Python Operators

- **What is Operator?**
- **What is Operand?**
- **Arithmetic Operators**
- **Comparison OperatorsS**
- **Python Assignment Operators**
- **Logical Operators**
- **Membership Operators**
- **Identity Operators**

What is Operator?

- Python Operators in general are used to perform operations on values and variables.
- These are standard symbols used for the purpose of logical and arithmetic operations.
- Operators: Are the special symbols.

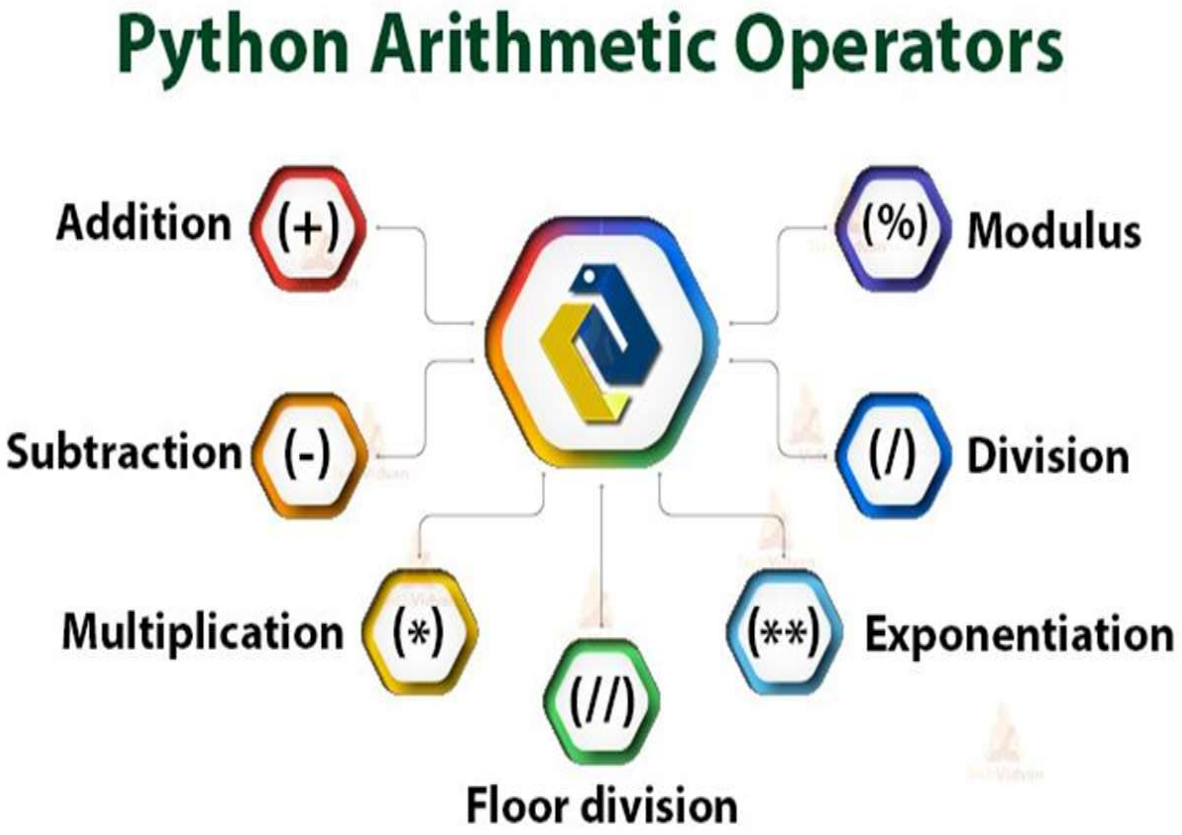
Eg- + , * , /, etc

What is Operand?

The operands in python, one of the programming languages , refer to the values on which the operator operates.

Arithmetic Operators :

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etc.



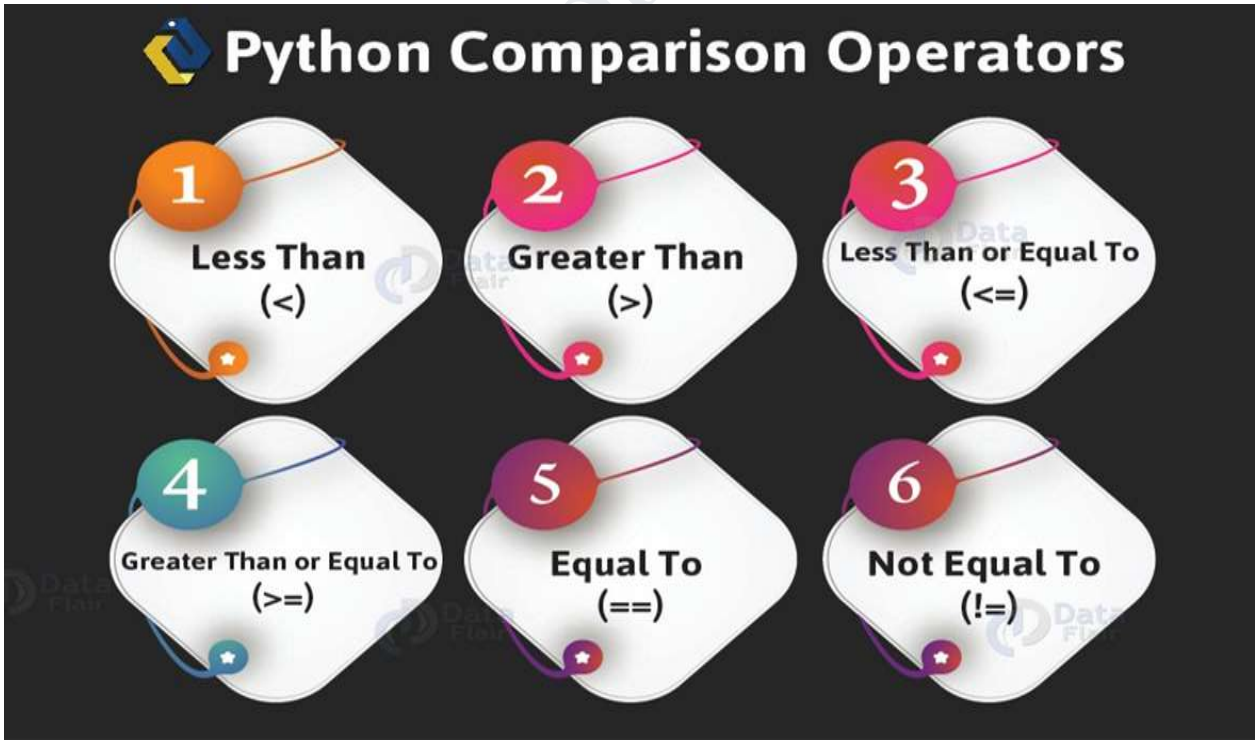
Operator	Meaning	Example
+	Add two operands or unary plus	x + y
-	Subtract right operand from the left or unary minus	x – y
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x/y
%	Modulus - remainder of the division of left operand by the right	x%y= (remainder of y/x)

//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

Comparison Operators:

In programming, you often want to compare a value with another value. To do that, you use comparison operators. It returns either true or false according to the condition.

Python has six comparison operators, which are as follows:



Operator	Meaning	Example
<	Less than - True if left operand is less than the right	x < y
>	Greater than - True if left operand is greater than the right	x > y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y

Python Assignment Operators

- Assignment operators are used in Python to assign values to variables.
- a=5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.
- There are various compound operators in Python like a +=5 that adds to the variable and later assigns the same. It is equivalent to a=a+5.

Operator	Meaning	Example
=	Assign value of right side of expression to left side operand	x = y + z
+=	Add and Assign: Add right side operand with left side operand and then assign to left operand	a += b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand: True if both operands are equal	a -= b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a *= b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b
%=	Modulus AND: Takes modulus using left and right operands and assign result to left operand	a%=b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a //= b
**=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	a **= b
&=	Performs Bitwise AND on operands and assign value to left operand	a &= b
|=	Performs Bitwise OR on operands and assign value to left operand	a |= b
^=	Performs Bitwise XOR on operands and assign value to left operand	a ^= b
>>=	Performs Bitwise right shift on operands and assign value to left operand	a >>= b
<<=	Performs Bitwise left shift on operands and assign value to left operand	a <<= b

Logical Operators

In Python, Logical operators are used on conditional statements (either True or False).

They perform Logical AND, Logical OR and Logical NOT operations.

Operators	Description
&& (Logical AND)	This operator returns true if all relational statements combined with && are true, else it returns false.
|| (Logical OR)	This operator returns true if at least one of the relational statements combined with || is true, else it returns false.
! (logical NOT)	It returns the inverse of the statement's result.
XOR(Logical XOR operator)	This operator returns true if either statement one is true or statement two is true but not both.
and (Logical AND)	This operator returns true if all relational statements combined with it are true, else it returns false.
or (Logical OR)	This operator returns true if atleast one of the relational statements combined with it is true, else it returns false.

Membership Operators

Membership operators are used to test if a sequence is presented in an object:

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

Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually 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

Control Statements

- **Conditional control statements**
- **If**
- **If-else**
- **If-elif-else**
- **Nested-if**
- **Looping statements**
- **for**
- **while**
- **Nested loops**
- **Break**
- **Continue**
- **Pass**
- **Return**

Conditional control statements

In programming languages, most of the time in large projects we have to control the flow of execution of our program and we want to execute some set of statements only if the given condition is satisfied, and a different set of statements when it's not satisfied.

Conditional statements are also known as decision-making statements. We need to use these conditional statements to execute the specific block of code if the given condition is true or false.

In Python we can achieve decision making by using the following statements:

if statements
if-else statements
elif statements
Nested if and if-else statements

elif ladder

If

Python if statement is one of the most commonly used conditional statements in programming languages. It decides whether certain statements need to be executed or not.

It checks for a given condition, if the condition is true, then the set of code present inside the ” if ” block will be executed otherwise not.

The if condition evaluates a Boolean expression and executes the block of code only when the Boolean expression becomes TRUE.

Syntax:

```
if (EXPRESSION == TRUE):
        Block of code
Else:
        Block of code
```


Here, the condition will be evaluated to a Boolean expression (true or false).

If the condition is true, then the statement or program present inside the ” if ” block will be executed and if the condition is false, then the statements or program present inside the “else” block will be executed.

If-else

The statement itself says if a given condition is true then execute the statements present inside the “if block” and if the condition is false then execute the “else” block.

The “else” block will execute only when the condition becomes false. It is the block where you will perform some actions when the condition is not true.

if-else statement evaluates the Boolean expression. If the condition is TRUE then, the code present in the “ if “ block will be executed otherwise the code of the “else“ block will be executed

Syntax :

```
if (EXPRESSION == TRUE):
        Statement (Body of the block)
else:
        Statement (Body of the block)
```


Here, the condition will be evaluated to a Boolean expression (true or false).

If the condition is true then the statements or program present inside the “if” block will be executed and if the condition is false then the statements or program present inside the “else” block will be executed.

If-elif-else

In Python, we have one more conditional statement called “elif” statements.

“elif” statement is used to check multiple conditions only if the given condition is false.

It’s similar to an “if-else” statement and the only difference is that in “else” we will not check the condition but in “elif” we will check the condition.

“elif” statements are similar to “if-else” statements but “elif” statements evaluate multiple conditions.

Syntax:

```
if (condition):
        #Set of statement to execute if condition is true
elif (condition):
                #Set of statements to be executed when if        condition is false and elif condition is true
else:
        #Set of statement to be executed when both if and elif conditions are false
```

Nested if-else

Nested “if-else” statements mean that an “if” statement or “if-else” statement is present inside another if or if-else block.

Python provides this feature as well, this in turn will help us to check multiple conditions in a given program.

An “if” statement is present inside another “if” statement which is present inside another “if” statements and so on.

Syntax:

```
if  boolean_expression:
        if boolean_expression:
```

```
    statement(s)
    else:
      statement(s)
else:
    statement(s)
```

Looping statements

The following loops are available in Python to fulfil the looping needs. Python offers 3 choices for running the loops. The basic functionality of all the techniques is the same, although the syntax and the amount of time required for checking the condition differ.

We can run a single statement or set of statements repeatedly using a loop command.

Name of the loop	Loop Type & Description
While loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
For loop	This type of loop executes a code block multiple times and abbreviates the code that manages the loop variable.
Nested loops	We can iterate a loop inside another loop.

for

It has the ability to iterate over the items of any sequence, such as a list or a string

Syntax:

```
for iterating_var in sequence:
    statement(s)
```

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable iterating_var.

Next, the statements block is executed. Each item in the list is assigned to iterating_var, and the statement(s) block is executed until the entire sequence is exhausted.

while

A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

The syntax of a while loop in Python programming language is -

while expression:

statements(s)

- Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non-zero value. The loop iterates while the condition is true.
- When the condition becomes false, program control passes to the line immediately following the loop.
- In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

Nested loops

Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax:

for iterating_var in sequence:

for iterating_var in sequence:

```
        statement(s)

  statement(s)
```

The syntax for a **nested while loop** statement in Python programming language is as follows -

```
while expression:
        while expression:
    statement(s)
 statement(s)
```

A final note on loop nesting is that you can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

Break

The break statement is used to break out of a loop.It is used inside for and while loops to alter the normal behavior. the break will end the loop it is in and control flows to the statement immediately below the loop.

Syntax:

```
Loop{

Condition:

        break

}
```

Example:

```
# break Statement Example
for i in range(1, 10):
if i == 4:
break
print(i)
```

Note: One to four numbers will execute in the above program

Continue

The continue keyword is used to end the current iteration in a for loop or a while loop and continues to the next iteration.

The continue statement is a temporary break in a loop.

Syntax

```
continue
```

Example:

```
# continue Statement Example
for i in range(1, 10):
if i == 4:
continue
print(i)
```

Note: One to 10 numbers will execute and except 4th number in the above program

Pass

The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

The pass statement is a *null* operation; nothing happens when it executes.

The pass is also useful in places where your code will eventually go, but has not been written yet.

Syntax:

```
pass
```


Example:

```
for letter in 'Python':
   if letter == 'h':
        pass
     print('This is pass block')
  print ('Current Letter :', letter)
print( "Good bye!")
```