

## PYTHON BASICS

**Python** is a high level, free open source and interpreted programming language.

Python is object-oriented language as well as procedural oriented language.

## FEATURES OF PYTHON:

- **Easy to code:**

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in python language and anybody can learn python basics in a few hours or days. It is also a developer-friendly language.

- **Free and Open Source:**

Python language is freely available at the official website and you can download it from the given download link below click on the **Download Python** keyword.

[Download Python](#)

Since it is open-source, this means that source code is also available to the public. So you can download it as, use it as well as share it.

- **Object-Oriented Language:**

One of the key features of python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, objects encapsulation, etc.

- **GUI Programming Support:**

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python.

PyQt5 is the most popular option for creating graphical apps with Python.

- **High-Level Language:**

Python is a high-level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.

- **Extensible feature:**

Python is a **Extensible** language. We can write us some Python code into C or C++ language and also, we can compile that code in C/C++ language.

- **Python is Portable language:**

Python language is also a portable language. For example, if we have python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

- **Python is Integrated language:**

Python is also an Integrated language because we can easily integrated python with other languages like c, c++, etc.

- **Interpreted Language:**

Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile python code this makes it easier to debug our code. The source code of python is converted into an immediate form called **bytecode**.

- **Large Standard Library**

Python has a large standard library which provides a rich set of module and functions so you do not have to write your own code for every single thing. There are many libraries present in python for such as regular expressions, unit-testing, web browsers, etc.

- **Dynamically Typed Language:**

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don't need to specify the type of variable.

## LITERAL CONSTANTS IN PYTHON

The word literal has been derived from literally. The value of a literal constant can be used directly in programs . For example, 7,3.9,'A', and "Hello" are literal constants

## NUMBERS:

There are four types of numbers in Python program. These include integers, long integers, floating point, and complex numbers. Note that the long integers must have 'l' or 'L' as the suffix

E.g.: 53633629843L is a long integer.

There is no limit to the size of an integer that can be represented in Python.

Numbers of the form  $a+bi$  ( $3+2i$ ) are complex numbers.

Numbers like 3.23 and  $91.5E-2$  are floating points and the 'E' notation indicates powers of 10. In this case  $91.5E-2=91.5*10^{-2}$ .

Floating point numbers have a **limited range** and **limited precision**. Usually the value ranges from -10308 to 10308 with 16 to 17 digits precision

Although floating points are helpful in case of handling large numbers there are certain errors that might rise up while dealing with them.

- **The Arithmetic Overflow Problem:** Occurs when multiplying two extra large numbers. It is a condition that occurs when a calculated result is too large in magnitude to be represented.  
E.g.: When you multiply  $2.7e200 * 4.3e200$  the output is inf, which means infinity. This result infinity denotes that an Arithmetic Overflow has occurred.
- **The Arithmetic Underflow Problem:** Occurs when dividing two extremely small numbers. It is a condition that occurs when a calculated result is too small in magnitude to be represented.  
E.g.: When you divide  $3.0e-400/5.0e200$  the output is 0.0. The value 0.0 indicates there was an Arithmetic Underflow in the result.
- **Loss of Precision:** The result of dividing  $1/3$  is known to be 0.33333.. where 3 is repeated infinitely many times. However due to a limited precision and range in Python, the output will be just an approximation of the true value or will display the true value up to the precision capacity of Python.

## SIMPLE OPERATION ON NUMBERS:

We are already aware of the basic addition and multiplication of numbers as in C and C++ languages. However, there are certain additional features in Python which are discussed below.

- **Division by zero:** Dividing a number by 0 in Python generates an error, unlike C++ which results in abnormal execution of the program.

- **Dividing two integers:** In C or C++ executing the code `3/5` results in an output to be 0 as division of two integers in C or C++ results in an integer output. However this issue is eliminated in Python as it implicitly converts every integer to a floating point (also known as type coercion) .Hence `3/5` in Python will give you the output as 0.6
- **Quotient and Remainder:** In case of dividing two numbers, for simple division use the division operator (`/`), for quotient use the floor division (`//`) and for remainder the modulo operator (`%`).
- **Exponentiation:** The `**` operator is used for exponentiation. The number to the left of the operator is the base and to the right is the exponent.

### **STRINGS:**

There is **no character data type** in Python.

String is a group of characters. You can use a string in the following way in a Python program.

1. **Using Single Quote:** (`'`) For example, a string can be written as `'Hello'`. The single quote in this case exhibits EOL (End Of Line).
2. **Using Double Quote:** (`"`) Strings in double quotes are exactly same as those in single quotes. Hence `'Hello'` is same as `"Hello"`. The double quote in this case exhibits EOL.
3. **Using Triple Quote:** (`''' '''`) You can specify multi-line strings using triple quotes. You can use as many single and double quotes you want in a string within triple quotes. The triple quote in this case exhibits EOL.  
Another way to print multiword strings is by using the escape sequence character (`\n`) **within the quotes**.

**E.g:** `print("Hello World. \n This is my first Python code.")`

The strings can be printed in Python by just simply writing the string in the appropriate quotes, or by using the **print()** function. There is a slight difference in the output when using the above two different methods. Observe the image below carefully

```
Python 3.8.3 Shell
File Edit Shell Debug Options Window Help
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> '''Good morning everyone.
Welcome to the world of Python.Happy reading the book 'Let Us Python'.
Happy reading'''
"Good morning everyone.\nWelcome to the world of Python.Happy reading the book '
Let Us Python'.\nHappy reading"
>>> print(''Good morning everyone.
Welcome to the world of Python.Happy reading the book 'Let Us Python'.
Happy reading'')
Good morning everyone.
Welcome to the world of Python.Happy reading the book 'Let Us Python'.
Happy reading
>>> |
```

Observe the `\n` in the output when a string is printed without the **print()** function. Hence in such a scenario the string is printed as it is observing the spaces, tabs, new lines and quotes (single as well as double).

**Note:** Using single or double quotes to print multiword strings generates error.

❖ Python concatenates two string literals that are placed side by side.

**E.g.:** `print('Beautiful weather' ' today' )`

**O/p:** Beautiful weather today.

Some characters (like " OR \) cannot be directly included in a string. Such characters must be escaped by placing a backslash (\) before them.

```
>>> print('What's your name?')
SyntaxError: invalid syntax
>>> print('What\'s your name?')
What's your name?
```

The syntax error was generated because Python got confused as to where the string starts and ends. Hence, we need to clearly specify that this single quote does not indicate the end of line.

However if we use double quotes to write a string then the single quote within the string is completely valid, it does not need a backslash ( \ ) character .

```
>>> print("What's your name?")
What's your name?
```

**RAW STRINGS:**

If you want to specify a string that should not handle any escape sequences and want to display exactly as specified, then you need to specify that string as a ***raw string***.

A Raw string is specified by prefixing r or R to the string.

```
print(R "What's your name?")
```

**OUTPUT:**    What's your name?

**PYTHON DATATYPES OVERVIEWS**

In Python language everything is referred to as an object including numbers and strings. The five standard data types supported by Python includes: numbers, string, list, tuple and dictionary.

**INPUT FUNCTION**

To make the program interactive in the sense that we can take some sort of input or information from the user and work on that input we use the **input( )** function. This function prompts the user to provide some information on which the program can work and give the result. However, we must always remember that the unput function takes user's input as **string**, irrespective of whether the entered element is a number or a character.

**RESERVED WORDS**

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

## ARITHMETIC OPERATORS

# Arithmetic Operators

a = 5  
b = 2

Operator	Meaning	Example	Result
+	Addition Operator. Adds two Values.	a + b	7
-	Subtraction Operator. Subtracts one value from another	a - b	3
*	Multiplication Operator. Multiplies values on either side of the operator	a * b	10
/	Division Operator. Divides left operand by the right operand.	a / b	2.5
%	Modulus Operator. Gives remainder of division	a % b	1
**	Exponent Operator. Calculates exponential power value. a ** b gives the value of a to the power of b	a ** b	25
//	Integer division, also called floor division. Performs division and gives only integer quotient.	a // b	2

## COMPARISON OPERATORS

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

## ASSIGNMENT AND IN-PLACE OPERATORS

The in-place operators also known as shortcut operators include +=, -=, \*=, /=, //=, \*\*=. They can be also used on other data types. (str1+=str2)



# Assignment Operators

```
x = 3
y = 2
z = 5
```

Operator	Example	Meaning	Result
=	z = x + y	Assignment operator. Stores right side value into left side variable.	z = 5
+=	z+=x	Addition assignment operator. Adds right operand to the left operand and stores the result into left operand.	z = 8
-=	z-=x	Subtraction assignment operator. subtracts right operand to the left operand and stores the result into left operand.	z = 2
*=	z*=x	Multiplication assignment operator. Multiplies right operand to the left operand and stores the result into left operand.	z = 15
/=	z/=y	Division assignment operator. Divides left operand with the right operand and stores the result into left operand.	z = 2.5
%=	z%=x	Modulus assignment operator. Divides left operand to the right operand and stores the result into left operand.	2
**=	z**=y	Exponential assignment operator. Performs power value and then stores the result into left operands	25
//=	z//=y	Floor division assignment operator. Performs floor division and then stores	2

## BINARY OPERATORS

Python does not support prefix and postfix increment or decrement operator.

Bit-wise	Prototype	Semantics (integers shown as bits)
and	& (int,int) -> int	0011 & 0101 returns the result 0001
inclusive or	(int,int) -> int	0011   0101 returns the result 0111
not	~ (int) -> int	~01 returns the result 10
exclusive or	^ (int,int) -> int	0011 ^ 0101 returns the result 0110
shift left	<< (int,int) -> int	101 << 2 returns the result 10100 (similar to 5 // 2**2)
shift right	>> (int,int) -> int	101 >> 2 returns the result 1 (similar to 5 * 2**2)

Bit-wise operations (& | ~ ^ << >>) compute their results on the individual bits in the binary representations of their integer operands. Note that BITWISE operators cannot be applied to float or double variables. The BINARY NOT, or complement, performs logical negation on each bit of the operand.

## MEMBERSHIP OPERATORS

These operators test for membership in a sequence such as strings, lists or tuples.



**in Operator:** The operator returns true if a variable is found in the specified sequence and False otherwise

**not in Operator:** The operator returns True if a variable is not found in the specified sequence and False otherwise.

## IDENTITY OPERATORS

These operators compare the memory locations of the two objects

**is Operator:** Returns True if operands or values on both sides of the operator point to the same object and False otherwise. For instance, a is b returns 1 if id(a) is same as id(b).

**is not Operator:** Returns True if operands or values on both sides of the operator do not point to the same object (memory location).

## OPERATIONS ON STRINGS

**CONCATENATION:** Like numbers you can also add two strings in python using the '+' operator.

**STRING REPETATION:** You cannot add a string and a number but you can definitely multiply a string and a number. When a string is multiplied with an integer n, the string is repeated n time. Hence the \* operator is also known as string repetition operator. The order of the string and the integer does not matter.

**SLICING A STRING:** You can extract subsets of a string by using the slice operator ( [ ] and [ : ] ). We need to specify index or the range of index of characters to be extracted. The index of the first character is 0 and that of the last is n-1, where n is the length of the string

If you want to extract characters starting from the end of the string, then you must specify the index as a negative number. For example, the index of the last character is -1

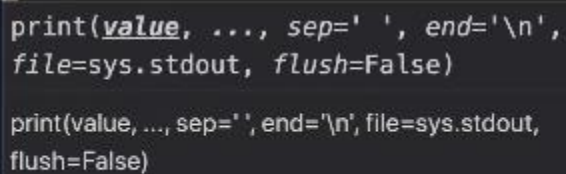
## F-STRINGS:

The idea behind f-strings is to make string interpolation simpler.

To create an f-string, prefix the string with the letter “f”. F-strings provide a concise and convenient way to embed python expressions inside string literals for formatting. For instance, look at the code below.

```
score = 0
height = 1.8
isWinning = True

print("your score is " + str(score))
```



Here in order to use the concatenation technique every variable would have to be typecasted into string and used. However F strings provide an alternative way to write such strings without any overhead of continuous typecasting.

```
1 score = 0
2 height = 1.8
3 isWinning = True
4 #f-String
5 print(f"your score is {score}, your height is {height}, you are winning is {isWinning}")
```

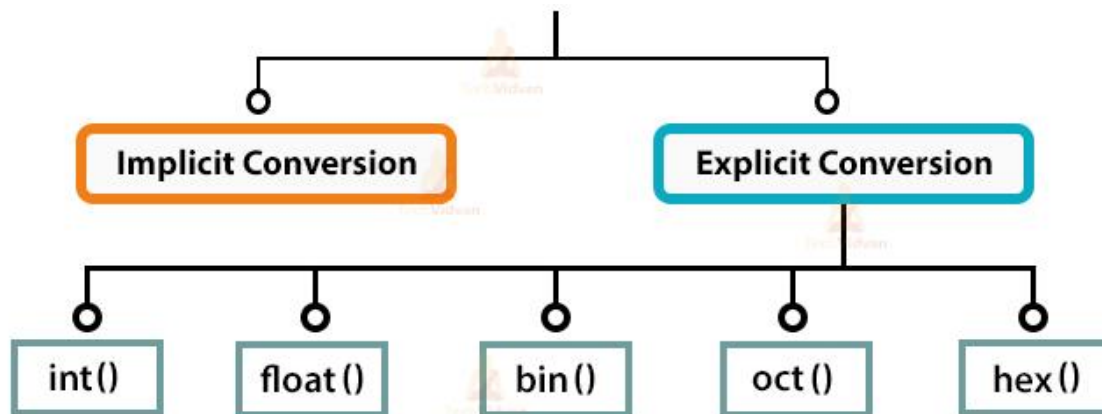
PRECEDENCE

## Python Operator Precedence

Precedence	Operator Sign	Operator Name
Highest	**	Exponentiation
	+X, -X, ~X	Unary positive, unary negative, bitwise negation
	*, /, //, %	Multiplication, division, floor, division, modulus
	+, -	Addition, subtraction
	<<, >>	Left-shift, right-shift
	&	Bitwise AND
	^	Bitwise XOR
		Bitwise OR
	==, !=, <, <=, >, >=, is, is not	Comparison, Identity
	not	Boolean NOT
	and	Boolean AND
Lowest	or	Boolean OR

TYPE CONVERSION

**type(obj)** is used to identify the type of the object.



### TYPE-CASTING VS TYPE COERCION

Explicit conversion of a value from one data type to another is known as type casting. However, in Python and most of the other programming languages there is an implicit conversion of data types done either during compile time or run time. This is known as type coercion. For instance,  $21+2.1=23.1$ . The compiler implicitly will convert the integer into floating point number so that the fractional part is not lost.

**NOTE:** In Python whenever one variable's value is assigned to another variable, no new copy of the variable is made instead they point to the same memory location. This is known as aliasing.

```
a=4
b=a
print(id(a)==id(b))
```

True