

# Chapter 1: Python Programming Basics

**Introduction to Python:** Python is a high-level programming language that contains features of functional programming language like C and object-oriented language like Java. Python was developed in 1991 by **Guido Van Rossum** of Netherland. Python is considered as an interpreted language since Python programs are executed by an interpreter. Python is a general-purpose language. It has wide range of applications from Web development, scientific and mathematical computing to desktop graphical user Interfaces. We also learn that Python shell can be used in two ways, viz., interactive mode and script mode.

Python wasn't named after a dangerous snake. Rossum was fan of a comedy series "Monty Python's Flying Circus". The name "Python" was adopted from the same series.

A simple program to add two numbers in Java is:

```
class Add
{
public static void main (String args [])
{
    int a,b;
    a=b=10;
    System.out.println("sum="+a+b);
}
}
```

The same program to add two numbers in C as:

```
#include<stdio.h>
void main()
{
    int a,b;
    a=b=10;
    printf("sum=%d",a+b);
}
```

When they want to develop functional aspects. Then to use C style coding and when they are in need of going for classes and objects, they will use Java style coding. Answer for their requirement is Python!

In python, the program to add two numbers will be as follows:

```
a=b=10
print("sum=",a+b)
```

**Features of python:** The following are some of the important features of python:

**1. Simple Easy to learn:** Python is a simple programming language. When we need a python program, we feel like reading English sentences. Python programs use very simple structure hence, developing and understanding programs will become easy.

**2. Open source:** There is no need to pay for python software. Python can be freely downloaded from [WWW.python.org](http://WWW.python.org) website. Its source code can be read, modified and can be used in programs as desired by the programmers.

**3. High level language:** Since python uses English words in its programs and hence it is called high level programming language.

**4. Dynamically typed:** In python, we need not declare anything. Statements binds a name to an object and the object can be of any type.

**5. Platform independent:** python programs are not dependent on any specific operating system since python virtual machine (PVM), runs the byte code instructions on any computer system. Hence python runs on almost all operating systems like UNIX, Linux, Windows etc...

**6. Portable:** When a program yields the same result on any computer in the world, then it is called a portable program.

**7. Procedure and object oriented:** Python is a procedure oriented as well as an object-oriented programming language.

**8. Extensible:** The programs written in C or C++ can be integrated into python and executed using PVM.

**9. Scripting language:** Python is considered as a scripting language like PHP as it is interpreted and it is used on the internet to support other software.

**10. Database connectivity:** python provides interfaces to connect its programs to all major databases like Oracle, Sybase or MySQL.

**11. Scalable:** A program would be scalable if it could be moved to another operating system or hardware and take full advantage of the new environment in terms of performance. Python Programs are scalable since they can run on any platform and use the features of the new platform effectively.

**12. Batteries included:** Python contains several small applications or small packages, which are already developed available to programmers. These small packages are called 'batteries included'.

\* CherryPy is an object-oriented HTTP framework.

\* Cryptography offers cryptographic techniques for the programmers.

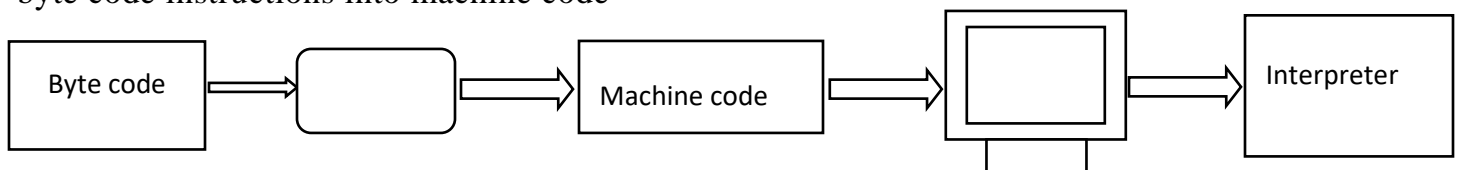
\* mysql-connector-python is a driver written in python to connect MySQL database.

\* numPy is a package for processing arrays of single or multidimensional type.

\* sciPy is the scientific library to do scientific and engineering calculations.

### Python Virtual Machine (PVM)

A python compiler converts the program source code into another code, called byte code. The size of each byte code instruction is 1 byte and hence these are called byte code instructions. The byte code instructions found in the .pyc file. This figure shows the role of virtual machine in converting byte code instructions into machine code



**Figure: The python virtual machine**

The role of PVM is to convert the byte code instructions into machine code so that the computer can execute those machine code instructions and display the final output. The interpreter converts the byte code into machine code and sends that machine code to the computer processor for execution. Since interpreter is playing the main role, often the python virtual machine is also called an interpreter.

### Differences between C and Python

C	Python
C is procedure-oriented language. It does not contain the features like classes, objects, inheritance, etc.	Python is object-oriented language. It contains features like classes objects, inheritance, polymorphism, etc.
C programs execute faster.	Python programs are slower compared to C.
It is compulsory to declare the datatypes	Type declaration is not required in python.
C language type discipline is static and weak.	Python type discipline is dynamic and strong.
Pointers concept is available in C.	Python does not use pointers.
C does not have exception handling facility. Hence C programs are weak.	Python handles exceptions and hence python programs are robust.
C has do while, while and for loops.	Python has while and for loops.
C has switch statement.	Python does not have switch statement.
The variable in for loop does not increment automatically.	The variable in the for loop increments automatically.
The allocation and deallocation of memory is done using malloc(), calloc(), realloc() or free() functions.	Memory allocation and deallocation is done automatically by PVM.
C does not contain a garbage collector.	Automatically garbage collector is available in python.
C supports single and multi-dimensional arrays.	Python supports only single dimensional arrays. To work with multidimensional arrays, we use the applications like numPy.
The array index should be positive integer.	Array index can be positive or negative integer number.
Indentation of statements is not necessary in C	Indentation is required to represent a block of statements.
A semicolon is used to terminate the statements and comma is used to separate expressions.	New line indicates end of the statements and semicolon is used to separate expressions.

## Differences between Java and Python

Java	Python
Java is object-oriented programming language.	Python blends the functional programming with oop features.
Java programs are verbose. It means they contain more number of lines.	Python programs are concise and compact. A big program can be written using very less number of lines.
It is compulsory to declare the data types of variables, arrays etc	Type declaration is not required in python.
Java language type discipline is static and weak.	Python type discipline is dynamic and strong.
Java has do while, while and for loops	Python has while and for loops.
Java has switch statement.	Python does not have switch statement.
The variable in for loop does not increment automatically.	The variable in the for loop increments automatically.
Memory allocation and deal location is done automatically by JVM.	Memory allocation and deal location is done automatically by PVM.
Java supports single and multidimensional arrays.	Python supports only single dimensional arrays. Multidimensional arrays use numpy.
The array index should be a positive integer.	Array index can be positive or negative integer number.
Indentation of statements is not necessary in java.	Indentation is required to represent a block of statement.
A semicolon is used to terminate the statements and comma is used to separate expressions	New line indicates end of the statements and semicolon is used as an expression separator.
In java, the collection objects like stack, linked list or vector store only objects but not primitive datatypes.	Python collection objects like lists and dictionaries can store objects of any type, including numbers and lists.

**Structure of a Python Program:** Python is a flexible and dynamic language, executed by command line window or the IDLE Development environment. Python shell can be used in two ways, viz., interactive mode and script mode. In interactive mode, it interacts with OS. That means, when we type Python expression / statement / command after the prompt (>>>), the Python immediately responses with the output.

e.g.: when we type print "WELCOME" after the prompt.

```
>>>print "WELCOME "
```

```
o/p: WELCOME
```

```
e.g 2: >>> x=10
      >>> y=20
      >>> print x*y
      200
```

In script mode, we type Python program in a file and then use interpreter to execute the content of the file.

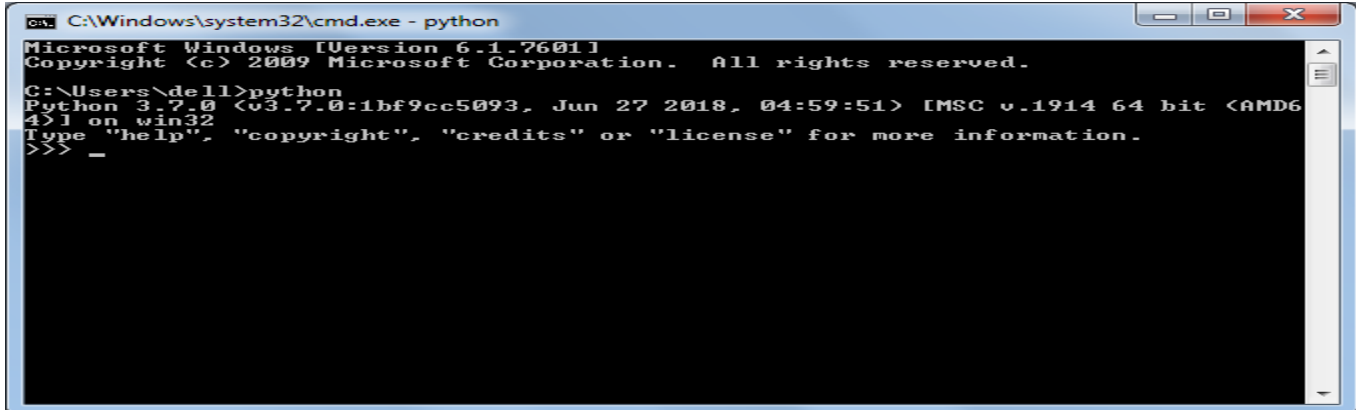
**Indentation:** Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation. A code block (body of function, loop etc.) starts with indentation & ends with the first unindented line. The amount of indentation is **up to you**, but it must be consistent throughout that block. Generally, four white spaces are used for indentation and is preferred over tabs.

```
e.g.: for i in range(1,11):
      print(i)
      if i == 5:
          break
```

**Python Shell:** Python is an interpreter language. It means it executes the code line by line. Python provides a Python Shell (also known as Python Interactive Shell) which is used to execute a single Python command and get the result.

Python Shell waits for the input command from the user. As soon as the user enters the command, it executes it and displays the result.

To open the Python Shell on Windows, open the command prompt, write python and press **enter**.



A Python prompt comprising of three greater than symbols (>>>) appear. Now, we can enter a single statement to get the result.

e.g.: enter an expression like `3 + 2`, press enter & it will display the result in the next line

**Python identifiers:** A Python identifier is a name used to identify a variable, function, class etc. An identifier consists of alphabets, underscore (\_) and digits (0 to 9). An identifier starts with a letter A to Z or a to z, or an underscore. Python does not allow any special symbol such as @, \$, and % within identifiers.

**The naming conventions for Python identifiers:**

- The class names start with an uppercase letter.
- All other identifiers start with a lower-case letter.

- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.

**Keywords in Python:** Keywords are the reserved words in python. We cannot use a keyword as variable name, function name or any other identifier. Here's a list of all keywords in Python programming

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

These keywords may get altered in different versions of Python. Some extra might get added or some might be removed. We can always get the list of keywords in the current version by typing the following in the prompt.

```
>>> import keyword
>>> print(keyword.kwlist)
```

**Comments in python:** There are two types of comments in python: single line comments and multiline comments.

**Single line comments:** Comments are non-executable statements. These comments start with a hash symbol (#) and are useful to mention that the entire line till end should be treated as comment. PVM will execute these comments,

e.g.: #to find sum of two numbers  
a=10 # store 10into variable a

**Multi line comments:** In multiline comments instead of starting every line with # symbol, we can write the previous block of code inside “ “ “ (triple double quotes) in the beginning and ending of the block as:” ” ”

## Data types in python

A data type represents the type of data stored into a variable or memory. The data types, which are already available in python language, are called Built-in data types. The data types, which can be created by the programmers, are called User-defined data types.

**Built-in data types:** The built-in data types are of five types:

- 1) None Type
- 2) Numeric Type
- 3) Sequences
- 4) Sets
- 5) Mapping

**The None type:** In python, the 'None' data type represents an object that does not contain any value. In language like java, it is called 'null' object. In a python program, maximum of only one 'None' object is provided. One of the uses of 'None' is that it is used inside a function as a default value of the arguments.

**Numeric type:** The numeric types represent numbers. Python numbers are of 4 types

- int
- float
- complex
- bool

Number data type stores Numerical Values. This data type is immutable i.e. value of its object cannot be changed.

Integer & Long to store whole numbers i.e. decimal digits without fraction part.

Float/floating point to store numbers with fraction part.

Complex to store real and imaginary part.

Bool to store true or false of an expression.

### Conversion of data types

Depending on the type of data, Python internally assumes the data type for the variable. But sometimes, the programmers want to convert one data type into another type explicitly. This is called type conversion or coercion. This is possible by mentioning the data type with parameters.

e.g.: To convert a number into integer type, we can write `int(num)`.

- `int(x)` is used to convert the number x into int type.  
`x=15.56`  
`int(x)` will display 15
- `float(x)` is used to convert x into float type.  
`num=15`  
`float(num)` will display 5.0
- `complex(x)` is used to convert x into a complex number with real part x and imaginary part zero.  
`n=10`  
`complex(n)` will display (10+0i)
- `complex(x,y)` is used to convert x and y into a complex number such that x will be the real part and y will be the imaginary part.  
`a=10`  
`b=-5`  
`complex(a,b)` will display (10-5i)

### Sequences in python

A sequence is an ordered collection of items, indexed by positive integers. It is a combination of mutable and immutable data types. There are three types of sequence data type available in Python; they are Strings, Lists & Tuples.

Generally, a sequence represents a group of items. There are 6 types of sequences

- |              |         |
|--------------|---------|
| * Str        | * list  |
| * Bytes      | * tuple |
| * Byte array | * range |

### str data type:

**In python**, str represents string data type. **String** is an ordered sequence of characters. They are enclosed in single quotes ( ' ') or double quotes ( " "). The quotes are not part of string. They only tell the computer about where the string constant begins and ends. They can have any character or sign, including space in them. These are **immutable**. A string with length 1 represents a character in Python.

In addition, we can use a triple quoted string that begins and ends with three quote characters (either three single quotes or three double quotes).

A string with length 1 represents a character in Python. Strings are represented by the immutable str data type which holds a sequence of Unicode characters. The str data type can be called as a function to create string objects with no arguments it returns an empty string, with a non-string argument it returns the string form of the argument, and with a string argument it returns a copy of the string.

### String Methods

Strings are the objects of the str class. The input function returns a string from the keyboard and the print function displays a string on the monitor.

**Creating Strings:** Strings are created by using the constructor, as follows:

```
s1 = str() # Create an empty string object
```

```
s2 = str("Welcome") # Create a string object for Welcome
```

Python provides a simple syntax for creating a string object by using a string value.

e.g.: `s1 = " "` # Same as `s1 = str()`

```
s2 = "Welcome" # Same as s2 = str("Welcome")
```

### Python's built-in functions

**Len()** function: It return the number of the characters in a string.

1. `>>> s = "Welcome"`
2. `>>> L=len(s)`
3. `>>> print(L)`
4. 7

**Index Operator [ ] :** A string is a sequence of characters. A character in the string can be accessed through the index operator using the syntax: `s [index]`

The indexes are 0 based; that is, they range from 0 to `len(s)-1`,

1. `>>> s = "Welcome"`
2. `>>> for i in range(0, len(s), 2):`
3. `... print(s[i], end = ' ')`
4. Wloe



**The Slicing Operator [start : end] :** The slicing operator returns a slice of the string using the syntax s[start : end]. The slice is a substring from index start to index end – 1.

1. >>> s = "Welcome"
2. >>> s[1 : 4]
3. 'elc'

s[1 : 4] returns a substring from index 1 to index 3.

The starting index or ending index may be omitted. In this case, by default the starting index is 0 and the ending index is the last index. For example,

1. >>> s = "Welcome"
2. >>> s[ : 6]
3. 'Welcom'
4. >>> s[4 : ]
5. 'ome'
6. >>> s[1 : -1]
7. 'elcom'
8. >>>

**The Concatenation (+) and Repetition (\*) Operators:** Two strings can be joined or concatenated, by using the concatenation operator (+). Also the repetition operator (\*) is used to concatenate the same string multiple times.

1. >>> s1 = "Welcome"
2. >>> s2 = "Python"
3. >>> s3 = s1 + " to " + s2
4. >>> s3
5. 'Welcome to Python'
6. >>> s4 = 3 \* s1
7. >>> s4
8. 'WelcomeWelcomeWelcome'
9. >>> s5 = s1 \* 3
10. >>> s5
11. 'WelcomeWelcomeWelcome'
12. >>>

## String methods

Syntax	Description
s.capitalize()	Returns a copy of str s with the first letter capitalized;
s.count(t, start, end)	Returns the number of occurrences of str t in str s (or in the start:end slice of s)
s.find(t, start, end)	Returns the leftmost position of t in s (or in the start:endslice of s) or -1 if not found.

s.isalnum()	Returns True if s is nonempty and every character in s is alphanumeric
s.isalpha()	Returns True if s is nonempty and every character in s is alphabetic
s.isdecimal()	Returns True if s is nonempty and every character in s is a Unicode base 10 digit
s.isdigit()	Returns True if s is nonempty and every character in s is an ASCII digit
s.isidentifier()	Returns True if s is nonempty and is a valid identifier
s.islower()	Returns True if s has at least one lower case character and all its lower case characters are lowercase;
s.isnumeric()	Returns True if s is nonempty and every character in s is a numeric Unicode character such as a digit or fraction
s.isupper()	Returns True if str s has at least one uppercaseable character and all its uppercaseable characters are uppercase;
s.lower()	Returns a lowercased copy of s; see also str.upper()
s.replace(t,u, n)	Returns a copy of s with every (or a maximum of n if given) occurrences of str t replaced with str u
s.swapcase()	Returns a copy of s with uppercase characters lowercased and lowercase characters uppercased.
s.title()	Returns a copy of s where the first letter of each word is uppercased and all other letters are lowercased;
s.upper()	Returns an uppercased copy of s.
s.center(width)	Returns a copy of this string centered in a field of the given width.
s.ljust(width)	Returns a string left justified in a field of the given width.
s.rjust(width)	Returns a string right justified in a field of the given width.
s.format(items)	Formats a string.

## String Representation

P	Y	T	H	O	N	
0	1	2	3	4	5	6
-6	-5	-4	-3	-2	-1	Reverse

## Bytes and byte array

Bytes and byte array objects contain single bytes. Bytes is immutable while byte array is a mutable sequence. Bytes objects can be constructed by using constructor, bytes(), and from literals. To construct byte arrays, the bytearray() function is used.

**bytes() function:** The **bytes() function** returns a new "bytes" object, which is an immutable sequence of small integers in the range  $0 \leq x < 256$ , print as ASCII characters when displayed. bytes are an

immutable version of byte array – it has the same non-mutating methods and the same indexing and slicing behavior.

**Syntax:** bytes([source[, encoding[, errors]]])

### **bytearray() function :**

Return a new array of bytes. The bytearray type is a mutable sequence of integers in the range  $0 \leq x < 256$ . It has most of the usual methods of mutable sequences, described in Mutable Sequence Types, as well as most methods that the bytes type has, see Bytes and Byte Array Methods.

**Syntax:** bytearray([source[, encoding[, errors]]])

**Lists:** List is also a sequence of values of any type. Values in the list are called elements or items. These are **mutable** and indexed. List is enclosed in square brackets ([]). The difference between a list and an array is that a list can store different types of elements but an array can store only one type of elements. Lists are represented using square brackets [] and the elements are written in [], separated by commas.

**List data type:** Lists in python are similar to arrays in C or Java.

e.g.: List= [10,-20,15.5, 'vijay','mary']

**Tuples:** Tuples are a sequence of values of any type and are indexed by integers. They are immutable. Tuples are enclosed in ().

**Tuple data type:** A tuple is similar to a list. *A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).* The elements in the tuple are separated by commas and enclosed in parentheses (). The list elements can be modified.

tp1=(10,-20,15.5,'vijay','mary')

### **The difference between the tuples and lists:**

In a tuple, the elements once assigned cannot be changed whereas in a list, elements can be changed.

**Range data type:** The range data type represents a sequence of numbers. The numbers in the range are not modifiable.

r=range(10)

Here, the range object is created with the numbers starting from 0 to 9. We can display these numbers using a for loop as:

```
for i in r:  
    print(i)
```

This statement will display numbers from 0 to 9.

**Sets:** A set is an unordered and unindexed collection of elements. Sets are written with curly brackets.

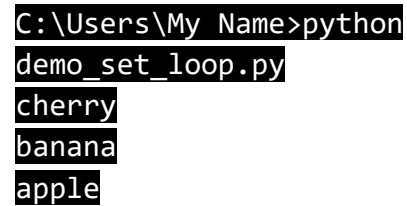
e.g.: Create a Set:

```
thisset={"apple", "banana", "cherry"}  
print(thisset)
```

```
C:\Users\My Name>python demo_set.py  
{'banana', 'cherry', 'apple'}
```

Since sets are unordered the items has no index, we cannot access items in a set by referring to an index, But we can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

```
e.g.:thisset={"apple", "banana", "cherry"}
for x in thisset:
    print(x)
```



```
C:\Users\My Name>python
demo_set_loop.py
cherry
banana
apple
```

**Frozen set data type:** The frozen set data type is same as the set data type. The difference is that the elements in the set data type can be modified; whereas, the elements of frozen set cannot be modified.

```
S={50,60,70,80,90}
Print(S) #Display {50,60,70,80,90}
fs=frozenset(s) create frozenset fs
print(fs) #Display frozen set ({50,60,70,80,90})
```

**Mapping types:** A map represents a group of elements in the form of key value pairs is given, we can retrieve the value associated with it.

The mapping objects are used to map hash table values to arbitrary objects. In python there is mapping type called **dictionary**. It is mutable.

The keys of the dictionary are arbitrary. As the value, we can use different kind of elements like lists, integers or any other mutable type objects.

Some dictionary related methods and operations are

#### **Method len(d)**

The len() method returns the number of elements in the dictionary.

#### **Operation d[k]**

It will return the item of d with the key 'k'. It may raise **KeyError** if the key is not mapped.

#### **Method iter(d)**

This method will return an iterator over the keys of dictionary. We can also perform this task by using **iter(d.keys())**.

#### **Method get(key[, default])**

The get() method will return the value from the key. The second argument is optional. If the key is not present, it will return the default value.

#### **Method items()**

It will return the items using (key, value) pairs format.

#### **Method keys()**

Return the list of different keys in the dictionary.

## Method values()

Return the list of different values from the dictionary.

## Method update(elem)

Modify the element elem in the dictionary.

## Operators

An operator is a symbol that performs an operation on some variables called operands.

e.g.: In  $a+b$ , the operator '+' is acting on two operands 'a' and 'b'.

Operators are classified as unary, binary and ternary operators.

1. If an operator acts on a single variable, it is called unary operator.
2. If an operator acts on two variables, it is called binary operator.
3. If an operator acts on three variables, then it is called ternary operator.

We can classify the operators depending upon their nature, as shown below:

- \* Arithmetic operators
- \* Assignment operators
- \* Relational operators
- \* Logical operators
- \* Boolean operators
- \* Bitwise operators
- \* Membership operators
- \* Identity operators

**Arithmetic operators:** These operators are used to perform basic arithmetic operations like addition, subtraction, division, etc.

There are 7 arithmetic operators available in python. They are 'binary operators'. Let's assume  $a=13$  and  $b=5$ , the effect of various arithmetic operations are shown below

Operator	Meaning	Example	Result
+	Addition operator. Adds two values	$a+b$	18
-	Subtraction operator. Subtracts one value from another	$a-b$	8
*	Multiplication operator. Multiplies values on either side of the operator	$a*b$	65
/	Division operator. Divides left operand by the right operand	$a/b$	2.6
%	Modulus operator. Gives remainder of division	$a\%b$	3
**	Exponent operator to calculate exponential power value $a**b$ . This gives the value of a to the power of b	$a**b$	371293
//	Integer division or floor division to perform division and give only integer quotient.	$a//b$	2

**Assignment Operators:** The assignment operator = is used to store right side value into left side variable, i.e.  $x + y$  is stored into  $z$ .  $z = x + y$

Other shorthand assignment operators are.

$+=$  : Additional assignment operator.

$-=$  : Subtraction assignment operator.

$*=$  : Multiplication assignment operator.

$/=$  : Division assignment operator.

$\%=$  : Modulus assignment operator.

$**=$  : Exponentiation assignment operator.

$//=$  : Floor division assignment operator.

### Relational operators or Comparison operators

Comparison operators are used to compare values. It either returns **True** or **False** according to the condition.

e.g.  $x = 10$

$y = 12$

`print('x > y is', x > y)` # Output: x > y is False

`print('x < y is', x < y)` # Output: x < y is True

`print('x == y is', x == y)` # Output: x == y is False

`print('x != y is', x != y)` # Output: x != y is True

`print('x >= y is', x >= y)` # Output: x >= y is False

`print('x <= y is', x <= y)` # Output: x <= y is True

**Logical Operators:** The logical operators are logical AND (&&), Logical OR || and logical NOT

Operator	Example	Meaning
AND	$x \text{ AND } y$	True if both the operands are true
OR	$x \text{ OR } y$	True if either of the operands is true
NOT	$x \text{ NOT } y$	True if operand is false and vice versa (complements the operand)

e.g.  $x = \text{True}$

$y = \text{False}$

`print('x and y is', x and y)` # Output: x and y is False

`print('x or y is', x or y)` # Output: x or y is True

`print('not x is', not x)` # Output: not x is False

**Bitwise Operators:** Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit, hence the name. There are 6 types of bitwise operators.

e.g.: Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

Operator	Meaning	Example
&	Bitwise AND	x & y = 0 (0000 0000)
	Bitwise OR	x   y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x >> 2 = 2 (0000 0010)
<<	Bitwise left shift	x << 2 = 40 (0010 1000)

**Membership Operators:** The membership operators are useful to test for membership in a sequence such as strings, lists, tuples or dictionaries.

e.g.: The presence of an element in the sequence can be confirmed using these operators. These are two membership operators:

1. in
2. not in

**in operator:** The 'in' operator is used to check if a value exists in a sequence or not. Evaluates to true if it finds a variable in the specified sequence and false otherwise.

#### Python program to illustrate 'in' operator

```
list1=[1,2,3,4,5]
list2=[6,7,8,9]
for item in list1:
    if item in list2:
        print("overlapping")
    else:
        print("not overlapping")
```

Output:  
not overlapping

**'not in' operator:** Evaluates to true if it does not find a variable in the specified sequence and false otherwise.

#### Python program to illustrate not in operator

```
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 present in given list

**Identity operators:** These operators compare the memory known whether the two objects are same or not. There are two identity operators:

1. is
2. is not

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

#### # Python program to illustrate the use of 'is' identity operator

```
x = 5
if (type(x) is int):
    print ("true")
else:
    print ("false")
```

Output:

true

**The 'is not' operator:** This 'is not' operator returns true, if the identity numbers of two objects being compared are not same. If they are same, then it will return false.

#### Python program to illustrate the use of 'is not' identity operator

```
x = 5.2
if (type(x) is not int):
    print ("true")
else:
    print ("false")
```

Output:

true

#### Increment and Decrement operators:

*There are no increment and decrement operators in Python.*

In Python to increment the value of a variable by 1 we write += or  $x = x + 1$ .

To decrement the value by 1 we use -= or  $x = x - 1$ .

Generally, ++ and -- is used in a loop. But in Python for loop is generally written as for i in range (0, 10) thus removing the need for a ++ operator.

```
e.g., for i in range(1, 11)
print(i)
```

This Python code prints the integer numbers from 1 to 10.

\*\*\*\*\*