



**SATHYABAMA**

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UNIT- I - Python Programming – SCS1619**

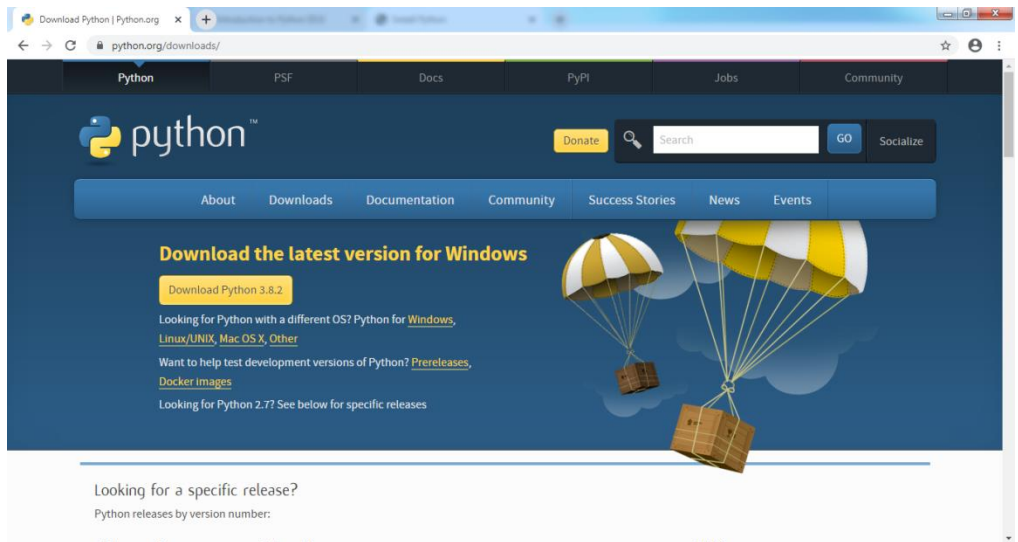
# UNIT I

## INTRODUCTION

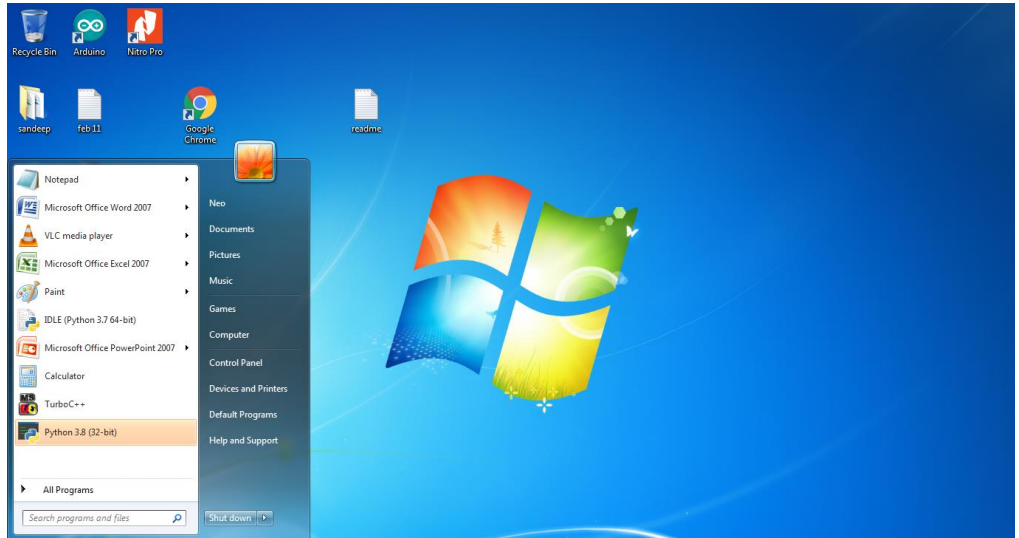
Introduction to the IDLE interpreter (shell) -Expressions – Data Types  
- Built-in function -Conditional statements - Iterative statements-  
Input/output -Compound Data Types - Nested compound statements –  
Introduction to Object Oriented Concepts.

### 1.1 INTRODUCTION TO THE IDLE INTERPRETER (SHELL)

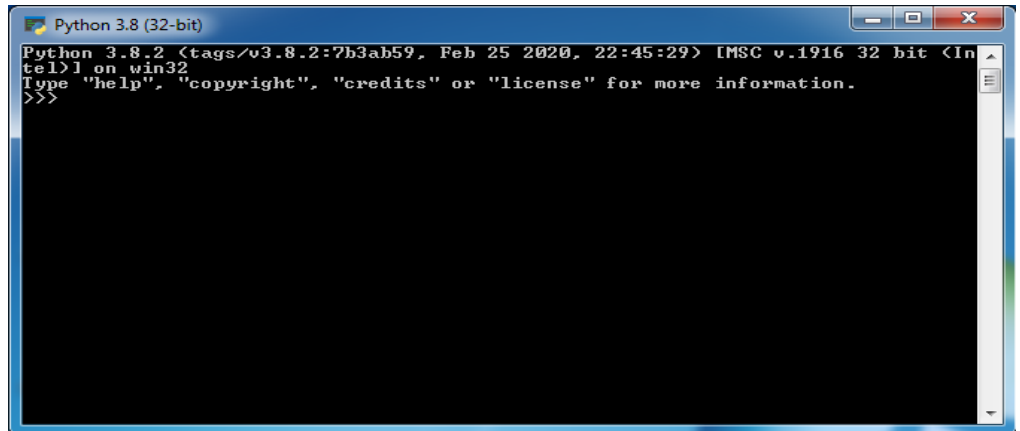
Python is a freeware that can be installed on your workstation or laptop. The current version of is Python 3.8.2. (Release date: Feb 24, 2020). Python can be downloaded from the <https://www.python.org/downloads/> website



After installation, shortcut for the python software will be available on the start menu. Select Start menu -> All Programs -> Python 3.8 -> IDLE Python 3.8 to invoke python IDE (Integrated Development Environment).



IDLE (Integrated Development and Learning Environment) is an IDE for Python. Python programs can also be executed in Python command line.



### 1.1.2 Program

A program performs a task in the computer. But, in order to be executed, a program must be written in the machine language of the processor of a computer. Unfortunately, it is extremely difficult for humans to read or write a machine language program. This is because a machine language is entirely made up of sequences of bits. However, high level languages are close to natural languages like English and only use familiar mathematical characters, operators and expressions. Hence, people prefer to write programs in high level languages like C, C++, Java, or Python. A high level program is translated into machine language by translators like compiler or interpreter.

### 1.1.3 About Python

Python is a high level programming language that is translated by the python **interpreter**. As is known, an interpreter works by translating line-by-line and executing. It was developed by Guido-van-rossum in 1990, at the National Research Institute for Mathematics and Computer Science in

Netherlands. Python doesn't refer to the snake but was named after the famous British comedy troupe, Monty Python's Flying Circus. Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it.

Python is Interactive: we can actually sit at a Python prompt and interact with the interpreter directly to write our programs.

Python is Object-Oriented: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Application of python used in Search engine. In mission critical projects in Naza, in processing financial transaction at New york stock Exchange.

The following are some of the features of Python:

- Python is an Open Source: It is freely downloadable, from the link "[http:// python.org/](http://python.org/)"
- Python is portable: It runs on different operating systems / platforms
- Python has automatic memory management
- Python is flexible with both procedural oriented and object oriented programming
- Python is easy to learn, read and maintain
- Python is Extendable. You can add low-level modules to the Python is Interpreted. These modules enable programmers to add to or customize their tools to be more efficient.

- Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.
- It supports functional and structured programming methods as well as OOP.

It is very flexible with the console program, Graphical User Interface (GUI) applications, Web related programs etc.

### Points to remember while writing a Python program

- Case sensitive : Example - In case of print statement use only lower case and not upper case, (See the snippet below)

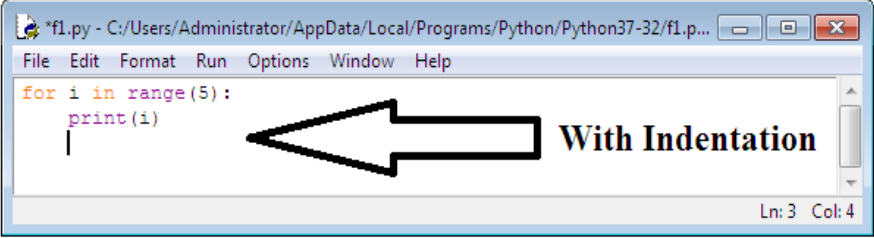
```

>>> print("hello") ← Valid
hello
>>> Print("hello") ← Invalid
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined
>>>

```

- Punctuation is not required at end of the statement
- In case of string use single or double quotes i.e. ' ' or " "

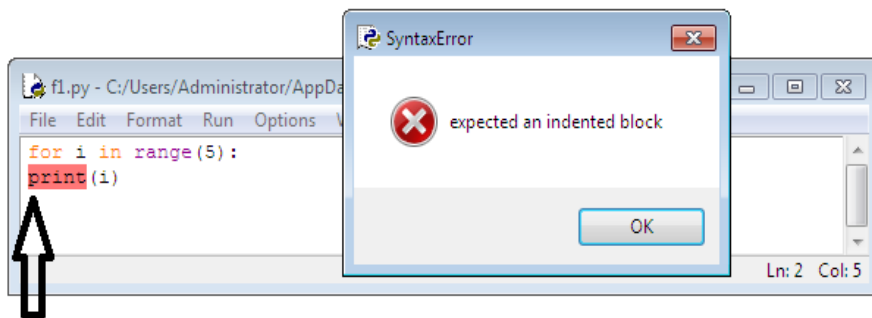
- Must use proper indentation. The screen shots given below show, how the value of “i” behaves with indentation and without indentation.



A screenshot of a Python IDE window titled "f1.py - C:/Users/Administrator/AppData/Local/Programs/Python/Python37-32/f1.p...". The code in the editor is:

```
for i in range(5):  
    print(i)
```

A large black arrow points from the text "With Indentation" to the indented line of code. The status bar at the bottom right shows "Ln: 3 Col: 4".



**Without Indentation**

- Special characters like (,),# etc. are used
- () ->Used in opening and closing parameters of functions
- #-> The Pound sign is used to comment a line

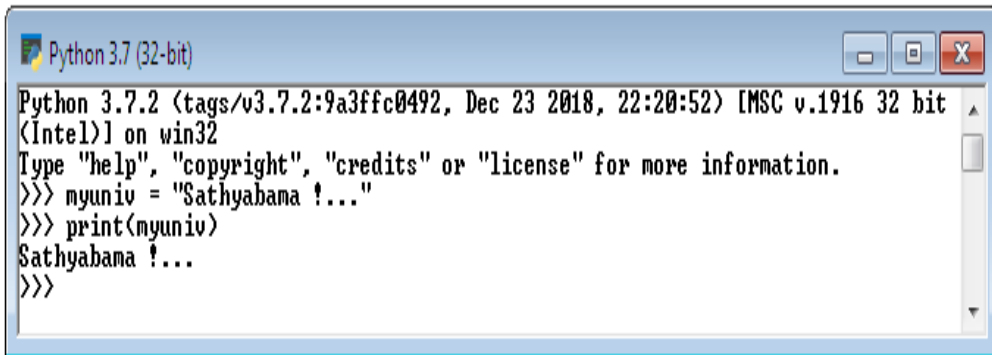
### 1.1.4 Two Modes of Python Program

Python Program can be executed in two different modes:

- Interactive mode programming
- Script mode programming

## Interactive Mode Programming

It is a command line shell which gives immediate output for each statement, while keeping previously fed statements in active memory. This mode is used when a user wishes to run one single line or small block of code. It runs very quickly and gives instant output. A sample code is executed using interactive mode as below.

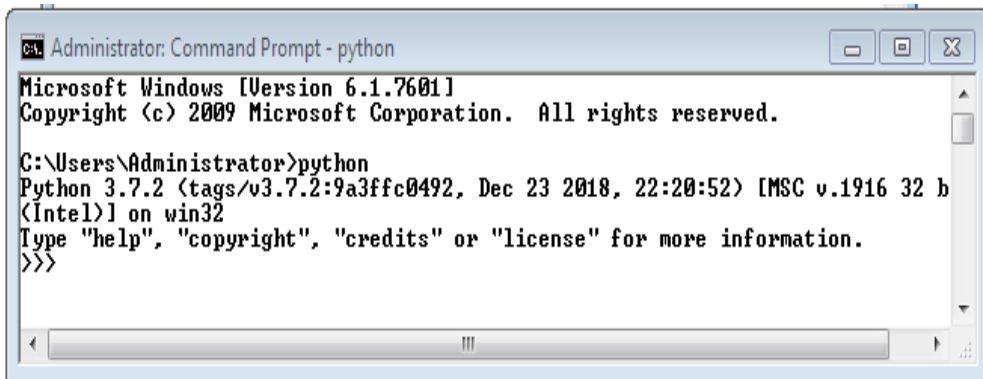
A screenshot of a Windows-style window titled "Python 3.7 (32-bit)". The window contains the Python 3.7.2 interactive shell. The text inside the window is as follows:

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit  
<Intel>] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> myuniv = "Sathyabama !..."  
>>> print(myuniv)  
Sathyabama !...  
>>>
```

Interactive mode can also be opened using the following ways:

- From command prompt `c :> users\\...>python`



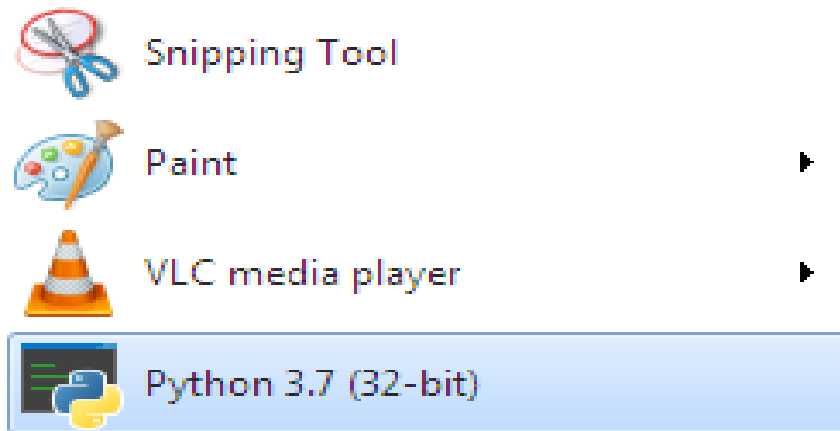


```
Administrator: Command Prompt - python
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 b
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The symbol “>>>” in the above screen indicates that the Python environment is in interactive mode.

ii) From the start menu select Python (As shown below)



Python interpreter in interactive mode is commonly known as Python Shell. >>> is the prompt for Python shell. It shows that shell is ready to accept your commands. Python shell allows you to type Python

code and see the result immediately. It is also known as REPL which stands Read-Eval-Print-Loop. REPL allows you to quickly test code snippets and see the output immediately. To quit the Python shell in Python Command line, hit Ctrl+Z followed by the Enter Key. To quit the Python shell in Idle, press Ctrl+Q.

## Script Mode Programming

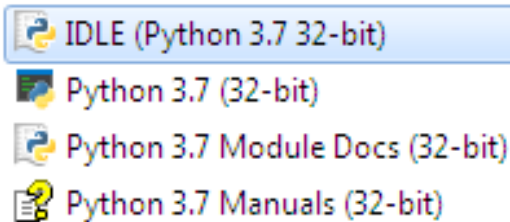
When the programmer wishes to use more than one line of code or a block of code, script mode is preferred. The Script mode works the following way:

- i) Open the Script mode
- ii) Type the complete program. Comment, edit if required.
- iii) Save the program with a valid name.
- iv) Run
- v) Correct errors, if any, Save and Run until proper output

The above steps are described in detail below:

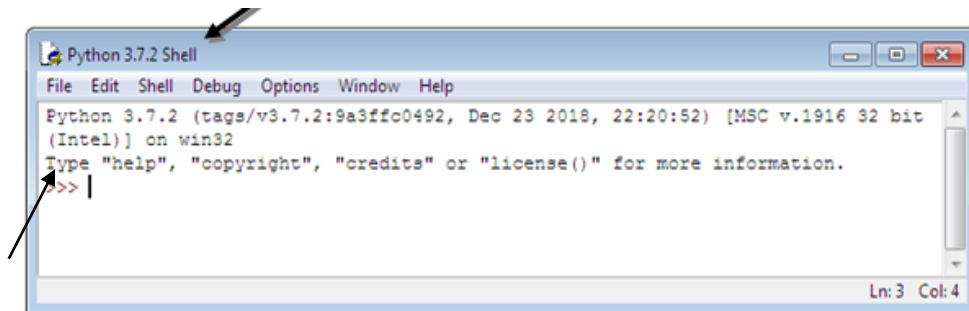
- i) To open script mode, select the menu “***IDLE (Python 3.7 32-bit)***” from start menu

## Programs (4)



## Documents (24)

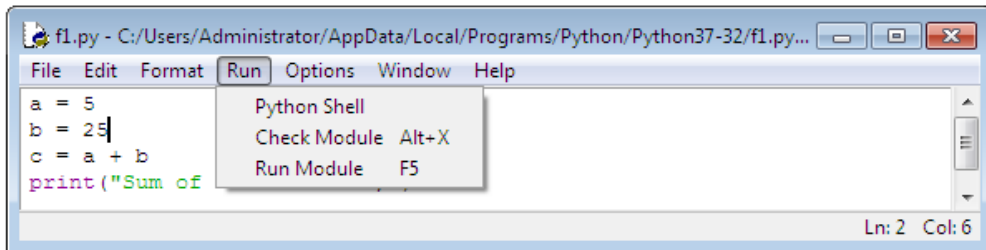
- ii) After clicking on the menu “**IDLE (Python 3.7 32-bit)**”, a new window with the text Python 3.7.2 Shell will be opened as shown below:



- iii) Select File → New, to open editor. Type the complete program.  
iv) Select File again; Choose Save.

This will automatically save the file with an extension “.py”.

- v) Select Run → Run Module or Short Cut Key **F5** ( *As shown in the screen below* )



The output of the program will be displayed as below:

```
>> Sum of a and b is: 30
```

Script mode is used to create, modify and execute Python programs. Script mode is used for executing a set of statements at any time and any number of times. The set of statements can be saved in a file with extension .py which can be executed at later time. The Python Interpreter in Script mode is used to execute the python code from a file.

## 1.2 VARIABLES

Variable is the name given to a reserved memory locations to store values. It is also known as Identifier in python.

### *Need for variable:*

Sometimes certain parameters will take different values at different time. Hence, in order to know the current value of such parameter we need to have a temporary memory which is identified by a name that name is called as

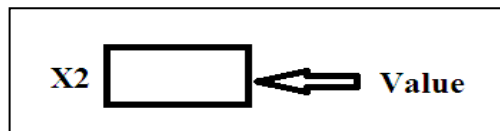
variable. For example, our surrounding temperature changes frequently. In order to know the temperature at a particular time, we need to have a variable.

### *Naming and Initialization of a variable*

1. A variable name is made up of alphabets (Both upper and lower cases) and digits and is case sensitive
2. No reserved words
3. Initialize before calling
4. Multiple variables initialized
5. Dynamic variable initialization

Consist of upper and lower case alphabets, Numbers (0-9). E.g. X2

- i. In the above example, a memory space is assigned to variable X2. The value of X2 is stored in this space.

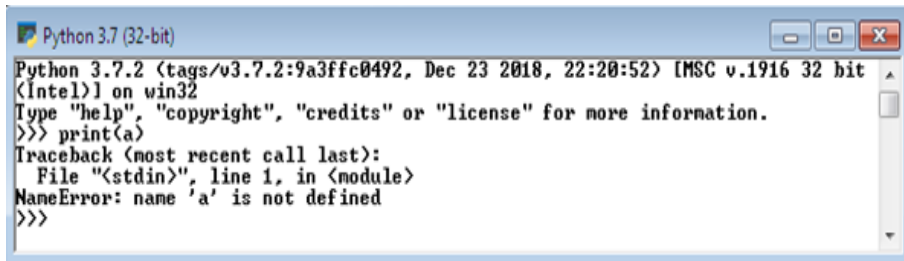


- ii. Reserved words should not be used as variables names.

```
Python 3.7 (32-bit)
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> # Valid Variable
>>> x2 = 25
>>> print(x2)
25
>>> # Invalid variable
>>> and = 25
File "<stdin>", line 2
    and = 25
    ^
SyntaxError: invalid syntax
```

In the above example “and” is a reserved word, which leads to Syntax error

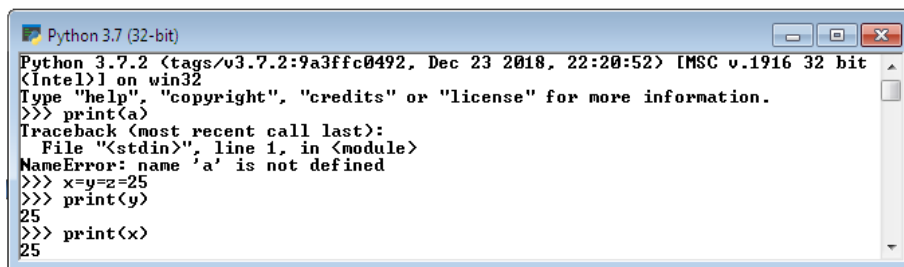
- iii. Variables must be initialized before it called , else it reports “is not defined ” error message as below E.g.: a=5 print(a)



```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit  
<Intel>] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print(a)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'a' is not defined  
>>>
```

In the above example “a” is called before it initialized. Hence, the python interpreter generates the error message: NameError: ‘a’ is not defined.

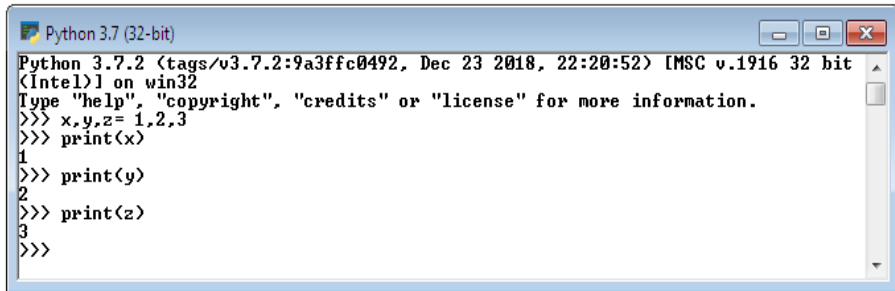
- iv. Multiple variables can be initialized with a common value.  
E.g.: x=y=z=25



```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit  
<Intel>] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print(a)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'a' is not defined  
>>> x=y=z=25  
>>> print(y)  
25  
>>> print(x)  
25
```

In the above three variables x, y, z is assigned with same value 25.

v. Python also supports dynamic variable initialization. E.g.:  
`x,y,z=1,2,3`



```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit  
<Intel>] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> x,y,z= 1,2,3  
>>> print(x)  
1  
>>> print(y)  
2  
>>> print(z)  
3  
>>>
```

Proper spacing should be given

- `print(10+20+30)` → bad style
- `print(20 + 30 + 10)` → good style

### 1.2.1 Expression

An expression is a combination of variables, operators, values and calls to functions. Expressions need to be evaluated.

#### *Need for Expression:*

Suppose if you wish to calculate area. Area depends on various parameters in different situations. E.g. Circle, Rectangle and so on...

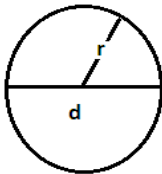
In order to find area of circle, the expression  $\pi * r * r$  must be evaluated and for the rectangle the expression is  $w * l$  in case of rectangle. Hence, in this case a variable / value / operator are not enough to handle such situation. So

expressions are used. Expression is the combination of variables, values and operations.

A simple example of an expression is  $10 + 15$ . An expression can be broken down into operators and operands. Few valid examples are given below.

### Circle

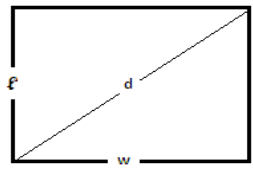
Solution For Area ▾

$$A = \pi r^2$$


$r$  Radius

### Rectangle

Solve for area ▾

$$A = w l$$


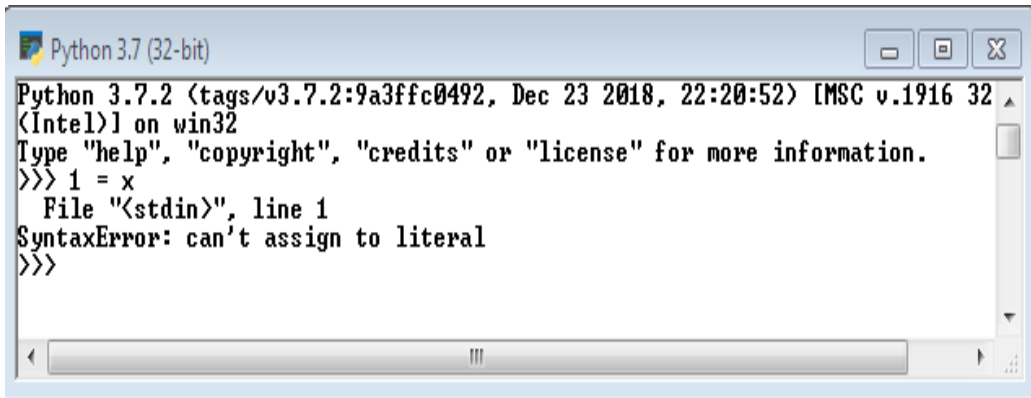
$l$  Length   
 $w$  Width

```
Python 3.7 (32-bit)
Python 3.7.2 <tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52> [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> # Example1
>>> diameter = 25.0
>>> radius = diameter / 2
>>> print (radius)
12.5
>>> # Example2
>>> i = 25 * (3/2) + 5 * 10
>>> print(i)
87.5
>>> # Example3
>>> area = radius * radius * 3.14
>>> print(area)
490.625
>>> # Example4
>>> 5 + 25
30
>>>
```

### *Invalid Expression*



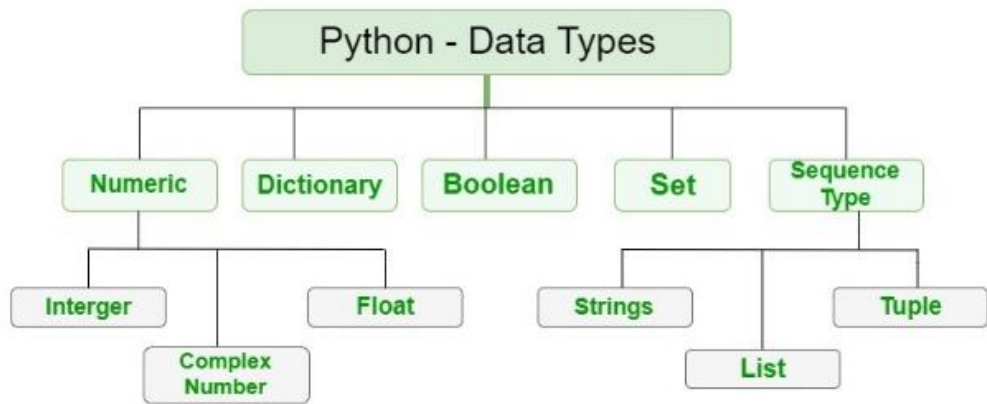
Always values should be assigned in the right hand side of the variable, but in the below example, the value is given in the left hand side of the variable, which is an invalid syntax for expression.

A screenshot of a Windows command prompt window titled "Python 3.7 (32-bit)". The window shows the Python 3.7.2 shell interface. The prompt displays the version and architecture: "Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 (Intel)] on win32". It then shows the prompt "Type 'help', 'copyright', 'credits' or 'license' for more information." followed by the user input ">>> 1 = x". The shell responds with an error: "File '<stdin>', line 1\nSyntaxError: can't assign to literal\n>>>". The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
Python 3.7 (32-bit)
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 = x
File "<stdin>", line 1
SyntaxError: can't assign to literal
>>>
```

### 1.3 DATA TYPES

A Data type indicates which type of value a variable has in a program. However a python variables can store data of any data type but it is necessary to identify the different types of data they contain to avoid errors during execution of program. The most common data types used in python are str(string), int(integer) and float (floating-point).



***Fig.1.1: Python Data Types***

### **1.3.1 Numeric**

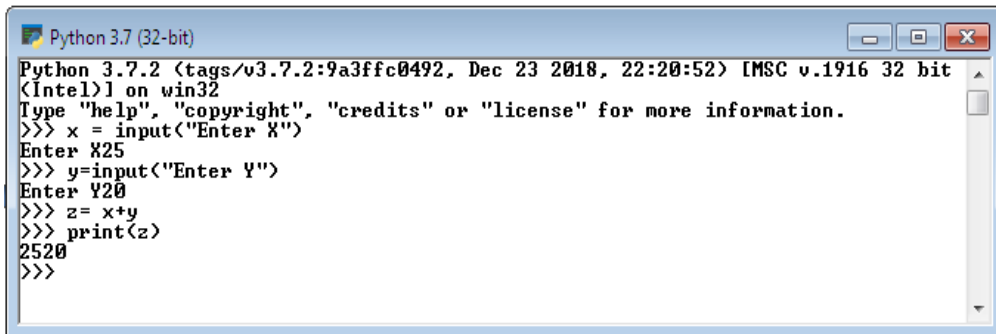
Numeric data type means the data will have numeric value.

Numeric value can be integer, floating number or even complex numbers.

- Integers – Int class is used to represent integers which may be a positive whole number or negative whole number. There is no restriction on limit for the value of integer. Integers are whole number values such as 50, 100, -3
- Float – Float class is used to represent floating point number which is a real number with floating point representation. It is specified by a decimal point. Float is a value that uses decimal point and therefore may have fractional point E.g.: 3.415, -5.15

- Complex Numbers – Complex class is used to represent Complex number specified as (real part) + (imaginary part)*j*. For example:  $5 + 7j$  where 5 is the real part and 7 is the imaginary part.

By default when a user gives input it will be stored as string. But strings cannot be used for performing arithmetic operations. For example while attempting to perform arithmetic operation add on string values it just concatenates (joins together) the values together rather performing addition. For example :  $'25' + '20' = '45'$  (As in the below Example)



```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit  
(Intel)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> x = input("Enter X")  
Enter X25  
>>> y=input("Enter Y")  
Enter Y20  
>>> z= x+y  
>>> print(z)  
2520  
>>>
```

Fortunately python have an option of converting one data type into another data type (Called as “Casting”) using build in functions in python. The build function `int()` converts the string into integer before performing operation to give the right answer. (As in the below Program)

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x = input("Enter A")
Enter A 25
>>> y=input("Enter B")
Enter B 2
>>> z = int(x) + int(y)
>>> print("Sum is", z)
Sum is 27
>>>
```

PROGRAM:

```
# Python program to demonstrate numeric value
a = 10
print("Type of a: ", type(a))
b = 20.0
print("\nType of b: ", type(b))
c = 5 + 7j
print("\nType of c: ", type(c))
```

Output:

```
Type of a: <class 'int'>
Type of b: <class 'float'>
Type of c: <class 'complex'>
```

### 1.3.2 Boolean

The Boolean data type has two built-in values True or False. It is denoted by the class bool.

Note – True and False with capital ‘T’ and ‘F’ are valid booleans value. otherwise python will throw an error.

PROGRAM:

```
# Python program to demonstrate boolean type
print(type(True))
print(type(False))
print(type(true))
```

Output:

```
<class 'bool'>
```

```
<class 'bool'>
```

Traceback (most recent call last):

File "/home/jesu/boolean.py", line 9, in

```
print(type(true))
```

NameError: name 'true' is not defined

### 1.3.3 Sequence Type

A sequence is an ordered collection of similar or different data items. Using sequence, Multiple values can be stored in the data type in an efficient manner. There are different types of sequence data type such as

- i) Strings
- ii) List
- iii) Tuple
- i) Strings

### **1.3.3.1 *String***

String is an array of bytes. Each byte represents a Unicode character. A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by str class.

Individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character and so on. Only Integers are allowed to be passed as an index, float or other types will cause a TypeError.

Updation or deletion of characters from a String is not allowed. This will cause an error because item assignment or item deletion from a String is not supported. This is because Strings are immutable, hence elements of a String cannot be changed once it has been assigned. Only new strings can be reassigned to the same name.

Strings: Sequence of characters inside single quotes or double quotes.

E.g. myuniv = "Sathyabama !.."

## PROGRAM:

```
# Python Program for String Manipulation
# Creating a String with single Quotes, double quotes, tripple quotes
String1 = 'Welcome'
String2 = "Sathyabama"
String3 = ""CSE""
# Triple Quotes allows multiple lines
String4 = ""Welcome
    To
        Sathyabama""
print("\nUsing Single quote")
print(String1)
print("\nUsing Double quote")
print(String2)
print("\nUsing Triple quote")
print(String3)
print("\nUsing Triple quote to print multiline")
print(String4)

#printing first character
print("\nPrint First Character")
print(String1[0])

#printing last character
print("\nPrint Last Character")
print(String1[-1])
```

```
#updating a single character
#String1[2] = 'p'
#Cannot Update because strings are immutable
```

```
# Deleting a character of the String
#del String1[2]
#Cannot Delete because strings are immutable
```

```
# Escaping Single Quote
String1 = 'I\'m "Trying"'
print("\nEscaping Single Quote: ")
print(String1)
```

```
# Escaping Double Quotes
String1 = "I'm a \"Trying\""
print("\nEscaping Double Quotes: ")
print(String1)
```

```
# Printing Paths with the
# use of Escape Sequences
String1 = "C:\\Python\\programs\\"
print("\nEscaping Backslashes: ")
print(String1)
```

Output:  
Using Single quote  
Welcome



Using Double quote  
Sathyabama

Using Triple quote  
CSE

Using Triple quote to print multiline  
Welcome  
    To  
        Sathyabama

Print First Character  
W

Print Last Character  
e

Escaping Single Quote:  
I'm "Trying"

Escaping Double Quotes:  
I'm a "Trying"

Escaping Backslashes:  
C:\Python\programs\  
>>>

### 1.3.3.2 List

The List is an ordered sequence of data items. It is one of the flexible and very frequently used data type in Python. All the items in a list are not necessary to be of the same data type.

Declaring a list is straight forward methods. Items in the list are just separated by commas and enclosed within brackets [ ].

```
>>> list1 =[3.141, 100, 'CSE', 'ECE', 'IT', 'EEE']
```

Methods used in list

list1.append(x)	To add item x to the end of the list “list1”
list1.reverse()	Reverse the order of the element in the list “list1”
list1.sort()	To sort elements in the list
list1.reverse()	To reverse the order of the elements in list1.

Lists are similar to arrays but they are homogeneous always which makes it the most powerful tool in Python. A single list may contain Data Types like Integers, Strings, as well as Objects. Lists are mutable. Hence, they cannot be modified once created. Lists are ordered and have definite count. The list index starts with 0. Duplication of elements is possible in list. The lists are implemented by list class.

Lists in Python can be created by just placing the sequence inside the square brackets[]. Unlike Sets, list doesn't need a built-in function for creation of list.

## PROGRAM:

```
# Python program to demonstrate List
```

```
List = []
```

```
print("Initial blank List: ")
```

```
print(List)
```

```
# Creating a List with the use of a String
```

```
List = ['Welcome To Sathyabama']
```

```
print("\nList with the use of String: ")
```

```
print(List)
```

```
# Creating a List with the use of multiple values
```

```
List = ["Welcome", "To", "Sathyabama"]
```

```
print("\nList containing multiple values: ")
```

```
print(List[0])
```

```
print(List[2])
```

```
# Creating a Multi-Dimensional List (By Nesting a list inside a List)
```

```
List = [['Welcome', 'To'], ['Sathyabama']]
```

```
print("\nMulti-Dimensional List: ")
```

```
print(List)
```

```
# Addition of Elements
```

```
# in the List
```

```
List.append(1)
```

```
List.append(2)
```

```
List.append(4)
```

```
print("\nList after Addition of Three elements: ")
print(List)
```

```
# Addition of elements in a List
```

```
# Creating a List
```

```
List = []
print("Initial blank List: ")
print(List)
```

```
# Addition of Elements
```

```
# in the List
```

```
List.append(1)
List.append(2)
List.append(4)
print("\nList after Addition of Three elements: ")
print(List)
```

```
# Addition of Element at
```

```
# specific Position
```

```
# (using Insert Method)
```

```
List.insert(3, 12)
List.insert(0, 'Sathyabama')
print("\nList after performing Insert Operation: ")
print(List)
```

```
# Addition of multiple elements
```

```
# to the List at the end
```

```
# (using Extend Method)
List.extend([8, 'Sathyabama', 'Always'])
print("\nList after performing Extend Operation: ")
print(List)
```

```
# Python program to demonstrate
# accessing of element from list
```

```
# Creating a List with
# the use of multiple values
List = ["Welcome", "To", "Sathyabama"]
```

```
# accessing a element from the
# list using index number
print("\nAccessing element from the list")
print(List[0])
print(List[2])
```

```
# accessing a element using
# negative indexing
print("\nAccessing element using negative indexing")
```

```
# print the last element of list
print(List[-1])
```

```
# print the third last element of list
print(List[-3])
```

```
List = [1, 2, 3, 4, 5, 6,  
        7, 8, 9, 10, 11, 12]  
print("\nIntial List: ")  
print(List)
```

```
# Removing elements from List  
# using Remove() method  
List.remove(3)  
List.remove(4)  
print("\nList after Removal of two elements: ")  
print(List)
```

```
List.pop()  
print("\nList after popping an element: ")  
print(List)
```

```
# Removing element at a  
# specific location from the  
# Set using the pop() method  
List.pop(2)  
print("\nList after popping a specific element: ")  
print(List)
```

OUTPUT:

Initial blank List:

```
[]
```

List with the use of String:

```
['Welcome To Sathyabama']
```

List containing multiple values:

```
Welcome
```

```
Sathyabama
```

Multi-Dimensional List:

```
[['Welcome', 'To'], ['Sathyabama']]
```

List after Addition of Three elements:

```
[['Welcome', 'To'], ['Sathyabama'], 1, 2, 4]
```

Initial blank List:

```
[]
```

List after Addition of Three elements:

```
[1, 2, 4]
```

List after performing Insert Operation:

```
['Sathyabama', 1, 2, 4, 12]
```

List after performing Extend Operation:

```
['Sathyabama', 1, 2, 4, 12, 8, 'Sathyabama', 'Always']
```

Accessing element from the list

Welcome  
Sathyabama

Accessing element using negative indexing  
Sathyabama  
Welcome

Initial List:  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

List after Removal of two elements:  
[1, 2, 5, 6, 7, 8, 9, 10, 11, 12]

List after popping an element:  
[1, 2, 5, 6, 7, 8, 9, 10, 11]

List after popping a specific element:  
[1, 2, 6, 7, 8, 9, 10, 11]  
>>>

### ***1.3.3.3 Tuple***

Tuple is also an ordered sequence of items of different data types like list. But, in a list data can be modified even after creation of the list whereas Tuples are immutable and cannot be modified after creation.

The advantages of tuples is to write-protect data and are usually very fast when compared to lists as a tuple cannot be changed dynamically.



The elements of the tuples are separated by commas and are enclosed inside open and closed brackets.

```
>>> t = (50,'python', 2+3j)
```

List	Tuple
<pre>&gt;&gt;&gt; list1[12,45,27] &gt;&gt;&gt; list1[1] = 55 &gt;&gt;&gt; print(list1) &gt;&gt;&gt; [12,55,27]</pre>	<pre>&gt;&gt;&gt; t1 = (12,45,27) &gt;&gt;&gt; t1[1] = 55 &gt;&gt;&gt; Generates Error Message # Because Tuples are immutable</pre>

The values stored in a tuple can be of any type, and they are indexed by integers. The important difference between a list and a tuple is that tuples are immutable. Tuples are hashable whereas lists are not. It is represented by tuple class.

In Python, tuples are created by placing sequence of values separated by ‘comma’ with or without the use of parentheses for grouping of data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, list, etc.). Tuples can also be created with a single element, but it is a bit tricky. Having one element in the parentheses is not sufficient, there must be a trailing ‘comma’ to make it a tuple.

Note – Creation of Python tuple without the use of parentheses is known as Tuple Packing.

#### PROGRAM:

```
# Python program to demonstrate Set
```

```
# Creating an empty tuple
```

```
Tuple1 = ()
```

```
print("Initial empty Tuple: ")
```

```
print (Tuple1)
```

```
# Creating a Tuple with the use of Strings
```

```
Tuple1 = ('Welcome', 'Sathyabama')
```

```
print("\nTuple with the use of String: ")
```

```
print(Tuple1)
```

```
# Creating a Tuple with the use of list
```

```
list1 = [1, 2, 4, 5, 6]
```

```
print("\nTuple using List: ")
```

```
print(tuple(list1))
```

```
# Creating a Tuple with the use of built-in function
```

```
Tuple1 = tuple('Sathyabama')
```

```
print("\nTuple with the use of function: ")
```

```
print(Tuple1)
```

```
# Creating a Tuple with nested tuples
```

```
Tuple1 = (0, 1, 2, 3)
```

```
Tuple2 = ('python', 'program')
Tuple3 = (Tuple1, Tuple2)
print("\nTuple with nested tuples: ")
print(Tuple3)
```

```
# demonstrate accessing tuple
tuple1 = tuple([1, 2, 3, 4, 5])
```

```
# Accessing element using indexing
print("Frist element of tuple")
print(tuple1[0])
```

```
# Accessing element from last
# negative indexing
print("\nLast element of tuple")
print(tuple1[-1])
```

```
print("\nThird last element of tuple")
print(tuple1[-3])
```

```
# demonstrate updation / deletion from a tuple
tuple1 = tuple([1, 2, 3, 4, 5])
print("Initial tuple")
print(tuple1)
```

```
# Updating an element of a tuple is not possible as it is immutable
#tuple1[0] = -1
```

```
# Deleting an element from a tuple is not possible as it is immutable
#del tuple1[2]
```

OUTPUT:

Initial empty Tuple:

()

Tuple with the use of String:

('Welcome', 'Sathyabama')

Tuple using List:

(1, 2, 4, 5, 6)

Tuple with the use of function:

('S', 'a', 't', 'h', 'y', 'a', 'b', 'a', 'm', 'a')

Tuple with nested tuples:

((0, 1, 2, 3), ('python', 'program'))

Frist element of tuple

1

Last element of tuple

5

Third last element of tuple

3

35

Initial tuple  
(1, 2, 3, 4, 5)  
>>>

#### 1.3.3.4 Set

The Set is an unordered collection of unique data items. Items in a set are not ordered, separated by comma and enclosed inside { } braces. Sets are helpful in performing operations like union and intersection. However, indexing is not done because sets are unordered.

List	Set
>>> L1 = [1,20,25] >>> print(L1[1]) >>> 20	>>> S1= {1,20,25,25} >>> print(S1) >>> {1,20,25} >>> print(S1[1]) >>>Error , Set object does not support indexing.

It is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements. The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set. Type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

## PROGRAM:

# Python program to demonstrate Set in Python

```
set1 = set()
```

```
print("Initial blank Set: ")
```

```
print(set1)
```

# Creating a Set with the use of a String

```
set1 = set("Welcome to Python")
```

```
print("\nSet with the use of String: ")
```

```
print(set1)
```

# Creating a Set with the use of a List

```
set1 = set(["Python", "is", "Simple"])
```

```
print("\nSet with the use of List: ")
```

```
print(set1)
```

# Creating a Set with a mixed type of values

# (Having numbers and strings)

```
set1 = set([1, 2, 'Python', 4, 'is', 6, 'simple'])
```

```
print("\nSet with the use of Mixed Values")
```

```
print(set1)
```

# Addition of elements in a Set

```
set1 = set()
```

```
print("Initial blank Set: ")
```

```
print(set1)
```

```
# Adding element and tuple to the Set
```

```
set1.add(8)
```

```
set1.add(9)
```

```
set1.add((6, 7))
```

```
print("\nSet after Addition of Three elements: ")
```

```
print(set1)
```

```
# Addition of elements to the Set using Update function
```

```
set1.update([10, 11])
```

```
print("\nSet after Addition of elements using Update: ")
```

```
print(set1)
```

```
# Accessing of elements in a set
```

```
# Creating a set
```

```
set1 = set(["Python", "is", "Excellent"])
```

```
print("\nInitial set")
```

```
print(set1)
```

```
# Accessing element using for loop
```

```
print("\nElements of set: ")
```

```
for i in set1:
```

```
    print(i, end = " ")
```

```
# Checking the element using in keyword
```

```
print("Great" in set1)
```

# Deletion of elements in a Set

# Creating a Set

```
set1 = set([1, 2, 3, 4, 5, 6,  
           7, 8, 9, 10, 11, 12])  
print("Initial Set: ")  
print(set1)
```

# Removing elements from Set using Remove() method

```
set1.remove(5)  
set1.remove(6)  
print("\nSet after Removal of two elements: ")  
print(set1)
```

# Removing elements from Set using Discard() method

```
set1.discard(8)  
set1.discard(9)  
print("\nSet after Discarding two elements: ")  
print(set1)
```

# Removing element from the Set using the pop() method

```
set1.pop()  
print("\nSet after popping an element: ")  
print(set1)
```

# Removing all the elements from Set using clear() method

```
set1.clear()  
print("\nSet after clearing all the elements: ")
```



```
print(set1)
```

OUTPUT:

Initial blank Set:

```
set()
```

Set with the use of String:

```
{ 't', ' ', 'h', 'o', 'e', 'm', 'W', 'P', 'c', 'y', 'n', 'l' }
```

Set with the use of List:

```
{ 'Python', 'is', 'Simple' }
```

Set with the use of Mixed Values

```
{ 1, 2, 4, 6, 'is', 'Python', 'simple' }
```

Initial blank Set:

```
set()
```

Set after Addition of Three elements:

```
{ 8, 9, (6, 7) }
```

Set after Addition of elements using Update:

```
{ 8, 9, (6, 7), 10, 11 }
```

Initial set

```
{ 'Python', 'is', 'Excellent' }
```

Elements of set:

```
Python is Excellent False
```

Initial Set:

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

Set after Removal of two elements:

```
{1, 2, 3, 4, 7, 8, 9, 10, 11, 12}
```

Set after Discarding two elements:

```
{1, 2, 3, 4, 7, 10, 11, 12}
```

Set after popping an element:

```
{2, 3, 4, 7, 10, 11, 12}
```

Set after clearing all the elements:

```
set()
```

```
>>>
```

### ***1.3.3.5 Dictionary***

Dictionary is an unordered collection of data values. It is used to store data values like a map. Dictionary holds key:value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair is separated by a colon :, whereas each key is separated by a 'comma'.

Dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'. Dictionary holds a pair of values, one being the Key and the other corresponding pair element being its Key:value. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can't be repeated and must be immutable.

Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing curly braces{ }.

Note – Dictionary keys are case sensitive, same name but different cases of Key will be treated distinctly.

Dictionaries are optimized for retrieving data when there is huge volume of data. They provide the key to retrieve the value.

```
>>> d1={1:'value','key':2}
```

```
>>> type(d)
```

**PROGRAM:**

```
# Creating an empty Dictionary
```

```
Dict = { }
```

```
print("Empty Dictionary: ")
```

```
print(Dict)
```

```
# Creating a Dictionary
```

```
# with Integer Keys
```

```
Dict = { 1: 'Python', 2: 'Is', 3: 'Powerful'}
```

```
print("\nDictionary with the use of Integer Keys: ")
```

```
print(Dict)
```

```
# Creating a Dictionary
```

```
# with Mixed keys
```

```
Dict = {'Name': 'Python', 1: [1, 2, 3, 4]}
```

```
print("\nDictionary with the use of Mixed Keys: ")
```

```
print(Dict)
```

```
# Creating a Dictionary
```

```
# with dict() method
```

```
Dict = dict({ 1: 'Python', 2: 'Is', 3:'Efficient'})
```

```
print("\nDictionary with the use of dict(): ")
```

```
print(Dict)
```

```
# Creating a Dictionary
```

```
# with each item as a Pair
```

```
Dict = dict([(1, 'Python'), (2, 'Programming')])
```

```
print("\nDictionary with each item as a pair: ")
```

```
print(Dict)
```

```
# Creating an empty Dictionary
```

```
Dict = { }
```

```
print("Empty Dictionary: ")
```

```
print(Dict)
```

```
# Adding elements one at a time
```

```
Dict[0] = 'Python'
```

```
Dict[2] = 'Program'
```

```
Dict[3] = 1
```

```
print("\nDictionary after adding 3 elements: ")
```

```
print(Dict)
```

```
    # Updating existing Key's Value
```

```
Dict[2] = 'Welcome'
```

```
print("\nUpdated key value: ")
```

```
print(Dict)
```

```
# Python program to demonstrate
```

```
# accessing a element from a Dictionary
```

```

# Creating a Dictionary
Dict = { 1: 'Python', 'name': 'Is', 3: 'Case-Sensitive'}
    # accessing a element using key
print("Accessing a element using key:")
print(Dict['name'])
# accessing a element using get() method
print("Accessing a element using get:")
print(Dict.get(3))
# Initial Dictionary
Dict = { 5 : 'Welcome', 6 : 'To', 7 : 'Python',
        'A' : { 1 : 'Python', 2 : 'Is', 3 : 'Simple'},
        'B' : { 1 : 'Python', 2 : 'Pramming'}}
print("Initial Dictionary: ")
print(Dict)
# Deleting a Key value
del Dict[6]
print("\nDeleting a specific key: ")
print(Dict)

# Deleting a Key
# using pop()
Dict.pop(5)
print("\nPopping specific element: ")
print(Dict)
# Deleting an arbitrary Key-value pair using popitem()
Dict.popitem()
print("\nPops an arbitrary key-value pair: ")

```

```
print(Dict)
# Deleting entire Dictionary
Dict.clear()
print("\nDeleting Entire Dictionary: ")
print(Dict)
OUTPUT:
Empty Dictionary:
{}
Dictionary with the use of Integer Keys:
{1: 'Python', 2: 'Is', 3: 'Powerful'}
```

Dictionary with the use of Mixed Keys:

```
{'Name': 'Python', 1: [1, 2, 3, 4]}
```

Dictionary with the use of dict():

```
{1: 'Python', 2: 'Is', 3: 'Efficient'}
```

Dictionary with each item as a pair:

```
{1: 'Python', 2: 'Programming'}
```

Empty Dictionary:

```
{}
```

Dictionary after adding 3 elements:

```
{0: 'Python', 2: 'Program', 3: 1}
```

Updated key value:

```
{0: 'Python', 2: 'Welcome', 3: 1}
```

Accessing a element using key:

Is

Accessing a element using get:

Case-Sensitive

Initial Dictionary:

```
{5: 'Welcome', 6: 'To', 7: 'Python', 'A': {1: 'Python', 2: 'Is', 3: 'Simple'}, 'B': {1: 'Python', 2: 'Pramming'}}
```

Deleting a specific key:

```
{5: 'Welcome', 7: 'Python', 'A': {1: 'Python', 2: 'Is', 3: 'Simple'}, 'B': {1: 'Python', 2: 'Pramming'}}
```

Popping specific element:

```
{7: 'Python', 'A': {1: 'Python', 2: 'Is', 3: 'Simple'}, 'B': {1: 'Python', 2: 'Pramming'}}
```

Pops an arbitrary key-value pair:

```
{7: 'Python', 'A': {1: 'Python', 2: 'Is', 3: 'Simple'}}
```

Deleting Entire Dictionary:

```
{}
```

```
>>>
```

## 1.4 PYTHON BUILT-IN FUNCTIONS

A function is a group of statements that performs a specific task. Python provides a library of functions like any other programming language. The built-in functions such as eval, input, print, and int are always available in the Python interpreter. You don't have to import any modules to use these functions.

**Table.1.1: Simple Python Built-in Functions**

Function	Description	Example
abs(x)	Returns the absolute value for x	abs(-2) is 2

max(x1, x2, ...)	Returns the largest among x1, x2, ...	max(1, 5, 2) is 5
min(x1, x2, ...)	Returns the smallest among x1, x2, ...	min(1, 5, 2) is 1
pow(a, b)	Returns ab. Same as a ** b.	pow(2, 3) is 8
round(x)	Returns an integer nearest to x. If x is equally close to two integers, the even one is returned.	round(5.4) is 5 round(5.5) is 6 round(4.5) is 4
round(x, n)	Returns the float value rounded to n digits after the decimal point.	round(5.466, 2) is 5.47 round(5.463, 2) is 5.46

**Table 1.2: Mathematical Functions**

Function	Description	Example
fabs(x)	Returns the absolute value for x as a float.	fabs(-2) is 2.0
ceil(x)	Rounds x up to its nearest integer and returns that integer.	ceil(2.1) is 3 ceil(-2.1) is -2
floor(x)	Rounds x down to its nearest integer and returns that integer.	floor(2.1) is 2 floor(-2.1) is -3
exp(x)	Returns the exponential function of x (ex).	exp(1) is 2.71828
log(x)	Returns the natural logarithm of x.	log(2.71828) is 1.0
log(x, base)	Returns the logarithm of x for the specified base.	log(100, 10) is 2.0
sqrt(x)	Returns the square root of x.	sqrt(4.0) is 2



sin(x)	Returns the sine of x. x represents an angle in radians.	sin(3.14159 / 2) is 1 sin(3.14159) is 0
asin(x)	Returns the angle in radians for the inverse of sine.	asin(1.0) is 1.57 asin(0.5) is 0.523599
cos(x)	Returns the cosine of x. x represents an angle in radians.	cos(3.14159 / 2) is 0 cos(3.14159) is -1
acos(x)	Returns the angle in radians for the inverse of cosine.	acos(1.0) is 0 acos(0.5) is 1.0472
tan(x)	Returns the tangent of x. x represents an angle in radians.	tan(3.14159 / 4) is 1 tan(0.0) is 0
degrees(x)	Converts angle x from radians to degrees.	degrees(1.57) is 90
radians(x)	Converts angle x from degrees to radians.	radians(90) is 1.57

#### PROGRAM:

```
import math # import math module to use the math functions
# Test algebraic functions
print("exp(1.0) =", math.exp(1))
print("log(2.78) =", math.log(math.e))
print("log10(10, 10) =", math.log(10, 10))
print("sqrt(4.0) =", math.sqrt(4.0))

# Test trigonometric functions
print("sin(PI / 2) =", math.sin(math.pi / 2))
print("cos(PI / 2) =", math.cos(math.pi / 2))
```

48

```
print("tan(PI / 2) =", math.tan(math.pi / 2))
print("degrees(1.57) =", math.degrees(1.57))
print("radians(90) =", math.radians(90))
```

OUTPUT:

```
exp(1.0) = 2.718281828459045
log(2.78) = 1.0
log10(10, 10) = 1.0
sqrt(4.0) = 2.0
sin(PI / 2) = 1.0
cos(PI / 2) = 6.123233995736766e-17
tan(PI / 2) = 1.633123935319537e+16
degrees(1.57) = 89.95437383553924
radians(90) = 1.5707963267948966
>>>
```

**Table 1.3: String Functions**

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value

expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case

lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case

zfill()	Fills the string with a specified number of 0 values at the beginning
---------	---

## 1.5 CONDITIONAL STATEMENTS


When there is no condition placed before any set of statements, the program will be executed in sequential manure. But when some condition is placed before a block of statements the flow of execution might change depends on the result evaluated by the condition. This type of statement is also called decision making statements or control statements. This type of statement may skip some set of statements based on the condition.

### *Logical Conditions Supported by Python*

- Equal to (==) Eg: a == b
- Not Equal (!=)Eg : a != b
- Greater than (>) Eg: a > b
- Greater than or equal to (>=) Eg: a >= b
- Less than (<) Eg: a < b
- Less than or equal to (<=) Eg: a <= b

### *Indentation*

C Program	Python
<pre>x = 500 y = 200 if (x &gt; y) {     printf("x is greater than y") } else if(x == y) {</pre>	<pre>x = 500 y = 200 if x &gt; y:     print("x is greater than y") elif x == y:     print("x and y are equal") else:     print("x is less than y")</pre>

<pre>printf("x and y are equal") } else { printf("x is less than y") }</pre>	 <p>Indentation (At least one White Space instead of curly bracket)</p>
--	--

### ***Structure of C- Program Vs Python***

To represent a block of statements other programming languages like C, C++ uses “{ ...}” curly – brackets , instead of this curly braces python uses indentation using white space which defines scope in the code. The example given below shows the difference between usage of Curly bracket and white space to represent a block of statement.

#### ***Without proper Indentation:***

```
x = 500
y = 200
if x > y:
print("x is greater than y")
```

In the above example there is no proper indentation after if statement which will lead to Indentation error.

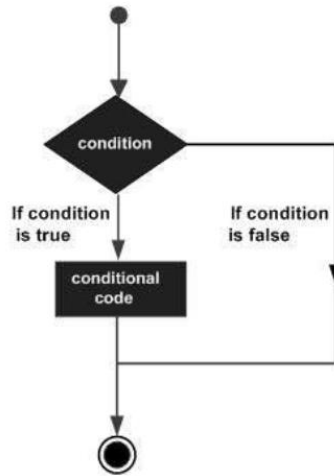
#### **1.5.1 If Statement**

The ‘if’ statement is written using “if” keyword, followed by a condition.If the condition is true the block will be executed. Otherwise, the control will be transferred to the firststatement after the block.

Syntax:

if<Boolean>:

<block>



*Fig.1.3: if statement*

In this statement, the order of execution is purely based on the evaluation of boolean expression.

**Example:**

x = 200

y = 100

if x > y:

    print("X is greater than Y")

print("End")

### **Output :**

X is greater than Y

End

In the above the value of x is greater than y , hence it executed the print statement whereas in the below example x is not greater than y hence it is not executed the first print statement

```
x = 100
```

```
y = 200
```

```
if x > y:
```

```
    print("X is greater than Y")
```

```
print("End")
```

### **Output :**

End

### **Elif**

The **elif** keyword is useful for checking another condition when one condition is false.



## Example

```
mark = 55

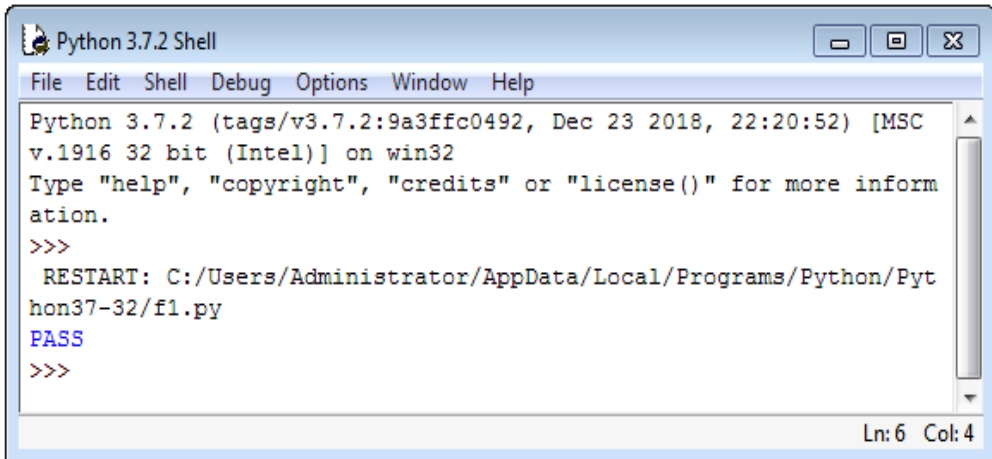
if (mark >=75):

print("FIRST CLASS")

elif mark >= 50:

print("PASS")
```

## Output :



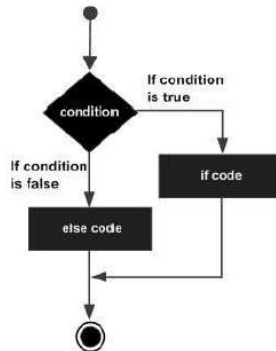
```
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC
v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more inform
ation.
>>>
  RESTART: C:/Users/Administrator/AppData/Local/Programs/Python/Pyt
hon37-32/f1.py
PASS
>>>
```

Ln: 6 Col: 4

In the above the example, the first condition (mark >=75) is false then the control is transferred to the next condition (mark >=50), Thus, the keyword **elif** will be helpful for having more than one condition.

## Else

The **else** keyword will be used as a default condition. i.e. When there are many conditions, when the **if-condition** is not true and all **elif-conditions** are also not true, then **else** part will be executed..

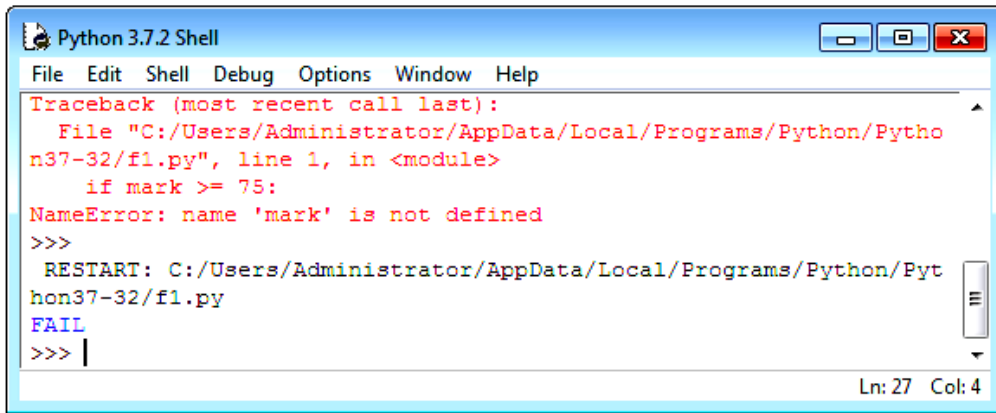


*Fig.1.4: if-else statement*

#### Example

```
mark = 10
```

```
if mark >= 75:  
    print("FIRST CLASS")  
elif mark >= 50:  
    print("PASS")  
else:  
    print("FAIL")
```

A screenshot of a Python 3.7.2 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows a traceback for a NameError. The error message is 'NameError: name 'mark' is not defined'. The traceback indicates the error occurred in the file 'C:/Users/Administrator/AppData/Local/Programs/Python/Python37-32/f1.py' at line 1. The code being executed is 'if mark >= 75:'. The prompt '>>>' is shown, followed by 'RESTART: C:/Users/Administrator/AppData/Local/Programs/Python/Python37-32/f1.py' and 'FAIL'. The status bar at the bottom right shows 'Ln: 27 Col: 4'.

In the example above, condition 1 and condition 2 fail. None of the preceding condition is true. Hence, the **else** part is executed.

### 1.5.2 Iterative Statements

Sometimes certain section of the code (block) may need to be repeated again and again as long as certain condition remains true. In order to achieve this, the iterative statements are used. The number of times the block needs to be repeated is controlled by the test condition used in that statement. This type of statement is also called as the “Looping Statement”. Looping statements add a surprising amount of new power to the program.

#### Need for Looping / Iterative Statement

Suppose the programmer wishes to display the string “Sathyabama !...” 150 times. For this, one can use the print command 150 times.

```
print("Sathyabama  
!...")  
print("Sathyabama  
!...")  
.....
```

The above method is somewhat difficult and laborious. The same result can be achieved by a loop using just two lines of code.(As below)

```
for count in  
range(1,150) :  
    print ("Sathyabama  
- ...")
```

Types of looping statements

- 1)     **for** loop
- 2)     **while** loop

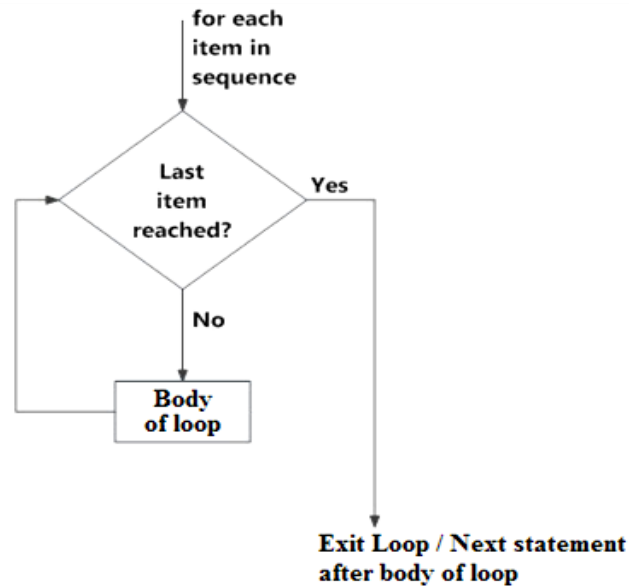
### ***1.5.2.1 The 'for' Loop***

The **for**loop is one of the powerful and efficient statements in python which is used very often. It specifies how many times the body of the loops needs to be executed. For this reason it uses control variables which keep tracks,the count of execution. The general syntax of a 'for' loop looks as below:

```
for<variable>in range (A,B):
```

```
<body of the loop >
```

**Flow Chart:**



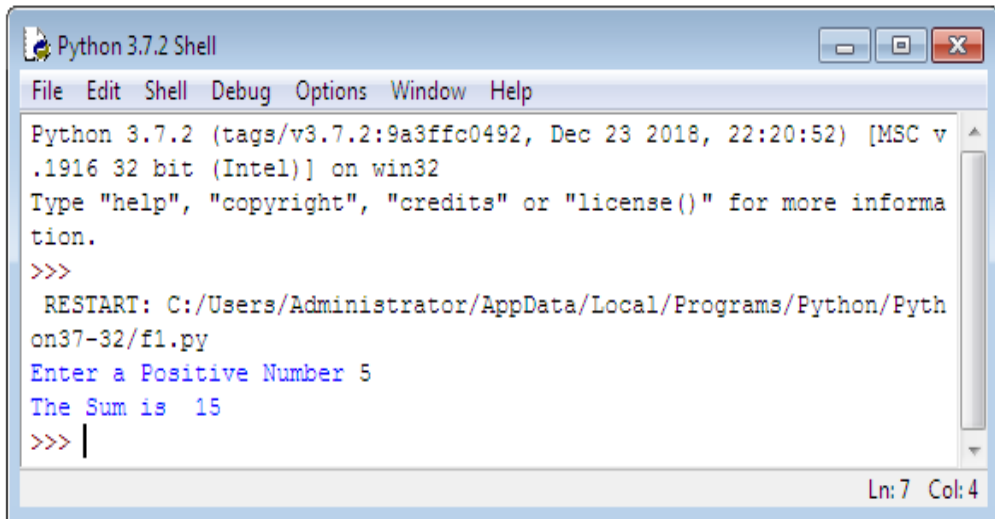
**Fig.1.5: for loop**

**Example 1:** To compute the sum of first n numbers (i.e.  $1 + 2 + 3 + \dots + n$ )

```
# Sum.py
total = 0
n = int (input ("Enter a Positive Number"))
for i in range(1,n+1):
    total = total + i
print ("The Sum is ", total)
```

**Note:Why (n+1)?** Check in table given below.

## Output:



```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/Administrator/AppData/Local/Programs/Python/Python37-32/f1.py
Enter a Positive Number 5
The Sum is 15
>>> |
```

In the above program, the statement `total = total + i` is repeated again and again 'n' times. The number of execution count is controlled by the variable 'i'. The range value is specified earlier before it starts executing the body of loop. The initial value for the variable i is 1 and final value depends on 'n'. You may also specify any constant value.

The **range( )** Function:

The **range()** function can be called in three different ways based on the number of parameters. All parameter values must be integers.

Type	Example	Explanation
<b>range(end)</b>	for i in range(5): print(i) <b>Output :</b> 0,1,2,3,4	This is begins at 0. Increments by 1. End just before the value of end parameter.
<b>range(begin,end)</b>	for i in range(2,5): print(i) <b>Output :</b> 2,3,4	Starts at begin, End before end value, Increment by 1
<b>range(begin,end,step)</b>	for i in range(2,7,2) print(i) <b>Output :</b> 2,4,6	Starts at begin, End before end value, increment by step value

**Example:**To compute Harmonic Sum (ie:  $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots \frac{1}{n}$ )

```
# harmonic.py
```

```
total = 0
```

```
n= int(input("Enter a Positive Integer:"))
```

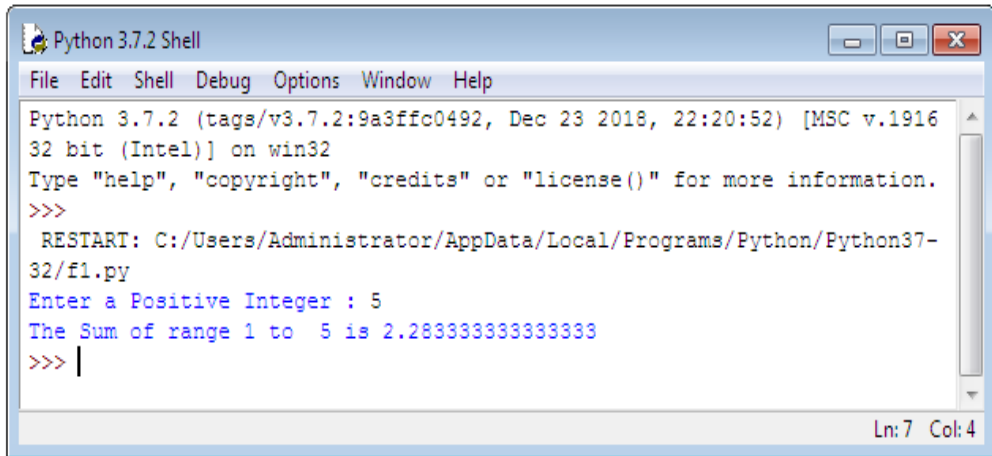
```
for i in range(1,n+1):
```

```
total+= 1/i
```

```
62
```

```
print("The Sum of range 1 to ",n, "is", total)
```

### Output:



```
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/Administrator/AppData/Local/Programs/Python/Python37-
32/f1.py
Enter a Positive Integer : 5
The Sum of range 1 to 5 is 2.2833333333333333
>>> |
```

### Example:

```
# Factorial of a number "n"
```

```
n= int(input("Enter a Number :"))
```

```
factorial = 1
```

```
# Initialize factorial value by 1
```

```
# Toverify whether the given number is negative / positive / zero
```

```
if n < 0:
```

```
print("Negative Number , Enter valid Number !...")
```

```
elif n == 0:
```

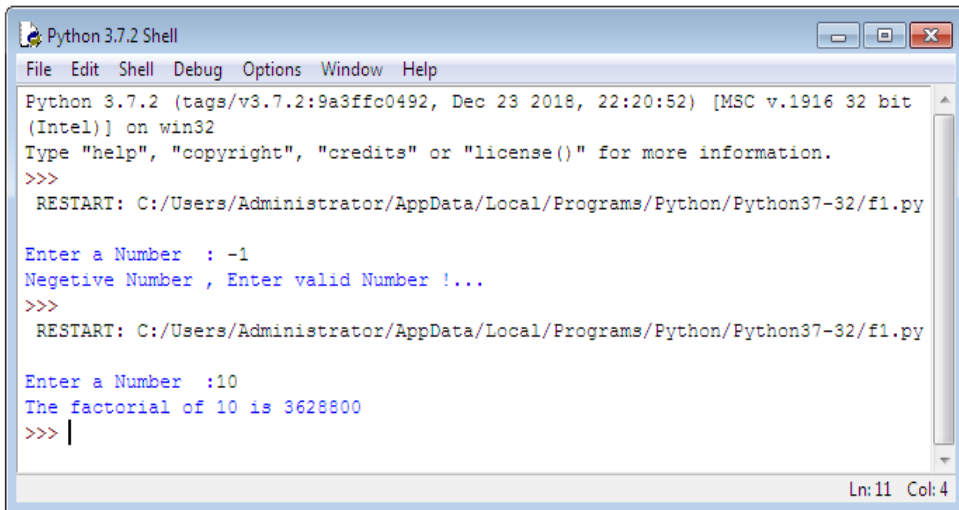


```
print("The factorial of 0 is 1")

else:

for i in range(1,n + 1):
factorial = factorial*i
print("The factorial of" ,n, "is", factorial)
```

### Output:



```
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/Administrator/AppData/Local/Programs/Python/Python37-32/f1.py

Enter a Number : -1
Negative Number , Enter valid Number !...
>>>
RESTART: C:/Users/Administrator/AppData/Local/Programs/Python/Python37-32/f1.py

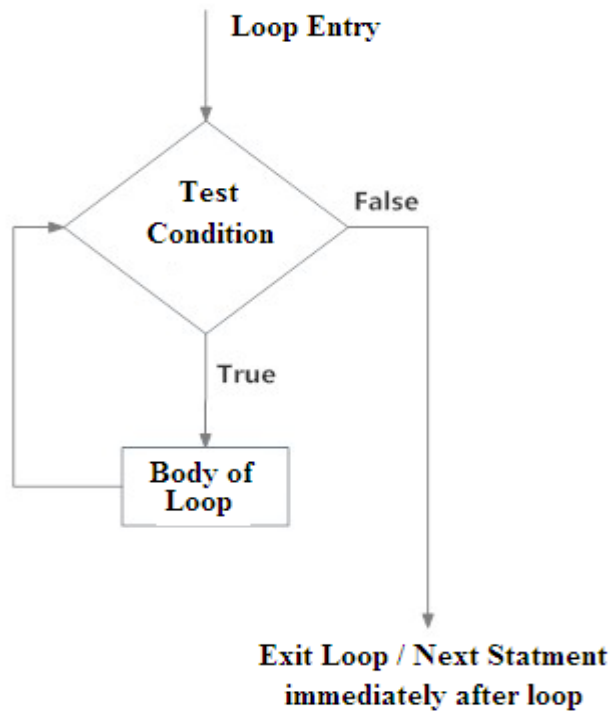
Enter a Number :10
The factorial of 10 is 3628800
>>> |
```

### 1.5.2.2 The while Loop

The **while** loop allows the program to repeat the body of a loop, any number of times, when some condition is true. The drawback of **while** loop

is that, if the condition not proper it may lead to infinite looping. So the user has to carefully choose the condition in such a way that it will terminate at a particular stage.

### Flow Chart:



*Fig.1.6: while loop*

### Syntax:

```
while (condition):

    <body of the loop>
```

In this type of loop, The execution of the loop body is purely based on the output of the given condition. As long as the condition is TRUE or in other words until the condition becomes FALSE the program will repeat the body of loop.

Valid Example	Invalid Example
<pre>i = 10 while i&lt;15 :     print(i)     i = i + 1</pre> <p><b>Output :</b> 10,11,12,13,14</p>	<pre>i = 10 while i&lt;15 :     print(i)</pre> <p><b>Output :</b> 10,10,10,10..... Indeterminate number of times</p>

**Example:** Program to display Fibonacci Sequence

```
# Program to Display Fibonacci Sequence based on number of terms
n

n = int(input("Enter number of terms in the sequence you want to
display"))
```

```

# n1 represents -- > first term and n2 represents --> Second term
n1 = 0
n2 = 1
count = 0
# count -- To check number of terms
if n <= 0:          # To check whether valid number of terms
    print ("Enter a positive integer")
elif n == 1:
    print("Fibonacci sequence up to",n,":")
    print(n2)
else:
    print("Fibonacci sequence of ",n, " terms :")
    while count < n:
        print(n1,end=' , ')
        nth = n1 + n2
        n1 = n2
        n2 = nth
        count = count + 1

```

## 1.6 INPUT / OUTPUT STATEMENT

Programmer often has a need to interact with users, either to get data or to provide some sort of result.

For Example: In a program to add two numbers, first the program needs to have an input of two numbers ( The numbers which they prefer to add) and after processing, the output should be displayed. So to get the input of two numbers, the program need to have an Input Statement and in order to display the result i.e. the sum of two numbers, it needs to have an Output Statement.

### 1.6.1 Input Statement

Helpful to take input from the user through input devices like keyboard. In Python, the standard input function is 'input()'

The syntax for input function is as follows:

**input()**

However, to get an input by prompting the user, the following form is used:

**input('prompt')**

where prompt is the string, which programmer wish to display on the screen to give more clarity about the input data. It is optional.

#### **Example:**

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

The above statement will wait till the user, enters the input value.

#### **Output:**

```
Enter a number:
```

```
>>>num
```

```
'10' # Input data entered by the user
```

### 1.6.2 Output Statement

The output statement is used to display the output in the standard output devices like monitor (screen). The standard output function “print()” is used.

#### Syntax:

```
print('prompt')
```

where `prompt` is the string, which programmer wish to display on the screen

#### Example 1:

```
print('Welcome to the Python World !')
```

#### Output:

Welcome to the Python World !

#### Example 2:

```
X = 5
```

```
print ('The value of a is', X)
```

#### Output:

The value of X is 5

#### Example 3:

69

```
print(1,2,3,4)
```

**Output: 1 2 3 4**

**Example 4:**

```
print(100,200,300,4000,sep='*')
```

**Output:**

100\*200\*300\*4000

**Example 5:**

```
print(1,2,3,4,sep='#',end='&')
```

**Output:**

1#2#3#4&

## **1.7 OBJECT ORIENTED PROGRAMMING**

Python supports object oriented programming concepts. The basic entities in object oriented programming are Class, Objects, and Methods. It also supports some of the techniques in real world entities like inheritance, Data hiding, Polymorphism, Encapsulation, MethodOverloading etc., in programming. Object orientation helps to utilize GUI environment efficiently. Some of the other programming languages which support OOPS concepts are C++, JAVA, C#.net, VB.net etc.

## Need for Object Oriented Programming

The object oriented programming is having certain advantage when compared to the normal procedure oriented programming. The main advantage is to provide access specifiers like Public, Private and Protected. OOps provide data hiding technique which is more secured than procedure oriented programming. Code reusability is one of the key features of OOPs Concept.

### Class

It is a template or blue print created by the programmer – which defines how the object's data field and methods are represented. Basically class consists of two parts: data member and function member (methods).

### Object

It is an instance of a Class;Any number objects can be created.

**Class Name: Student**

***Data Fields:***

Name,  
Mark1,Mark2,Mark3

***Methods:***

Average ()  
Rank ()



A Class is a template for creating an object. Python provides a special method, `__init__`, called as initializer, to initialize a new object when it is created.

**Example :**

```
class Student:
    def __init__(self, name, regno):
        self.name = name
        self.regno = regno
s1 = Student("John", 36)
print(s1.name)
print(s1.regno)
```

In the above example “Student” is the class name, name and regno are the data fields and s1 is the created object,

Note :

`__init__` is a method or constructor in Python. This method is automatically called to allocate memory when a new object/ instance of a class is created. All classes have the `__init__` method.

**Output :**

```
>>> John
```

```
36
```

Let us create a method (Function member) for the above class

```

class Student:
    def __init__(self,name, regno):
        self.name = name
    self.regno = regno
    def display(self):
        print("Name of the student is " + self.name )
s1 = Student("James", 43)
s1.display()

```

In the above example “display” is the method used to display the student name.

### 1.7.1 Inheritance

Inheritance allows to create anew class (Child Class) from the existing class (Parent Class).The child class inherits all the attributes of its parent class.

**Parent class** is the class, whose properties are being inherited by subclass. Parent class is also called as Base class or Super Class.

**Child class** is the class that inherits properties from another class. The child class is also called as Sub class or Derived Class.

#### Example :

```

class Person:
    def __init__(self, fname, lname):

```

```
self.firstname = fname
self.lastname = lname
def printdetails(self):
    print(self.firstname, self.lastname)
#Use the Person class to create an object and then execute the
printdetails method:
x = Person("John", "Doe")
x.printdetails()
class Employee(Person):
    pass
y = Employee("Mike", "Olsen")
y.printdetails()
```

### **Output :**

```
>>>
```

RESTART:

C:/Users/Administrator/AppData/Local/Programs/Python/Python37-32/f1.py

John Doe

Mike Olsen

```
>>>
```

In the above example the base class is Person. The first object “x” is created through the base class “Person” and the method printdetails() is invoked with that object which produces an output “John Doe”. Again, another object “y” is created through derived class “Employee” and the same method printdetails() (belongs to base class) is invoked to produce the output “Mike Olsen”. Thus, the derived class is having the ability to invoke the method from base class just because of the inheritance property which reduces the code length or in other words it is helpful for reusability of code.

**Note:** Use the pass keyword when the programmer does not wish to add any other properties or methods to the derived class.

### **Example 2:**

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname
    def printdetails(self):
        print(self.firstname, self.lastname)
```

#Object For Base Class

```
x = Person("Paul", "Benjamin")
x.printdetails()
```

```

class Employee(Person):
    def __init__(self, fname, lname):
        Person.__init__(self, fname, lname)
        self.doj = 2019
    def greetings(self):
        print("Welcome", self.firstname, self.lastname, "who joined in the
        year ", self.doj)
# Object for derived class
y = Employee("Samuel", "Ernest")
y.printdetails()
y.greetings()

```

In the above example a new method `greetings()` is included in the derived class, Thus the derived class object is capable of invoking the method present inside base class as well as its own methods.

*printdetails()* -- method present inside base class Person.

*greetings()* -- method present inside derived class Employee.

The object “y” is able to invoke both the methods `printdetails()` and `greetings()`.

**Questions :**

1. Compare a) List and Tuple b) List and Set
2. What is type conversion in Python?
3. Is indentation required in python?
4. What is `__init__`?
5. How can you randomize the items of a list in place in Python?
6. How do you write comments in python?
7. What is a dictionary in Python?
8. Does Python have OOPs concepts?
9. Write a program in Python to check if a sequence is a Palindrome.

10. Write a program in Python to check if a number is prime.
11. How to create an empty class in Python?
12. Write a sorting algorithm for a numerical dataset in Python.



**SATHYABAMA**

INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UNIT-II - Python Programming – SCS1619**



## UNIT II

### FILE HANDLING

File Operations –Iterators - Exception handling - Regular Expressions

#### 2.1 FILE OPERATIONS

An object that stores data, settings or programming commands in a computer system is called as a file. The data's used in a program is temporary. The data is lost when the program terminates. A file is used to store data permanently. There are three major file operations:

- Opening a file
- Performing file operations using Read or Write
- Closing the file

##### 2.1.1. File Open

To open a file create a file object and use the open() function. The open function returns a file object for the filename. The access mode specifies how the file is used

**Method:**open()

**Purpose:** To open a file

### Syntax:

File\_object=open(filename,Access\_mode,buffering)

### Attributes:

- i. Filename – Name of the file
- ii. Access\_mode- Mode of Access (Read, Write, Append)

Access Mode	Description
"r"	Opens a file for reading
"rb"	Opens a file for reading binary data
"w"	Opens a file for writing
"wb"	Opens a file for writing binary data
"a"	Opens a file for appending
"ab"	Opens a file for appending binary data

- iii. Buffering – 0 (no buffer), 1 (buffer)

### Example:

```
f= open('abc.txt') (or)  
f=open("D:/Mypython/abc.txt")
```

### ***2.1.1.1 File Access Modes***

**Table 2.1: File Access Modes**

<b>File Mode</b>	<b>Description</b>
r	Read mode
w	Write mode
x	Create and open a file
a	Appending at end of file
t	Text mode
b	Binary mode
+	Update mode

#### **Example:**

```
f= open('abc.txt', r)
```

The above statement opens the file 'abc.txt' in read mode.

### *2.1.1.2 Example for File Access modes and Properties*

```
fo=open('aa.txt','w')
print('Filename: ', fo.name)
print('Filemode: ', fo.mode)
print('File closed: ', fo.closed)
fo.close()
print('Fileclosed: ', fo.closed)

'''
output:
Filename:  aa.txt
Filemode:  w
File closed:  False
Fileclosed:  True
'''
```

The above code is a sample snippet for understanding the file modes and file properties.

(e.g)

To open a file in current directory

```
obj = open ("abc.txt", "r")
```

To open a file in specified directory

```
obj = open (r"c:\python\abc.txt", "r")
```

(Here, the prefix r represents that the string is a raw string where the backslash characters are treated as literal backslashes

(or)

```
obj = open ("c:\\python\\abc.txt", "r")
```

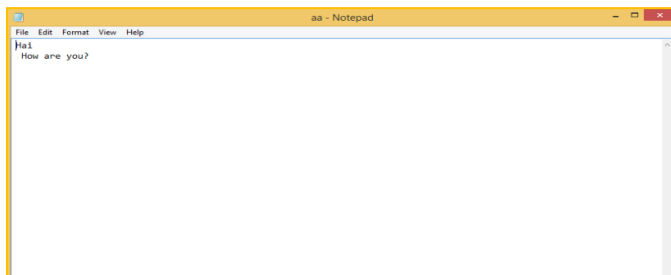
## 2.1.2. File Reading and Writing

### 2.1.2.1. File write:

write() method is used to write the contents to a file. The following code is for writing the contents to the file aa.txt.

```
fo=open('aa.txt','w')  
  
fo.write('hai \n how are you?')  
  
fo.close()
```

### Output:



In the above example, the contents of the file can be viewed by opening the file 'aa.txt'.

Example

```
# Python code to create a file
file = open('good.txt','w')
file.write("This is the write command")
file.write("It allows us to write in a particular file")
file.close()
```

#### ***2.1.2.2. Reading a file:***

read() method is used to read the contents from a file. The following code is for reading the first 10 bytes of the file 'aa.txt'.

```
fo=open('aa.txt','r')
print(fo.read())
#reading 10 bytes
fo.read(10)
fo.close()
'''
output:
Hai
How are you?
'''
```

There is more than one way to read a file in Python. If you need to extract a string that contains all characters in the file then we can use `file.read()`.

```
# Python code to illustrate read() mode
```

```
file = open("file.text", "r")
```

```
print file.read()
```

Another way to read a file is to call a certain number of characters like in the following code ,the interpreter will read the first five characters of stored data and return it as a string:

```
# Python code to illustrate read() mode character wise
```

```
file = open("file.txt", "r")
```

```
print file.read(5)
```

### **2.1.3. File Positions**

To know about the file offset positions in Python, the following methods are used:

- `seek()`
- `tell()`

**seek():**

**Syntax:** seek(offset, from)

**Description:** Sets the file's current position at the offset. The offset values are as follows:

0 : reference (beginning of file( default))

1 : current (current file position)

2 : end (end of file)

**tell() :**

**Description:** Prints the current position of file pointer.

#### ***2.1.3.1.File Offset***

'h'	'a'	'i'		,		'h'	'o'	'w'		'a'	'r'	'e'		'y'	'o'	'u'	'?'
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17



```

fo=open('aa.txt','r')
print('current position',fo.tell())
print(fo.read(10))
print('current position',fo.tell())
fo.seek(2,0) # to skip first 2
print(fo.read())
'''
output
current position 0
Hai , How
current position 10
i , How are you?
'''

```

In the above code, initially the position of the file pointer is at 0. After reading the contents, the position of the file pointer is moved to 10 (from 0 to 9). Now up on giving the command seek(2,0), the file will be read from the beginning after skipping the first 2 positions.

### ***Detailed Example:***

```

f=open('aa.txt','r')
pos=f.tell()
print(pos)
#output : 0
line=f.readline()
print(line)
#output: prints first line Hai , How are you?
pos=f.tell()
print(pos)

#20
line=f.readline()
print(line)
#
f.seek(0,0)
pos=f.tell()
print(pos)
line=f.readline()

print(line)
pos=f.tell()
print(pos)
line=f.readline()
print(line)

''' output
0
Hai , How are you?
20
Welcome to Sathyabama
0
Welcome to Sathyabama
43
School of Computing
'''

```

The contents of the file aa.txt is now:

Hai , How are you?  
Welcome to Sathyabama  
School of Computing  
Department of Computer Science & Engineering

### ***2.1.3.2. Reading a file Line by line***

In order to read a file till the End of File(EoF), while loop is used.

```
f=open('f8.txt','r')
line=f.readline()
while line!='':
    print(line)
    line=f.readline()
f.close()
'''output
kdskfa

dsafldk

kdafslj'''
```

### ***2.1.3.3. Modifying a file***

```
f=open('aa.txt','a')
f.write('aa bb cc dd')
f.close()
f=open('aa.txt','r')
print(f.read())
#prints the entire file

#go to 5th position using seek(5)
f.seek(5,0)
print('from 5th posn',f.read())
f.seek(30)
line=f.readline()
#prints posn of line from 30th posn
print('line at 30', line)
f.seek(0)
#print(f.read())#prints full file
print('current posn before reading',f.tell())
f.close()
...
Output
Hai , How are you?
Welcome to Sathyabama
School of Computing
Department of Computer Science & Engineeringaa bb cc ddaa bb cc dd
from 5th posn How are you?
Welcome to Sathyabama
School of Computing
Department of Computer Science & Engineeringaa bb cc ddaa bb cc dd
line at 30 Sathyabama
current posn before reading 0
...
```

#### 2.1.4. Alternate way for opening and closing a file:

##### Syntax:

*with open('filename') as file object:*

- No need to close the file

```
with open('aa.txt') as f:
    for line in f:
        print(line)

'''output
Hai , How are you?

Welcome to Sathyabama

School of Computing

Department of Computer Science & Engineeringaa bb cc ddaa bb cc dd'''
```

#### 2.1.5. read() &readline()

- read() – read entire file content from current position
- readline() – read the particular line of file pointer

To read data from a file read() function is used.

read specifies the number of bytes to be read

Syntax

fileobject.read (count)

## Syntax

`seek (offset, from)`

offset – number of bytes to be moved

from – specifies the reference position from where the bytes are to be moved

0 – use beginning of file as reference position

1 – use current position as reference position

2 – end of file as reference position

(e.g)

```
f = open ("abc.txt", "r") str = f.read(10)
```

```
print ("The string is", str) # current position pos = fo.tell()
```

```
print ("The current file position is", pos) #Reposition at beginning
```

```
pos = fo.seek(0,0) str = f.read(10)
```

```
print ( "The string read again is", str) f.close()
```

## Output

The string is : Python is

The current file position is : 10

The string read again is : Python is

### **2.1.6. Appending Data to an existing file**

`append()` used to append data to an existing file

# Python code to illustrate `append()` mode

```
file = open('geek.txt','a')
```

```
file.write("This will add this line")  
file.close()
```

### **2.1.7. Closing a File**

The close() function is used to close a file

#### **Syntax**

```
fileobject.close()
```

## **2.2 ITERATORS**

Iterator in python is any python type that can be used with a 'for in loop'. Python lists, tuples, dicts and sets are all examples of inbuilt iterators.

An iterator is an object that can be iterated upon, meaning that you can traverse through all the values. Iterators are objects that can be iterated upon. They are implemented within for loops, generators etc. but hidden in plain sight

The iterator object must implement two special methods `__iter__()` and `__next__()` collectively called iterator protocol. The `iter()` function returns an iterator. The `next()` function is used to manually iterate through all the items of an iterator.

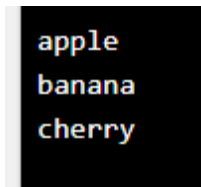
Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods `__iter__()` and `__next__()`.

Return an iterator from a tuple, and print each value:

```
mytuple = ("apple", "banana", "cherry")  
myit = iter(mytuple)
```

```
print(next(myit))  
print(next(myit))  
print(next(myit))
```

Output



```
apple  
banana  
cherry
```

Iterate the values of a tuple:

```
mytuple = ("apple", "banana", "cherry")  
  
for x in mytuple:  
    print(x)
```

```
apple  
banana  
cherry
```

Iterate the characters of a string:

```
mystr="banana" =  
for x in mystr:  
    print(x)
```

```
b  
a  
n  
a  
n  
a
```

### **List as iterator:**

```
for i in [1, 2, 3, 4]:  
    print(i)
```

Iterator in Python is a type which could be implemented in for loops. An iterator is an object that returns data one at a time.

For example if we have a list A=[1,2,3] , then iterator is used to return the items in the list one at a time.

There are two special Methods:

- `__iter__()` : returns iterator from list
- `__next__()`: returns next element in the list

Iterable objects in Python are:

- List
- Tuple
- String

### 2.2.1. Example Iterator:

```
mylist=[4,7,0,3]
myiter=iter(mylist)
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
'''
output
4
7
0
3
Error
'''
```



In the above code the list items of mylist object are retrieved one by one using 'next()' method. When the list reaches its end and if next() method is used , it shows error in the output.

### 2.2.2. Example for \_\_next\_\_()

Alternate way for retrieving the items is to use for loop and retrieve the item using \_\_next\_\_() inside the for loop. To find the length of the list 'len()' method is used.

```
list=[3,4,5,6]
iterobj=iter(list)
print()
for i in range(0,len(list)):
    print(iterobj.__next__())
'''
3
4
5
6
'''
```

### 2.2.3. Building User defined iterators

We can also build our own iterators. The following code is for implementing user defined iterators for finding powers of two.

```

class pow2:
    #To implement an iterator of powers of two
    def __init__(self,max=0):
        self.max=max
    def __iter__(self):
        self.n=0
        return self
    def __next__(self):
        try:
            if self.n<=self.max:
                res=2**self.n
                self.n+=1
                return res
            else:
                raise StopIteration
        except StopIteration:
            quit(0)
a=pow2(4)
i=iter(a)
print(next(i))
while True:
    print(next(i))
'''output
1
2
4
8
16
...
'''

```

#### 2.2.4. Python Infinite Iterators:

There are two Arguments in infinite iterators:

- Callable Object: A built in function
- Sentinels: The terminating value

The following is an example for infinite iterator. `next(inf)` will always return 0, since the sentinel 1 not at all reaches.

```

>>> int()
0
>>> inf=iter(int,1)
>>> next(inf)
0
>>> next(inf)
0

```

Similarly , the following code uses while loop to print the odd numbers starting from 1 to infinite number of times. The execution is manually terminated by providing keyboard interrupt(Ctrl+c).

```
class infin:
    def __iter__(self):
        self.num=1
        return self
    def __next__(self):
        num=self.num
        self.num+=2
        return num
a=iter(infin())
while True:
    print(next(a))
'''output
1
3
5
7
9
11
13
15
17
19.....'''
```

### 2.2.5. Python Generators

Generator functions are alternates for iterators that contain one or more **yield()** statements. Methods like `__iter__()`, `__next__()` are implemented and are iterated using `next()` automatically. Local variables are remembered between successive calls. When function terminates, `StopIterator` exception is raised automatically.

### 2.2.5.1.Example

In the following code, n value is initiated to 1 in the first step. In the second step n is incremented by two and the value yielded is now 3. In the last step n is incremented by 1 and now the value is 4.

```
def my_gen():
    n=1
    print('first')
    yield n
    n+=2
    print('second')
    yield n
    n+=1
    print('last')
    yield n
for i in my_gen():
    print(i)

'''output
first
1
second
3
last
4'''
```

The following is an example for reversing a String using python Generator. Here the string 'hello' is passed to the function 'rev()'. Using for loop, the string is yielded from the last character(len-1) to -1(0<sup>th</sup> position minus 1) as per the syntax.

```
def rev(mystr):
    len1=len(mystr)
    for i in range(len1-1,-1,-1):
        yield mystr[i]

for c in rev('hello'):
    print(c)

'''output
o
l
l
e
h'''
```

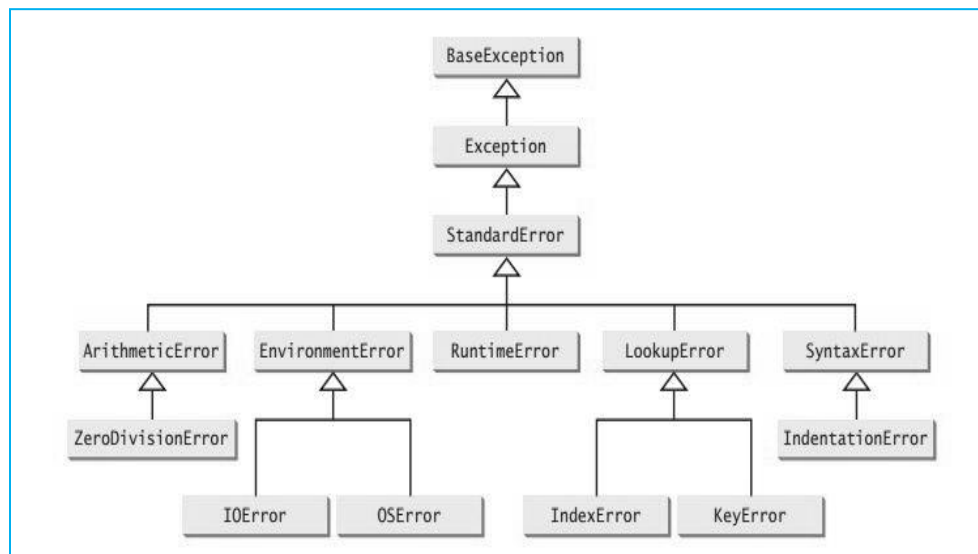
### 2.2.5.2. Advantages of Generators

- Easy to implement
- Memory efficient
- Represents infinite stream
- Generators can be pipelined

## 2.3. EXCEPTION HANDLING

Exception is an event that occurs during execution of a Python program disrupting the normal flow of execution. Exceptions are handled using try and except blocks in Python. There are built in exception classes for handling common exceptions. BaseException is the parent class for all built in Exception classes. Fig 2.1 represents the Standard Exception class hierarchy.

An error that occurs at runtime is called an exception. It enables a program to deal with exceptions and continue its normal execution. The try block lets you test a block of code for errors. The except block lets you handle the error. The finally block lets you execute code, regardless of the result of the try- and except blocks.



**Fig 2.1 Standard Exception class hierarchy**

### 2.3.1. Exception Handling Syntax and Examples

While handling exception, keep the suspicious code in try block and following the try block, include except: statement

```

try:
    suspicious block
except Exception1:
    #statement1
except Exception2:
    #statement2
.....
else:
    no exception

```

The following code raises exception when a run time error occurs upon writing the file 'aa.txt'. In case of normal program flow, the else clause will be invoked and the statements in else block will be executed.

```

try:
    fo=open('aa.txt','w')
    fo.write('Exception for exception')
except IOError:
    print('cant write')
else:
    print('written successfully')

#output:
''' written successfully'''
#content has been written to file aa.txt

```

IOError exception is also invoked when we intend to write a file when it is opened in 'read' mode. The following code depicts this case.

```
try:
    fo=open('aa.txt','r')
    fo.write('Exception handling example')
except IOError:
    print('cant write in read mode')
else:
    print('written successfully')

#output:
''' cant write in read mode'''
```

#### ***2.3.1.1. Except Clause without specifying any exception***

In python, we can also have except clause with no specific exception. In this case any type of exception can be handled. The following is the syntax for except statement with no specific exception type.

***Syntax:***

```
try:
    #Error code
except:
    #Execute block with Any exception
else:
    #No exception
```

***Example:***

In the following code, except clause is alone given, without mentioning the type of exception. In the sample runs when the value of 'b' is given as 0,



exception is caught and 'divide by zero error' is printed. Whereas, in case of normal run, the output is displayed.

```
a,b=eval(input('Enter two nos.'))
try:
    c=a/b
except:
    print('divide by zero error')
else:
    print('Normal execution & the value is',c)

'''Sample outputs:
Run1:

Enter two nos.2,0
divide by zero error

Run 2:

Enter two nos.3,6
Normal execution & the value is 0.5
'''
```

### 2.3.1.2. Except Clause with Multiple exceptions:

There is another way of specifying multiple exceptions in the single except clause. When multiple exceptions are thrown, the first exception which is being caught will alone be handled. The syntax is given as follows.

*Syntax:*

```
try:
    #Error code
except (Exception 1, Exception2,...):
    #Execute block with Any exception
else:
    #No exception
```

*Example:*

```
a=input('Enter the value of a')
b =input('Enter the value of b')
try:
    c=a/b
except (TypeError, ZeroDivisionError):
    if TypeError:
        print('Type error')
    elif ZeroDivisionError:
        print('Divide by zero error')
else:
    print('Normal execution & the value is',c)
'''
Sample output:
Enter the value of a : 6
Enter the value of b : a
Type error
'''
```

### 2.3.1.3 Optional finally clause

Like other object oriented programming languages, try has optional finally clause. The statements given in finally block will be executed even after the exceptions are handled.

```
try:
    f = open("aa.txt",'r')
    f.write('exception handling')
except:
    print('file write exception')
finally:
    f.close()
    print('normal flow')
'''
sample output:
file write exception
normal flow
'''
```

### 2.3.2. Raising Exceptions

Exception can be raised from a function:

*raise ExceptionClass('Something Wrong')*

**Example:**

```
ex=RunTimeError('Something Wrong')
```

```
raise ex
```

OR

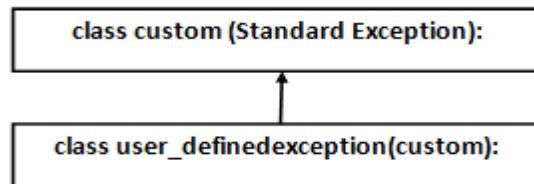
Raise RuntimeError('Something Wrong')

```
try:
    a = int(input("Enter a positive integer: "))
    if a <= 0:
        raise ValueError("That is not a positive number!")
except ValueError as er:
    print(er)

'''Sample output:
Enter a positive integer: -7
That is not a positive number!
'''
```

### 2.3.3. Custom Exception/User Defined Exception

In Python custom exception or otherwise called as user defined exception can be handled by creating a new user defined class which is a derived class from Exception class.



**Fig. 2.2: Inheriting the Standard Exception class**

In the following example two user defined exception classes are derived from the parent class Error which inherits the standard Exception class. The number guessed is 10. When any number greater than 10 is given as input TooLargeErr exception is thrown and when the number is less than 10, TooSmallErr exception is thrown.

```
class Error(Exception):
    pass
class TooSmallErr(Error):
    pass
class TooLargeErr(Error):
    pass

n=10
while True:
    try:
        x=int(input('enter a number'))
        if x<n:
            raise TooSmallErr
        elif x>n:
            raise TooLargeErr
        break
    except TooSmallErr:
        print('value is small, try again!..')
        print()
    except TooLargeErr:
        print('value is large, try again!..')
        print()
print('Wow! Guess is correct!')

'''output
enter a number23
value is large, try again!..

enter a number1
value is small, try again!..

enter a number10
Wow! Guess is correct!'''
```

## 2.4 REGULAR EXPRESSIONS

Regular Expressions can also be called as RE/regex/regex patterns .RE's are specialized programming languages embedded inside Python. RE's are available by importing **re** module. RE patterns are compiled into a series of bytecodes when executed by a matching engine written in C language. REs could not perform all string processing tasks. REs are applicable in Pattern recognition problems. RE module has to imported for calling re methods like: split(), findall(), search() etc.

### ***Syntax:***

```
import re
```

### 2.4.1 RE matching characters

Character matching is very important for identifying patterns and matching them with the given input. The following table describes some of the important matching characters used in Python REs.

**Table: 2.2 Python Character Matching**

Matching Character	Description
[ ]	Finding a range of characters [a-z]
\w	Alphanumeric character [a-zA-Z0-9]

\W	Non alpha numeric characters : ^ [a-zA-Z0-9]
*	Repeating a character [0] or more times
()	Grouping or including
+	1 or more
?	0 or 1
{x}	Exact no. of match
{a,b}	In range from a to b
\any_number	Matching the group of same number.
\A	Only at the start of the string.
\Z	Only at the end of the string
\b	Empty string only at the beginning or end of a word.
\B	Empty string match not at the beginning or end of a word
\d	[0-9]
\D	^[0-9]
\s	Space
\S	Non space

## 2.4.2. RE Methods

### 2.4.2.1. *The search() method*

**Method:**search()

**Description:** Returns true if the search string is found.

**Example:**

```
import re
m = re.search('info','information')
if m:
    print(m,"is found")
else:
    print('not found')

''' output
<re.Match object; span=(0, 4), match='info'> is found
'''
```

The above code returns the Match object with a span position from 0 to n-1 when the search information is found.

#### ***2.4.2.2. The split() method***

**Method:**split()

**Description:** For creating space in the string.

**Example:**



```
import re
print(re.split(r'(\s)','This is a string'))
print()
print(re.split(r'[a-i]','This is a string'))
'''
output:
    ['This', ' ', 'is', ' ', 'a', ' ', 'string']

    ['T', '', 's ', 's ', ' str', 'n', '']
'''
```

In the above code, split() method is applied twice on the string, ‘This is a string’. When the matching character \s is applied, the spaces in the string are split up. When the regular expression **r'([a-i])** is applied, the string is split ignoring the range of characters from a to i.

#### ***2.4.2.3. The findall () method***

**Method:**findall()

**Description:** Finds all the matches and returns them as a list of strings.

**Example:**

```
import re
n='123 1234 12345 636525 1478523'
print(re.findall('\d{5,7}',n))
'''output
returns digits of length from 5 to 7
['12345', '636525', '1478523']
'''
```

#### 2.4.2.4. *The match() method*

**Method:**match()

**Description:**To match the RE pattern to string with optional flags.

**Example:**

```
import re
list=['csea','cseb','cse a and b']
for e in list:
    z=re.match('(c\w+)',e)
    if z:
        print(z.groups())
'''
Sample output:
The first word of the list items matching the letter c is grouped up
('csea',)
('cseb',)
('cse',)
'''
```

#### ***2.4.2.5. The finditer() method***

**Method:** finditer()

**Description:** Generating an iterator.

**Example:**

```
import re
str='welcome to cse dept and it dept of Soc'
for i in re.finditer('dept',str):
    localtime=i.span()
    print(localtuple)

'''output:
returns start index and end index of the string
'dept' which occurs in 2 places:

(15, 19)
(27, 31)
'''
```

#### ***2.4.2.6. The compile() method***

**Method:** compile()

**Description:** Compiling a pattern without rewriting it.

**Example:**

```

import re
pattern=re.compile('Python')
result=pattern.findall('Welcome to Python programming. Python is Object Oriented.')
print(result)
result2=pattern.findall('Learning Python is Simple')
print(result2)
'''output
['Python', 'Python']
['Python']
'''

```

In the above code the compiled pattern is ‘Python’. The result objects return each and every occurrence of the matched pattern line by line. Other Regular Expression methods are given in Table 2.2 and RE Compilation flags are given in Table 2.3.

**Table 2.3: Other RE methods**

Method/Attribute	Purpose
group()	Returns the string matched by the RE
start()	Returns the starting position of the match
end()	Returns the ending position of the match

span()	Returns a tuple containing the starting and ending positions of the match
sub()	Replaces the RE pattern and returns the modified string

**Table 2.4: RE Compilation Flags**

Flag	Syntax	Description
ASCII	re.A	Makes several escapes like \w,\b,\s and \d and match only on ASCII characters
DOTALL	re.S	Match any character including newline
IGNORECASE	re.I	Case insensitive matches
MULTILINE	re.M	Multiline matching affecting ^ and \$
LOCALE	re.L	Locale aware match(Localization API)
VERBOSE	re.X	Enables verbose RE

*Example:*

```
import re
list='''csea
nseb
dsea1 and b'''

m1=re.findall(r'^\w',list)
m2=re.findall(r'^\w',list,re.MULTILINE)

print(m1)
print()
print(m2)

'''output
['c'] <- returns only the first character of first line

['c', 'n', 'd'] <-returns all first characters since it is multiline
'''|
```

### 2.4.3. Case Studies on Pattern Matching:

#### *Case Study 1: Phone number verification*

```
import re
ph='412-555-342-4533'
if re.search('\w{3}-\w{3}-\w{3}-\w{4}',ph):
    print('valid phone no')
else:
    print('invalid phone no')
'''
output:
valid phone no'''
```

#### *Case Study 2: Validating First name & Last name*

```
import re
name='arthi rathna'
if re.search('\w',name):
    print('valid full name')
else:
    print('invalid name')
'''
output:
valid full name'''
```

### *Case Study 3: Email Address Verification*

```
import re
n='abc@gmail.com, x3@, @abc.com, az2@abc.in'
print(re.findall('[\w._/]{1,20}@[ \w.-]{2,20}.[A-Za-z]{2,3}',n))
'''output
returns valid emailaddresses:
['abc@gmail.com', 'az2@abc.in']
'''
```

### *Case Study 4: Web Scrapping*

```
import urllib.request
from re import findall
url='http://www.sathyabama.ac.in/sitepagethree.php?mainref=23/'
resp=urllib.request.urlopen(url)
html=resp.read()
htmlstr=html.decode()
pdata=findall('\d{4}\s-\s\d{3}\s-\s\d{4}',htmlstr)
for item in pdata:
    print(item)

'''
output:
1800 - 425 - 1770
'''
```



## 2.8. EXERCISES

1. What is the output of the following code?

```
f1= None
for i in range (5):
    with open("data.txt", "w") as f1:
        if i > 2:
            break
print(f1.closed)
```

2. Write a Python code to read a String, character by character and print the String as a whole using iterators.
3. Write a Python program that matches any string that has an *a* followed by one or more t's.
4. Write a Python program to insert spaces between words starting with capital letters.
5. Write a Python program to remove the parenthesis area in a string using REs.  
Sample data : ["abc (.com)", "w3schools", "google (.com)"]  
Expected Output:  
abc  
w3schools  
google
6. Write a Python program to do a case-insensitive string replacement.
7. Write a Python code to print the given list in reverse order.
8. What is the output of the snippet of code shown below?

```
import numpy as np
a = np.array([[ 1, 2, 3],[4,5,6],[7,8,9]])
print(a[1])
```

9. Write a Python code to append a file 'aa.txt' and then read and display the contents of the file line by line.
10. Check whether the methods today() and now() of datetime library are same or not. Prove the same using a Python code.

## REFERENCES:

1. Timothy A.Budd, Exploring Python, Tata McGraw Hill Education Private Limited, New Delhi, 2011.
2. Python basics: <https://www.tutorialspoint.com/python> , Accessed on May 2019.
3. Y. Daniel Liang, Introduction to Programming Using Python, Pearson, 2013.
4. Python Libraries: <http://cs231n.github.io/python-numpy-tutorial/>, Accessed on May 2019.
5. Scipy: <https://www.guru99.com/scipy-tutorial.html>, Accessed on May 2019.
6. Python Exercises: <https://www.w3resource.com/python-exercises/re/> , Accessed on May 2019.



**SATHYABAMA**

INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE  
[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UNIT-III Python - Programming – SCS1619**

## UNIT III

### GUI PROGRAMMING WITH PYTHON

GUI Programming in Python - Introduction to GUI library - Layout management - Events and bindings - Fonts – Colors - Canvas - Widgets (frame, label, button, check box, entry, list box, message, radio button, text, spin box).

In python text only programs can be created using Command line Interface. Graphical user interface(GUI) can be created using tkinter module in python.

#### 3.1 INTRODUCTION TO GUI LIBRARY IN PYTHON

**Tkinter** is a module in the Python standard library which serves as an interface to Tk (ie) simple *toolkit*. There are many other toolkits also available to create GUI.

Tkinter provides the following widgets:

- button
- canvas
- checkbutton
- combobox
- entry
- frame
- label

- listbox
- menu
- message
- progressbar
- radiobutton
- scrollbar
- spinbox
- text

Tkinter also provides three layout managers:

- place - It positions widgets at absolute locations
- grid - It arranges widgets in a grid
- pack - It packs widgets into a cavity

## **3.2 LAYOUT MANAGEMENT**

The Layout Managers are used to arrange components in a particular manner. It is used to organize the components. There are three Layout Management in python:

1. Pack Layout
2. Grid Layout
3. Place Layout

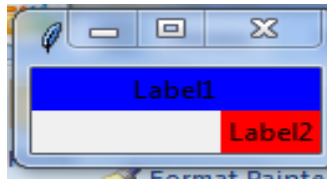
### **3.2.1 Pack Layout Manager**

It is a simple layout manager. Here widgets can be organized in horizontal and vertical boxes. It is used to place each widget next to previous widget. It will be called without any arguments and it will position and size the widgets in a reasonable way. Whenever the user wants to have a series of widgets in a vertical or horizontal row, the pack layout manager is fairly simple to use. The layout is controlled with the fill, expand, and side options.

***Example:***

```
from tkinter import *  
top=Tk()  
l1=Label(top,text="Label1",bg="blue")  
l2=Label(top,text="Label2",bg="red" )  
l1.pack(fill=X,side=TOP,expand=True)  
l2.pack(fill=X,side=RIGHT)  
top.mainloop()
```

***Output:***



***Explanation:*** Label l1 has been placed in top position, it is filled in X axis. Label l2 has been placed in Right Position and it is also filled in X axis. Since expand attribute has the value True for Label l1, it can be stretched.

## Padding Option in Pack Layout:

The pack() manager has four padding options:

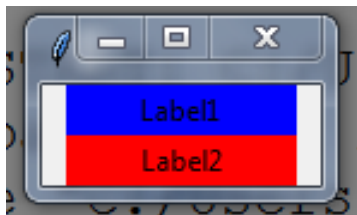
1. Internal Padding
2. External padding
3. Padding in X Direction.
4. Padding in Y Direction.

### External Padding in Horizontal direction(padx)

#### *Example:*

```
from tkinter import *  
top=Tk()  
l1=Label(top,text="Label1",bg="blue")  
l2=Label(top,text="Label2",bg="red" )  
l1.pack(fill=X,side=TOP,expand=True,padx=10)  
l2.pack(fill=X,side=TOP,padx=10)  
top.mainloop()
```

#### *Output:*



## External Padding in Vertical direction (pady)

### *Example:*

```
from tkinter import *  
  
top=Tk()  
  
l1=Label(top,text="Label1",bg="blue")  
  
l2=Label(top,text="Label2",bg="red" )  
  
l1.pack(fill=X,side=TOP,expand=True,pady=10)  
  
l2.pack(fill=X,side=TOP,pady=10)  
  
top.mainloop()
```

### *Output:*





## Internal Padding in Horizontal direction(ipadx)

### *Example:*

```
from tkinter import *  
  
top = Tk()  
  
l1 = Label(top, text="Label1", bg="blue")  
l2 = Label(top, text="Label2", bg="red" )  
  
l1.pack(fill=X, side=TOP, expand=True, ipadx=10)  
l2.pack(fill=X, side=TOP, ipadx=10)  
  
top.mainloop()
```

### *Output:*



## Internal Padding in Y Direction(ipady):

### *Example:*

```
from tkinter import *
```

```

top=Tk()

l1=Label(top,text="Label1",bg="blue")

l2=Label(top,text="Label2",bg="red" )

l1.pack(fill=X,side=TOP,expand=True,ipadx=10)

l2.pack(fill=X,side=TOP,ipady=10)

top.mainloop()

```

***Output:***



### **3.2.2. Place Layout**

Place is the most complex manager out of the 3 managers. It uses absolute positioning, when we choose place lay out in our design, then we need to specify the position of the widgets using x and y coordinates. The size and position of the widgets will not be changed when we resize the window.

***Example:***

```

fromtkinter import *

top=Tk()

l1=Label(top,text="Label1",bg="blue")

```

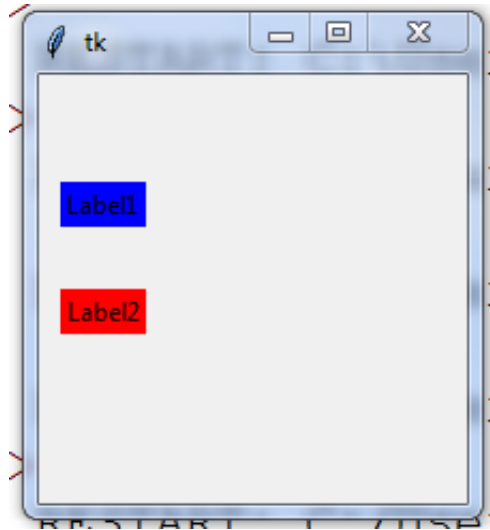
```
l2=Label(top,text="Label2",bg="red" )
```

```
l1.place(x=10,y=50)
```

```
l2.place(x=10,y=100)
```

```
top.mainloop()
```

***Output:***



***Explanation:***

Here Label1 is placed in the position (10,50) and label2 is placed in the position (10,100).

### **3.2.3 Grid Layout**

Pack Layout is not easy to understand and it is difficult to change the existing design. By using place layout, we can control the positioning of

widgets but it is complex than pack. Grid is one of the most versatile layout manager out of the three layout managers in python. By using Grid layout, the widgets can be placed in rows and columns.

***Example:***

```
from tkinter import *  
  
top=Tk()  
  
l1=Label(top,text="Label1",bg="blue")  
  
l2=Label(top,text="Label2",bg="red" )  
  
l3=Label(top,text="Label2",bg="green" )  
  
l1.grid(row=0,column=0)  
  
l2.grid(row=0,column=1)  
  
l3.grid(row=1,column=1)  
  
top.mainloop()
```

***Output:***



***Explanation:***

Here Label 1 is placed in 0<sup>th</sup> row and 0<sup>th</sup> column. Label 2 is placed in 0<sup>th</sup> row and 1<sup>st</sup> column and Label 3 is placed in 1<sup>st</sup> row and 1<sup>st</sup> column.

### **3.3 FONT**

There are three ways to specify font in python.

- 1.By using Font Tuple
- 2.By using Font Object
- 3.By using XFont

#### **3.3.1 Simple Font Tuple**

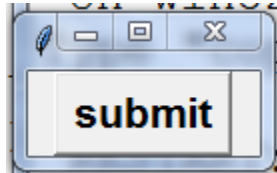
Font can be specified using tuple. Here the font tuple consists of three elements. First element specifies font family, second element specifies font size and third element specifies font style.

Ex: t=("Arial",14,"Bold")

***Example:***

```
from tkinter import *  
  
top=Tk()  
  
b1=Button(text="submit",font=("Arial","16","bold"))  
  
b1.pack()  
  
top.mainloop()
```

***Output:***



Explanation:

Text for the Button has been set in the Arial font with size 16 and Bold style.

### **3.3.2 Font Object**

Font object can be created by importing tkFont module.

Syntax for Font class constructor is:

Import tkFont

Font fl=tkFont.Font(parameters,.....)

Here is the list of parameters:

- |            |   |
|------------|---|
| Family     | – The font family name as a string.   |
| size       | – The font height as an integer in points. To get a font n pixels high, use -n. |
| weight     | – "bold" for boldface, "normal" for regular weight.                             |
| Slant      | – "italic" for italic, "roman" for unslanted.                                   |
| underline  | – 1 for underlined text, 0 for normal.  |
| Overstrike | – 1 for overstruck text, 0 for normal   |

***Example:***

```
fromtkinter import *
```

```
fromtkFont import *
```

```
top=Tk()
```

```
f1=Font(family="Helvetica",size=20,weight="bold",slant="italic",underline=1,overstrike=1)
```

```
l1=Label(top,text="Label1",bg="blue",font=f1)
```

```
l1.pack()
```

```
top.mainloop()
```

### **X Window Fonts:**

If you are running under the X Window System, you can use any of the X font names.

## **3.4 COLORS**

Tkinter represents colors with strings. There are two general ways to specify colors in Tkinter :

- We can use a string specifying the proportion of red, green and blue in hexadecimal digits. For example,
  - "#fff"            -- white,
  - "#000000"        -- black,
  - "#000fff000"     -- pure green
  - "#00ffff"        -- pure cyan
- We can also use any locally defined standard following color names.
  - "white"
  - "black"



- "red"
- "green"
- "blue"
- "cyan"
- "yellow"
- "magenta"

The common color options are :

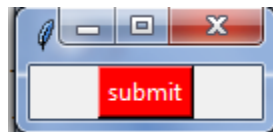
- |                    |  |
|--------------------|--|
| Active background  | – Specifies Background color for the widget when the widget is active.           |
| activeforeground   | – Specifies Foreground color for the widget when the widget is active.           |
| background         | – Specifies Background color for the widget. This can also be represented as bg. |
| disabledforeground | – Specifies Foreground color for the widget when the widget is disabled.         |
| foreground         | – Specifies Foreground color for the widget. This can also be represented as fg. |

- highlightbackground – Specifies Background color of the highlight region when the widget has focus.
- highlightcolor – Specifies Foreground color of the highlight region when the widget has focus.
- selectbackground – Specifies Background color for the selected items of the widget.
- selectforeground – Specifies Foreground color for the selected items of the widget.

***Example:***

```
from tkinter import *  
  
top=Tk()  
  
b1=Button(text="submit",bg="red",fg="white")  
  
b1.pack()  
  
top.mainloop()
```

***Output:***



***Explanation:***

Here the back ground of the button is red in color and foreground color of the button is white in colour.

### 3.5 CANVAS

The Canvas is a rectangular area used for drawing pictures or other complex layouts. Graphics, text, widgets or frames can be placed on a Canvas.

#### *Syntax:*

w = Canvas ( top, option=value, ... )

top            –    It represents the parent window.

Options       –    commonly used options for this widget. These options can be used as key-value pairs separated by commas.

Commonly used Options are:

bd            -    Border Width of the canvas

bg            -    Background color of the canvas

cursor       -    Cursor used in the canvas like circle, arrow and dot.

relief        -    Type of the border

width        -    Width of the canvas

Items supported by canvas:

1.Arc

2.Image

3.Line

4.Oval

## 5.Polygon

### **ARC**

Creates an arc item, which can be a chord or a simple arc.

#### ***Syntax:***

`create_arc(x0, y0, x1, y1, options.....)`

`x0,y0,x1,y1`-Top Left and Bottom Right coordinates of Bounding Rectangle

Commonly used Options:

`start,extend`-Specifies which section to draw

#### ***Example:***

```
fromtkinter import *
```

```
root=Tk()
```

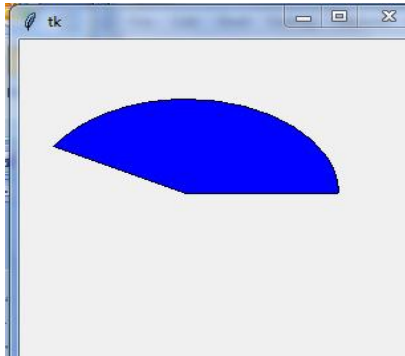
```
w = Canvas(root, width=500, height=500)
```

```
coord = 10, 50, 240, 210
```

```
arc = w.create_arc(coord, start=0, extent=150, fill="blue")
```

```
w.pack()
```

#### ***Output:***



### ***Explanation:***

Here Arc is drawn with blue color and within the bounded rectangle with top left(10,50)position and bottom right(240,210) position and started from angle 0 and extended till 150 degree.

### **3.5.1 Image**

Creates an image , which can be an instance of either the BitmapImage or the PhotoImage classes.

### ***Syntax:***

Create\_image(x,y,options....)

x,y-Specifies the position of the image

commonly used options:

anchor=Where to place the image relative to the given position.  
Default is CENTER.

image=image object

***Example:***

```
fromtkinter import *  
root=Tk()  
w = Canvas(root, width=500, height=500)  
w.create_image("F:\img2",50,50)  
w.pack()  
root.mainloop()
```

### **3.5.2 Line**

Creates a line item.

***Syntax:***

```
canvas.create_line(x0, y0, x1, y1, ...,xn, yn, options)
```

x0,y0,x1,y1->coordinates of line

Commonly used options:

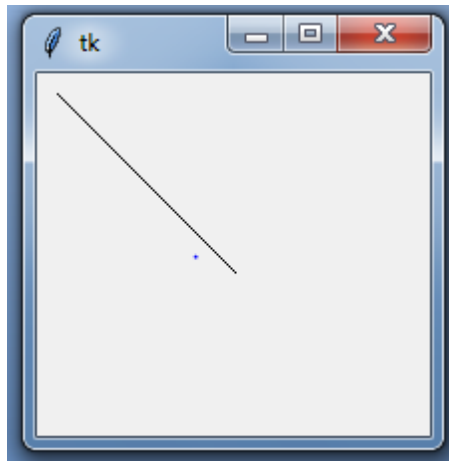
activefill-Color of the line when it is active

width        -Width of the line

***Example:***

```
from tkinter import *  
root=Tk()  
w = Canvas(root, width=500, height=500)  
w.create_line(10,10,100,100,activefill="red")  
w.pack()  
root.mainloop()
```

***Output:***



**3.5.3 Oval**

Creates a circle or an ellipse at the given coordinates. It takes two pairs of coordinates; the top left and bottom right corners of the bounding rectangle for the oval.

***Syntax:***

```
canvas.create_oval(x0, y0, x1, y1, options)
```

x0, y0, x1, y1- the top left and bottom right corners of the bounding rectangle

Options:

activefill-Color of the oval when it is active

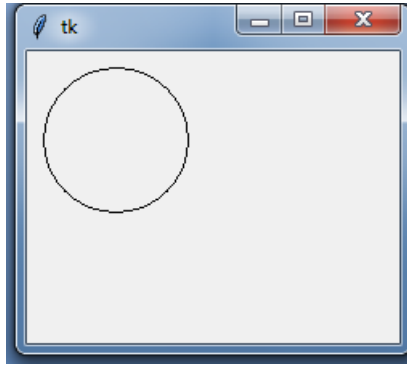
width -Width of the line

***Example:***

```
fromtkinter import *  
  
root=Tk()  
  
w = Canvas(root, width=500, height=500)  
  
w.create_oval(10,10,100,100,activefill="red")  
  
w.pack()  
  
root.mainloop()
```



***Output:***



### **3.5.4 Polygon**

Creates a polygon item that must have at least three vertices.

***Syntax:***

```
canvas.create_polygon(x0, y0, x1, y1,...xn, yn, options)
```

x0, y0, x1, y1,...xn, yn-Coordinates of polygon

Options:

Activefill-Color of the oval when it is active

width -Width of the line

***Example***

```
fromtkinter import *
```

```
root=Tk()

w = Canvas(root, width=500, height=500)

w.create_polygon(50,50,20,20,100,100,activefill="red")

w.pack()

root.mainloop()
```

### **3.6 WIDGETS IN PYTHON**

Widgets are standard graphical user interface (GUI) elements, like different kinds of buttons and menus.

#### **3.6.1 Label**

A Label widget shows text to the user about other widgets used in the application. The widget can be updated programmatically.

Syntax to create Label:

```
w=Label (root ,options)
```

root    - Parent Window

List of commonly used options are given below:

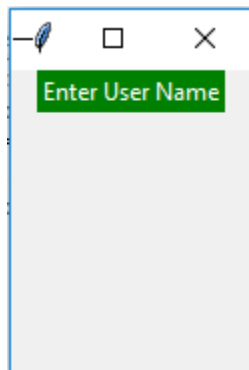
**Table 3.1: Options for Label Widget**

Option	Description
anchor	It specifies the exact position of the text within the size provided to the widget. The default value is CENTER, which is used to center the text within the specified space.
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor. eg: dot, arrow, circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text
relief	Specifies type of border
text	Text to be displayed in the widget
underline	Underline the label text

***Example:***

```
from tkinter import *  
  
root=Tk()  
  
l1=Label(root,text="Enter User Name",bg="green",fg="white")  
  
l1.pack()  
  
root.mainloop()
```

***Output:***



***Explanation:***

Here Label has been created with green background color and white foreground color with the text “Enter User Name”.

## ENTRY

The Entry widget is used to create the single line text-box to the user to accept a value from the user. It can accept the text strings from the user. It can receive one line of text from the user. For multiple lines of text, the text widget will be used.

Syntax for creating Entry Widget:

```
w=Entry(root, options)
```

root-Main Window

**Table 3.2: List of commonly used options for Entry Widget**

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor.eg:dot,arrow,circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text

Option	Description
relief	Specifies type of border
text	Text to be displayed in the widget
underline	Underline the label text
selectbackground	Background color of the selected text
selectforeground	Foreground color of the selected text
show	Specifies the character used to mask characters in the text box

***Example:***

```

fromtkinter import *

root=Tk()

l1=Label(root,text="Enter User Name",bg="green",fg="white")

e1=Entry(root,show="*")

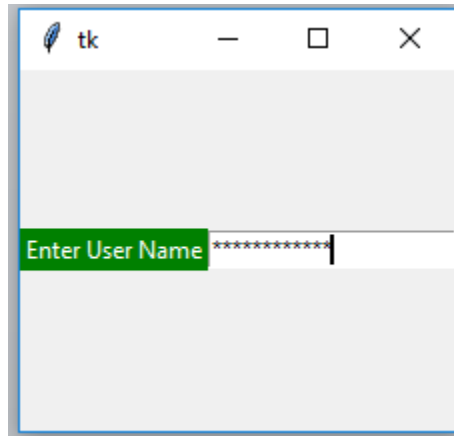
l1.pack(side=LEFT)

e1.pack(side=RIGHT)

root.mainloop()

```

***Output:***



***Explanation:***

Here Label and entry widgets are created. Since the show attribute value is \*, the characters entered in the text box appeared as “\*”.

### **3.6.2 Button**

Button Widget is used to create various kinds of buttons. The user can interact with the button. They can contain text or images.

Syntax for creating Button:

```
b=Button(root,options)
```

root-main window

**Table 3.3: List of commonly used options for Button**

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor.eg:dot,arrow,circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text
relief	Specifies type of border
text	Text to be displayed in the widget
underline	Underline the label text
command	It is set to function name which will be called the button is clicked

***Example:***

```
fromtkinter import *  
root=Tk()
```

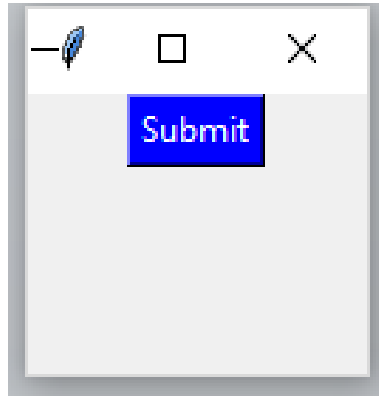


```
b1=Button(root,text="Submit",bg="blue",fg="white")
```

```
b1.pack()
```

```
root.mainloop()
```

***Output:***



### **3.6.3 Checkbutton**

The Checkbutton is used to track the user's choices provided to the application. Checkbutton is used to implement the on/off selections. The Checkbutton can contain the or images or text. The Checkbutton is mostly used to provide many choices to the user among which, the user needs to choose the one.

Syntax for creating Check Button:

```
b=CheckButton(root,options)
```

root-main window

**Table 3.4: List of possible options for Checkbutton**

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor.eg:dot,arrow,circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text
relief	Specifies type of border
text	Text to be displayed in the widget
underline	Underline the label text
command	It is set to function name which will be called the button is clicked
offvalue	Set value to the control variable if the button is checked.Default Value is 1
onvalue	Set value to the control variable if the button is unchecked.Default Value is 0
selectcolor	Set color of the check button when it is checked.
selectimage	Set the image to be shown when it is checked.

***Example:***

```
fromtkinter import *
```

```
root=Tk()
```

```
c1 = Checkbutton(root, text = "C", onvalue = 1, offvalue = 0, height = 2,  
width = 10)
```

```
c2 = Checkbutton(root, text = "C++", onvalue = 1, offvalue = 0, height = 2,  
width = 10)
```

```
c3 = Checkbutton(root, text = "JAVA", onvalue = 1, offvalue = 0, height = 2,  
width = 10)
```

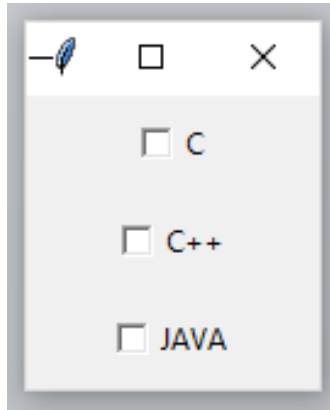
```
c1.pack()
```

```
c2.pack()
```

```
c3.pack()
```

```
root.mainloop()
```

***Output:***



### 3.6.4 Radiobutton

The Radiobutton widget is used to implement one-of-many selection. It shows multiple options to the user out of which, the user can select only one option. It is possible to display the multiple line text or images on the radiobuttons. To keep track the user's selection, the radiobutton is associated with a single variable. Each Radio button displays a single value for that particular variable.

Syntax for creating Radio Button:

```
b=RadioButton(root,options)
```

root-main window

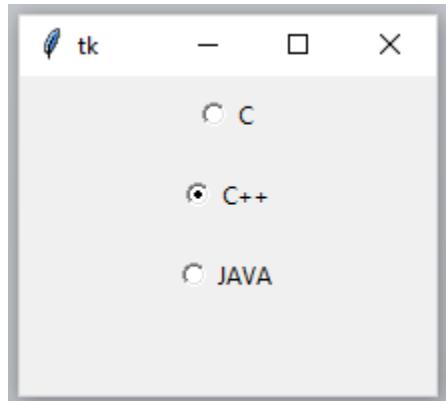
**Table 3.5: List of possible options for Radiobutton**

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor.eg:dot,arrow,circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text
relief	Specifies type of border
text	Text to be displayed in the widget
underline	Underline the label text
command	It is set to function name which will be called the button is clicked
value	Set value to the control variable if the button is selected.
selectcolor	Set color of the check button when it is checked.
selectimage	Set the image to be shown when it is checked.
variable	It is used to keep track of user choices.

***Example:***

```
from tkinter import *  
  
root = Tk()  
  
r1 = Radiobutton(root, text = "C", value = 1, height = 2, width = 10)  
r2 = Radiobutton(root, text = "C++", value = 2, height = 2, width = 10)  
r3 = Radiobutton(root, text = "JAVA", value = 3, height = 2, width = 10)  
  
r1.pack()  
r2.pack()  
r3.pack()  
  
root.mainloop()
```

***Output:***



### 3.6.5 Listbox

The Listbox widget is used to display the list items to the user. The user can choose one or more items from the list depending upon the configuration.

Syntax for creating Listbox:

```
b=Listbox(root,options)
```

root-main window

**Table 3.6: List of possible options for Listbox**

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor. eg: dot, arrow, circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text

relief	Specifies type of border
value	Set value to the control variable if the button is selected.
selectbackground	Set back ground color of the selected text.
xscrollcommand	User can scroll the list box horizontally
yscrollcommand	User can scroll the list box vertically

***Example:***

```

fromtkinter import *

top = Tk()

lbl = Label(top,text = "A list of favourite countries...")

listbox = Listbox(top)

listbox.insert(1,"India")

listbox.insert(2, "USA")

listbox.insert(3, "Japan")

listbox.insert(4, "Austrelia")

lbl.pack()

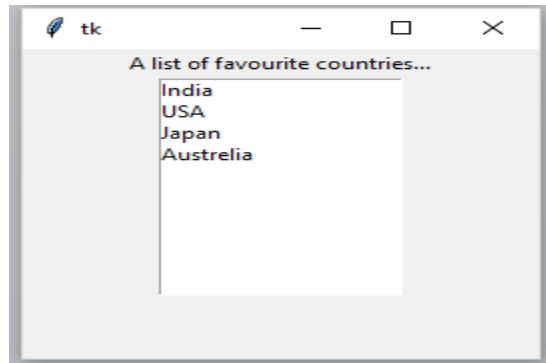
listbox.pack()

top.mainloop()
37

```



***Output:***



### **3.6.6 Message**

Its functionality is very similar to Label widget, except that it can automatically wrap the text, maintaining a given width.

Syntax for creating Message:

```
m=Message(root,options)
```

root-main window

**Table 3.7: List of possible options for Message**

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor.eg:dot,arrow,circle

Option	Description
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text
relief	Specifies type of border

***Example:***

```
fromtkinter import *

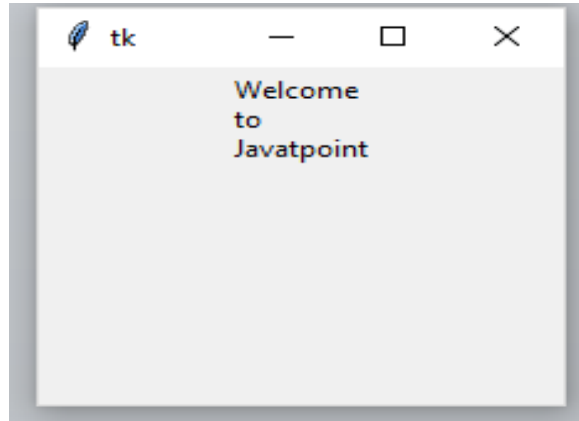
top = Tk()

msg = Message( top, text = "Welcome to Javatpoint")

msg.pack()

top.mainloop()
```

***Output:***



### 3.6.7 Text

Tkinter provides us the Entry widget which is used to implement the single line text box. Text widget provides advanced capabilities that allow us to edit a multiline text and format the way it has to be displayed, such as changing its color and font. We can also use the structures like tabs and marks to locate specific sections of the text, and apply changes to those areas.

Syntax for creating Message:

```
T=Text(root,options)
```

root-main window

**Table 3.8: List of possible options for Text**

Option	Description
bg	Specifies background color of the widget

bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor.eg:dot,arrow,circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text
relief	Specifies type of border
xscrollcommand	User can scroll the text widget horizontally
yscrollcommand	User can scroll the text widget vertically
selectbackground	Background color of the selected text

**Table 3.9: General Methods**

Method	Description
delete(startindex, endindex)	This method is used to delete the characters of the specified range
get(startindex,endindex)	It returns the characters present in the specified range.
insert(index, string)	It is used to insert the specified string at the given

	index.
--	--------

### Mark Handling Methods:

Marks are used to bookmark the specified position between the characters of the associated text.

**Table 3.10: List of Mark handling methods**

Method	Description
mark_set(mark,index)	It is used to create mark at the specified index.
mark_unset(mark)	It is used to clear the given mark
mark_names()	It is used to return names of all the marks

### Tag Handling Methods:

The tags are the names given to the specific areas of the text. The tags are used to configure the different areas of the text separately.

**Table 3.11: The list of tag-handling methods**

Method	Description
tag_add(tagname, startindex, endindex)	It is used to tag the characters in the given range
tag_config()	It is used to configure the tag properties

tag_delete(tagname)	It is used to delete the given tag
tag_remove(tagname, startindex, endindex)	It is used to remove the tag from the specified range

***Example:***

```

fromtkinter import *

top = Tk()

text = Text(top)

text.insert(INSERT, "Name.....")

text.insert(END, "Salary.....")

text.pack()

text.tag_add("Write Here", "1.0", "1.4")

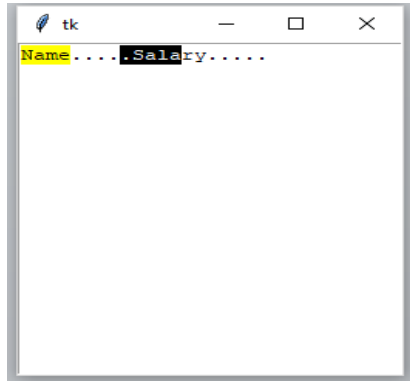
text.tag_add("Click Here", "1.8", "1.13")

text.tag_config("Write Here", background="yellow", foreground="black")

text.tag_config("Click Here", background="black", foreground="white")

```

***Output:***



### ***Explanation:***

The tag “Write Here” tags the characters from the index 0 to 4. The tag “Click Here” tags the characters from the index 8 to 13. These tags are configured using the method `tag_config()`.

### **3.6.8 Spinbox**

The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values.

Syntax:

```
w = Spinbox( master, option, ... )
```

Parameters

- **master** – This represents the parent window.
- **options** – Here is the list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.

- The Spinbox control is an alternative to the Entry control. It provides the range of values to the user, out of which, the user can select only one value. It is used in the case where a user is given some fixed number of values to choose from.
- Syntax for creating Message:
- `S=Spinbox(root,options)`
- root-main window

**Table 3.12: List of options for Spinbox**

Sr.No.	Option & Description
1	<b>activebackground</b> The color of the slider and arrowheads when the mouse is over them.
2	<b>bg</b> The color of the slider and arrowheads when the mouse is not over them.
3	<b>bd</b> The width of the 3-d borders around the entire perimeter of the trough, and also the width of the 3-d effects on the arrowheads and slider. Default is no border around the trough, and a 2-pixel border around the arrowheads and slider.



4	<b>command</b> A procedure to be called whenever the scrollbar is moved.
5	<b>cursor</b> The cursor that appears when the mouse is over the scrollbar.
6	<b>disabledbackground</b> The background color to use when the widget is disabled.
7	<b>disabledforeground</b> The text color to use when the widget is disabled.
8	<b>fg</b> Text color.
9	<b>font</b> The font to use in this widget.
10	<b>format</b> Format string. No default value.

11	<b>from_</b> The minimum value. Used together with to to limit the spinbox range.
12	<b>justify</b> Default is LEFT
13	<b>relief</b> Default is SUNKEN.
14	<b>repeatdelay</b> Together with repeatinterval, this option controls button auto-repeat. Both values are given in milliseconds.
15	<b>repeatinterval</b> See repeatdelay.
16	<b>state</b> One of NORMAL, DISABLED, or "readonly". Default is NORMAL.
17	<b>textvariable</b>

	No default value.
18	<b>to</b> See from.
19	<b>validate</b> Validation mode. Default is NONE.
20	<b>validatecommand</b> Validation callback. No default value.
21	<b>values</b> A tuple containing valid values for this widget. Overrides from/to/increment.
22	<b>vcmd</b> Same as validatecommand.
23	<b>width</b> Widget width, in character units. Default is 20.

24	<b>wrap</b> If true, the up and down buttons will wrap around.
25	<b>xscrollcommand</b> Used to connect a spinbox field to a horizontal scrollbar. This option should be set to the set method of the corresponding scrollbar.

## Methods

**Table 3.13: Methods of Spinbox objects**

Sr.No.	Methods & Description
1	<b>delete(startindex [,endindex])</b> This method deletes a specific character or a range of text.
2	<b>get(startindex [,endindex])</b> This method returns a specific character or a range of text.
3	<b>identify(x, y)</b>

	Identifies the widget element at the given location.
4	<b>index(index)</b> Returns the absolute value of an index based on the given index.
5	<b>insert(index [,string]...)</b> This method inserts strings at the specified index location.
6	<b>invoke(element)</b> Invokes a spinbox button.

### Example

Try the following example yourself –

```
from Tkinter import *
master = Tk()
```

```
w = Spinbox(master, from_=0, to=10)
w.pack()

mainloop()
```

When the above code is executed, it produces the following result –



***Example:***

```
from tkinter import *

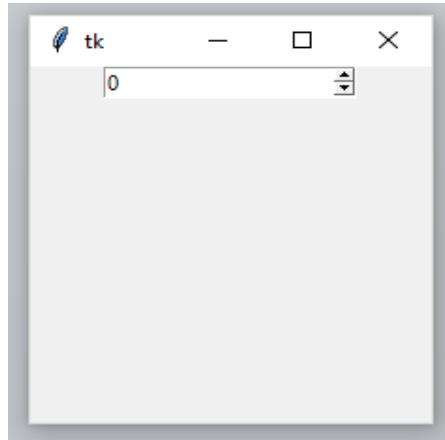
top = Tk()

spin = Spinbox(top, from_= 0, to = 25)

spin.pack()

top.mainloop()
```

***Output:***



### 3.7 FRAME

Frame widget is used to organize the group of widgets. It acts like a container which can be used to hold the other widgets. The rectangular areas of the screen are used to organize the widgets to the python application.

Syntax for creating Frame:

```
S=Frame(root,options)
```

root-main window

**Table 3.14: List of possible options for Frame**

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels

cursor	Specifies type of cursor.eg:dot,arrow,circle
height	Height of the widget
width	Width of the widget
Relief	Specifies type of border

***Example:***

```

fromtkinter import *
top = Tk()
Topframe = Frame(top)
Topframe.pack(side = TOP)
Bottomframe = Frame(top)
Bottomframe.pack(side =BOTTOM)
btn1 = Button(Topframe, text="Submit", fg="red",activebackground = "red")
btn1.pack(side = LEFT)
btn2 = Button(Topframe, text="Remove", fg="brown", activebackground =
"brown")
btn2.pack(side = RIGHT)
btn3 = Button(Bottomframe, text="Add", fg="blue", activebackground =
"blue")
btn3.pack(side = LEFT)

```

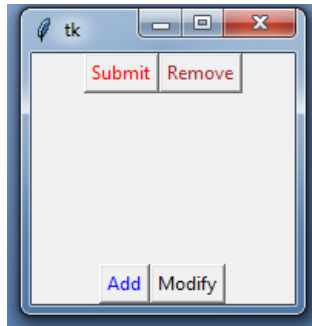


```
btn4 = Button(Bottomframe, text="Modify", fg="black", activebackground =  
"white")
```

```
btn4.pack(side = RIGHT)
```

```
top.mainloop()
```

***Output:***



***Explanation:***

Here two frames (Top Frame and Bottom Frame) have been created. Topframe contains submit and remove buttons and Bottom frame contains Add and modify buttons .

### **3.8 EVENTS AND BINDINGS IN PYTHON**

Binding function is used to deal with the events. We can bind Python's Functions and methods to an event as well as we can bind these functions to any particular widget. Events can come from various sources, including key presses and mouse operations by the user. Tkinter provides a powerful

mechanism to let you deal with events yourself. For each widget, you can bind Python functions and methods to events.

```
widget.bind(event, handler)
```

If an event matching the event description occurs in the widget, the given handler is called with an object describing the event.

A Tkinter application spends most of its time inside an event loop (entered via the **mainloop** method). Events can come from various sources, including key presses and mouse operations by the user, and redraw events from the window manager.

Tkinter provides a powerful mechanism to deal with events. For each widget, you can **bind** Python functions and methods to events.

```
widget.bind(event, handler)
```

If an event matching the *event* description occurs in the widget, the given *handler* is called with an object describing the event.

Here's a simple example:

#### **Example Program1: Capturing clicks in a window**

```
from tkinter import *
window = Tk()
def callback(event):
    print ("clicked at", event.x, event.y)
frame = Frame(window, width=100, height=100)
frame.bind("<Button-1>", callback)
frame.pack()
window.mainloop()
```

In this example, the **bind** method of the frame widget is used to bind a callback function to an event called **<Button-1>**. Run this program and click in the window that appears. Each time you click, a message like “**clicked at 44 63**” is printed to the console window.

Keyboard events are sent to the widget that currently owns the keyboard focus. The **focus\_set** method can be used to move focus to a widget:

#### **Example Program2: Capturing keyboard events**

```
from tkinter import *
window = Tk()
def key(event):
    print("pressed", repr(event.char))
def callback(event):
    frame.focus_set()
    print("clicked at", event.x, event.y)
frame = Frame(window, width=100, height=100)
frame.bind("<Key>", key)
frame.bind("<Button-1>", callback)
frame.pack()
window.mainloop()
```

If you run this script, you'll find that you have to click in the frame before it starts receiving any keyboard events.

Some of the commonly used events and some event properties are listed below:

**Table 3.15: Events**

Event	Description
<Bi-Motion>	An event occurs when a mouse button is moved while being held down on the widget.
<Button- <i>i</i> >	Button-1, Button-2, and Button-3 identify the left, middle, and right buttons. When a mouse button is pressed over the widget, Tkinter automatically grabs the mouse pointer's location. ButtonPressed- <i>i</i> is synonymous with Button- <i>i</i> .
<ButtonReleased- <i>i</i> >	An event occurs when a mouse button is released.
<Double-Button- <i>i</i> >	An event occurs when a mouse button is double-clicked.
<Enter>	An event occurs when a mouse pointer enters the widget.
<Key>	An event occurs when a key is pressed.
<Leave>	An event occurs when a mouse pointer leaves the widget.
<Return>	An event occurs when the <i>Enter</i> key is pressed. You can bind any key such as <i>A</i> , <i>B</i> , <i>Up</i> , <i>Down</i> , <i>Left</i> , <i>Right</i> in the keyboard with an event.
<Shift+A>	An event occurs when the <i>Shift</i> + <i>A</i> keys are pressed. You can combine <i>Alt</i> , <i>Shift</i> , and <i>Control</i> with other keys.
<Triple-Button- <i>i</i> >	An event occurs when a mouse button is triple-clicked.

**Table 3.16: Event Properties**

<b>Event Property</b>	<b>Description</b>
<b>char</b>	The character entered from the keyboard for key events.
<b>keycode</b>	The key code (i.e., Unicode) for the key entered from the keyboard for key events.
<b>keysym</b>	The key symbol (i.e., character) for the key entered from the keyboard for key events.
<b>num</b>	The button number (1, 2, 3) indicates which mouse button was clicked.
<b>widget</b>	The widget object that fires this event.
<b>x and y</b>	The current mouse location in the widget in pixels.
<b>x_ _root</b> and <b>y_ _root</b>	The current mouse position relative to the upper-left corner of the screen, in pixels.

The program **MouseKeyEventDemo** processes mouse and key events. It displays the window as shown in Figure 1a. The mouse and key events are processed and the processing information is displayed in the command window, as shown in Figure 1 b.

### **Example Program3 : MouseKeyEventDemo**

```
from tkinter import * # Import all definitions from tkinter
class MouseKeyEventDemo:
    def __init__(self):
        window = Tk() # Create a window
        window.title("Event Demo") # Set a title
        canvas = Canvas(window, bg = "white", width = 200, height = 100)
        canvas.pack()
        # Bind with <Button-1> event
        canvas.bind("<Button-1>", self.processMouseEvent)
        # Bind with <Key> event
        canvas.bind("<Key>", self.processKeyEvent)
        canvas.focus_set()
        window.mainloop() # Create an event loop
    def processMouseEvent(self, event):
        print("clicked at", event.x, event.y)
        print("Position in the screen", event.x_root, event.y_root)
        print("Which button is clicked? ", event.num)
    def processKeyEvent(self, event):
        print("keysym? ", event.keysym)
        print("char? ", event.char)
        print("keycode? ", event.keycode)
MouseKeyEventDemo() # Create GUI
```

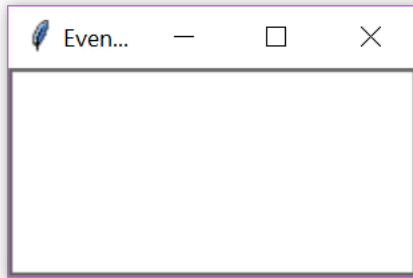


Figure 1 a Output Window of Program3

```

Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: F:/evebin31.py =====
clicked at 30 24
Position in the screen 894 226
Which button is clicked? 1
keysym? k
char? k
keycode? 75
clicked at 193 88
Position in the screen 1057 290
Which button is clicked? 1
keysym? l
char? l
keycode? 76

```

Figure 1 bOutput of Program3 for mouseclick and keypress

The program creates a canvas and binds a mouse event **<Button-1>** with the callbackfunction **processMouseEvent** on the canvas. Nothing is drawn on the canvas. So it is blank as shown in Figure 1a. When the left mouse button is clicked on the canvas, an event is created. The **processMouseEvent** is invoked to process an event that displays the mouse pointer's location on the canvas, on

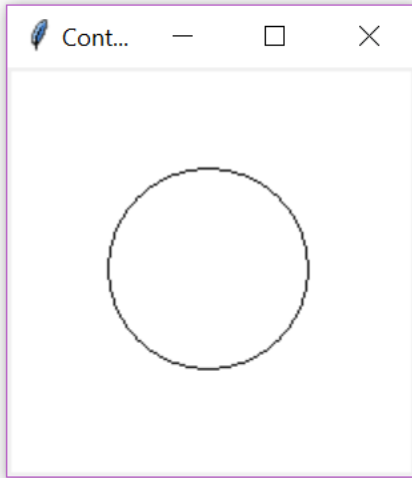
the screen, and which mousebutton is clicked. The **Canvas** widget is also the source for the key event. The program binds a key event with the callback function **processKeyEvent** on the canvas and sets the focus on the canvas so that the canvas will receive input from the keyboard.

The example Program EnlargeShrinkCircle displays a circle on the canvas. The circle radius is increased with a left mouse click and decreased with a right mouse click, as shown in Figure 2 a, 2 b, 2 c.

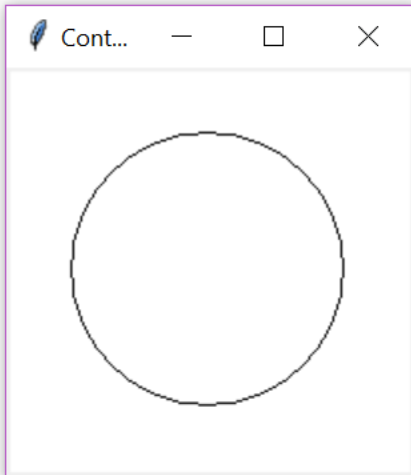


#### **Example Program4 : EnlargeShrinkCircle**

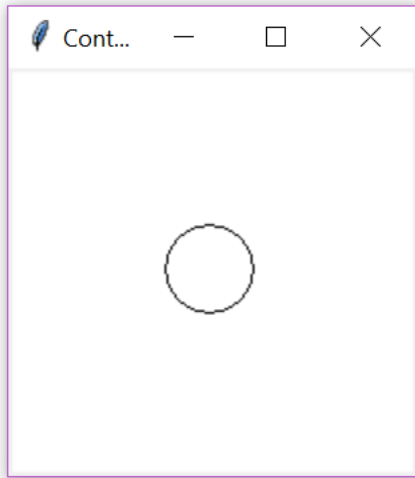
```
from tkinter import * # Import all definitions from tkinter
class EnlargeShrinkCircle:
    def __init__(self):
        self.radius = 50
        window = Tk() # Create a window
        window.title("Control Circle Demo") # Set a title
        self.canvas = Canvas(window, bg = "white",width = 200, height = 200)
        self.canvas.pack()
        self.canvas.create_oval(100 - self.radius, 100 - self.radius,100 + self.radius,
        100 + self.radius, tags = "oval")
        # Bind canvas with mouse events
        self.canvas.bind("<Button-1>", self.increaseCircle)
        self.canvas.bind("<Button-3>", self.decreaseCircle)
        window.mainloop() # Create an event loop
        def increaseCircle(self, event):
            self.canvas.delete("oval")
            if self.radius< 100:
                self.radius += 2
            self.canvas.create_oval(100 - self.radius, 100 - self.radius,100 + self.radius,
            100 + self.radius, tags = "oval")
            def decreaseCircle(self, event):
                self.canvas.delete("oval")
                if self.radius> 2:
                    self.radius -= 2
            self.canvas.create_oval(100 - self.radius, 100 - self.radius,100 + self.radius,
            100 + self.radius, tags = "oval")
EnlargeShrinkCircle() # Create GUI
```



**Figure 2 a Output Window of Program4**



**Figure 2 b Circle Radius Enlarged using Left Mouse of Program4**



**Figure 2 c Circle Radius Shrunk using Right Mouse Button of Program4**

The program creates a canvas and displays a circle on the canvas with an initial radius of **50**. The canvas is bound to a mouse event **<Button-1>** with the handler **increaseCircle** and to a mouse event **<Button-3>** with the handler **decreaseCircle**. When the left mouse button is pressed, the **increaseCircle** function is invoked to increase the radius and redisplay the circle. When the right mouse button is pressed, the **decreaseCircle** function is invoked to decrease the radius and redisplay the circle.

Another simple example is given below that shows how to use the motion event, i.e. if the mouse is moved inside of a widget:

### **Example Program5 : MouseMove**

```
from tkinter import *
def motion(event):
    print("Mouse position: (%s %s)" % (event.x, event.y))
    return
window = Tk()
display_message = "Python Programming and Machine Learning"
msg = Message(window, text = display_message)
msg.config(bg='lightgreen', font=('times', 24, 'italic'))
msg.bind('<Motion>',motion)
msg.pack()
mainloop()
```

Every time the mouse is moved in the Message widget, the position of the mouse pointer will be printed.

### **3.8.1 Handling Mouse Button Event in Python**

#### ***Example:***

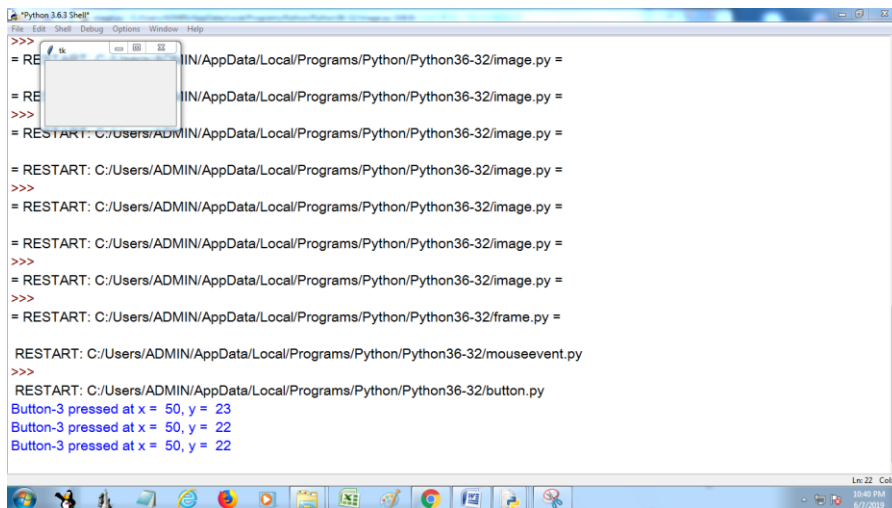
```
fromtkinter import *
fromtkinter.ttk import *
# creates tkinter window or root window
root = Tk()
# function to be called when button-2 of mouse is pressed
def pressed2(event):
    print('Button-2 pressed at x = % d, y = % d'%(event.x, event.y))
# function to be called when button-3 of mouse is pressed
```

```

def pressed3(event):
print('Button-3 pressed at x = % d, y = % d'%(event.x, event.y))
    ## function to be called when button-1 is double clicked
defdouble_click(event):
print('Double clicked at x = % d, y = % d'%(event.x, event.y))
frame1 = Frame(root, height = 100, width = 200)
# Binding mouse buttons with the Frame widget
frame1.bind('<Button-2>', pressed2)
frame1.bind('<Button-3>', pressed3)
frame1.bind('<Double 1>', double_click)
frame1.pack()
root.mainloop()

```

## Output:



```

Python 3.6.3 Shell
>>> = RE C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/image.py =
= RE C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/image.py =
>>> = RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/image.py =
= RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/image.py =
>>> = RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/image.py =
>>> = RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/image.py =
>>> = RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/frame.py =
RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/mouseevent.py
>>> RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/button.py
Button-3 pressed at x = 50, y = 23
Button-3 pressed at x = 50, y = 22
Button-3 pressed at x = 50, y = 22

```

### 3.8.2 Handling Key Press Event In Python

*Example:*

```
fromtkinter import *

fromtkinter.ttk import *


# function to be called when
# keyboard buttons are pressed
defkey_press(event):

    key = event.char

    print(key, 'is pressed')


# creates tkinter window or root window
root = Tk()

root.geometry('200x100')

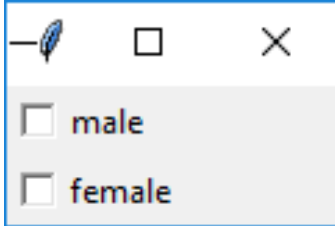

# here we are binding keyboard
# with the main window
```

67



## QUESTIONS

1. Write the Python Program to create simple window.
2. Write a Python Program to create label, entry and button components and arrange the components using Grid Layout.
3. Write a Python Program to validate user name and password.
4. Write a Python Program to display the basic shapes.
5. Write a Python program to create a following GUI design



6. Write the GUI program to create List Box for shopping cart.
7. Write a python Program to create simple calculator.
8. Write a Python Program to add image on the button.
9. Write a Python program to create simple application form.
10. Write a Python program to create check button for selecting multiple hobbies.





**SATHYABAMA**

INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE  
[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UNIT-IV - Python Programming – SCS1619**

## UNIT IV

### DATABASE AND NETWORK

*Database (using NoSQL): Connector Module –Cursor – Statements - Exceptions in database.*  
*Network connectivity: Socket module - Client – Server – Email – URL Access*

Data is very important for any organization to continue its operations. The data may be related to employees in the organization or the operational data like products information, raw material prices, sales information, profits and losses. Without data, no organization will survive. Hence, data is very important and it should never be lost.

#### **DBMS**

To store data, a file or database can be used. A file stores data in the secondary storage device like hard disk, either in the text format or binary format.

A database represents collection of data. Data can be stores in the database. Once the data is stored in the database, various operations can be performed on the data. For example, modifying the existing data, deleting the unwanted data, or retrieving the data from the database and etc. To perform such

operations, a database comes with software. This is called a database management system.

DBMS= Database + Software to manage the data

Example DBMS are MySQL, Oracle, Sybase,, SQL server etc.

Types of databases used with Python

#### **4.1 DATABASE SUPPORT**

- SQL
- NoSQL

As more and more data become available as unstructured or semi-structured, the need of managing them through NoSql database increases. Python can also interact with NoSQL databases in a similar way as is interacts with Relational databases. In this chapter we will use python to interact with MongoDB as a NoSQL database.

##### **4.1.1 MongoDB**

MongoDB stores data in JSON-like documents, which makes the database very flexible and scalable.

Where to Use MongoDB?

- Big Data
- Content Management and Delivery

- Mobile and Social Infrastructure
- User Data Management
- Data Hub

download a free MongoDB database at <https://www.mongodb.com>.

## PyMongo

Python needs a MongoDB driver to access the MongoDB database.

In this tutorial we will use the MongoDB driver "PyMongo".

We recommend that you use PIP to install "PyMongo".

PIP is most likely already installed in your Python environment.

Navigate your command line to the location of PIP, and type the following:

Download and install "PyMongo":

```
C:\Users\Your          Name\AppData\Local\Programs\Python\Python36-32\Scripts>python -m pip install pymongo
```

Now you have downloaded and installed a mongoDB driver.

Where to Use MongoDB?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management

- Data Hub

## Test PyMongo

To test if the installation was successful, or if you already have "pymongo" installed, create a Python page with the following content:

```
demo_mongodb_test.py:
```

```
import pymongo
```

### Creating a Database

To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database you want to create.

MongoDB will create the database if it does not exist, and make a connection to it.

### Example

Create a database called mydatabase

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
```

```
mydb = myclient["mydatabase"]
```

MongoDB waits until you have created a collection (table), with at least one document (record) before it actually creates the database (and collection).

### Creating a Collection

To create a collection in MongoDB, use database object and specify the name of the collection you want to create.

MongoDB will create the collection if it does not exist.

### Program

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
```

**MongoDB waits until you have inserted a document before it actually creates the collection.**

### Python MongoDB Insert Document

#### Insert Into Collection

To insert a record, or *document* as it is called in MongoDB, into a collection, we use the `insert_one()` method.

The first parameter of the `insert_one()` method is a dictionary containing the name(s) and value(s) of each field in the document you want to insert.

## Example

Insert a record in the “Customers” Collection:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
```

```
mydb = myclient["mydatabase"]
```

```
mycol = mydb["customers"]
```

```
mydict = { "name": "John", "address": "Highway 37" }
```

```
x = mycol.insert_one(mydict)
```

## Insert Multiple Documents

To insert multiple documents into a collection in MongoDB, we use the `insert_many()` method.

The first parameter of the `insert_many()` method is a list containing dictionaries with the data you want to insert:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
```

```
mydb = myclient["mydatabase"]
```

```
mycol = mydb["customers"]
```

```
mylist = [
```

```
{ "name": "Amy", "address": "Apple st 652"},  
{ "name": "Hannah", "address": "Mountain 21"},  
{ "name": "Michael", "address": "Valley 345"},  
{ "name": "Sandy", "address": "Ocean blvd 2"},  
{ "name": "Betty", "address": "Green Grass 1"},  
{ "name": "Richard", "address": "Sky st 331"},  
{ "name": "Susan", "address": "One way 98"},  
{ "name": "Vicky", "address": "Yellow Garden 2"},  
{ "name": "Ben", "address": "Park Lane 38"},  
{ "name": "William", "address": "Central st 954"},  
{ "name": "Chuck", "address": "Main Road 989"},  
{ "name": "Viola", "address": "Sideway 1633"}  
]
```

```
x = mycol.insert_many(mylist)
```

## Python MongoDB Find

In MongoDB we use the **find** and **findOne** methods to find data in a collection.

Just like the **SELECT** statement is used to find data in a table in a MySQL database.

### Find One

To select data from a collection in MongoDB, we can use the `find_one()` method.



The `find_one()` method returns the first occurrence in the selection.

### Example

Find the first document in the customers collection:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

x = mycol.find_one()

print(x)
```

### Output

```
{'_id': 1, 'name': 'John', 'address': 'Highway37'}
```

### Find All

To select data from a table in MongoDB, we can also use the `find()` method.

The `find()` method returns all occurrences in the selection.

The first parameter of the `find()` method is a query object. In this example we use an empty query object, which selects all documents in the collection.

## Example

Return all documents in the "customers" collection, and print each document:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

for x in mycol.find():

    print(x)

{'_id': 1, 'name': 'John', 'address': 'Highway37'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
```

## Filter the Result

When finding documents in a collection, you can filter the result by using a query object.

The first argument of the `find()` method is a query object, and is used to limit the search.

### Example

Find document(s) with the address "Park Lane 38":

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Park Lane 38" }

mydoc = mycol.find(myquery)

for x in mydoc:
    print(x)
```

output

```
{ '_id': 11, 'name': 'Ben', 'address': 'Park Lane 38' }
```

## Example

Find documents where the address starts with the letter "S" or higher:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": { "$gt": "S" } }

mydoc = mycol.find(myquery)

for x in mydoc:
    print(x)
```

## Output

```
{ '_id': 5, 'name': 'Michael', 'address': 'Valley
345' }
{ '_id': 8, 'name': 'Richard', 'address': 'Sky st
331' }
{ '_id': 10, 'name': 'Vicky', 'address': 'Yellow
Garden 2' }
{ '_id': 14, 'name': 'Viola', 'address': 'Sideway
1633' }
```

## Return Only Some Fields

The second parameter of the `find()` method is an object describing which fields to include in the result.

This parameter is optional, and if omitted, all fields will be included in the result.

### Example

Return only the names and addresses, not the `_ids`:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

for x in mycol.find({}, { "_id": 0, "name": 1, "address": 1 }):
    print(x)
```

### Output

```
{'name': 'John', 'address': 'Highway37'}
{'name': 'Peter', 'address': 'Lowstreet 27'}
{'name': 'Amy', 'address': 'Apple st 652'}
{'name': 'Hannah', 'address': 'Mountain 21'}
{'name': 'Michael', 'address': 'Valley 345'}
```

```
{'name': 'Sandy', 'address': 'Ocean blvd 2'}  
{'name': 'Betty', 'address': 'Green Grass 1'}  
{'name': 'Richard', 'address': 'Sky st 331'}  
{'name': 'Susan', 'address': 'One way 98'}  
{'name': 'Vicky', 'address': 'Yellow Garden 2'}  
{'name': 'Ben', 'address': 'Park Lane 38'}  
{'name': 'William', 'address': 'Central st 954'}  
{'name': 'Chuck', 'address': 'Main Road 989'}  
{'name': 'Viola', 'address': 'Sideway 1633'}
```

Sort the Result

Use the sort() method to sort the result in ascending or descending order.

The sort() method takes one parameter for "fieldname" and one parameter for "direction" (ascending is the default direction).

Example

Sort the result alphabetically by name:

```
import pymongo  
  
myclient = pymongo.MongoClient("mongodb://localhost:27017/")  
mydb = myclient["mydatabase"]  
mycol = mydb["customers"]  
  
mydoc = mycol.find().sort("name")
```

```
for x in mydoc:  
    print(x)
```

## OUTPUT

```
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}  
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}  
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}  
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}  
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}  
{'_id': 1, 'name': 'John', 'address': 'Highway37'}  
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}  
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}  
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}  
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}  
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}  
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}  
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}  
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
```

Sort Descending

Use the value -1 as the second parameter to sort descending.

```
sort("name", 1) #ascending  
sort("name", -1) #descending
```

Example

Sort the result reverse alphabetically by name:

```

import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mydoc = mycol.find().sort("name", -1)

for x in mydoc:
    print(x)

```

## Output

```

{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 1, 'name': 'John', 'address': 'Highway37'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}

```

## Python MongoDB Delete Document

To delete one document, we use the `delete_one()` method.



The first parameter of the `delete_one()` method is a query object defining which document to delete.

**Note:** If the query finds more than one document, only the first occurrence is deleted.

Example

Delete the document with the address "Mountain 21":

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Mountain 21" }

mycol.delete_one(myquery)
```

Delete Many Documents

To delete more than one document, use the `delete_many()` method.

The first parameter of the `delete_many()` method is a query object defining which documents to delete.

Example

Delete all documents where the address starts with the letter S:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": { "$regex": "^S" } }

x = mycol.delete_many(myquery)

print(x.deleted_count, " documents deleted.")

output
```

### **2 documents deleted.**

#### Delete All Documents in a Collection

To delete all documents in a collection, pass an empty query object to the `delete_many()` method:

#### Example

Delete all documents in the "customers" collection:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
```

```
x = mycol.delete_many({})

print(x.deleted_count, " documents deleted.")
```

Output:

**11 documents deleted**

Python MongoDB Drop Collection

Delete Collection

You can delete a table, or collection as it is called in MongoDB, by using the drop() method.

Example

Delete the "customers" collection:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mycol.drop()
```

The drop() method returns true if the collection was dropped successfully, and false if the collection does not exist.

## Python MongoDB Update

You can update a record, or document as it is called in MongoDB, by using the `update_one()` method.

The first parameter of the `update_one()` method is a query object defining which document to update.

**Note:** If the query finds more than one record, only the first occurrence is updated.

### Example

Change the address from "Valley 345" to "Canyon 123":

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Valley 345" }
newvalues = { "$set": { "address": "Canyon 123" } }

mycol.update_one(myquery, newvalues)

#print "customers" after the update:
```

```
for x in mycol.find():  
    print(x)
```

## OUTPUT

```
{'_id': 1, 'name': 'John', 'address': 'Highway37'}  
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}  
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}  
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}  
{'_id': 5, 'name': 'Michael', 'address': 'Canyon 123'}  
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}  
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}  
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}  
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}  
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}  
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}  
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}  
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}  
{'_id': 14, 'name': 'Viola', 'address': 'Sideway'}
```

## Update Many

To update *all* documents that meets the criteria of the query, use the `update_many()` method.

## Example

Update all documents where the address starts with the letter "S":

```
import pymongo
```

```

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": { "$regex": "^S" } }
newvalues = { "$set": { "name": "Minnie" } }
x = mycol.update_many(myquery, newvalues)

print(x.modified_count, "documents updated.")

```

Output

**2 documents updated.**

Python MongoDB Limit

o limit the result in MongoDB, we use the limit() method.

The limit() method takes one parameter, a number defining how many documents to return.

Consider you have a "customers" collection:

```

{'_id': 1, 'name': 'John', 'address': 'Highway37'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}

```

```
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}  
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}  
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}  
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}  
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}  
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}  
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}  
{'_id': 14, 'name': 'Viola', 'address': 'Sideway'}
```

## Example

Limit the result to only return 5 documents:

```
import pymongo  
  
myclient = pymongo.MongoClient("mongodb://localhost:27017/")  
mydb = myclient["mydatabase"]  
mycol = mydb["customers"]  
  
myresult = mycol.find().limit(5)  
  
#print the result:  
for x in myresult:  
    print(x)
```

## OUTPUT

```
{'_id': 1, 'name': 'John', 'address': 'Highway37'}  
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
```

```
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}  
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}  
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
```

## 4.2 CURSOR CLASS

To work with MySQL in python, connector sub module of mysql module.

```
import mysql.connector;
```

to establish connection with MySQL database, we use the connect() method of mysql.connector module as:

```
conn=mysql.connector.connect(host='localhost',database='university',user='root', password='***')
```

The connect() method returns MySQLConnection class object 'conn'.

The next step is to create cursor class object by calling the cursor() method on 'conn' object as:

```
cursor=con.cursor()
```

Cursor object is useful to execute SQL commands on the database.

it is done by execute() method of cursor object.

```
cursor.execute( sql query)
```



```
example: cursor.execute("select * from emptab")
```

The resultant rows retrieved from the table are stored in cursor object. the result can be fetched using fetchone() or fetchall() methods.

```
example: row = cursor.fetchone() # get 1 row
```

```
row = cursor.fetchall() # get all rows
```

Finally, the connection with MySQL can be closed by closing the cursor and connection objects as:

```
cursor.close()
```

```
conn.close()
```

Program: A python program to retrieve and display all rows from the student table:

```
import mysql.connector;
```

```
conn=mysql.connector.connect(host='localhost',database='university',user='root', password='***')
```

```
cursor=con.cursor()
```

```
cursor.execute("select * from stutab")
```

```
row = cursor.fetchone()
```

```
while row is not None:
```

```
print(row)

row=cursor.fetchone()

cursor.close()

conn.close()
```

Output:

```
(1001, 'Ajay', 8.5)
(1002, 'Alan', 7.5)
(1001, 'Joe', 9.00)
```

## 4.3 EXCEPTIONS CLASSES

Interacting with a database is an error prone process, so we must always implement some mechanism to handle errors.

### Built in Exceptions

**Table 4.1: Types of Exceptions**

Exception	Description
<b>Warning</b>	Used for non-fatal issues. Must subclass

	StandardError.
<b>Error</b>	Base class for errors. Must subclass StandardError.
<b>InterfaceError</b>	Used for errors in the database module, not the database itself. Must subclass Error.
<b>DatabaseError</b>	Used for errors in the database. Must subclass Error.
<b>DataError</b>	Subclass of DatabaseError that refers to errors in the data.
<b>OperationalError</b>	Subclass of DatabaseError that refers to errors such as the loss of a connection to the database. These errors are generally outside of the control of the Python scripter.
<b>Exception</b>	Description
<b>IntegrityError</b>	Subclass of DatabaseError for situations that would damage the relational integrity, such as uniqueness constraints or foreign keys.
<b>InternalError</b>	Subclass of DatabaseError that refers to errors internal to the database module, such as a cursor no longer being active.
<b>ProgrammingError</b>	Subclass of DatabaseError that refers to errors such as a bad table name and other things that can safely be blamed on you.

## 4.4 NETWORKING

For a specific purpose if things are connected together, are referred as a NETWORK. A network can be of many types, like a telephone network, television network, computer network or even a people network.

Similarly, a COMPUTER NETWORK is also a kind of setup, where it connects two or more devices to share a range of services and information in the form of e-mails and messages, databases, documents, web-sites, audios and videos, Telephone calls and video conferences etc among them.

A PROTOCOL is nothing but set of defined rules, which has to be followed by every connected devices across a network to communicate and share information among them. To facilitates End to End communication, a number of protocols worked together to form a Protocol Suites or Stacks.

Some basic Protocols are:

- IP : Internet Protocol
- FTP : File Transfer Protocol
- SMTP : Simple Mail Transfer Protocol
- HTTP : Hyper Text Transfer Protocol

The Network reference models were developed to allow products from different manufacturers to interoperate on a network. A network reference model serves as a blueprint, detailing standards for how protocol communication should occur.

The most widely recognized reference models are, the Open Systems Interconnect ( OSI ) Model and Department of Defense ( DoD, also known as TCP/IP ) model.

Network Types are often categorized by their size and functionality. According to the size, the network can be commonly categorized into Three types.

- **LANs (Local Area Networks)**
- **MANs (Metropolitan Area Networks)**
- **WANs (Wide Area Networks)**

An **Internetwork** is a general term describing multiple networks connected together. The Internet is the largest and most well-known internetwork.

Some networks are categorized by their function, as opposed to their size.

For example:

- **SAN (Storage Area Network):** A SAN provides systems with high-speed, lossless access to high-capacity storage devices.
- **VPN (Virtual Private Network):** A VPN allows for information to be securely sent across a public or unsecure network, such as the Internet. Common uses of a VPN are to connect branch offices or remote users to a main office.

In a network, any connected device is called as *host*. A host can serve as following ways:

- A host can act as a *Client*, when he is requesting information.
- A host can act as a *Server*, when he provides information.
- A host can also request and provide information, is called *Peer*.

## 4.5 SOCKET MODULE

### What Are Sockets?

A socket is a link between two applications that can communicate with one another (either locally on a single machine or remotely between two machines in separate locations).

Basically, sockets act as a communication link between two entities, i.e. a server and a client. A server will give out information being requested by a client. For example, when you visited this page, the browser created a socket and connected to the server.

### The socket Module

In order to create a socket, you use the `socket.socket()` function, and the syntax is as simple as:

```
import socket

s= socket.socket (socket_family, socket_type, protocol=0)
```

Here is the description of the arguments:

- **socket\_family:** Represents the address (and protocol) family. It can be either AF\_UNIX or AF\_INET.
- **socket\_type:** Represents the socket type, and can be either SOCK\_STREAM or SOCK\_DGRAM.
- **protocol:** This is an optional argument, and it usually defaults to 0.

After obtaining your socket object, you can then create a server or client as desired using the methods available in the socket module.

- s.recv() –It receives TCPmessage
- s.send() – It transmits TCP message
- s.recvfrom() – It receives UDPmessage
- s.sendto() – It transmits UDP message
- s.close() – It closes socket
- socket.gethostname() – It returns thehostname

## 4.6 Create a Simple CLIENT

Before we get started, let's look at the client socket methods available in Python.

```
s= socket.socket(socket.AF_INET, socket.sock_STREAM)
```

```
s.connect()Initiates a TCP server connection.
```

To create a new socket, you first import the socket method of the socket class.

```
import socket
```

Next, we'll create a stream (TCP) socket as follows:

```
stream_socket = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
```

The `AF_INET` argument indicates that you're requesting an Internet Protocol (IP) socket, specifically IPv4. The second argument is the transport protocol type `SOCK_STREAM` for TCP sockets. Additionally, you can also create an IPv6 socket by specifying the socket `AF_INET6` argument.

Specify the server.

```
server = "localhost"
```

Specify the port we want to communicate with.

```
port = 80
```

Connect the socket to the port where the server is listening.

```
server_address = ((host, port))
```

```
stream_socket.connect(server_address)
```

It's important to note that the host and port must be a tuple.

Send a data request to the server:

```
message = 'message'
```

```
stream_socket.sendall(message)
```



Get the response from the server:

```
data = sock.recv(10)
```

```
print data
```

To close a connected socket, you use the close method:

```
stream_socket.close()
```

Below is the full code for the Client/Server.

```
import socket
```

```
import sys
```

```
# Create a TCP/IP socket
```

```
stream_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
# Define host
```

```
host = 'localhost'
```

```
# define the communication port
```

```
port = 8080
```

```
# Connect the socket to the port where the server is listening
```

```
server_address = ((host, port))
```

```
print "connecting"
```

```
32
```

```
stream_socket.connect(server_address)

# Send data

message = 'message'

stream_socket.sendall(message)

# response

data = stream_socket.recv(10)

print data

print 'socket closed'

stream_socket.close()
```

## **4.7 BUILD A SIMPLE SERVER**

Now let's take a look at a simple Python server. The following are the socket server methods available in Python.

`s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)`

`s.bind():` Binds address (hostname, port number) to socket.

`s.listen():` Sets up and starts TCP listener.

`s.accept():` Accepts TCP client connection.

We will follow the following steps:

- Create a socket.
- Bind the socket to a port.
- Start accepting connections on the socket.

Here is the server program.

```
import socket

import sys

# Create a TCP/IP socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Define host

host = 'localhost'

# define the communication port

port = 8080

# Bind the socket to the port

sock.bind((host, port))

# Listen for incoming connections

sock.listen(1)

# Wait for a connection

34
```

```
print 'waiting for a connection'

connection, client = sock.accept()

print client, 'connected'

# Receive the data in small chunks and retransmit it

data = connection.recv(16)

print 'received "%s"' % data

if data:

    connection.sendall(data)

else:

    print 'no data from', client

# Close the connection

connection.close()
```

The server is now ready for incoming connections.

Now run the client and server programs in separate terminal windows, so they can communicate with each other.

### ***Server Output***

```
$ python server.py
```

waiting for a connection

('127.0.0.1', 47050) connected

received "message"

### ***Client Output***

```
$ python client.py
```

connecting

message

socket closed

## **4.8 SENDING EMAIL USING SMTP**

Simple Mail Transfer Protocol (SMTP) is a protocol, which handles sending e-mail and routing e-mail between mail servers.

Python provides **smtplib** module, which defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.

Here is a simple syntax to create one SMTP object, which can later be used to send an e-mail –

```
import smtplib
```

```
smtpObj = smtplib.SMTP( [host [, port [, local_hostname]]] )
```

Here is the detail of the parameters –

- **host** – This is the host running your SMTP server. You can specify IP address of the host or a domain name like `tutorialspoint.com`. This is optional argument.
- **port** – If you are providing *host* argument, then you need to specify a port, where SMTP server is listening. Usually this port would be 25.
- **local\_hostname** – If your SMTP server is running on your local machine, then you can specify just *localhost* as of this option.

An SMTP object has an instance method called **sendmail**, which is typically used to do the work of mailing a message. It takes three parameters –

- The *sender* – A string with the address of the sender.
- The *receivers* – A list of strings, one for each recipient.
- The *message* – A message as a string formatted as specified in the various RFCs.

### Example

Here is a simple way to send one e-mail using Python script. Try it once –

```
import smtplib
```

```
sender = 'from@fromdomain.com'
```

```
receivers = ['to@todomain.com']

message = """From: From Person from@fromdomain.com

To: To Person to@todomain.com

Subject: SMTP e-mail test

This is a test e-mail message.

"""

try:

    smtpObj = smtplib.SMTP('localhost')

    smtpObj.sendmail(sender, receivers, message)

    print "Successfully sent email"

except SMTPException:
```

Here, you have placed a basic e-mail in message, using a triple quote, taking care to format the headers correctly. An e-mail requires a **From**, **To**, and **Subject** header, separated from the body of the e-mail with a blank line.

To send the mail you use *smtpObj* to connect to the SMTP server on the local machine and then use the *sendmail* method along with the message, the from address, and the destination address as parameters (even though the from and to addresses are within the e-mail itself, these aren't always used to route mail).

If you are not running an SMTP server on your local machine, you can use *smtplib* client to communicate with a remote SMTP server. Unless you are using a webmail service (such as Hotmail or Yahoo! Mail), your e-mail provider must have provided you with outgoing mail server details that you can supply them, as follows – `smtplib.SMTP('mail.your-domain.com', 25)`





**SATHYABAMA**

INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UNIT-V - Python Programming – SCS1619**

## UNIT V

### CASE STUDY

Web Programming using Python.

Image Processing–Face Book Analysis–Twitter Analysis

#### 5.1 WEB PROGRAMMING USING PYTHON

Python is one of the most acceptable language for web application development for its efficiency and readability. There are different frameworks supported by python. A framework is a bundle of packages and modules that allows us to create web application very easily without having to handle low level activities such as thread management, process management and protocol management. We can build our application very effectively with the help of frameworks.

Below are the popular web frameworks in python.

##### 1. Django

Django is a popular python web framework and is used for larger applications. It contains everything needed for web development bundled with the framework itself. Users no need to handle database administration, routing and authentication. It works with all important databases like Oracle,MySQL,PostgreSQL,SQLite,etc.

## **Features**

1. Ridiculously fast- It is designed to handle the applications from beginning to end as quickly as possible.
2. Fully loaded – It handles user authentication, context administration, site maps and many tasks.
3. Security- It helps the developer to avoid common security mistakes such as SQL injection, cross-site scripting and cross site request forgery.
4. Scalability- It handle the heaviest traffic demands.

## **2. Flask**

Flask is a micro framework for python and good choice for building smaller applications and web services. It implements the commonly used core components of a web application framework such as URL routing, request and response objects and templates. Database access, form generation and validation are not built in functions of Flask.

## **3. Pyramid**

Pyramid is the most flexible python framework and is used for mid-high scale applications. Anyone can start to work with Pyramid without any prior knowledge about it. It comes with only some important tools which are needed for developing application. It is a finishing framework with the ability to start small application and allow us to code a solid foundation for our solution and to scale up as needed.

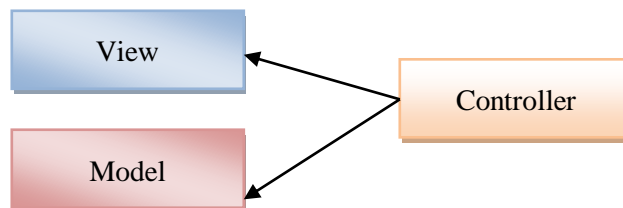
### 5.1.1 Developing simple application using Django

Web framework provide tools and libraries to simplify the task of web development operations. It solve the issues and it will make our worka lot easier. Django web framework is written on quickly and helps in building the clean and maintainable web applications.

### 5.1.2 Django Architecture

It follows a MVC-MVT architecture. MVC stands for Model View Controller. It is used for developing the web applications. It consists of three segments like model, view and controller. The following fig 1 shows the MVC architecture.

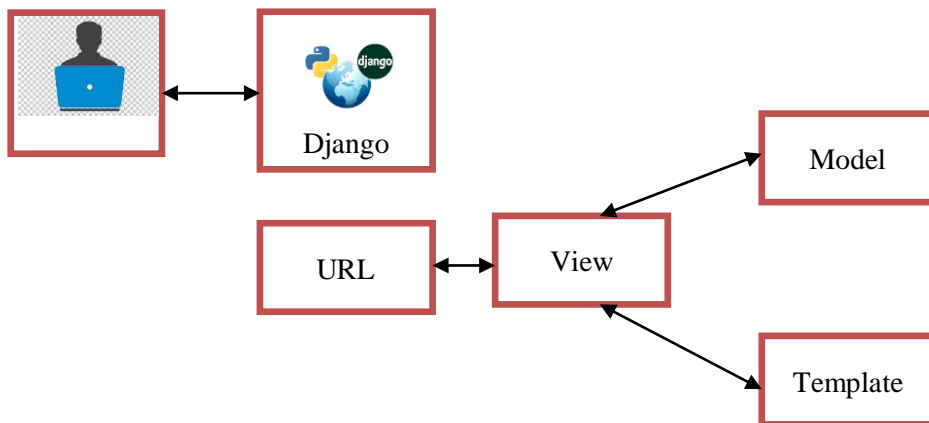
- Model : It is used for storing and maintaining our data. It is the backend where our database is defined.
- Views: views are in html. Whatever user is seeing ,it is defined as view.
- Controller: Controller is business logic that interact with the model and the view.



***Fig. 5.1: MVC Architecture***

### 5.1.3 Django MVT pattern

MVT stands for Model View Template. In MVT, predefined template is used for user interface. User no need to rewrite the code again by using template. Django will acts as controller in this part. Template is our front end which will interact with the view and the model will be used as backend. View will access both the model and templates and maps it to a URL. Fig 2 describes the MVT pattern.



*Fig. 5.2: MVT Pattern*

### 5.1.4 Django Installation

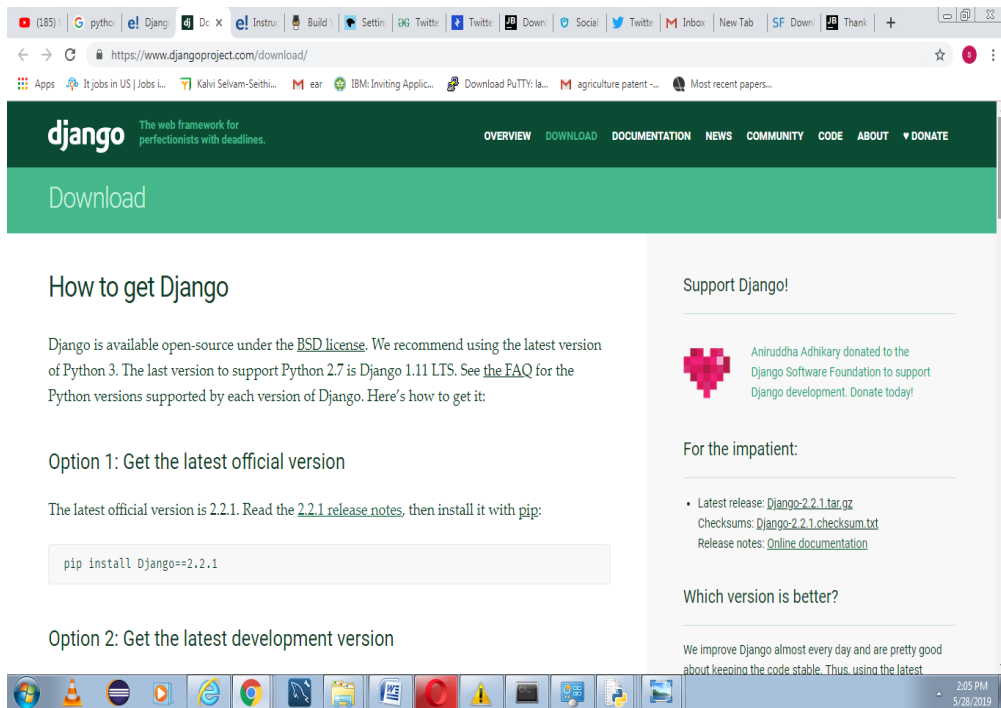
Step 1: Go to the link: <https://www.djangoproject.com/download/>. It is described in fig 3.

Step 2: Type the pip command on command prompt which run as a administrator. Fig 4 shows the installation of Django

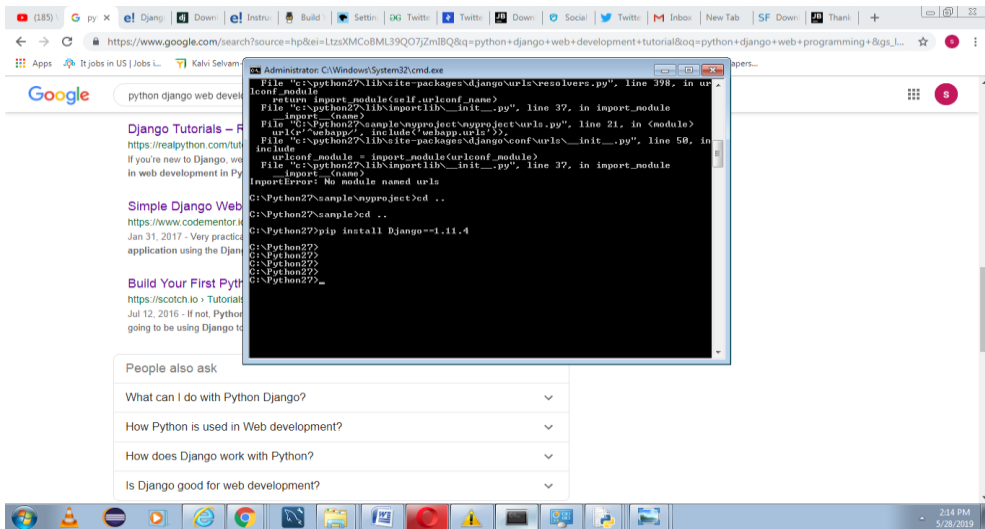
Pip install Django ==1.11.4

Step 3: Build our web application, first let's create a project. Enter in to our project folder. Execute the following command in the command prompt

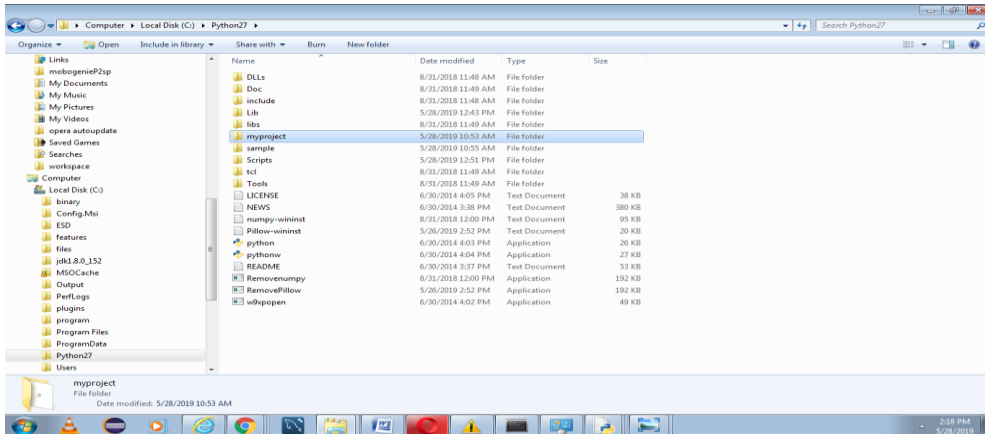
Django-admin start project myproject



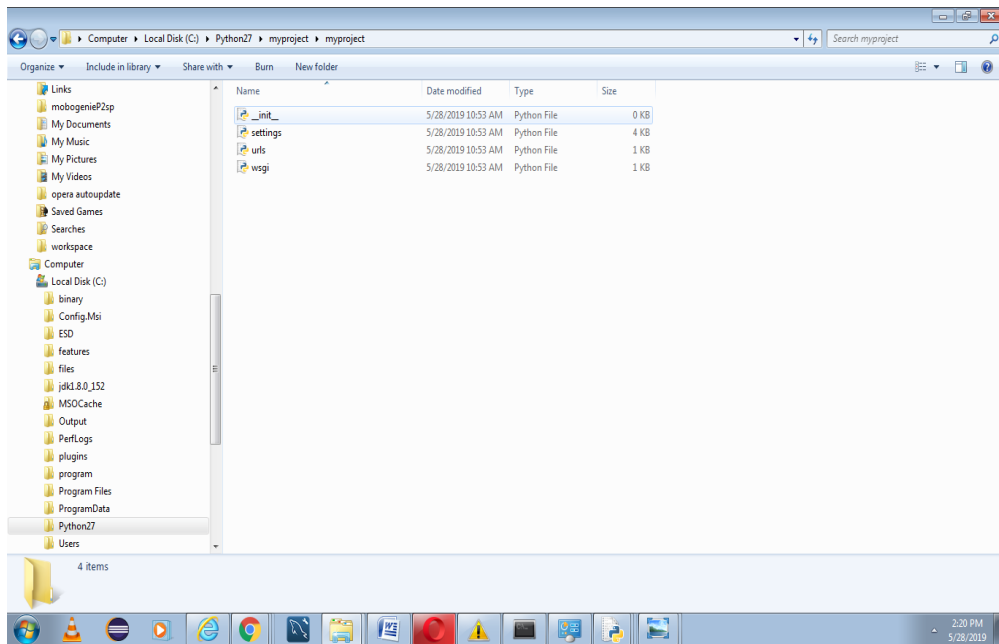
**Fig.5.3: Django Website**



**Fig.5.4: Installation of Django**



**Fig.5.5: Folder Creation in Python Environment**



***Fig.5.6: Files in Directory***

Fig 5 and 6 describes the folder creation and list of files in directory. Our project is created now. We will see the list of files in directory. Let's discuss about the following files.

1. Manage.py- It is a command line utility
2. Myproject –It is actual python package in our project.
3. Init.py-Python package
4. Settings.py- It manages all the settings of our project

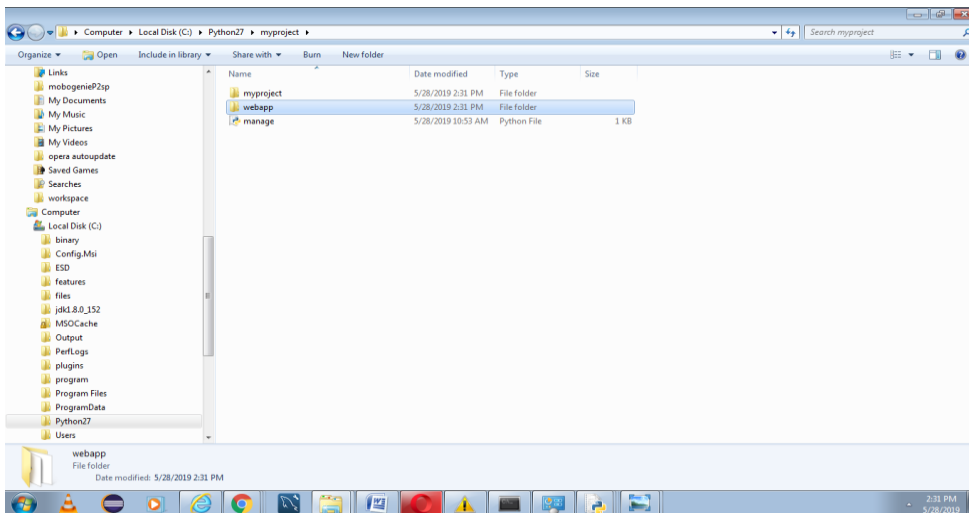


5. `Urls.py`-Main controller which maps it to our web site.
6. `Wsgi.py`- It acts as an entry point for WSGI (Web Server Gateway Interface) compatible web servers

Step 4: Create our web application and make sure that we are in the same directory as `mangae.py` and type the following command

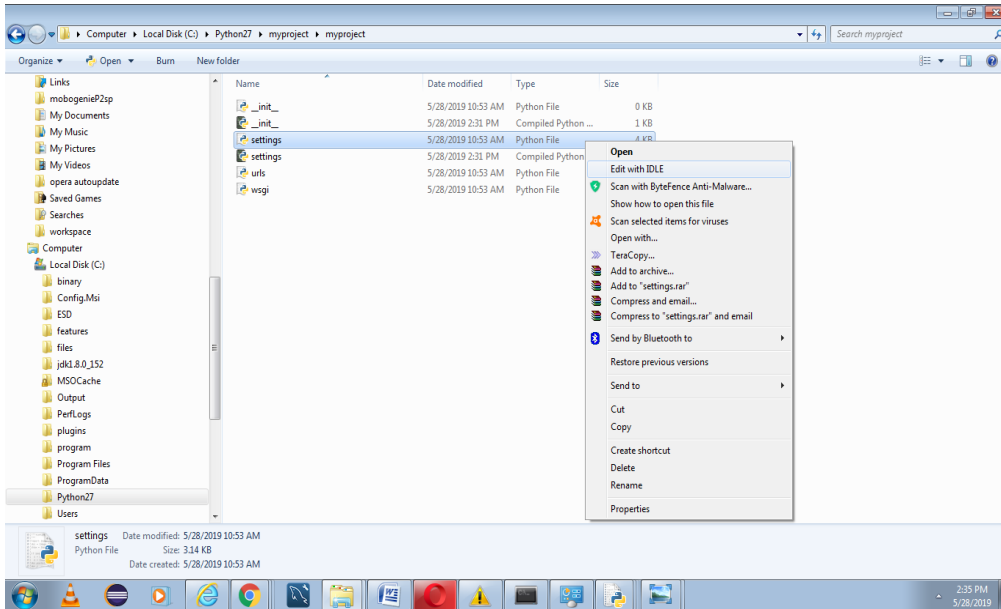
```
python manage.py startapp webapp
```

Now `webapp` is added in our project folder and few elements are added in `web app` like `view`, `test` and `model`. It is shown in fig 7.



***Fig.5.7: Creation of Web App***

Step 5: Now open our myproject/settings.py and our webapp manually. The following fig 8 shows the settings file.



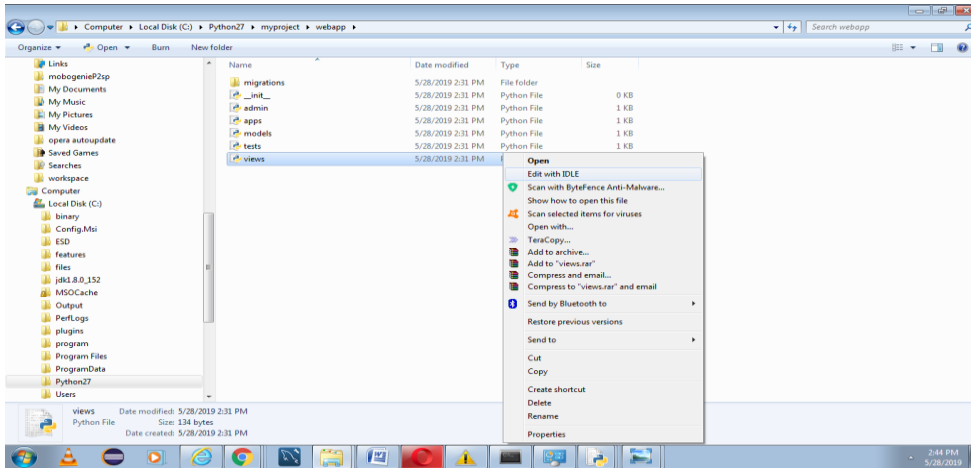
***Fig.5.8: Settings File***

```
INSTALLED_APPS = [  
  
    'webapp',  
  
    'django.contrib.admin',  
  
    'django.contrib.auth',  
  
    'django.contrib.contenttypes',
```

```
'django.contrib.sessions',  
  
'django.contrib.messages',  
  
'django.contrib.staticfiles',  
  
]
```

Step 6 : Once we have installed our app, now create a view which is shown fig 9. Open our webapp/views.py and enter the following code.

```
from django.shortcuts import render  
  
from django.http import HttpResponse  
  
def index(request):  
  
    return HttpResponse("<H2>HEY! Welcome to Sathyabama! </H2>")
```



***Fig.5.9: View File***

Step 7: We have created a view that returns http response and map this view to a URL. We need to create a “url.py” inside our webapp and enter the following code.

```
from django.conf.urls import url
from . import views
urlpatterns = [
    url(r'^$', views.index, name='index'),
]
```

Step 8:

Point the root URLconf at the webapp.urls module. Open our myproject/urls.py file and write the following code.

```
fromdjango.conf.urls import include, url
fromdjango.contrib import admin
urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^webapp/', include('webapp.urls')),
]
```

Step 9: Now start the server by type the following command

Python manage.py runserver

After running the server, goto <http://localhost:8000/webapp/> in our browser and see the “Hey!nWelcome to sathyabama “ which we defined in the index view.

## **5.2 IMAGE PROCESSING**

Image processing is a method of converting an image into digital form and perform the operations like smoothing, sharpening, contrasting and stretching on image to get an enhanced image and to extract the useful information from it. It can increase the readability of the image and enhance the quality of the image. Image processing is a part of computer vision. Computer vision is an important field in the area of artificial intelligence.

Python supports lot of libraries for image processing, including

- Open-CV- It is mainly focused on real time computer vision with variety of applications such as two dimensional and three dimensional Open-CV is an open source computer vision library for real time image and video processing. It supports a lot of algorithms related to computer vision. It supports a variety of languages like C++, Python and Java. It is available on different platforms including Windows, Linux, Android and iOS.
- Numpy and Scipy libraries- Numpy is a optimized library for numerical operations. Open-CV array structures are converted to Numpy arrays. Both are used for image manipulation and processing.
- Python Imaging Library(PIL) – It is mainly used for performing basic operations such as resize, rotation and convert between different file formats.
- Matplotlib- It is an optional choice for displaying frames from images or videos.

The following Python packages are needed to be downloaded and installed to their default locations.

- Python-2.7.x
- Numpy
- Matplotlib

Steps for installation of packages:

1. Python will be installed in C://Python27/.
2. After installation , open Python IDE and enter import numpy and verify that Numpy is working fine.
3. Download the latest Open-CV release from the internet and double click to extract it.
4. Go to opencv/build/python/2.7 folder
5. Copy cv2.pyd to C://Python27/lib/site-packages
6. Open Python IDE and type the following codes in python terminal  

```
>>>import cv2  
>>>print cv2._version_
```

### **5.2.1 Gray Scale Image**

Below are the some of the examples for demonstrating the use of libraries for image processing. The given program shows the image in gray scale. Import the all the libraries and read the image using imread function. Fig 10 shows the image in gray scale.

#### **Code:**

```
import cv2
```

```
import numpy as np
```

```
from matplotlib import pyplot as plt  
  
im = cv2.imread('boat.jpg',cv2.IMREAD_GRAYSCALE)  
  
cv2.imshow('image',im)  
  
cv2.waitKey(0)  
  
cv2.destroyAllWindows()
```

Output:



*Fig.5.10: Gray scale Image*



## 5.2.2 Geometric Transformation of Image

### 5.2.2.1 *Resize Image*

Scaling is just resizing of the image. The size of the image can be specified manually or specify with scaling factor. Resizing an image is changing the dimensions of it, be it width alone, height alone or both. The following syntax specifies the resize function.

```
cv2.resize(src,dsize, Interpolation)
```

where `src` specifies source image

`dsize` specifies destination image

Interpolation represents the different function such as `cv.INTER_AREA` for shrinking and `cv.INTER_CUBIC` for zooming operation.

Fig 11 shows the output of scaling.

Code:

```
import cv2

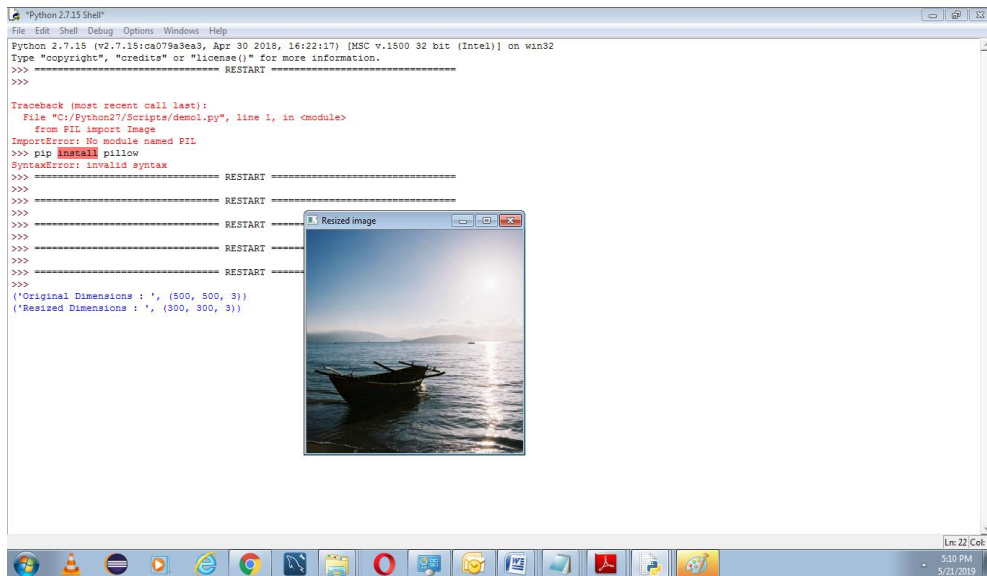
img = cv2.imread('boat.jpg', cv2.IMREAD_UNCHANGED)

print('Original Dimensions : ',img.shape)

scale_percent = 60 # percent of original size
```

```
width = int(img.shape[1] * scale_percent / 100)
height = int(img.shape[0] * scale_percent / 100)
dim = (width, height)
# resize image
resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
print('Resized Dimensions : ',resized.shape)

cv2.imshow("Resized image", resized)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



***Fig.5.11: Scaling***

### ***5.2.2.2 Translation***

Translation is the shifting of object's location from (x,y) direction to (tx,ty) location.

The transformation matrix M is represented as follows:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

Code:

```

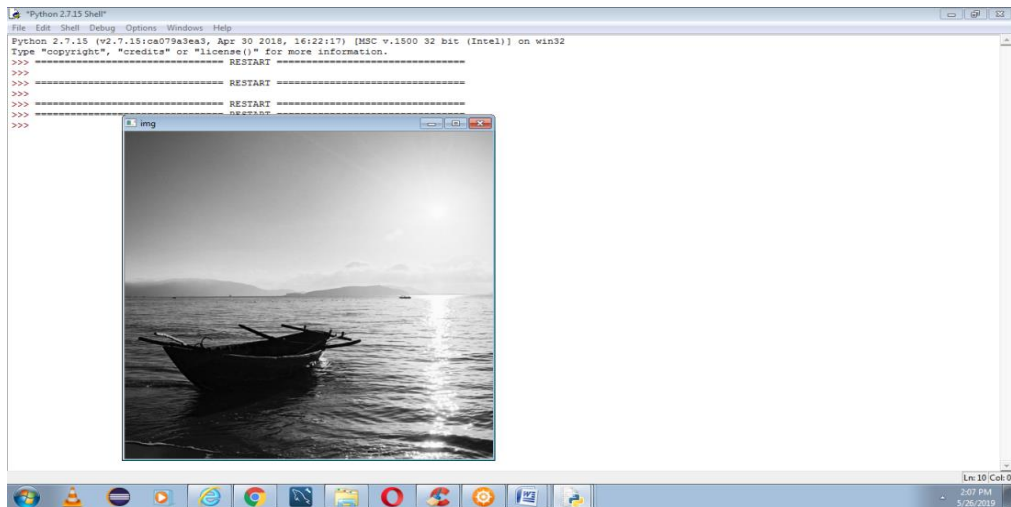
import numpy as np
18

```

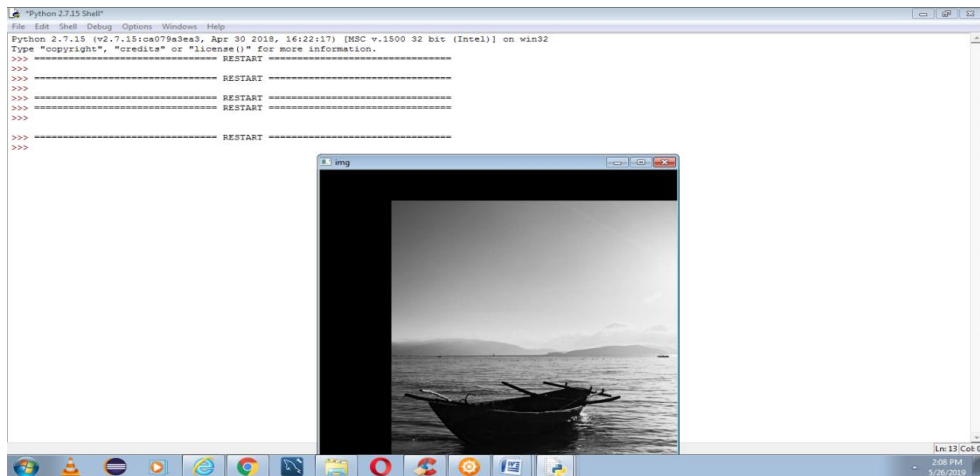
```
import cv2 as cv  
  
img = cv.imread('boat.jpg',0)  
  
rows,cols = img.shape  
  
M = np.float32([[1,0,100],[0,1,50]])  
  
dst = cv.warpAffine(img,M,(cols,rows))  
  
cv.imshow('img',dst)  
  
cv.waitKey(0)  
  
cv.destroyAllWindows()
```

where `cv.warpAffine` function specifies size of the output image.

Fig 12 and 13 describes the original image and translation result.



***Fig.5.12: Original Image***



***Fig .5.13: Translation***

### 5.2.3 Thresholding

Thresholding is a simplest method of image segmentation. It converts a gray scale image into a binary image. If a pixel is greater than a threshold value, it is assigned with one value(White), else it is assigned another value (Black). The threshold function is described as below:

`Cv2.threshold(src, thresh, maxval, type[, dst])`

This function is used to get a binary image out of a grayscaleimage for removing a noise.

1. src-Input array. This is the source image.
2. thresh-threshold value which is used for classifying the pixel.
3. maxval-Maxval which represents the value to given if pixel is more than the threshold value.
4. Type- Thresholding type. Different types are mentioned as below:
  - a. `cv2.THRESH_BINARY` (Threshold Binary)
  - b. `cv2.THRESH_BINARY_INV` (Threshold Binary Inverted)
  - c. `cv2.THRESH_TRUNC` (Truncate)
  - d. `cv2.THRESH_TOZERO` (Threshold to Zero)
  - e. `cv2.THRESH_TOZERO_INV` (Threshold to Zero Inverted)

The following fig 14 shows the outputs for different threshold functions.

Code:

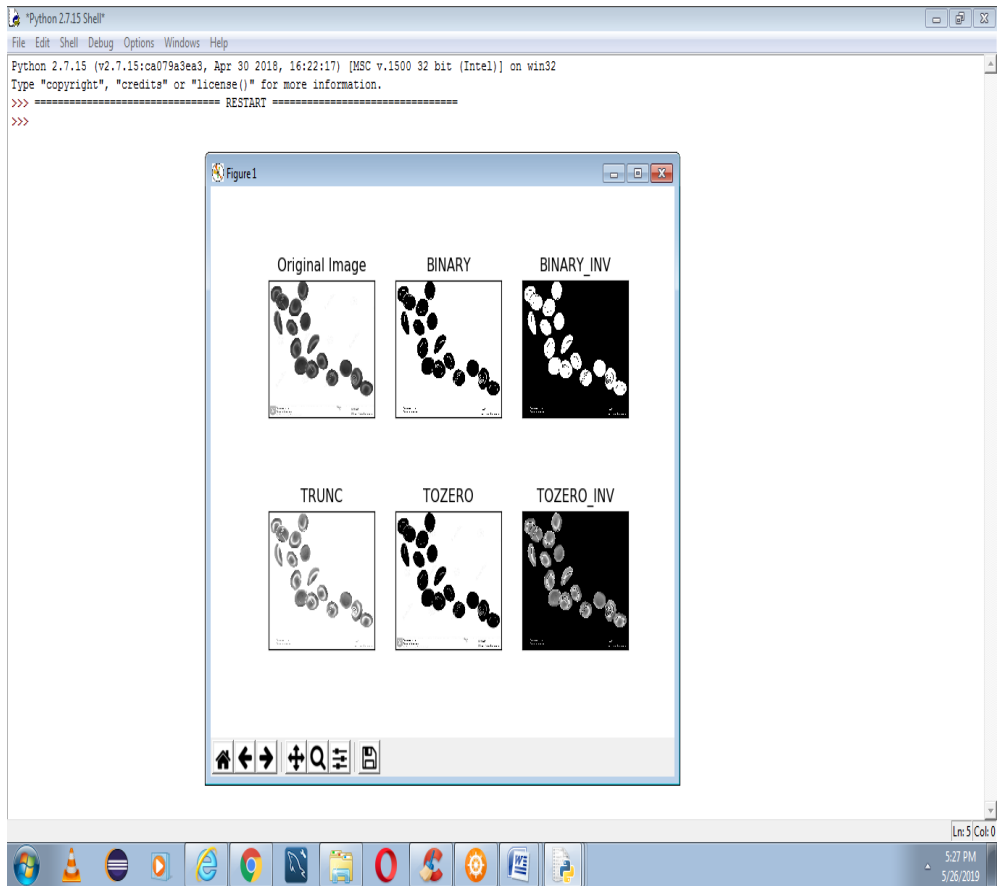
```

import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('bloodcells.jpg',0)
ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

titles = ['Original
Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in xrange(6):
plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
plt.title(titles[i])
plt.xticks([],plt.yticks([]))
plt.show()

```



***Fig.5.14: Thresholding***

### **5.2.4 Image Blurring (Image Smoothing)**

Image blurring is achieved by removing the outlier pixels in the image. It removes high frequency content from the image resulting in edges being



blurred when the filter is applied. Here the following section describes the examples of blurring techniques.

#### **5.2.4.1Averaging**

It takes the average of all the pixels under kernel area and replaces the central element with this average. This is achieved by using `cv2.blur()`. A  $3 \times 3$

filter is described as below:  $K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

Code:

```
import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('bloodcells.jpg')

blur = cv2.blur(img,(5,5))

plt.subplot(121),plt.imshow(img),plt.title('Original')

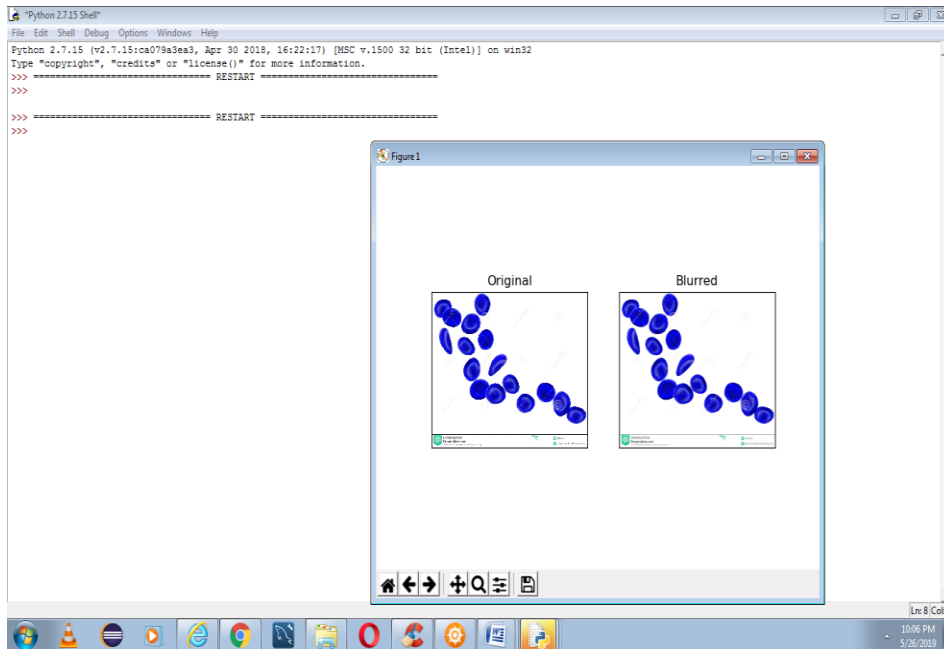
plt.xticks([], plt.yticks([]))

plt.subplot(122),plt.imshow(blur),plt.title('Blurred')

plt.xticks([], plt.yticks([]))

plt.show()
```

Fig 15 shows the image averaging output.



***Fig.5.15: Image Averaging***

#### ***5.2.4.2 Median Filtering***

Median filter is effectively used for removing salt and pepper noise. It computes the medial of all pixels under the kernel window and the central pixel is replaced by the median value. Central element is always replaced by some pixel value in the image. It reduces the noise effectively. Fig 16 shows the output of blurred image.

Code:

```
import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('noise.jpg')

median = cv2.medianBlur(img,5)

plt.subplot(121),plt.imshow(img),plt.title('Original')

plt.xticks([], plt.yticks([]))

plt.subplot(122),plt.imshow(median),plt.title('Blurred')

plt.xticks([], plt.yticks([]))

plt.show()
```

Output:



unwanted pixels in the edges. In hysteresis thresholding decides which are the edges are really edges or not by using two threshold values minval and maxval. Any edges with intensity gradient is more than maxval are considered as edges and those below minval are considered as non edged and also discarded. The following fig 17 shows the result of canny edge detection.

Code

```
import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('noise.jpg',0)

edges = cv2.Canny(img,100,200)

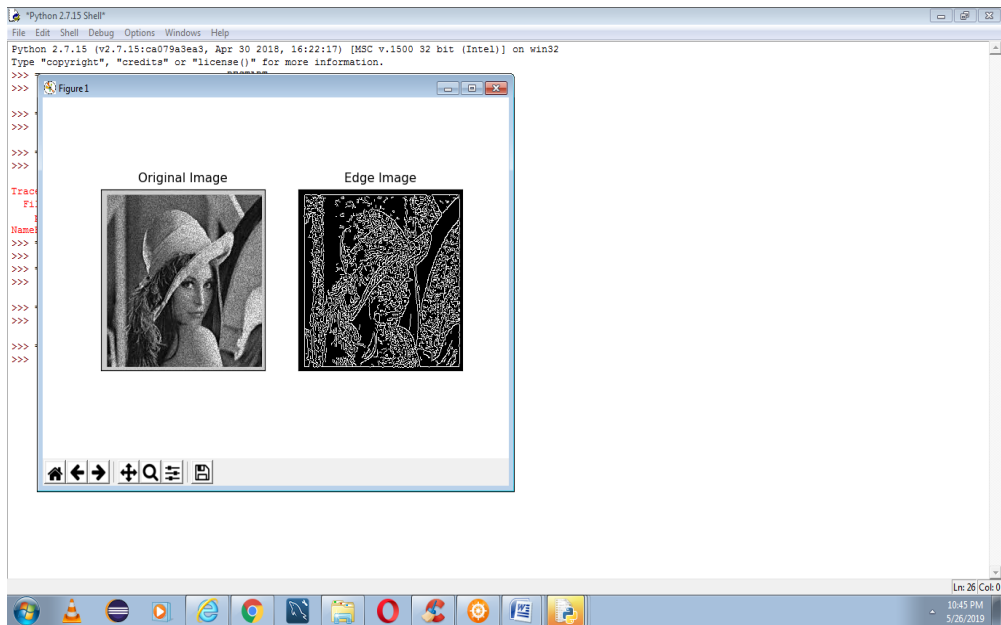
plt.subplot(121),plt.imshow(img,cmap = 'gray')

plt.title('Original Image'), plt.xticks([]), plt.yticks([])

plt.subplot(122),plt.imshow(edges,cmap = 'gray')

plt.title('Edge Image'), plt.xticks([]), plt.yticks([])

plt.show()
```



***Fig.5.17: Canny Edge Detection***

## 5.2.6 Histograms

Histogram is a graph or plot which gives us an overall idea about the intensity distribution of an image. It is a plot with pixel values(ranging from 0 to 255) in X axis and corresponding number of pixels the Y axis. `Cv2.calcHist()` function is used to find the histogram. It is described as given below:

`Cv2.calcHist(images,channels,mask,histsize,ranges[,hist[,accumulate]])`

1. Images- Source image
2. Channels-If the input is grayscale image ,its value is [0]. For color image, we can pass [0],[1],[2] to calculate histogram of blue,green or red respectively.
3. Mask- If we want to find the histogram of particular region of the image, we have to create a mask image for that one.
4. Histsize: BIN count is 256 which represent the number of pixels for every pixel value from 0 to 255.
5. Ranges- Normally it is [0,256]

Fig 18 shows the histogram output with mask.

Code:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('boat.jpg',0)
# create a mask
mask = np.zeros(img.shape[:2], np.uint8)
mask[100:300, 100:400] = 255
```

```

masked_img = cv2.bitwise_and(img,img,mask = mask)

# Calculate histogram with mask and without mask

# Check third argument for mask

hist_full = cv2.calcHist([img],[0],None,[256],[0,256])

hist_mask = cv2.calcHist([img],[0],mask,[256],[0,256])

plt.subplot(221), plt.imshow(img, 'gray')

plt.subplot(222), plt.imshow(mask,'gray')

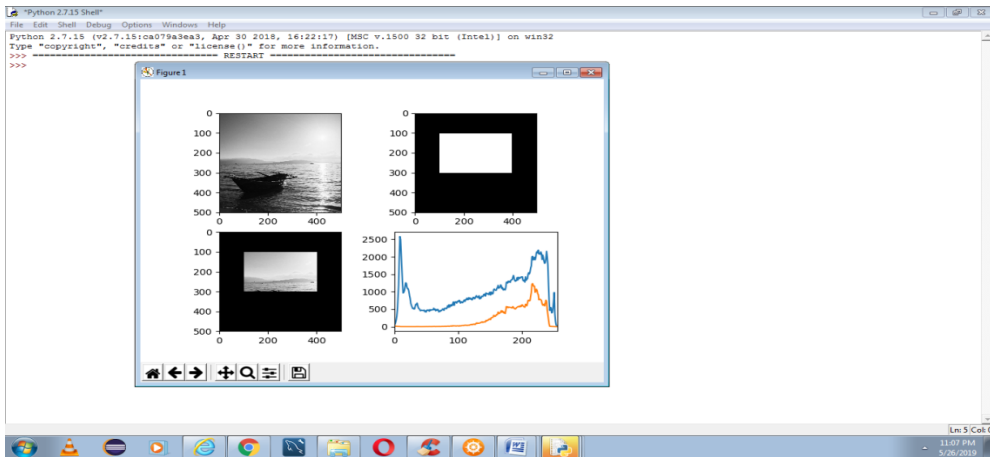
plt.subplot(223), plt.imshow(masked_img, 'gray')

plt.subplot(224), plt.plot(hist_full), plt.plot(hist_mask)

plt.xlim([0,256])

plt.show()

```



**Fig.5.18: Histograms**



### 5.3 FACE BOOK DATA ANALYSIS

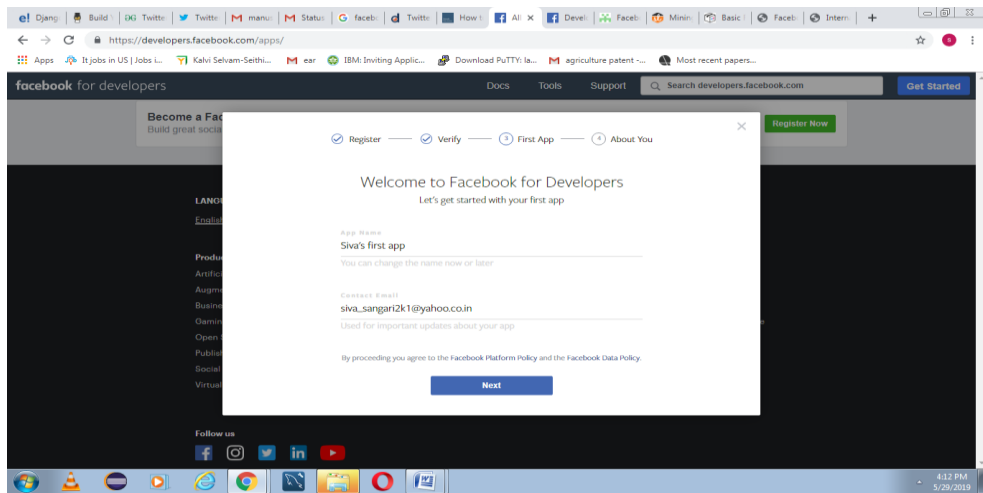
Python is used for extract data from facebook . We need to register as developer on facebook. Here the steps are listed below.

1. Go to the link [developers.facebook.com](https://developers.facebook.com) and create an there.
2. Go to the link [developers.facebook.com/tools/explorer](https://developers.facebook.com/tools/explorer).
3. Go to Myapps drop down in the top right corner and select add a new app. Choose the display name and category and then create APP ID.
4. Again go to the link [developers.facebook.com/tools/explorer](https://developers.facebook.com/tools/explorer). We will see “Graph API Explorer below “Myapps” in the top right corner. From “Graph API Explorer” drop down, select our App.
5. Select “Get Token”. From this menu select “Get user access Token “. Select permissions from the menu that appears and then select “Get access Token”

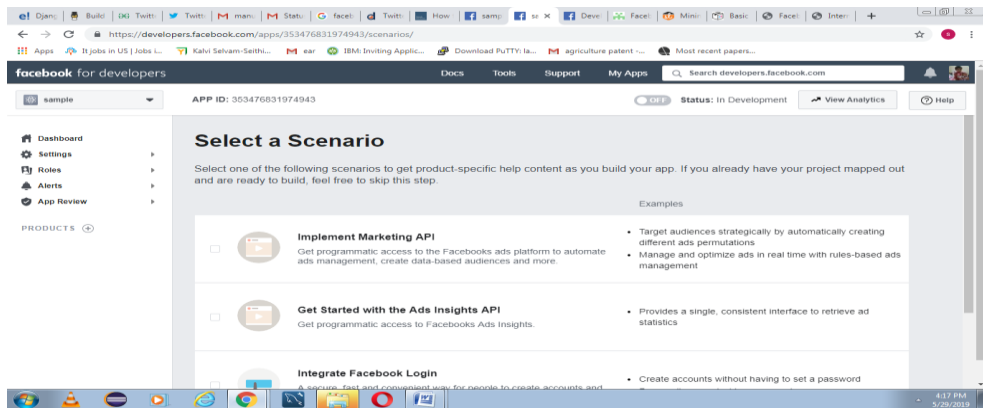
We can download datasets from other Face book pages and get these stats for each post:

- Number of likes
- Number of shares
- Number of comments

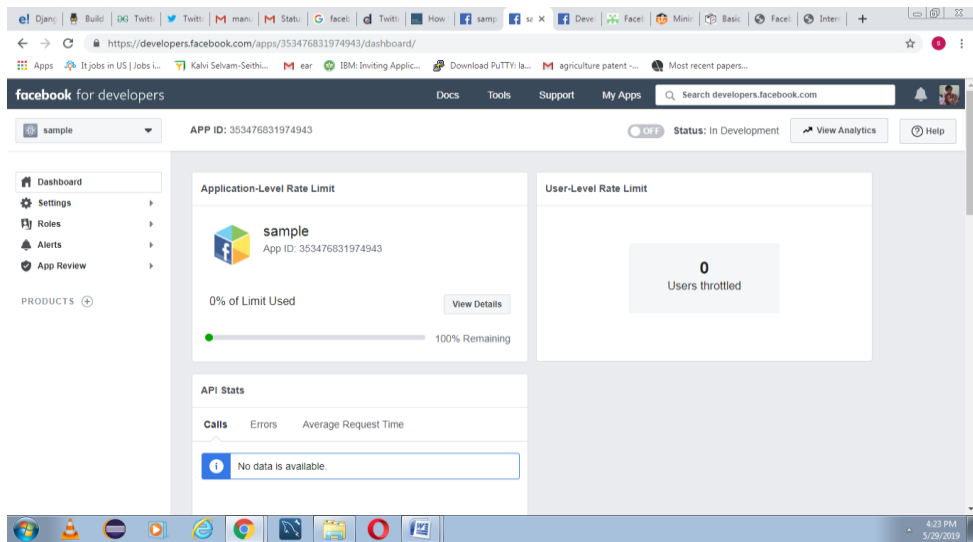
Then we can analyze this data using Excel or Tableau or Python or any software used for data analysis. Fig 19 shows the login access in facebook developer account. App creation details are described in fig 20,21 and 22.



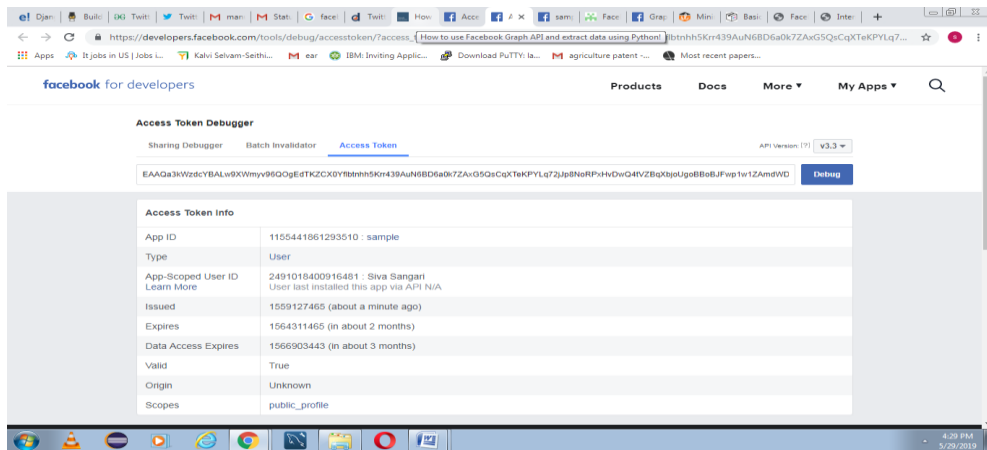
**Fig.5.19: Login in face bookdevelopers account**



**Fig.5.20: Creation of App**



**Fig.5.21: App Dashboard**



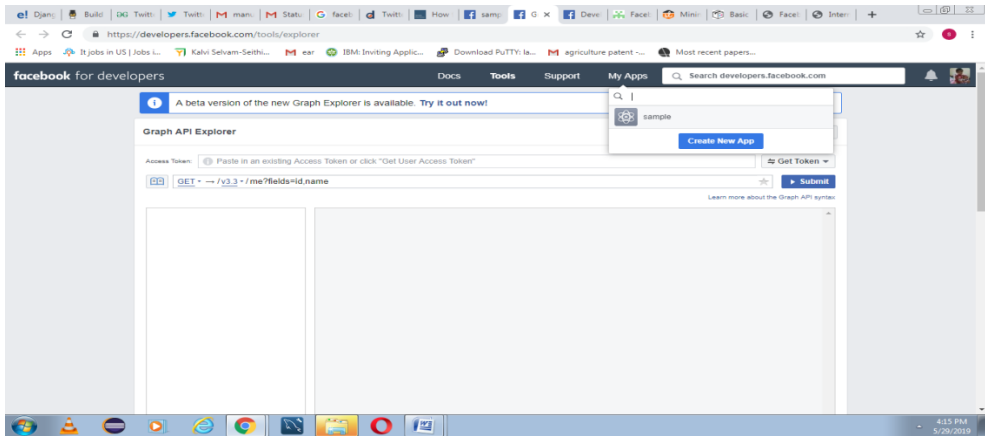
**Fig.5.22: Access Token Details**

The graph API is called social graph. It is a representation of information in face book. It consists of the following elements.

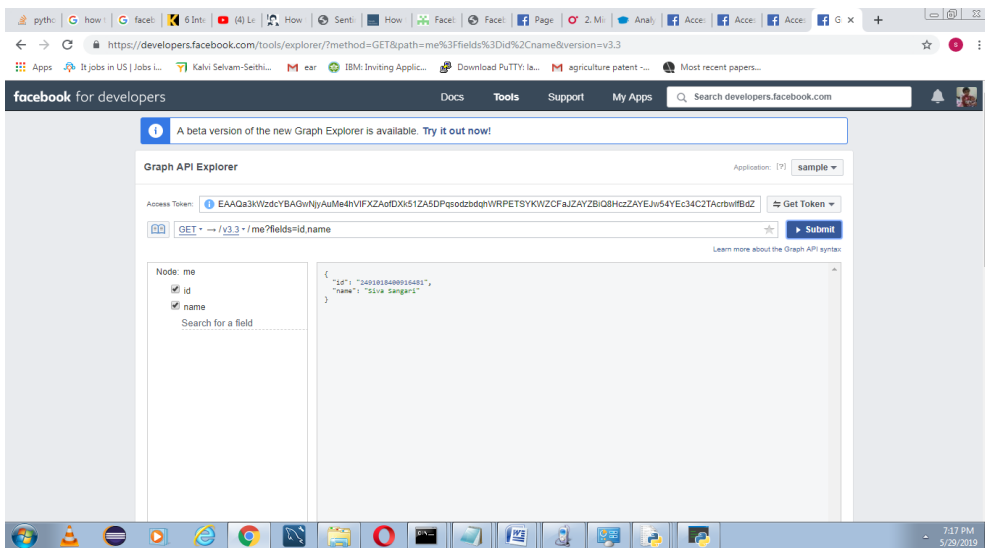
- Nodes- Individual objects such as user, photo ,page or comment
- Edges- Connection between a collection of objects and a single object such as photos or comments on a photo.
- Fields- Data about an object such as birthday or a page's name.

We can use nodes to get data about a specific object, use edges to get collections of objects on a single object and fields to get about a single object or each object in a collection. Graph API is HTTP based and works with any language.

Google graph API provides us a way by which we can get data from face book. We can put our data in facebook platform. It is a REST based API and used to query data, manage our ads on facebook, upload photos, videos and post our new stories to facebook automatically. We can this API to get our own Facebook account data. But, we need to get other users data for this we need to take several permissions from users . We need to implement oAuth protocol to implement this operation. Anyonecan authenticate and grant our permissions. Fig 23 and 24 show the face bookgraph API and node information.



**Fig.5.23: Facebook Graph API**



**Fig.5.24: Node Information**

Code

```
import json
```

```
import facebook
```

```
def main():
```

```
    token =
```

```
    "{EAAQa3kWzdcYBAKdzunCHWEixLKLvLSb5lnd8Ohs5Jh6zBef  
    MCgOPPJdYq4mTvkgpl15y1th6XpRSO5pxlnijQSCZAHShENSP06x  
    tF4WZAAD0CPFq988ZBdZAZAG8nx0DrTZA vIZBcfsYskP3JXsg7  
    GN973Q39XwhKORlmxxR5kZA5GYN3ZCyNM3uL3waUh3dm91H  
    ruwWM63ZAtYQZDZD}"
```

```
    graph = facebook.GraphAPI(token)
```

```
    page_name = raw_input("Enter a page name: ")
```

```
    # list of required fields
```

```
    fields = ['id','name','about','likes']
```

```
    fields = ','.join(fields)
```

```
page = graph.get_object(page_name, fields=fields)
```

```
print(json.dumps(page,indent=4))
```

```
if __name__ == '__main__':
```

```
    main()
```

Output:

Enter page name

Smith

Name: Smith

Id: 13456234578

Likes: 23

## **5.4 TWITTER ANALYSIS**

Sentiment analysis is the process of determining whether a piece of writing is positive, negative or neutral. In business field, companies use it to develop their strategies, understand the customer's feelings related to particular product ,product launches and reasons for not buying the particular

products. In political field , it is used to detect the consistency and inconsistency statements.

Installation:

Before we start coding, we need to register for the Twitter API <https://apps.twitter.com/>. Here we need to register an app to generate various keys associated with our API. The following keys are used for authentication

- API key
- API secret Key
- Access Token
- Access Token Secret

After creating the app we need to install the following commands.

Tweepy: Python client for the official Twitter API. Install it using following pip command.

Pip install tweepy

Textblob: Python library for processing textual data.

Pip install textblob

.

Code:



```

import re

import tweepy

from tweepy import OAuthHandler

from textblob import TextBlob


class TwitterClient(object):

    """
    Generic Twitter Class for sentiment analysis.
    """

    def __init__(self):

        """
        Class constructor or initialization method.

        # keys and tokens from the Twitter Dev Console

        consumer_key = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXX'

        consumer_secret = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXX'

```

```

access_token = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
access_token_secret = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXX'

    # attempt authentication

try:

    # create OAuthHandler object
    self.auth = OAuthHandler(consumer_key, consumer_secret)

    # set access token and secret
    self.auth.set_access_token(access_token, access_token_secret)

    # create tweepy API object to fetch tweets
    self.api = tweepy.API(self.auth)

except:

    print("Error: Authentication Failed")

def clean_tweet(self, tweet):

    """

```

Utility function to clean tweet text by removing links, special characters  
using simple regex statements.

```
"""  
  
return ''.join(re.sub("(@[A-Za-z0-9]+)|(^0-9A-Za-z \t)|  
|(\w+:\w+\S+)", " ", tweet).split())
```

```
def get_tweet_sentiment(self, tweet):
```

```
    """  
  
    Utility function to classify sentiment of passed tweet  
    using textblob's sentiment method
```

```
    """  
  
    # create TextBlob object of passed tweet text  
    analysis = TextBlob(self.clean_tweet(tweet))  
  
    # set sentiment  
  
    if analysis.sentiment.polarity > 0:  
  
    return 'positive'
```

```
elif analysis.sentiment.polarity == 0:
```

```
    return 'neutral'
```

```
else:
```

```
    return 'negative'
```

```
def get_tweets(self, query, count = 10):
```

```
    """
```

```
    Main function to fetch tweets and parse them.
```

```
    """
```

```
        # empty list to store parsed tweets
```

```
        tweets = []
```

```
        try:
```

```
            # call twitter api to fetch tweets
```

```
            fetched_tweets = self.api.search(q = query, count = count)
```

```
            # parsing tweets one by one
```

```

for tweet in fetched_tweets:

    # empty dictionary to store required params of a tweet
    parsed_tweet = {}

    # saving text of tweet
    parsed_tweet['text'] = tweet.text

    # saving sentiment of tweet
    parsed_tweet['sentiment'] = self.get_tweet_sentiment(tweet.text)

    # appending parsed tweet to tweets list
    if tweet.retweet_count > 0:

        # if tweet has retweets, ensure that it is appended only once
        if parsed_tweet not in tweets:
            tweets.append(parsed_tweet)
        else:
            tweets.append(parsed_tweet)

```

```

        # return parsed tweets

return tweets

except tweepy.TweepError as e:

    # print error (if any)

print("Error : " + str(e))


def main():

    # creating object of TwitterClient Class

    api = TwitterClient()

    # calling function to get tweets

    tweets = api.get_tweets(query = 'Donald Trump', count = 200)


    # picking positive tweets from tweets

    ptweets = [tweet for tweet in tweets if tweet['sentiment'] == 'positive']

```

```

# percentage of positive tweets

print("Positive tweets percentage: { }
%.format(100*len(ptweets)/len(tweets)))

# picking negative tweets from tweets

ntweets = [tweet for tweet in tweets if tweet['sentiment'] == 'negative']

# percentage of negative tweets

print("Negative tweets percentage: { }
%.format(100*len(ntweets)/len(tweets)))

# percentage of neutral tweets

print("Neutral tweets percentage: { } % \

    ".format(100*len(tweets - ntweets - ptweets)/len(tweets)))

# printing first 5 positive tweets

print("\n\nPositive tweets:")

for tweet in ptweets[:10]:

    print(tweet['text'])

```

```
# printing first 5 negative tweets

print("\n\nNegative tweets:")

for tweet in ntweets[:10]:

    print(tweet['text'])


if __name__ == "__main__":

    # calling main function

    main()
```

Output:

Positive tweet percentage: 22%

Negative tweet percentage: 16%



## QUESTIONS

1. List out the frame works of python in web programming.
2. Mention the libraries for image processing.
3. Explain different types of threshold function types?
4. Illustrate about canny edge detection algorithm?
5. How do you find the intensity distribution of the image?
6. Describe about the parameters of histogram function?
7. Evaluate the procedure for getting access token in Face Book data analysis?
8. Illustrate the implementation of Django web framework?
9. Elaborate about the method for removing noise from the image?
10. Assess the methods used in geo metric transformation of the image?
11. Analyze the steps involved in face book data analysis?
12. Elaborate about twitter data analysis?