PYTHON INTRODUCTION

Python is an interpreted high-level object-oriented computer programming language best known for its simplest easy to use codes

Python programming language was created by Guido Van Rossum.

Because of its simplicity in coding and powerful features, it works brilliantly for both beginner to experts.

HISTORY OF PYTHON

Python was introduced by Guido Van Rossum in 1989.

•Python laid its foundation in the late 1980s.
•The implementation of Python was started in December 1989 by **Guido Van Rossum** at CWI(Centrum voor Wiskunde en Informatica) in Netherland.
•In February 1991, **Guido Van Rossum** published the code (labeled version 0.9.0) to alt.sources.
•In 1994, Python 1.0 was released with new features like lambda, map, filter, and reduce.
•Python 2.0 added new features such as list comprehensions, garbage collection systems.
•On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify the fundamental flaw of the language.
•*ABC programming language* is said to be the predecessor of Python language, which was capable of Exception Handling and interfacing with the Amoeba Operating System.
•The following programming languages influence Python:

- ABC language.
- Modula-3

Why the Name Python?

There is a fact behind choosing the name Python. Guido van Rossum was reading the script of a popular BBC comedy series "Monty Python's Flying Circus". It was late on-air 1970s.

Van Rossum wanted to select a name which unique, sort, and little-bit mysterious. So he decided to select naming Python after the "Monty Python's Flying Circus" for their newly created programming language.

The comedy series was creative and well random. It talks about everything. Thus it is slow and unpredictable, which made it very interesting.

Python is also versatile and widely used in every technical field, such as Machine Learning, Artificial Intelligence, Web Development, Mobile Application, Desktop Application, Scientific Calculation, etc.

Python Version List

Python programming language is being updated regularly with new features and supports. There are lots of update in Python versions, started from 1994 to current release.

A list of Python versions with its released date is given below.

Python Version	Released Date
Python 1.0	January 1994
Python 1.5	December 31, 1997
Python 1.6	September 5, 2000
Python 2.0	October 16, 2000
Python 2.1	April 17, 2001
Python 2.2	December 21, 2001
Python 2.3	July 29, 2003
Python 2.4	November 30, 2004
Python 2.5	September 19, 2006
Python 2.6	October 1, 2008
Python 2.7	July 3, 2010

Python Version	Released Date
Python 3.0	December 3, 2008
Python 3.1	June 27, 2009
Python 3.2	February 20, 2011
Python 3.3	September 29, 2012
Python 3.4	March 16, 2014
Python 3.5	September 13, 2015
Python 3.6	December 23, 2016
Python 3.7	June 27, 2018
Python 3.8	October 14, 2019

Python Programming: Features

There are many features of Python programming making it more popular among developers with every passing day.

•Simplicity in coding

Simplicity is what isolates python programming from other programming languages like C, C++, Java and others. Python is best for beginners as its simple and elegant syntax is easy to get along yet being so much powerful at the same time.

•Easy to learn, read and interpret

The simplicity in structure and easily defined syntax of Python programming which makes it super easy to learn, even for beginners. Python is like simple English with some predefined super easy instructions.

Python Programming: Features

•Free and Open Source: Thus easy to maintain

Python is constantly improved by the community of developers working every day to make it efficient and better. The best part of Python is that it is absolutely free for distributing copies, reading and editing source codes. It is free for both domestic and commercial use.

•Broad range of standard libraries

There is a huge number of standard Python libraries available for free. Thus, it makes it super easy to solve even complicated task using these libraries because almost everything is predefined and coded in these libraries. It is the availability of these libraries that have made Python one of the most popular programming language.

Python Programming: Features

•Portable and Extensible

Portable means, Python programs, and applications can be used on multiple platforms without any changes. With the same interface, it can work perfectly fine in cross platforms. Also, we can integrate and add modules to python interpreter for making our tools more efficient.

•Embeddable and Scalable

To increase the capabilities and scope of the program we can embed Python code with C/C++ programs. Python is also scalable in a sense that, it has better structure and support for large programs instead of just scripting in a shell.

•Interpreted

This is also one of the key features of Python programming. When we run a program, the compiler loads the program from the memory heap, compiles it and starts running it. On the contrary, an interpreter runs Python program directly from source code. We don’t need to worry about compiling and other lower level tasks which make it super easy for a programmer.

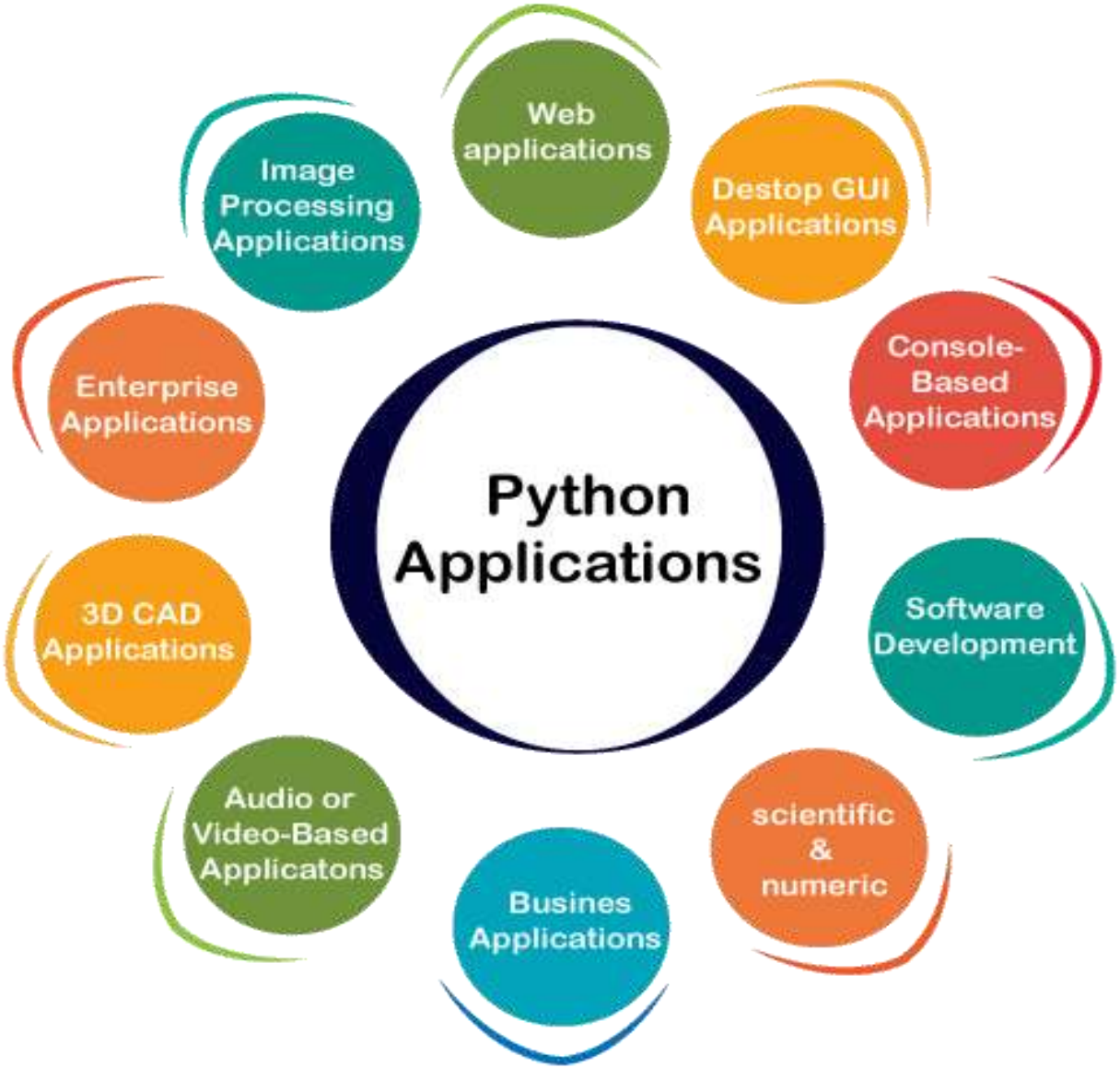
Applications of Python

Before knowing about Python's application, one must know the big companies and applications that use Python programming languages.

To name a few are:

- Google
- Facebook
- Instagram
- NASA
- Mozilla (built in Python)
- Dropbox
- Quora and many others

Besides these tech giants, millions of applications are built on a regular basis using Python because of its astounding simplicity in coding and many other core features.


Python Applications
Web applications
Destop GUI Applications
Console-Based Applications
Software Development
scientific & numeric
Busines Applications
Audio or Video-Based Applicatons
3D CAD Applications
Enterprise Applications
Image Processing Applications

1) Web Applications

We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, beautifulSoup, Feedparser, etc. One of Python web-framework named Django is used on **Instagram**. Python provides many useful frameworks, and these are given below:

- Django and Pyramid framework(Use for heavy applications)
- Flask and Bottle (Micro-framework)
- Plone and Django CMS (Advance Content management)

2) Desktop GUI Applications

The GUI stands for the Graphical User Interface, which provides a smooth interaction to any application. Python provides a **Tk GUI library** to develop a user interface. Some popular GUI libraries are given below.

- Tkinter or Tk
- wxWidgetM
- Kivy (used for writing multitouch applications)
- PyQt or Pyside

3) Console-based Application

Console-based applications run from the command-line or shell. These applications are computer program which are used commands to execute. This kind of application was more popular in the old generation of computers. Python can develop this kind of application very effectively. It is famous for having REPL, which means **the Read-Eval-Print Loop** that makes it the most suitable language for the command-line applications.

Python provides many free library or module which helps to build the command-line apps. The necessary **IO** libraries are used to read and write. It helps to parse argument and create console help text out-of-the-box. There are also advance libraries that can develop independent console apps.

4) Software Development

Python is useful for the software development process. It works as a support language and can be used to build control and management, testing, etc.

- **SCons** is used to build control.
- **Buildbot** and **Apache** Gumps are used for automated continuous compilation and testing.
- **Round** or **Trac** for bug tracking and project management.

5) Scientific and Numeric

This is the era of Artificial intelligence where the machine can perform the task the same as the human. Python language is the most suitable language for Artificial intelligence or machine learning. It consists of many scientific and mathematical libraries, which makes easy to solve complex calculations.

Implementing machine learning algorithms require complex mathematical calculation. Python has many libraries for scientific and numeric such as Numpy, Pandas, Scipy, Scikit-learn, etc. If you have some basic knowledge of Python, you need to import libraries on the top of the code. Few popular frameworks of machine libraries are given below.

- SciPy
- Scikit-learn
- NumPy
- Pandas
- Matplotlib

6) Business Applications

Business Applications differ from standard applications. E-commerce and ERP are an example of a business application. This kind of application requires extensively, scalability and readability, and Python provides all these features.

Oddo is an example of the all-in-one Python-based application which offers a range of business applications. Python provides a **Tryton** platform which is used to develop the business application.

7) Audio or Video-based Applications

Python is flexible to perform multiple tasks and can be used to create multimedia applications. Some multimedia applications which are made by using Python are **TimPlayer, cplay,** etc. The few multimedia libraries are given below.

- Gstreamer
- Pyglet
- QT Phonon

8) 3D CAD Applications

The CAD (Computer-aided design) is used to design engineering related architecture. It is used to develop the 3D representation of a part of a system. Python can create a 3D CAD application by using the following functionalities.

- Fandango (Popular)
- CAMVOX
- HeeksCNC
- AnyCAD
- RCAM

9) Enterprise Applications

Python can be used to create applications that can be used within an Enterprise or an Organization. Some real-time applications are OpenERP, Tryton, Picalo, etc.

10) Image Processing Application

Python contains many libraries that are used to work with the image. The image can be manipulated according to our requirements. Some libraries of image processing are given below.

- OpenCV
- Pillow
- SimpleITK

Usage of Python

Python is a general purpose, open source, high-level programming language and also provides number of libraries and frameworks. Python has gained popularity because of its simplicity, easy syntax and user-friendly environment. The usage of Python as follows.

Desktop Applications
Web Applications
Data Science
Artificial Intelligence
Machine Learning
Scientific Computing
Robotics
Internet of Things (IoT)
Gaming
Mobile Apps
Data Analysis and Preprocessing

Python: Environment Setup and Installation


Installing Python on Windows
www.tryitoprogram.com

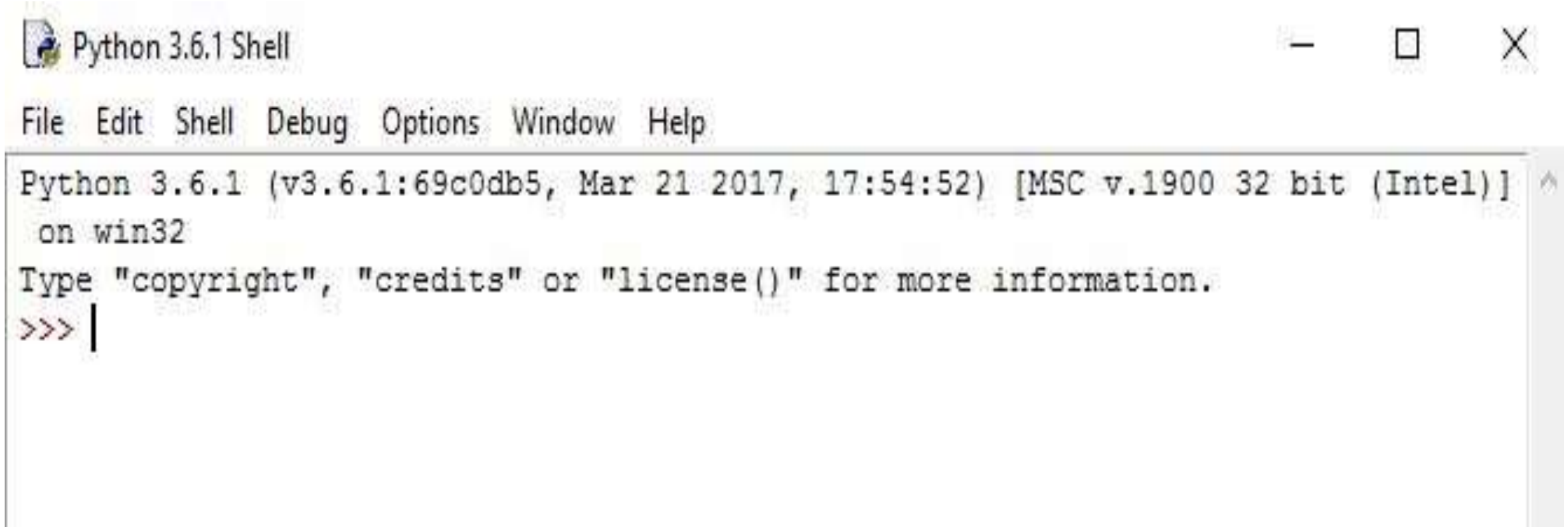
Installing Python is a simple task. Follow following steps to install Python on windows and you don’t need an external editor to write codes as IDLE comes in default while installing Python.

Step 1: Go to the official download page of Python and click on the download link to download the latest version or any version. There will be many downloadable files available. Make sure you download the executable file for the ease.

Step 2: Once the executable file is downloaded, click and install it following the instructions.

That’s it and you have installed Python on your windows. Now, the one thing you need is an editor to write Python scripts. But, as mentioned above Python comes with default editor IDLE.

When you open following screen will pop up.



Python Programming: Text Editors/Interpreters

With the installation of Python, IDLE comes as default and free which is amazing in its own ways. But, if you are working on complex projects, there are many other text editors and interpreters available which make coding in Python even easier.

Here are some of the popular IDEs for Python:

PyCharm by Jetbrains
PyDev with Eclipse
Spyder Python
Komodo IDE
Wing IDE

SIMPLE PROGRAM

```
>>> Print(“hello world”)
hello world

>>> 1+1
2
```

```
num1 = input('Enter first number: ')
num2 = input('Enter second number: ')

# Add two numbers
sum = float(num1) + float(num2)
# Subtract two numbers
min = float(num1) - float(num2)
# Multiply two numbers
mul = float(num1) * float(num2)
#Divide two numbers
div = float(num1) / float(num2)
# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))

# Display the subtraction
print('The subtraction of {0} and {1} is {2}'.format(num1, num2, min))
# Display the multiplication
print('The multiplication of {0} and {1} is {2}'.format(num1, num2, mul))
# Display the division
print('The division of {0} and {1} is {2}'.format(num1, num2, div))
```

Python Keywords

Keywords are predefined, reserved words used in Python programming that have special meanings to the compiler.

We cannot use a keyword as a variable name, function name, or any other identifier. They are used to define the syntax and structure of the Python language.

All the keywords except True, False and None are in lowercase and they must be written as they are. The list of all the keywords is given below.

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

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

Python Keywords List

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

Python Identifiers

Identifiers are the name given to variables, classes, methods, etc. For example,
language = 'Python’

Rules for Naming an Identifier

- Identifiers cannot be a keyword.
- Identifiers are case-sensitive.
- It can have a sequence of letters and digits. However, it must begin with a letter or _. The first letter of an identifier cannot be a digit.
- It's a convention to start an identifier with a letter rather _.
- Whitespaces are not allowed.
- We cannot use special symbols like !, @, #, $, and so on.

Python Comments

Comments are completely ignored by the interpreter. They are meant for fellow programmers.

Types of Comments in Python
In Python, there are two types of comments:

- single-line comment
- multi-line comment

Single-line Comment in Python

A single-line comment starts and ends in the same line. We use the # symbol to write a single-line comment. For example,

```
# create a variable
name = ‘Koshys'

# print the value
print(name)
```

Multi-line Comment in Python

Python doesn't offer a separate way to write multiline comments. However, there are other ways to get around this issue.

We can use # at the beginning of each line of comment on multiple lines. For example,

```
# This is a long comment
# and it extends
# to multiple lines
```

Another way of doing this is to use triple quotes, either ''' or """.

These triple quotes are generally used for multi-line strings. But if we do not assign it to any variable or function, we can use it as a comment.

The interpreter ignores the string that is not assigned to any variable or function.

Let's see an example,

```
''' This is also a
perfect example of
multi-line comments '''
```

Python Variables

In programming, a variable is a container (storage area) to hold data. For example,
number = 10

Assigning values to Variables in Python

We use the assignment operator = to assign a value to a variable.

Changing the Value of a Variable in Python

```
site_name = ‘google.com'
print(site_name)

# assigning a new value to site_name
site_name = 'apple.com'
print(site_name)
```

Assigning multiple values to multiple variables

```
a, b, c = 5, 3.2, 'Hello'

print(a)  # prints 5
print(b)  # prints 3.2
print(c)  # prints Hello
```

Rules for Naming Python Variables

- Constant and variable names should have a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).
- Create a name that makes sense. For example, vowel makes more sense than v.
- If you want to create a variable name having two words, use underscore to separate them.
- Python is case-sensitive. So num and Num are different variables.
- Avoid using keywords like if, True, class, etc. as variable names.

Assigning single value to multiple variables

Eg:

```
x=y=z=50
print(x)
print(y)
print(z)
```

Python Literals

Literals are representations of fixed values in a program. They can be numbers, characters, or strings, etc.
For example, 'Hello, World!', 12, 23.0, 'C', etc.

Literals are often used to assign values to variables or constants.

Python Numeric Literals

Numeric Literals are immutable (unchangeable).
Numeric literals can belong to 3 different numerical types: Integer, Float, and Complex.

Type	Example	Remarks
Decimal	5, 10, -68	Regular numbers.
Binary	0b101, 0b11	Start with 0b.
Octal	0o13	Start with 0o.
Hexadecimal	0x13	Start with 0x.
Floating-point Literal	10.5, 3.14	Containing floating decimal points.
Complex Literal	6 + 9j	Numerals in the form a + bj, where a is real and b is imaginary part

Python Boolean Literals

There are two boolean literals: True and False.

String and Character Literals in Python

Character literals are unicode characters enclosed in a quote.

Special Literal in Python

Python contains one special literal None. We use it to specify a null variable.

Python Constants

The term refers to a value or quantity that never changes. In programming, constants refer to names associated with values that never change during a program’s execution.

Why Use Constants

In most programming languages, constants protect you from accidentally changing their values somewhere in the code. Constants also help you make your code more readable and maintainable.

Advantage

- Improved readability
- Clear communication of intent
- Better maintainability
- Lower risk of errors
- Reduced debugging needs
- Thread-safe data storage

```
# declare constants
PI = 3.14
GRAVITY = 9.8
```

```
# import constant file we created above
import constant

print(constant.PI) # prints 3.14
print(constant.GRAVITY) # prints 9.8
```

Python Data Types

In computer programming, data types specify the type of data that can be stored inside a variable. For example,

num = 24

Here, 24 (an integer) is assigned to the num variable. So the data type of num is of the int class.

Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

GETTING THE DATA TYPE

```
num1 = 5
print(num1, 'is of type',type(num1))

num2 = 2.0
print(num2, 'is of type',type(num2))

num3 = 1+2j
print(num3, 'is of type',type(num3))
```

Python has the following data types built-in by default, in these categories:

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview
None Type:	NoneType

Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview
x = None	NoneType

Python Numeric Data type

In Python, numeric data type is used to hold numeric values.

Integers, floating-point numbers and complex numbers fall under Python numbers category. They are defined as int, float and complex classes in Python.

- int - holds signed integers of non-limited length.
- float - holds floating decimal points and it's accurate up to 15 decimal places.
- complex - holds complex numbers.

Python String Data Type

String is a sequence of characters represented by either single or double quotes. For example,

```
name = 'Python'
print(name)

message = “Python for beginners”
print(message)
```

Output

```
Python
Python for beginners
```

Python List Data Type

List is an ordered collection of similar or different types of items separated by commas and enclosed within brackets [ ]. For example,

languages = ["Swift", "Java", "Python"] Access List Items

To access items from a list, we use the index number (0, 1, 2 ...). For example,

```
languages = ["Swift", "Java", "Python"]

# access element at index 0
print(languages[0])   # Swift

# access element at index 2
print(languages[2])   # Python
```

Python Tuple Data Type

Tuple is an ordered sequence of items same as a list. The only difference is that tuples are immutable. Tuples once created cannot be modified.

In Python, we use the parentheses () to store items of a tuple. For example,

```
product = ('Xbox', 499.99)
```

Access Tuple Items

Similar to lists, we use the index number to access tuple items in Python . For example,

```
# create a tuple
product = ('Microsoft', 'Xbox', 499.99)
# access element at index 0
print(product[0])   # Microsoft
# access element at index 1
print(product[1])   # Xbox
```

Python Set Data Type

Set is an unordered collection of unique items. Set is defined by values separated by commas inside braces { }. For example,

```
# create a set named student_id
student_id = {112, 114, 116, 118, 115}

# display student_id elements
print(student_id)

# display type of student_id
print(type(student_id))
```

Dictionary

Dictionary is an unordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Key can hold any primitive data type, whereas value is an arbitrary Python object.

The items in the dictionary are separated with the comma (,) and enclosed in the curly braces {}.

Consider the following example.

```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'}

# Printing dictionary
print (d)
```

```
# Accesing value using keys
print("1st name is "+d[1])
print("2nd name is "+ d[4])

print (d.keys())
print (d.values())
```

Output:

```
1st name is Jimmy
2nd name is mike
{1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}
dict_keys([1, 2, 3, 4])
dict_values(['Jimmy', 'Alex', 'john', 'mike'])
```

```
# create a dictionary named capital_city

capital_city = {'Nepal': 'Kathmandu',
'Italy': 'Rome', 'England': 'London'}

print(capital_city['Nepal'])  # prints
Kathmandu

print(capital_city['Kathmandu'])  #
throws error message
```

Python Set Data Type

Set is an unordered collection of unique items. Set is defined by values separated by commas inside braces { }. For example,

```
# create a set named student_id
student_id = {112, 114, 116, 118, 115}

# display student_id elements
print(student_id)

# display type of student_id
print(type(student_id))
```

Boolean

Boolean type provides two built-in values, True and False. These values are used to determine the given statement true or false. It denotes by the class bool. True can be represented by any non-zero value or 'T' whereas false can be represented by the 0 or 'F'. Consider the following example.

```
# Python program to check the boolean type
print(type(True))
print(type(False))
print(false)
```

Output:

```
<class 'bool'>
<class 'bool'>
NameError: name 'false' is not defined
```

OPERATORS

Operators are special symbols that perform operations on variables and values.
For example,
print(5 + 6) # 11

Types of Python Operators

- Arithmetic operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- Special Operators

1. Python Arithmetic Operators

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

sub = 10 - 5 # 5

Operator	Operation	Example
+	Addition	5 + 2 = 7
-	Subtraction	4 - 2 = 2
*	Multiplication	2 * 3 = 6
/	Division	4 / 2 = 2
//	Floor Division	10 // 3 = 3
%	Modulo	5 % 2 = 1
**	Power	4 ** 2 = 16

```
a = 7
b = 2

# addition
print ('Sum: ', a + b)

# subtraction
print ('Subtraction: ', a - b)

# multiplication
print ('Multiplication: ', a * b)

# division
print ('Division: ', a / b)

# floor division
print ('Floor Division: ', a // b)

# modulo
print ('Modulo: ', a % b)

# a to the power b
print ('Power: ', a ** b)
```

2. Python Assignment Operators

Assignment operators are used to assign values to variables. For example,

```
# assign 5 to x
var x = 5
```

Operator	Name	Example
=	Assignment Operator	a = 7
+=	Addition Assignment	a += 1 # a = a + 1
-=	Subtraction Assignment	a -= 3 # a = a - 3
*=	Multiplication Assignment	a *= 4 # a = a * 4
/=	Division Assignment	a /= 3 # a = a / 3
%=	Remainder Assignment	a %= 10 # a = a % 10
**=	Exponent Assignment	a **= 10 # a = a ** 10

3. Python Comparison Operators

Comparison operators compare two values/variables and return a boolean result: True or False. For example,

```
a = 5
b =2

print (a > b)    # True
```

Operator	Meaning	Example
==	Is Equal To	3 == 5 gives us **False**
!=	Not Equal To	3 != 5 gives us **True**
>	Greater Than	3 > 5 gives us **False**
<	Less Than	3 < 5 gives us **True**
>=	Greater Than or Equal To	3 >= 5 give us **False**
<=	Less Than or Equal To	3 <= 5 gives us **True**

```
a = 5

b = 2

# equal to operator
print('a == b =', a == b)

# not equal to operator
print('a != b =', a != b)

# greater than operator
print('a > b =', a > b)

# less than operator
print('a < b =', a < b)

# greater than or equal to operator
print('a >= b =', a >= b)

# less than or equal to operator
print('a <= b =', a <= b)
```

4. Python Logical Operators

Logical operators are used to check whether an expression is True or False. They are used in decision-making. For example,

```
a = 5
b = 6

print((a > 2) and (b >= 6))     # True
```

Operator	Example	Meaning
and	a **and** b	**Logical AND**: True only if both the operands are True
or	a **or** b	**Logical OR**: True if at least one of the operands is True
not	**not** a	**Logical NOT**: True if the operand is False and vice-versa.

5. Python Bitwise operators

Bitwise operators act on operands as if they were strings of binary digits. They operate bit by bit, hence the name.

For example, 2 is 10 in binary and 7 is 111.

In the table below: 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)

6. Python Special operators

Python language offers some special types of operators like the identity operator and the membership operator. They are described below with examples.

Identity operators

In Python, is and is not are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]

print(x1 is not y1)  # prints False

print(x2 is y2)  # prints True

print(x3 is y3)  # prints False
```

Membership operators

In Python, in and not in are the membership operators. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

In a dictionary we can only test for presence of key, not the value.

Operator	Meaning	Example
in	True if value/variable is **found** in the sequence	5 in x
not in	True if value/variable is **not found** in the sequence	5 not in x

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

# check if 'H' is present in x string
print('H' in x)  # prints True

# check if 'hello' is present in x string
print('hello' not in x)  # prints True

# check if '1' key is present in y
print(1 in y)  # prints True

# check if 'a' key is present in y
print('a' in y)  # prints False
```

Python Operator Precedence

Operator precedence in python determines the order in which operators are evaluated in an expression. The below table shows operator precedence as follows.

Operator	Description
**	Exponentiation
'~ + -'	Bitwise NOT, Unary Plus and Minus
<'* / % //'	Multiplication, division, modulus, and floor division
'+ -'	Addition and subtraction
'>> <<'	Bitwise shift right and left
&	Bitwise AND
^	Bitwise XOR
and	Logical and
not	Logical not
or	Logical or
'== != > >= < <='	Comparison Operators
= %= /= //= -= += *= **=	Assignment Operators
Is, is not	Identity Operators
in not in	Membership Operators
not or and	Logical Operators

Associativity of Python Operators

When two operators have the same precedence, associativity helps to determine the order of operations.

Associativity is the order in which an expression is evaluated that has multiple operators of the same precedence. Almost all the operators have left-to-right associativity.

For example, multiplication and floor division have the same precedence. Hence, if both of them are present in an expression, the left one is evaluated first.

```
# Left-right associativity
# Output: 3
print(5 * 2 // 3)

# Shows left-right associativity
# Output: 0
print(5 * (2 // 3))
```

Operator	Description	Associativity
( )	Parentheses	left to right
**	Exponent	right to left
* / %	Multiplication / division / modulus	left to right
+ -	Addition / subtraction	left to right
<< >>	Bitwise left shift / Bitwise right shift	left to right
< <= > >=	Relational operators: less than / less than or equal to / greater than / greater than or equal to	left to right
== !=	Relational operators: is equal to / is not equal to	left to right
is, is not in, not in	Identity operators Membership operators	left to right
&	Bitwise AND operator	left to right
^	Bitwise exclusive OR operator	left to right
|	Bitwise inclusive OR operator	left to right
not	Logical NOT	right to left
and	Logical AND	left to right
or	Logical OR	left to right
= += -= *= /= %= &= ^= |= <<= >>=	Assignment operators: Addition / subtraction Multiplication / division Modulus / bitwise AND Bitwise exclusive / inclusive OR Bitwise shift left / right shift	

Python Type Conversion

In programming, type conversion is the process of converting data of one type to another. For example: converting int data to str.

There are two types of type conversion in Python.

Implicit Conversion - automatic type conversion
Explicit Conversion - manual type conversion

Python Implicit Type Conversion

In certain situations, Python automatically converts one data type to another. This is known as implicit type conversion.

```
#program to demonstrate implicit type conversion
#initializing the value of a
a = 10
print(a)
print("The type of a is ", type(a))
#initializing the value of b
b = 4.5
print(b)
print("The type of b is ", type(b))
#initializing the value of c
c = 4.0
print(c)
print("The type of c is ", type(c))
#initializing the value of d
d = 5.0
print(d)
print("The type of d is ", type(d))
```

```
#performing arithmetic operations
res = a * b
print("The product of a and b is ", res)
add = c + d
print("The addition of c and d is ", add)

print("The type of d is ", type(res))


print("The type of d is ", type(add))
```

Explicit Type Conversion
In Explicit Type Conversion, users convert the data type of an object to required data type.

We use the built-in functions like int(), float(), str(), etc to perform explicit type conversion.

This type of conversion is also called typecasting because the user casts (changes) the data type of the objects.

```
x = 1 # int
y = 2.8 # float
z = 1j # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)
```

```
print(a)
print(b)
print(c)
```

Python Output

In Python, we can simply use the print() function to print output. For example,

```
print('Python is powerful')

# Output: Python is powerful
```

Syntax of print()

```
print(object= separator= end= file= flush=)
```

Here,

object - value(s) to be printed
sep (optional) - allows us to separate multiple objects inside print().
end (optional) - allows us to add add specific values like new line "\n", tab "\t"
file (optional) - where the values are printed. It's default value is sys.stdout (screen)
flush (optional) - boolean specifying if the output is flushed or buffered. Default: False

Example 1: Python Print Statement

```
print('Good Morning!')
print('It is rainy today’)
```

Example 2: Python print() with end Parameter

```
# print with end whitespace
print('Good Morning!', end= ' ')

print('It is rainy today’)
```

Example 3: Python print() with sep parameter

```
print('New Year', 2024, 'See you soon!', sep= '. ‘)
```

OUTPUT:

1. Good Morning!
   It is rainy today

2. Good Morning! It is rainy today
3. New Year. 2024. See you soon!

Output formatting

Sometimes we would like to format our output to make it look attractive. This can be done by using the str.format() method. For example,

```
x = 5
y = 10

print('The value of x is {} and y is {}'.format(x,y))
```

Here, the curly braces {} are used as placeholders. We can specify the order in which they are printed by using numbers (tuple index).

Python Input

While programming, we might want to take the input from the user. In Python, we can use the input() function.

Syntax of input()

```
input(prompt)
```

Here, prompt is the string we wish to display on the screen. It is optional.

```
# using input() to take user input
num = input('Enter a number: ')

print('You Entered:', num)

print('Data type of num:', type(num))
```

To convert user input into a number we can use int() or float() functions as:

```
num = int(input('Enter a number: '))
```

Python Indentation

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

Unlike many other programming languages, where flower braces {} or other mechanism is used to define scope or a block of code, Python does not have those. Instead, it uses indentation.

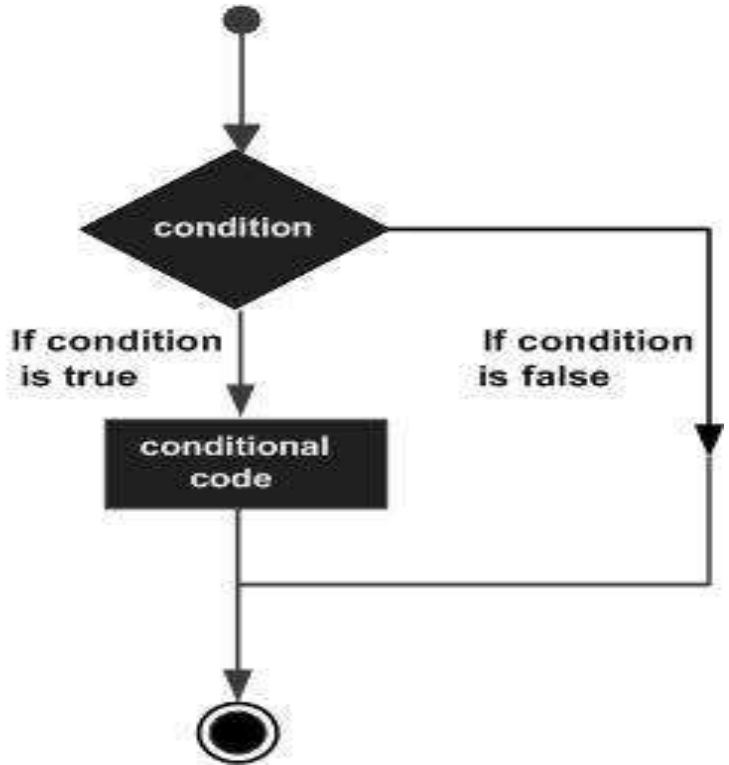
You can use any number of spaces for indentation. But, you have to provide same number of spaces for statement in a code block.

DECISION MAKING

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

Following is the general form of a typical decision making structure found in most of the programming languages



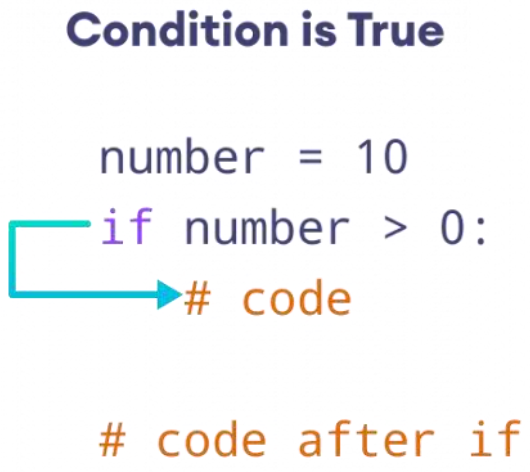
In Python, there are three forms of the if...else statement.

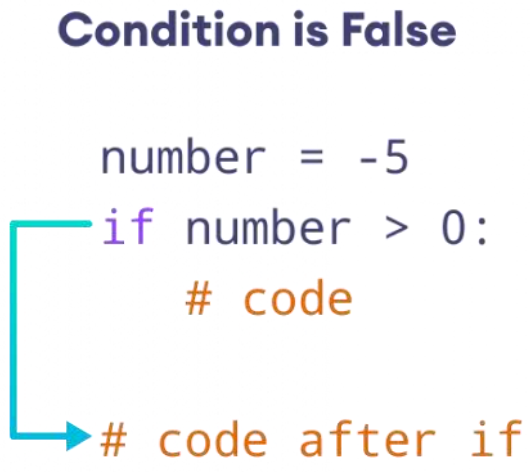
- if statement
- if...else statement
- if...elif...else statement

1. Python if statement
The syntax of if statement in Python is:

```
if condition:
    # body of if statement
```

The if statement evaluates condition.





If condition is evaluated to True, the code inside the body of if is executed.
If condition is evaluated to False, the code inside the body of if is skipped.

```
number = 10

# check if number is greater than 0
if number > 0:
    print('Number is positive.')

print('The if statement is easy')
```

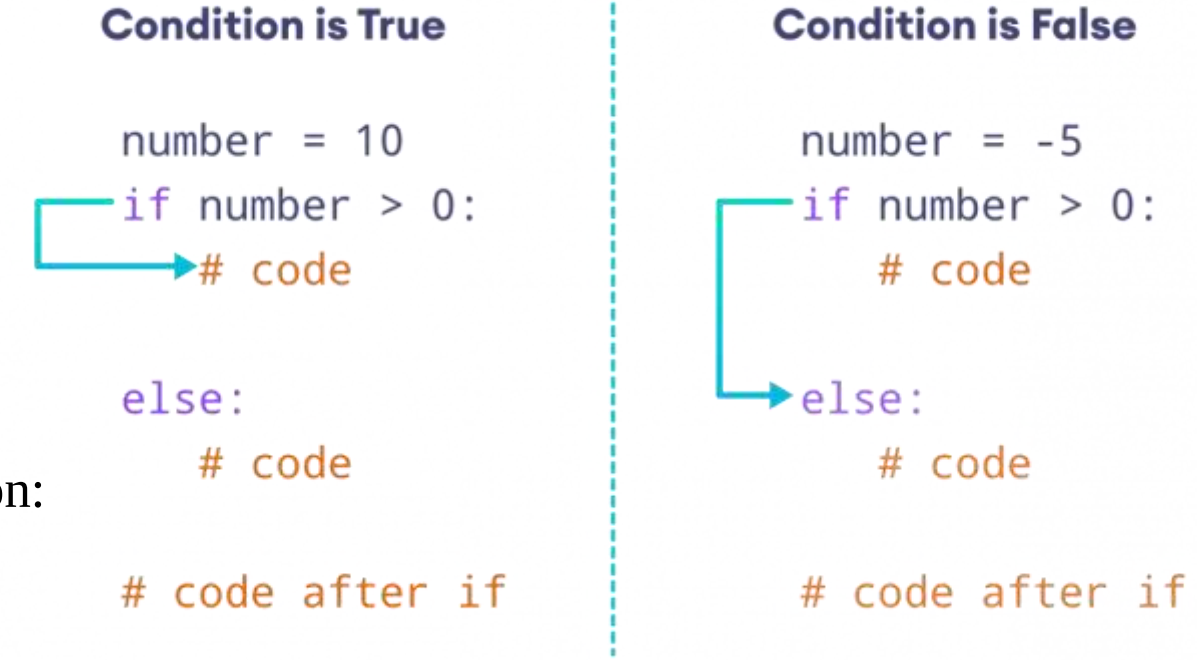
2. Python if...else Statement

An if statement can have an optional else clause.

The syntax of if...else statement is:

```
if condition:
    # block of code if condition is True

else:
    # block of code if condition is False
```

The if...else statement evaluates the given condition:



If the condition evaluates to True,

- the code inside if is executed
- the code inside else is skipped

If the condition evaluates to False,

- the code inside else is executed
- the code inside if is skipped

```
number = 10

if number > 0:
    print('Positive number')

else:
    print('Negative number')

print('This statement is always executed')
```

3. Python if...elif...else Statement

The if...else statement is used to execute a block of code among two alternatives.

However, if we need to make a choice between more than two alternatives, then we use the if...elif...else statement.

The syntax of the if...elif...else statement is:

```
if condition1:
    # code block 1

elif condition2:
    # code block 2

else:
    # code block 3
```

Here,

If condition1 evaluates to true, code block 1 is executed.
If condition1 evaluates to false, then condition2 is evaluated.

- If condition2 is true, code block 2 is executed.
- If condition2 is false, code block 3 is executed.

1st Condition is True

```
let number = 5
if number > 0 :
    # code

elif number < 0 :
    # code

else :
    # code


# code after if
```

2nd Condition is True

```
let number = -5
if number > 0 :
    # code

elif number < 0 :
    # code

else :
    # code


# code after if
```

All Conditions are False

```
let number = 0
if number > 0 :
    # code

elif number < 0 :
    # code

else :
    # code


# code after if
```

```
number = 0

if number > 0:
    print("Positive number")

elif number == 0:
    print('Zero')
else:
    print('Negative number')

print('This statement is always executed')
```

Python Nested if statements

We can also use an if statement inside of an if statement. This is known as a nested if statement.

The syntax of nested if statement is:

```
# outer if statement
if condition1:
    # statement(s)

    # inner if statement
    if condition2:
        # statement(s)
```

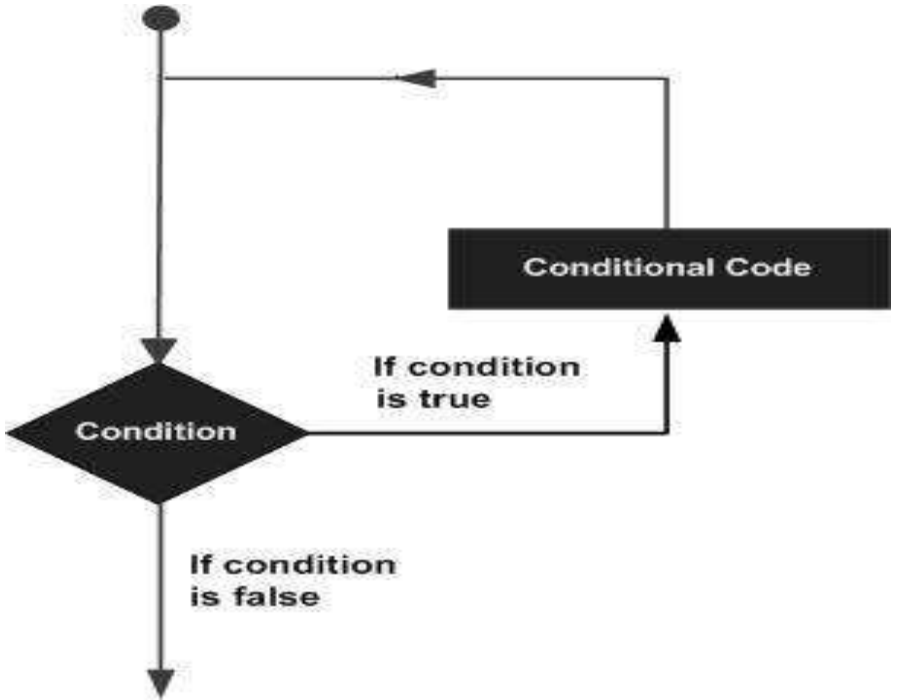
```
number = 5

# outer if statement
if (number >= 0):
    # inner if statement
    if number == 0:
      print('Number is 0')

    # inner else statement
    else:
        print('Number is positive')

# outer else statement
else:
    print('Number is negative')

# Output: Number is positive
```

LOOPING STATEMENTS

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement



There are 2 types of loops in Python:

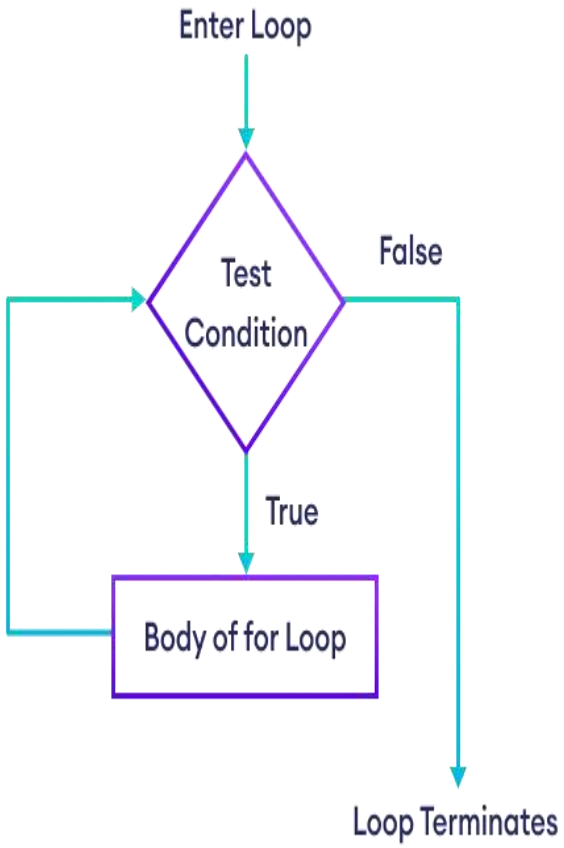
- for loop
- while loop

The for Loop

Python's for loop is designed to repeatedly execute a code block while iterating through a list, tuple, dictionary, or other iterable objects of Python. The process of traversing a sequence is known as iteration.

Syntax of the for Loop

```
for value in sequence:
    { code block }
```



EXAMPLE:

```
languages = ['Swift', 'Python', 'Go', 'JavaScript']

# run a loop for each item of the list
for language in languages:
    print(language)
```

Example: Loop Through a String

```
for x in 'Python':
    print(x)
```

Python for Loop with Python range()

A range is a series of values between two numeric intervals.We use Python's built-in function range() to define a range of values. For example,

values = range(4)
Here, 4 inside range() defines a range containing values 0, 1, 2, 3.

In Python, we can use for loop to iterate over a range. For example,

```
# use of range() to define a range of values
values = range(4)
# iterate from i = 0 to i = 3
for i in values:
    print(i)
```

Using a for Loop Without Accessing Items

It is not mandatory to use items of a sequence within a for loop. For example,

```
languages = ['Swift', 'Python', 'Go']

for language in languages:
    print('Hello')
    print('Hi’)
```

Python for loop with else

A for loop can have an optional else block. The else part is executed when the loop is exhausted (after the loop iterates through every item of a sequence). For example,

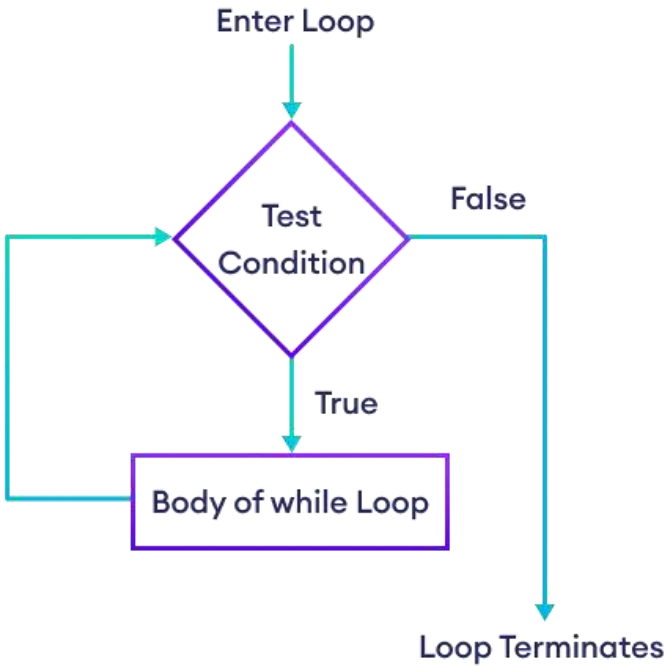
```
digits = [0, 1, 5]

for i in digits:
    print(i)
else:
    print("No items left.")
```

Python while Loop

Python while loop is used to run a block code until a certain condition is met.

The syntax of while loop is:

```
while condition:
    # body of while loop
```



Example 1: Python while Loop

```
# program to display numbers from 1 to 5

# initialize the variable
i = 1
n = 5

# while loop from i = 1 to 5
while i <= n:
    print(i)
    i = i + 1
```

Example 2: Python while Loop

```
# program to calculate the sum of numbers
# until the user enters zero

total = 0

number = int(input('Enter a number: '))

# add numbers until number is zero
while number != 0:
    total += number    # total = total + number

    # take integer input again
    number = int(input('Enter a number: '))


print('total =', total)
```

Infinite while Loop in Python

If the condition of a loop is always True, the loop runs for infinite times (until the memory is full). For example,

```
age = 32

# the test condition is always True
while age > 18:
    print('You can vote')
```

Python While loop with else

In Python, a while loop may have an optional else block.Here, the else part is executed after the condition of the loop evaluates to False.

```
counter = 0
while counter < 3:
    print('Inside loop')
    counter = counter + 1
else:
    print('Inside else')
```

Python break Statement

The break statement is used to terminate the loop immediately when it is encountered.

The syntax of the break statement is:

break

```
for val in sequence:
    # code
    if condition:
        break

    # code
```

```
while condition:
    # code
    if condition:
        break

    # code
```

Python break Statement with for Loop

We can use the break statement with the for loop to terminate the loop when a certain condition is met.
For example,

```
for i in range(5):
    if i == 3:
        break
    print(i)
```

Python break Statement with while Loop

We can also terminate the while loop using the break statement. For example,

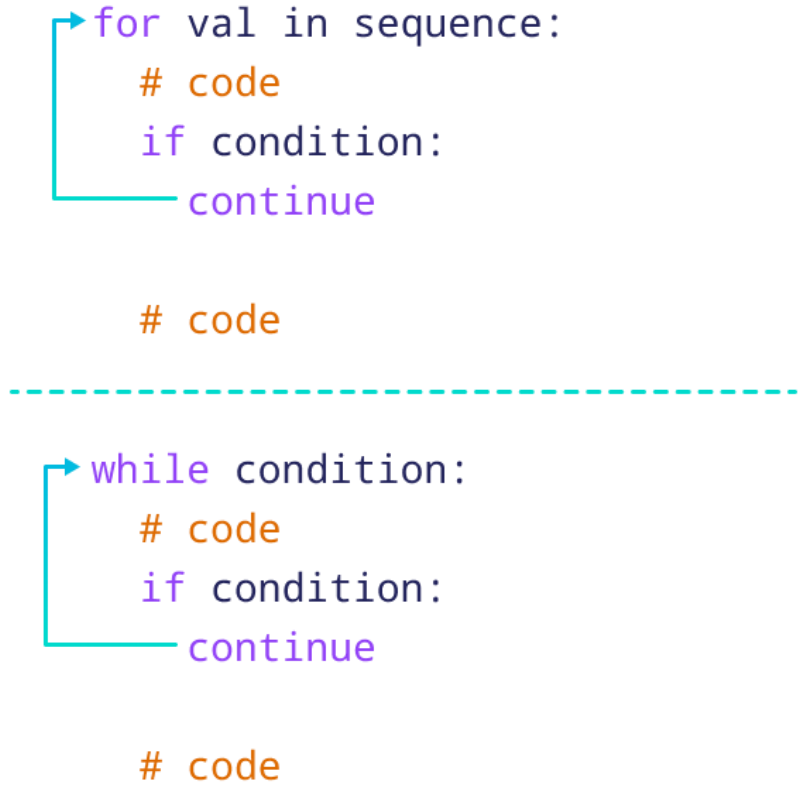
```
# program to find first 5 multiples of 6

i = 1

while i <= 10:
    print('6 * ',(i), '=',6 * i)

    if i >= 5:
        break

    i = i + 1
```

Python continue Statement

The continue statement is used to skip the current iteration of the loop and the control flow of the program goes to the next iteration.

The syntax of the continue statement is:

Continue



Python continue Statement with for Loop

We can use the continue statement with the for loop to skip the current iteration of the loop. Then the control of the program jumps to the next iteration. For example,

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

Python continue Statement with while Loop

In Python, we can also skip the current iteration of the while loop using the continue statement. For example,

```
# program to print odd numbers from 1 to 10
num = 0
while num < 10:
    num += 1
        if (num % 2) == 0:
        continue
    print(num)
```

PASS

In Python programming, the pass statement is a null statement which can be used as a placeholder for future code.

Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future. In such cases, we can use the pass statement.

The syntax of the pass statement is:

pass

```
Using pass With Conditional Statement
n = 10

# use pass inside if statement
if n > 10:
    pass

print('Hello')
```