the variable value is positive, it will be converted into negative and vice versa.

```
n = 10
print (-n)

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

```
     -10
     10
```

Relational Operator

Relational operators are used to compare two quantitites.

Using these operators it can be understood whether two values are same or whether one value is bigger or smaller than other

These operators will result in True or False depending on the values compared

1. Greater than (>): If the value of left operand is greater than the value of right operand, it gives True else it gives False.
2. Greater than or equal to (>=): If the value of left operand is greater or equal than that of right operand, it gives True else it gives False.
3. Less than (<): If the value of left operand is less than the value of right operand, it gives True else it gives False.
4. Less than or equal to (<=): If the value of left operand is lesser or equal than that of right operand, it gives True else it gives False.
5. Equals operator (==): If the value of left operand is equal to the value of right operand, it gives True else it gives False.
6. Not equals (!=): If the value of left operand is not equal to the value of right operand, it returns True else it returns False.

```
a = 1
b = 2

print( "Greater than:", a > b )
print( "Greater than or equal to:",a >= b )
print( "Less than:",a < b )
```

```
print( "Less than or equal to:",a <= b )
print( "Equal to:",a == b )
print( "Not equal to:",a != b )
```

```
Greater than: False
Greater than or equal to: False
Less than: True
Less than or equal to: True
Equal to: False
Not equal to: True
```

Relational Operators are generally used to construct conditions in if statement

```
a = 5
b = 7

if ( a > b ):
  print ("Yes")
  print('This is if')
else:
  print ("No")
  print('This is else')
```

```
No
This is else
```

Relational operators can be chained.

It means a single expression can hold more than one relational operator

In the chain of relational operators, if we get all True, then only the final result will be True.

If any comparison is false then overall result is also False

```
x = 15

print("Stmt 1:",10 < x < 20)
print("Stmt 2:",10 >= x < 20)
print("Stmt 3:",10 < x > 20)

print("Stmt 4:",1 < 2 < 3 < 4)

print("Stmt 5:",1 < 2 > 3 < 4)

print("Stmt 6:",4 > 2 >= 2 > 1)
```

```
Stmt 1: True
Stmt 2: False
```

```
Stmt 3: False
Stmt 4: True
Stmt 5: False
Stmt 6: True
```

Logical Operator

Logical operators are useful to construct compound conditions.

A compound condition is a combination of more than one simple condition.

Each of the simple condition is evaluated to True or False and then the decision is taken to know whether the total condition is True or False

In case of logical operators, False indicates 0 and True indicates any other number

1. and operator(and): x and y - If x is False, it returns x otherwise it returns y
2. or operator(or): x or y - If x is False, it returns y, otherwise it returns x
3. not operator(not): not x - If x is False, it returns True, otherwise False

```
x = 100
y = 200
z = 0
print (x and y)
print (x or y)
print (not x)
print (not z)

x = 1; y = 2; z = 3

if (x<y and y<z):    # Both relations are True
  print('Yes')
else:
  print('No')

if (x>y or y<z):    # One relation is false other is True
  print('Yes')
else:
  print('No')
```

```
200
100
False
True
Yes
Yes
```

Boolean Operator

Boolean operators act upon 'bool' type literals and they provide 'bool' type output.

The result provided by boolean operators will be either True or False.

1. Boolean and operator (and): x and y - If both x and y are True, then it returns True, otherwise False
2. Boolean or operator (or): x or y - If either x or y is True, then it returns True, else False
3. Boolean not operator (not): not x - If x is True, it returns False, else True

```
a = True
b = False

print ( "Stmt 1:",a and a)
print ( "Stmt 2:",a and b)
print ( "Stmt 3:",a or a)
print ( "Stmt 4:",a or b)
print ( "Stmt 5:",b or b)
print ( "Stmt 6:",not a)
print ( "Stmt 7:",not b)
```

```
Stmt 1: True
Stmt 2: False
Stmt 3: True
Stmt 4: True
Stmt 5: False
Stmt 6: False
Stmt 7: True
```

Bitwise Operators

These operators act on individual bits (0 and 1) of the operand.

Bitwise operators operates directly on binary numbers or on integers.

While operating on integers, these numbers are converted into bits and then bitwise operators act upon those bits.

The results are always given in the form of integers

There are six types of bitwise operators:

1. Bitwise Complement (~)
2. Bitwise AND (&)
3. Bitwise OR (|)

4. Bitwise XOR (^)
5. Bitwise Left Shift (<<)
6. Bitwise Right Shift (>>)

1. Bitwise Complement Operator (~)

It gives the complement form of a given number.

The symbol is pronounced as tilde

Complement form of a positive number can be obtained by changing 0's as 1's and vice versa

Example:

If x = 10 find ~x

x = 10 = 0000 1010

~x = -11 = 1111 0101

The above answer is -11 in decimal

---

2. Bitwise AND Operator (&)

This operator performs AND operation on the individual bits of numbers

The symbol is pronounced as ampersand

Example:

If x = 10 and y = 11 then find x & y

x = 10 = 0000 1010

y = 11 = 0000 1011

x & y = 0000 1010

The above answer is 10 in decimal

---

3. Bitwise OR Operator (|)

This operator performs OR operation on the individual bits of numbers

The symbol is pronounced as pipe

Example:

If x = 10 and y = 11 then find x | y

x = 10 = 0000 1010

y = 11 = 0000 1011

x | y = 0000 1011

The above answer is 11 in decimal

---

4. Bitwise XOR Operator (^)

This operator performs exclusive OR operation on the individual bits of numbers

The symbol is pronounced as cap or carat

Example:

If x = 10 and y = 11 then find x ^ y

x = 10 = 0000 1010

y = 11 = 0000 1011

x ^ y = 0000 0001

The above answer is 1 in decimal

---

5. Bitwise Left Shift Operator (<<)

This operator shifts the bits of the number towards left with a specified number of positions

The symbol is read as double less than

x << n : To shift the bits of x towards left n positions. Rightmost n bits will be padded with '0's

Example:

If x = 10 perform x << 2

x = 0000 1010

x << 2 = 0010 1000

The above answer is 40 in decimal

---

5. Bitwise Right Shift Operator (>>)

This operator shifts the bits of the number towards right with a specified number of positions

The symbol is read as double greater than

x >> n : To shift the bits of x towards right n positions. Leftmost n bits will be padded with '0's

Example:

If x = 10 perform x >> 2

x = 0000 1010

x >> 2 = 0000 0010

The above answer is 2 in decimal

```
x = 10
y = 11

print("Bitwise Not:",~x)
print("Bitwise AND:", x & y)
print("Bitwise OR:", x | y)
print("Bitwise XOR:", x ^ y)
print("Bitwise Left Shift:", x << 2)
print("Bitwise Right Shift:", x >> 2)
```

```
Bitwise Not: -11
Bitwise AND: 10
Bitwise OR: 11
Bitwise XOR: 1
Bitwise Left Shift: 40
Bitwise Right Shift: 2
```

Membership Operators

The membership operators are useful to test for membership in a sequence such as strings, lists, tuples or dictionaries

If an element is found in the sequence or not can be asserted using these operators.

There are two membership operators are

1. in operator: It returns 'True' if an element is found in the specified sequence. If not found then it returns 'False'

2. not in operator: It returns 'True' if an element is not found in the sequence otherwise it returns 'False'

```
names = ['James', 'Vivek', 'Advait', 'Sunny']

for k in names:          # To display the members of the list using for loop
  print (k)

# In above example the variable k holds the each and every value of list 'names'
# The operator in will check whether each of these values is a member of list or not
```

```
James
Vivek
Advait
Sunny
```

```
postal = {'Delhi':110001, 'Chennai':600001, 'Kolkata':700001, 'Bengluru':560001}

for city in postal:
  print(city, postal[city])
```

```
Delhi 110001
Chennai 600001
Kolkata 700001
Bengluru 560001
```

```
x='Hello World'
y={1:'a',2:'b'}

print('H' in x)
print('hello' not in x)
print(1 in y)
print('a' in y)
print('b' not in y)
```

```
True
True
True
False
True
```

Identity Operators

These operators compare memory locations of two objects.

It is possible to know whether the two objects are same not

The memory location of an object can be seen using **id( ) function**

The 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 an object referred by name(a)

```
a = 25
b = 25

print ("a and b holds same value")
print(id(a))
print(id(b))

b = 30

print ("The value of b has been modified")
print(id(a))
print(id(b))
```

```
a and b holds same value
10915264
```

```
10915264
The value of b has been modified
10915264
10915424
```

There are two identity operators:

1. is operator:

It is useful to compare whether two objects are same or not.

It will internally compare the identity number of the object.

If the identity number of the objects are same then it will return True otherwise it will return False

2. is not operator:

It returns True if the identity of the two objects being compared are not same.

If they are same then it will return False

Please NOTE:

Above operators compare the identity numbers or memory locations of the object.

If we want to compare the values then we should use equality operator (==)

```
a = 100
b = 100

if ( a is b):
  print("a and b have same identity")
else:
  print("a and b do not have same identity")
```

```
a and b have same identity
```

```
lst1 = [1, 2, 3, 4]
lst2 = [1, 2, 3, 4]

if (lst1 is lst2):
  print("lst1 and lst2 are internally same")
else:
  print("lst1 and lst2 are internally different")

print("id of lst1",id(lst1))
print("id of lst2",id(lst2))

if (lst1 == lst2):
  print("lst1 and lst2 are same in values")
else:
```

```
print("lst1 and lst2 are different in values")
```

```
lst1 and lst2 are internally different
id of lst1 139826450633928
id of lst2 139826450633864
lst1 and lst2 are same in values
```

Operator Precedance and Associativity

Operator Precedance:

The sequence of execution of the operators

Precedance level represents the priority level of the operator

The operator with higher precedance will be executed first than that of lower precedance

The following list shows the operator precedance in Python. The number 1 has highest precedance.

1. Parenthesis ( )
2. Exponentiation **
3. Unary Minus, Bitwise Complement -, ~
4. Multiplication, Division, Floor Division, Modulus *, /, //, %
5. Addition, Subtraction +, -
6. Bitwise Left Shift, Bitwise Right Shift <<, >>
7. Bitwise AND &
8. Bitwise XOR ^
9. Bitwise OR |
10. Relational Operators >, >=, <, <=, ==, !=
11. Assignment Operators =, %=, /=, //=, -=, +=, =, *=
12. Identity Operators is, is not
13. Membership Operators in, not in
14. Logical not operator: not
15. Logical or operator: or
16. Logical and operator: and

Associativity

It is the order in which an expression is evaluated that has multiple operators with the same precedance.

The associativity can be from Left-to-Right or Right-to-Left.

In python, almost all the operators have left-to-right associativity

---

Example:

value = 3 / 2 * 4 + 3 + **(10 / 4)** ** 3 - 2

value = 3 / 2 * 4 + 3 + **2.5 ** 3** - 2

value = **3 / 2** * 4 + 3 + 15.625 - 2

value = **1.5 * 4** + 3 + 15.625 - 2

value = **6.0 + 3** + 15.625 - 2

value = **9.0 + 15.625** - 2

value = **24.625 - 2**

value = 22.625

Mathematical Function

Built in functions can be used to perform various advanced operations

In python, module is a file that contains a group of useful objects like functions, classes or variables.

'math' is a module that contains several functions to perform mathematical operations.

A module has to be imported in the program before its use in Python Program.

A module is imported using keyword 'import'

It can be used in three different ways

1. Use name of the module to call the function

Example:

To import math module and use its square root function

Here function is called using module name

```
import math
x = math.sqrt(16)
```

2. Use proxy name for the module and then refer it to call fuctions

Example:

To import math module and use its square root function

Here function is called using dummy name m for module math

```
import math as m
x = m.sqrt(16)
```

3. import only required functions and then use it without referring module name

Example:

To import math module and use its square root function

Here only sqrt function is imported from math module

```
from math import sqrt
x = sqrt(16)
```

Important Mathematical Functions

Some of the important mathematical functions are listed below:

1. ceil(x): Raises value to next higher integer. If x is integer then same value will be returned.
2. floor(x): Decreases x value to the previous integer value. If x is integer then same value will be returned.
3. degrees(x): converts angle value x from radians into degrees
4. radians(x): converts x value from degrees into radians
5. sin(x): returns sine value of x. Here x value is given in radians
6. cos(x): returns cosine value of x. Here x value is given in radians
7. tan(x): returns tangent value of x. Here x value is given in radians
8. exp(x): returns exponentiation of x. This is similar to e ** x
9. fab(x): Gives positive value or absolute value of x.
10. factorial(x): returns factorial value of x. Raises 'Value Error' if x is not integer or x is negative

Note: These functions cannot be used with complex numbers. For complex numbers operations a separate module 'cmath' is available

Constant in Math Module

Some of the constants in Math Module

1. pi: The mathematical constant Pi = 3.141592...

2. e: The mathematical constant e = 2.718281
3. inf: A floating point positive infinity
4. nan: 'Not a Number' (NaN) value

```
import math
r = 15.5
area = math.pi * r ** 2
print("Area of Circle:", area)
```

```
Area of Circle: 754.7676350249478
```