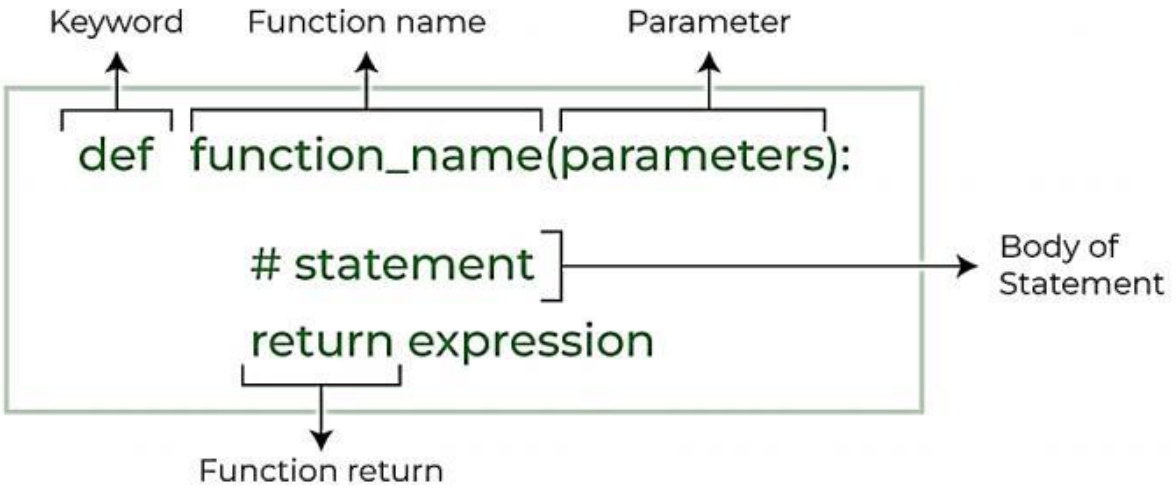
Unit- 1 Functions, Modules, OOP

Difference between Function and Method:

METHODS	FUNCTIONS
1)A method implemented **inside** of a class. It belongs to an object.	1)A function implemented **outside** of a class. It does not belong to an object.
2)A method is **called by only its associated object** (dependent). i.e. object.method()	2)A function is an independent block of code that **is not associated with any objects.** i.e. function (object)
3)We **need to declare** the **class.**	3)We **don't ne**ed to **declare the class**.
4)Methods can **access all the data** provided in the given class.	4)Functions can **only work** with the provided data.
5)A method always requires **'self'** as the first argument.	5)A function doesn't take '**self'** as an argument.

Functions :-
If a **group of statements** is **repeatedly required** then it is not possible **to write these statements every time separately**. We have to **define these statements as a single unit** and we can call that **unit any number of times** based on our requirement **without rewriting.** This unit is **called function**.

A function contains a group of statements or a block of reusable code to perform a certain task as per requirement. Once a function is written, it can be used repeatedly. Python function defined with the def keyword.



def keyword - Every function in python should start with the keyword 'def'.
Ex: def get_info():
Name of the function- Every function should be given a name, which can used to call it.
Parenthesis-After the name '()' parentheses are required
Parameters- The parameters included within the parenthesis.
Colon symbol ':'- The function definition always ends in a colon (:).
Body –The body of the function should have an indentation of one level with respect to the line containing the 'def' keyword. The **function body contains one or more statements that perform some actions**. It can also use Pass keyword. Python uses the docstring to generate documentation for the function automatically.
Ex:

```
def sample(): #function definition
print ("Hello world") #function body
sample() #function calling
```

The return statement:

The return statement is used to exit a function and go back to the place from where it was called also it returns some value. In python we can return multiple values from function.

Syntax of return:-

return [expression_list]

Ex.

```
def sub(p,q):
 r=p-q
return r
m=sub(15,5)
print("\n Subtraction is",m)
print("\n Subtraction is=",sub(25,10))
```

Advantage of functions:

1. **Increasing modularity of code –**User to divide the large programs into small groups and solve them individually, so that read and debug program faster and better.
2. **Minimizes Redundancy –** Reducing duplication of code i.e. writing the same code multiple times
3. **Maximizes Code Reusability –** Once a function is created, it can be call many times.
4. **Information hiding** :- By using hide information, We can create more secure and maintainable code.
5. Make programs **simpler to read and understand**
6. It encourages us to **call the same method** with different **inputs multiple times.**

Types of Function :-

- Functions are of two types
  1) Built-In functions
  2) User Defined Functions:

1) Built-In functions : The Functions which **are already defined in python library** are called as Built-In functions. When we want to use, then we only have to call them.

Eg. print(), len() ,min(),max() ,type() etc. are built-in functions.

2) User Defined Functions:

The functions which **are defined by user** are called as user defined function.

To Use a user definedfunction we have to follow just Two steps.

a) Defining a Function: The function is **defined by user** is called as user defined function.To define a user defined function following syntax will be used.

```
def  functionname(para1,para2,…) :
                    _____- logic-_____-___
                    -_____________________
```

Where,

def : is a keyword

Note: Logic of function should be write after indentation.

b) Calling a function : A function cannot run on its own, It **runs only when we call it**.So our next step is to call the function. And the syntax is,

Function name(para1,para2,…)

A Simple program of user defined function.

```
def sample():  # 1. defining a function
    print("This is user defined function")


sample()   # 2. Calling a function
```

Output:
This is user defined function

Function Parameter / Argument:

Parameters are **data values** , which are used in python program. Parameters can be literals variables , arrays ,lists , etc.

There are two Types of parameters

1) Actual Parameter: The parameters which **belongs to function call** are called as Actual Parameter.

2) Formal Parameter: The parameters which **belong to function definition** are called as Formal parameter.

Example of Actual and Formal Parameters :-

```
def ar(x,y):  #x and y are formal parameters.
    print("addition is",(x+y))

p=20
q=30
ar(p,q) #call (p and q are actual parameters)
```

Output: addition is 50

Parameter Passing Techniques:

- **Pass by value:** In this technique a **copy of value present in Actual parameter is send to the formal parameter.**

If we **do changes in formal parameter the changes can't reflect on Actual parameter.**

```
def Byvalue(y):
    print("value of y",y)
    y=40
    print("new value of y",y)

x=25
Byvalue(x)  #call
print("value of x",x)
```

Output:
value of y 25
new value of y 40
value of x 25

Categories of function: Based on Arguments/Parameters and return statement function have 4categories.

1) Function without Argument, without return value
2) Function without Argument, with return value
3) Function with Argument, without return value
4) Function with Argument, with return value

1) Function without Argument, without return value :

In this category,

A function **does not** contains any Arguments/ Parameters and it **do not** returns any valuefrom its definition.

```
def add():
    a=10
    b=5
    c=a+b
    print("addition=",c)

add()  #call
```

In this program add() does not accepts Any arguments and add() do not returnAny value from its definition.

Output: addition= 15

2) Function without Argument, with return value :

In this category,

A function **does not** contains Arguments/ Parameters but it returns values from its definition.

```
def add():
    a=2
    b=5
    c=a+b
    return c


x=add() #call
print("addition=",x)
```

In this program add() **does not acceptsAny arguments but add() returns**
A value present in variable c from its definition which is assigned to variable xpresent to right of function call.

Output: addition= 7

3) Function with Argument, without return value :

In this category,

A function contains Arguments/ Parameters and it **do not** returns any value from its definition.

```
def add(p,q):
    r=p+q
    print("addition=",r)


add(12,5)  #call
```

In this program our function contains **Arguments p and q called as formal Parameters** which accepts values from**Actual parameters** 12 and 3.
Our function does not returns any value.Output:

addition= 17

4) Function with Argument, with return value :

In this category,

A function contains Arguments/ Parameters also it returns values from its definition.

```
def add(p,q):
    r=p+q
    return r


x=add(10,10) #call
print("addition=",x)
```

In this program add()contains Arguments p and q called as formal Parameters which accepts values fromActual parameters 10 and 10.
Also add() returns a value presentin variable r from its definition which is assigned to variable x present to right of function call.

Output: addition=20

Anonymous/Lambda Function : In Python, an anonymous function is a function that is defined **without a name**. While normal functions are defined using the def keyword in Python, anonymous functions are defined using the **lambda keyword**.
Hence, anonymous functions are also called lambda functions.
Syntax :-

lambda arguments: expression

```
z= lambda x: x * 2
     print(z(5))
```

Output: 10

In the above program, lambda x: x * 2 is the lambda function.
Here x is the argument and x * 2 is the expression that gets evaluated and returned.

Global and Local Variables:

Global Variables : In Python, a **variable declared outside of the function** or in **global scope** is known as a global variable. This means that a global variable can be **accessed inside or outside of the function.**

```
x = 100      #global variable
def show():
    print("x inside:", x)

show()  #call to show function
print("x outside:", x)
```

Output: x inside: 100 x outside: 100

Local Variables : A variable which is **declared inside the function's** body or in the **local scope** isknown as a local variable.

```
def show():
    x = 10 #local variable
    print("x inside:", x)

show()  #call to show function
#print("x outside:", x)
```

Output: x inside: 10

Modules :

Modules **refer to a file** containing **Python statements and definitions.**
We use modules **to break down large programs into small manageable and organized files.**
Furthermore, modules provide **reusability of code.**
We can define our most used functions in a module **and import it**, instead of copying their definitions into different programs.

- Modules are of two types

1) Pre defined Module 2)User defined Module

1) Pre defined Module: The module which is **already defined by Python**, is called as Pre definedModule e.g. math, random, time , etc. module
To use predefined module firstly we have **to import** it as,

import modulename

where, import is a keyword
After this we can use the methods of module by **calling them** as,

Modulename.methodname(arg1,arg2,…)

There are large number of modules present in python library we are going to understandfew of them.

a)math Module:
Python has a built-in module called math module that can **use for mathematical tasks.**
The math module contains many predefined functions to **perform many mathematical operations**.Let's see some of methods present in math module .

* **sqrt() method**: This method is present in math module to **find square root of a number.**

```
import math
print(math.sqrt(4))
```

Output: 2.0

* **cos() method :** is used to find cosine.

```
import math
print(math.cos(0))
```

Output: -1

Also some constants defined under math module **pi** is one of them

```
import math
print (math.pi)
```

Output : 3.141

Like this many more methods and constants present in math module.

b)Time module : Python has a module named **time** , to **handle time-related tasks.**
***sleep() method**: The sleep() function **suspends (delays) execution of the current thread** for thegiven number of seconds.

```
import time
print("This is printed
immediately.")time.sleep(2.4)
print("This is printed after 2.4 seconds.")
```

Output: This is printed immediately.This is printed after 2.4 seconds.

Like this time module have many more methods.

c)Random module : Python has a built-in module that we can use to **make random numbers.**

***shuffle() Method:** Shuffle a list (**reorganize the order** of the list items):

```
import random
mylist = ["apple","banana","cherry"]
random.shuffle(mylist)
print(mylist)
```

Output : ['cherry', 'banana', 'apple']

Like this random module also have many more methods .

2) User defined module:

The module which is **defined by User** , is called as User defined Module.

Let us **create a User defined module**. Type the following and save it as **example.py**

```
def add(a, b):
    result = a + b
    return result
```

Here example is our module name and We defined add method in our moduleWhich will be accessed in further Programs.

Now let's **access our add() method** of **example** module to do addition in our program as,

```
import example
print(example.add(4,3))
```

Output: 7

*** Import with renaming :**

We can import a module by **renaming it** as follows:

```
import math as m
print("The value of pi is", m.pi)
```

Output : 3.141

*** Python from...import statement**

We can **import specific names** from a module without importing the module as a **whole**. Here is anexample.

```
from math import pi
      print("The value of pi is", pi)
```

NumPy :-

NumPy stands for **numeric python** which is a python package for the **computation and processing of the multidimensional and single dimensional array elements.**

Need of NumPy :-

1. NumPy performs array-oriented computing.
2. It efficiently implements the multidimensional arrays.
3. It performs scientific computations.
4. It is capable of performing Fourier Transform and rewriting the data stored in multidimensional arrays.
5. NumPy provides the in-built functions for linear algebra and random number generation.

NumPy doesn't come bundled with Python. We have to install it using **the python pip** installer. Execute the following command.

pip install numpy

NumPy Datatypes

The NumPy provides a higher range of numeric data types than that provided by the Python. A list of numeric data types is given in the following table.

SN	Data type	Description
1	bool_	It represents the boolean value indicating true or false. It is stored as a byte.
2	int_	It is the default type of integer. It is identical to long type in C that contains 64 bit or 32-bit integer.
3	intc	It is similar to the C integer (c int) as it represents 32 or 64-bit int.
4	intp	It represents the integers which are used for indexing.
5	int8	It is the 8-bit integer identical to a byte. The range of the value is -128 to 127.
6	int16	It is the 2-byte (16-bit) integer. The range is -32768 to 32767.
7	int32	It is the 4-byte (32-bit) integer. The range is -2147483648 to 2147483647.
8	int64	It is the 8-byte (64-bit) integer. The range is -9223372036854775808 to 9223372036854775807.
9	uint8	It is the 1-byte (8-bit) unsigned integer.
10	uint16	It is the 2-byte (16-bit) unsigned integer.
11	uint32	It is the 4-byte (32-bit) unsigned integer.
12	uint64	It is the 8 bytes (64-bit) unsigned integer.
13	float_	It is identical to float64.
14	float16	It is the half-precision float. 5 bits are reserved for the exponent. 10 bits are reserved for mantissa, and 1 bit is reserved for the sign.
15	float32	It is a single precision float. 8 bits are reserved for the exponent, 23 bits are reserved for mantissa, and 1 bit is reserved for the sign.
16	float64	It is the double precision float. 11 bits are reserved for the exponent, 52 bits are reserved for mantissa, 1 bit is used for the sign.
17	complex_	It is identical to complex128.
18	complex64	It is used to represent the complex number where real and imaginary part shares 32 bits each.
19	complex128	It is used to represent the complex number where real and imaginary part shares 64 bits each.

Example 1: numpy.array()

```
1. import numpy as np
2. arr=np.array([1,2,3])
3. print(arr)
```

Output:

```
[1, 2, 3]
```

In the above code

- We have imported numpy with alias name np.
- We have declared the 'arr' variable and assigned the value returned by np.array() function.
- In the array() function, we have passed only the elements, not axis.
- Lastly, we have tried to print the value of arr.

Example 2: Multidimensional :-

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

Output:

```
array([[1., 2., 3.],
       [4., 5., 7.]])
```

SciPy :-

The SciPy is an **open-source scientific library of Python** . It is used to solve the complex scientific and mathematical problems. It is built on top of the Numpy extension, which means if we import the SciPy, there is no need to import Numpy. The **Scipy** is pronounced as **Sigh pi**, and it depends on the Numpy, including the appropriate and fast N-dimension array manipulation.

It provides many user-friendly and effective numerical functions for numerical integration and optimization.

The **SciPy** library supports **integration, gradient optimization, special functions, ordinary differential equation solvers, parallel programming tools**, and many more. We can say that **SciPy** implementation exists in every complex numerical computation.

pip install scipy

Travis Oliphant, **Eric Jones**, and **Pearu Peterson** merged code they had written and called the new package **SciPy**.

Why use SciPy?

SciPy contain significant mathematical algorithms that provide easiness to develop sophisticated and dedicated applications. Being an open-source library, it has a large community across the world to the development of its additional module, and it is much beneficial for scientific application and data scientists.

Numpy vs. SciPy

Numpy and SciPy both are used for mathematical and numerical analysis. Numpy is suitable for basic operations such as sorting, indexing and many more because it contains array data, whereas SciPy consists of all the numeric data.

Numpy contains many functions that are used to resolve the linear algebra, Fourier transforms, etc. whereas SciPy library contains full featured version of the linear algebra module as well many other numerical algorithms.

A linear algebra problem can be solved by typing the following **scipy function**:

linalg.solve()

Linear Equation :-

The **linalg.solve** is used to solve the linear equation a*x + b*y = Z, for the unknown x, y values.

x + 3y +10z = 10
2x + 12y + 7z = 18
5x + 8y + 8z = 30

Here we will solve above linear equation by using the linear.solve command for the faster calculation.

Example :-

```
import numpy as np
from scipy import linalg
    # We are trying to solve a linear algebra system which can be given as
```

```
        #       x + 3y +10z = 10
        #       2x + 12y + 7z = 18
        #       5x + 8y + 8z = 30
    # Creating input array
a = np.array([[1, 3, 10], [2, 12, 7], [5, 8, 8]])
    # Solution Array
b = np.array([[10], [18], [30]])
    # Solve the linear algebra
x = linalg.solve(a, b)
    # Print results
print(x)
    # Checking Results
print("\n Checking results, Vectors must be zeros")
print(a.dot(X) - b)
```

Output:

```
[[4.55393586]
 [0.51311953]
 [0.39067055]]

 Checking results, Vectors must be zeros
[[0.]
 [0.]
 [0.]]
```

Object Oriented Programming :-

Features of object-oriented programming in Python:-

1)Encapsulation: This feature allows you **to group data and functions into a single unit** called a class. The data is **hidden from the outside world**, and only the methods or functions defined in the class can access it.

2)Inheritance: Inheritance is a way **to create a new class from an existing class**. The new class inherits the properties and behavior of the existing class and can also add its own properties and behavior.

3)Polymorphism: Polymorphism is the **ability of an object to take on many forms**. In Python, polymorphism is achieved through method overloading and method overriding.

4)Abstraction: Abstraction is **the process of hiding unnecessary details** while showing only the essential features of an object. It helps in reducing complexity and making code easier to understand and maintain.

5)Class: A class is a **blueprint or template that defines the properties and behavior of an ob**ject. It contains variables (called attributes) and functions (called methods) that can be used to manipulate the object.

6)Object: An **object is an instance of a class**. It contains data (attributes) and functions (methods) that operate on that data.

7)Method: A method is a **function defined in a class that can be used to manipulate the object**.

8)Instance: An instance is a **unique copy of an object** that is created from a class.

Class:

A class is user defined data type which is **collection of variables and methods.**
These variables are called ‘properties or attributes’ and methods are called ‘actions’.
All variables and methods are also available in objects, because they are created from class therefore they are called ‘instance variable’ and ‘instance method’.

Class Definition Syntax:

Where,
class: is a keyword
Classname: should be an identifier

```
class ClassName:
      # Statement-1
      .
      .
      # Statement-N
```

Object:

An Object is **an instance of a Class**. A class is like a blueprint while an instance is a copy of the classwith actual values.

An object consists of :

State : It is represented by attributes of an object. It also reflects the properties of an object.

Behavior : It is represented by methods of an object. It also reflects the response of an object withother objects.

Identity : It gives a unique name to an object and enables one object to interact with otherobjects.

Syntax to create an object

objectname=Classname()

Example

s=Student()

A simple program

```
class Student:
    roll=10         #variable
    nm="ABhi”       #variable

    #Method
    def show(self):
        print("Roll Number-",self.roll)
        print("Name-",self.nm)

s=Student()   #object creation
s.show()      #call to method
```

In python instance methods of a class contains one compulsory argument called **self** which refersmemory address of current invoking object.
In above program we accessed variables by using self keyword.Also
show() method is called using help of object.

Variables :

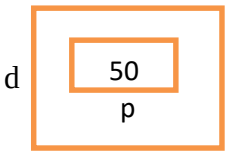
We know that variable is a **name provided to memory location**, Where data is saved.Types of variable-

a) instance variable b)class or static variable

a) Instance variable: Variables which are **declared inside the constructor and method** of class are called as Instance variables.
They are **tied to the particular object** instance of the class.

```
class Demo:
    def dis(self):
        self.p=50    # instance variables
        print("Value of p",self.p)


d=Demo()  #object creation
d.dis()   #method call
```

Output: Value of p 50



In above program p is a instance variable because it is declared in dis() method.
p variable is accessed within that method by using self as **self.p** because all instance variables areof a class is accessed as **self.variablename** .

b) Class or static variable:
A variable which is **declared inside the class definition** (**but outside any of the instance methods**)are called as class variables.
They are **not tied to any particular object of the class**, hence shared across all the objects of theclass.
Modifying a class variable affects all objects instance at the same time.

```
class Demo:
    v=100        # class variables
    def dis(self):
        print("Value of v",self.v)

d=Demo()  #object creation
d.dis()   #method call
```

Output: Value of v 100

In above program variable v is declared in class Demo but it is defined outside dis() method soIt is called as **class variable**.

Class Variables are also accessed by using object of that class in which it is declared.

Self variable :

Self is a **default variable that contains memory address of the instance of the current class.**So we can use **self to access** instance variables and instance methods.
The self keyword is used as **first parameter of a method or first parameter of constructor**
E.g.

def show(self)

Methods:

A method can access and modify the data attributes of the class and perform specific task on the instance.

A method in python is somewhat **similar to a function**, **except it is associated with object/classes**.There are 3 types of methods in python

A) Instance method B) Class method C)static method

A) Instance method :

Instance method is a very basic and **easy method that we use regularly** when we create classes inpython. If we want to **call the instance method**, we must **create an object** of that required class and we should **call the instance method by using that object.**

The **self** should be used as a first argument of instance method.Syntax for method

```
class ClassName:
      def method_name(self):
            # Method_body
```

Example of Instance method

```
class A:
    def add(self):
        self.a=10
        self.b=10
        self.c=self.a+self.b
        print("Addition is",self.c)


x=A()  #object creation
x.add()   #method call
```

Output: Addition is 20

In above program we defined add() instance method which contains **self** as first argument.Self used to access instance variables inside add() method.

We called add() method by using object **x** because instance method is called using object of thatclass.

B) Class Method :

- A class method is a method which **is bound to the class and not the object of the class.**
- They have the access to the state of the class as **it takes a cls (class) parameter** that points to the class **and not the object instance.**
- It can modify a class state that would apply across all the instances of the class. Forexample it can modify a class variable that will be applicable to all the instances.

The **@classmethod decorator, is a builtin function decorator** that is an expression thatgets evaluated after your function is defined.

syntax

```
class classname:
        @classmethod
        def fun(cls, arg1, arg2, ...):
                body
```

Class method example :

```
class Demo:
    v=100

    @classmethod
    def show(cls):
              return (cls.v)

  d=Demo()
  print("call using object",d.show()) #call to class method
  print("call using class name",Demo.show()) #call to class method
```

C)Static Method :

A static method is a method which is marked with a **@staticmethod decorator** to flag it as static.

It does not receive an implicit first argument (neither self nor cls).

static methods can neither modify the object state nor class state. They are primarily a wayto namespace our methods.

Call to static method is generated by its class name as,**Classname.methodname(arg1,arg2,….)**

Syntax to define static method

```
class Classname:
                @staticmethod
                 def fun(arg1, arg2, ...):
                        body
```

Example of static method

```
class Demo:
    @staticmethod
    def add():
        a=12
        b=5
        c=a+b
        print("addition is",c)

Demo.add() #call to static method by using class name
```

Output: addition is 17

Constructor :

A constructor is a special type of method (function) which is used to **initialize the instance members of the class.**
In C++ or Java, the constructor has the same name as its class, but it **treats constructor differently in Python.**
In Python the **_init_()** method is called the **constructor** and is **always called when anobject is created.**
Syntax of constructor definition:

```
def_____init____(self):
        # body of the constructor
```

There are two types of constructor in python
1) Default Constructor
2) Parameterised constructor

1) Default constructor :-

The default constructor is simple constructor which **doesn't accept any arguments**.
It's definition has **only one implicit argument which is self** (a reference to the instance being constructed).

```
class Demo:
    def_init_(self):          #default constructor
        self.a=10
        self.b=5
        self.c=self.a+self.b
        print("addition is",self.c)
 d=Demo() #call to default constructor
```

Output : addition is 15

2) Parameterised constructor :-

The **constructor with parameters** is known as parameterized constructor.
The parameterized constructor take **its first argument as self** and the **rest of the arguments are provided by the programmer**

```
class Demo:
    def __init__(self,x,y):      #parameterised constructor
        self.a=x
        self.b=y
        self.c=self.a+self.b
        print("addition is",self.c)

d=Demo(10,20) #call to default constructor
```

Output:

addition is 30

In above program we took **two explicit formal parameters** x and y in our constructor sothat the constructor called as **parameterised constructor.**
We passed two values 10 and 20 to x and y from actual parameters present in constructorcall.

Constructor with default Arguments :

The constructor which contains **formal parameters with default assigned values** is called asconstructor with default arguments.

It is nothing but a **parameterised constructor.**

In this concept there is **no need to pass values to this constructor while calling it.**

```
class Demo:
    #constructor with default arguments
    def __init__(self,x=100,y=30):
        self.a=x
        self.b=y
        self.c=self.a+self.b
        print("addition is",self.c)

d=Demo() #call to default constructor
```

Output: addition is
130

In above program we have a parameterised constructor which contains two formal parameters x and y which are assigned by default values 100 and 30 simultaneouslySo our constructor is called as constructor with default arguments .

In this case there is no need to pass values to formal parameters from actual parameters.So our constructor call is without actual parameters.

Passing members of one class To another Class by using object :

We know that all class data is in object of that class.

We can pass members(variables and methods) of one class to another class for that we canuse an object of that class.

```
class Emp:
    def __init__(self):
        self.id=101
        self.name="Nitin"

class Myclass:
    @staticmethod
    def show(e):
        print(e.id)
        print(e.name)

x=Emp()
Myclass.show(x)   #passing object to Myclass
```

Inner classes/ Nested class :

A class defined in another class is known as inner class or nested class.

An outer class can have one or more inner class but generally **inner classes are avoided**.Syntax to define inner class

```
class outerclassname:
        class Innerclass name:
                body of inner class
```

Syntax to create innerclass object

```
obj=outerclassobject.InnerclassName()
```

Example of inner class

```
class College:
    def __init__(self):
        print("outer class constructor")
    def show(self):
        print("Outer class method")
     #inner class
     class Student:
        def __init__(self):
            print("Inner class constructor")
        def display(self):
            print("Inner class method")
x=College()   #outer class object
y=x.Student() #inner class object
x.show()
y.display()
```

Output:

```
outer class constructor
Inner class constructor
Outer class method
Inner class method
```

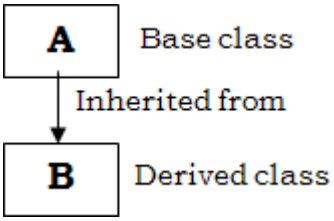
Inheritance :

We know that, inheritance is one of the most important object oriented concept

The main concept behind inheritance **is reusability of code (Software)** is possible by addingsome extra feature into existing software without modifying it.

Definition : The mechanism of **creating new class from existing class** is called asInheritance".

- The existing or old class is called as '**Base class' or 'Parent class' or 'Super class'** andnew class is called as '**Derived class' or 'Child class' or 'Sub class'**.
- While deriving the new class from exiting one then,the derived class will have someor all the features of existing class and also it can add its own features i.e sub class will acquires features of base class without modifying it.



In inheritance the base class is defined in normal way as we define our class, but thederived class is defined by following syntax

```
class DerivedClassname (BaseClassname):
                    Body of derived  class
```

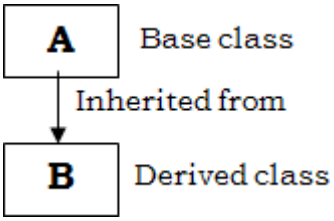
Types of Inheritance :

Types of Inheritance depends upon the number of child and parent classes involved. Thereare five types of inheritance in Python:

1) Single Inheritance 2)Multiple Inheritance

3)Multilevel Inheritance 4) Hierarchical Inheritance 5) Hybrid Inheritance

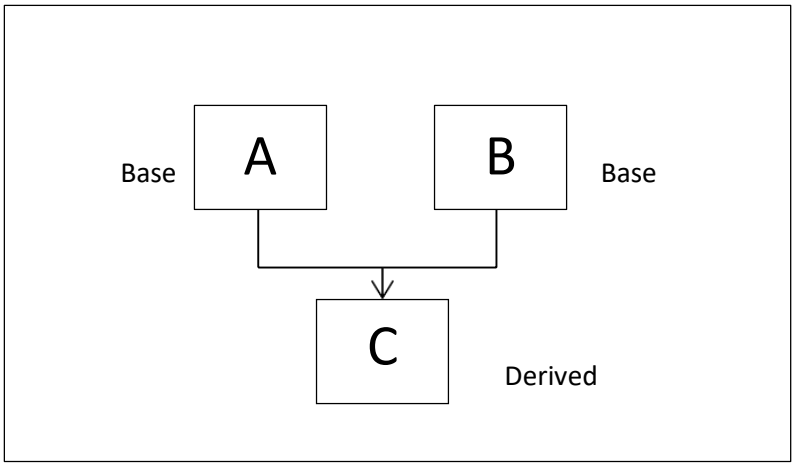
1) Single Inheritance : In case of Single inheritance there **is only one base class from whichwe derive only one new class.**




```
class A:
    p=30
    q=10

class B(A):
    def show(self):
        self.r=self.p - self.q
        print("Subtraction is",self.r)

x=B()   #always create derived class object
x.show()
```

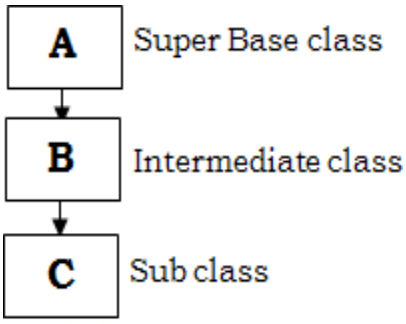
Output- Subtraction is 20

2) Multiple Inheritance : In case of Multiple inheritance there **are multiple(two or morethan two) base classes from which we derive only one new class.**




```
class A:
    p=30

class B:
    q=10

class C(A,B):
    def show(self):
        self.r=self.p - self.q
        print("Subtraction is",self.r)

x=C()      #always create derived class object
x.show()
```

3)Multilevel Inheritance :- In case of multilevel inheritance we can **derive new class from another derived classes, further from derived classes we again derive new class and so on.**Also, there are some levels of inheritances therefore it is called as “Multilevel inheritance”

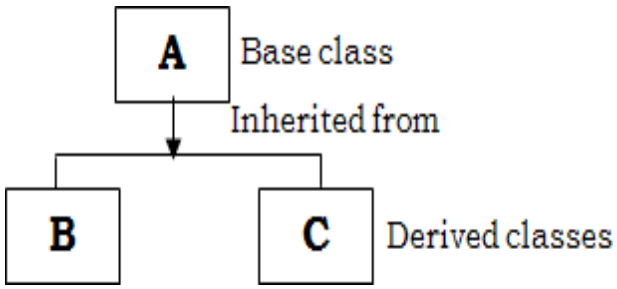


Example of multilevel Inheritance

```
class A:
    p=30

class B(A):
    q=10

class C(B):
    def show(self):
        self.r=self.p*self.q
        print("Multiplication is",self.r)

x=C()      #always create derived class object
x.show()
```

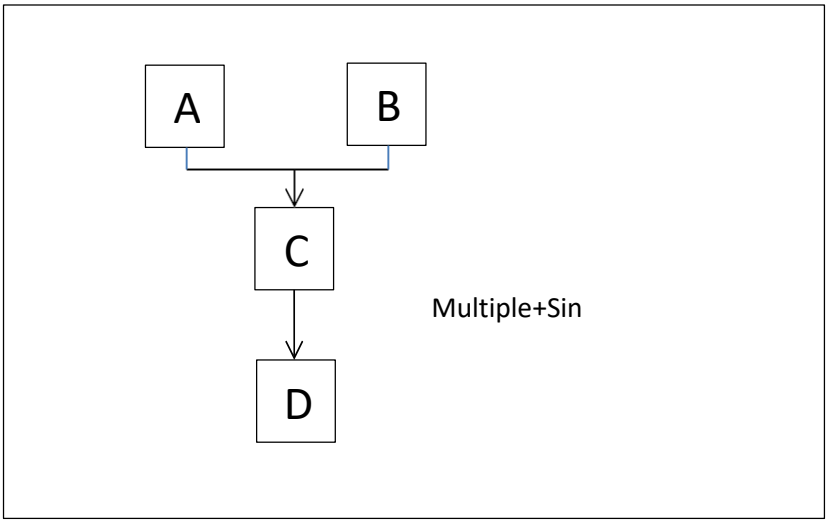
4) Hierarchical Inheritance : In case of hierarchical inheritance there is only **one base class from that class we can derive multiple new classes.**

The base class provides all its features to all derived classes which are common to all sub classes.The hierarchical inheritance comes into picture when certain feature of one level is shared by many other derived classes.



```
class A:
    p=30
    q=10

class B(A):
    def add(self):
        self.r=self.p+self.q
        print("Addition is",self.r)

class C(A):
  def sub(self):
        self.m=self.p-self.q
        print("Subtraction is",self.m)


x=B()   #always create derived class object
x.add()
y=C()   #always create derived class object
y.sub()
```

5) Hybrid Inheritance : Hybrid inheritance is the **combination of two or more forms ofinheritances.**



```
class A:
    p=30

class B:
    q=10

class C(A,B):
    r=5

class D(C):
    def show(self):
        self.m=self.p+self.q+self.r
        print("Addition is",self.m)

x=D()      #always create derived class object
x.show()
```

Constructor in Inheritance :

We can add constructor in stayed of methods in classes which are used in Inheritance.
If we **use constructor in base class** then there **is no need to call base class constructor Explicitly**.
When we **create derived class object then Base class constructor called automatically.**

Example of constructor in Inheritance :-

```
# Base class
class A:
     def___init__(self):   #base class constructor
         self.a=10
         self.b=5

# Derived class
class B(A):
     def show(self):
          print("Addition",(self.a+self.b))

x=B()     #call to base class constructor
x.show()
```

Output:

Addition 15

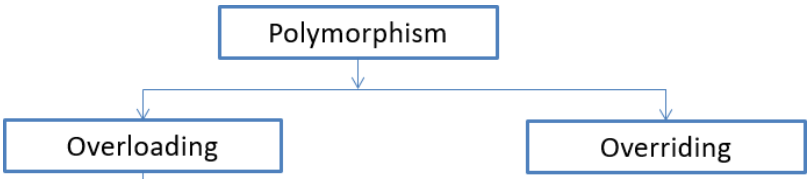
Super() method : The super() builtin returns a **proxy object** (temporary object of the superclass)that allows us to **access methods of the base class.**

In Python, super() has two major use cases:

Allows us to avoid using the base class name explicitly
Working with Multiple Inheritance

```
class college:
   def __init__(self,p):
       print(p)

class student(college):
  def __init__(self):
       print('all students are clever')
          super().__init__('BCA')     #call to base class constructor

d1 =student()#call to derived class constructor
```

Polymorphism: Polymorphism is one of the important feature of OOP.
Polymorphism is the **ability of an object to take on many forms** .The word polymorphism **means one name and many forms.**
In python we can implement polymorphism by Overloading and Overriding.



Method Overloading : Method Overloading is the ability of a function to behave in different ways **based on the parameters that are passed to the function**(function call) .

In Python, we can create a method that can be called in different ways. So, we can have a method that has zero, one or more number of parameters. Depending on the method definition, we can call it with **zero, one or more arguments**.

Program of method overloading

```
class Moverload:
    def show(self,a=None,b=None):
          print(a,b)

x=Moverload()
x.show()   #1st call
x.show(10) #2nd call
x.show(5,3)#3rd call
```

Output : None
None10
None
5 3

Constructor Overloading : constructor Overloading is the ability of a constructor(parameterised constructor) to behave in different ways based on the parameters that are passed to the constructor (constructor call) .

```
class Coverload:
    def__init__(self,a=None,b=None):
          print(a,b)

x=Coverload()    #1st  call
y=Coverload(10)  #2nd call
z=Coverload(5,3) #3rd call
```

Output:
None None
10 None
5 3

Operator Overloading : Python operators work for built-in classes. But the same operator behaves differently with different types.
For example, the + operator will perform arithmetic addition on two numbers, merge two lists, or concatenate two strings.
Ex. 2+3=5
'he'+'llo' -> 'Hello'
This feature in Python that allows the same operator to have different meaning according to the context is called operator overloading.
In Operator overloading special functions are user which starts from_____(double underscore) we can overload other operators as well. The special function that we need to implement is tabulated below.

Operator	Internally
+	__add__(self, other)
–	__sub__(self, other)
*	__mul__(self, other)
/	__truediv__(self, other)
//	__floordiv__(self, other)
%	__mod__(self, other)
**	__pow__(self, other)
>>	__rshift__(self, other)
<<	__lshift__(self, other)
&	__and__(self, other)
|	__or__(self, other)
^	__xor__(self, other)

Lets see_____add__() function example
When + operator used in between then add function called by int and execute as,

Output : 5

```
print(int.__add__(3,2))
```

When + operator used in between **String** operands then add function called by **str** and execute as,

```
print(str.__add__('he','llo'))
```

Output: Hello

Method Overriding:

Method overriding is a concept of object oriented programming that allows us to **change the implementation of a function in the child class that is defined in the parent class**. It is the ability of a child class to change the implementation of any method which is already provided by one of its parent class(ancestors).

In short we can say that in **method overriding multiple methods in Inheritance having same name and same signature(parameters)**

Program for Method Overriding

```
class A:
          def show(self):
               print("In Base class A")
class B(A):
          def show(self):
               print("In derived class B")
y=B()
y.show()
```

Output :
In derived class B

In method overriding base and derived class methods have same name and same signature. The base class method get overridden by derived class method.

Constructor Overriding (Overriding base class Constructor): When base and derived class **constructors are** of **same type(same name and same signature)** then that is called as constructor overriding.

In constructor overriding the derived class constructor gets executed.

```
class A:
          def ___init___(self):
               print("In Base class A")
class B(A):
          def___init__(self):
               print("In derived class B")

y=B() #call to derived class constructor
```

Output:

In derived class B

In constructor overriding The base class constructor get overridden by derived class constructor.

Abstract Class and Abstract Method:

Abstract Class: A class which **contains one or more abstract methods** is called an abstract class. An abstract class can be considered as a blueprint for other classes. It allows us to create a set of methods that must be created within any child classes built from the abstract class.

- An abstract class doesn't contain all of the method implementations.
- An abstract class cannot be instantiated.
- A derived subclass must implement the abstract methods to create a concrete class that fits the interface defined by the abstract class. Therefore it cannot be instantiated unless all of its abstract methods are overridden.

- To implement abstract class **ABC class** should be imported from **abc module**.
- Our Abstract class should be derived by ABC class as,
- An Abstract class can contain Abstract and concrete methods.

class classname(ABC):

Abstract Method: An abstract method is **a method that is declared**, **but not implemented in the code.**

The Abstract method should **compulsory redefined in derived classes.**

Before definition of Abstract method the **@abstractmethod** decorator should be declared.

Example of abstract class and Abstract method

```
from abc import ABC,abstractmethod

#Abstract class
class First(ABC):
    @abstractmethod
    def show(self):
          pass

class Second(First):
    def show(self):
        self.a=21
        self.b=33
        self.c=self.a+self.b
        print("Addition",self.c)


d=Second()
d.show()
```

Interface :

A class which **contains only abstract methods** that class is called as Interface.

In python we **cannot create an interface** like created in java.

In other words we say that an abstract class which contains **only abstract** methods that **abstractclass work as an Interface in python.**

```
from abc import ABC,abstractmethod

# Interface
class First(ABC):
    @abstractmethod
    def show(self):
         pass

class Second(First):
    def show(self):
        print("In derived class")

d=Second()
d.show()
```

Class method vs static method :

Class Method	Static Method
The class method takes cls (class)as first argument.	The static method does not take any specific parameter.
Class method can access and modify the class state.	Static Method cannot access or modify theclass state.
The class method takes the class as parameter to know about the stateof that class.	Static methods do not know about class state. These methods are used to do some utility tasks by taking some parameters.
@classmethod decorator is usedhere.	@staticmethod decorator is used here.

Namespaces :-

A namespace is a **collection of names and the details of the objects referenced by the names**. It is a python dictionary , which maps object names to objects. The keys of the dictionary correspond tothe names and the values correspond to the objects in python.

A Python namespace is an efficient system where each and every object in Python has a unique name.

- **Built-in namespace –** A Python namespace is created when we start using the interpreter, and it exists as long as to use. When the Python interpreter runs without any user-defined modules, methods, etc., built-in functions like print() and id() are always present and available from any part of a program. These functions exist in the built-in namespace.
- **Global namespace –** A global namespace is created when you create a module.
- **Local namespace –** This namespace is created when you create new localfunctions.

A namespace's lifetime depends on the scope of a variable. If the variable's scope ends, then pythonnamespace scope ends.

Behavior of local variables	**Using global variables within functions**	**built-in namespace**

def fun():
  a = 4
  print(a)
fun()
	a = 10
	def fun():
	  global a
	print(a)
	  b=4
	  print(b)
	fun()
	print(a)
		a = 0
		def fun():
		  int = 3
		  input = 4
		  print(int)
		  print(input)
		fun()
		print(int("123"))
		print(a)

Unit -2

Exception Handling and Threading , Python File Operation

Threading :-

Thread:- A small part of our program code are known as thread.

It is lightweight as compared to process

Uses of Threading :-

some common uses of threading in Python as follows.

1)Concurrent Execution: Threading enables you to execute multiple tasks concurrently. For example, you can use threads to handle network requests, perform time-consuming I/O operations, or carry out computations in parallel, making your program more efficient.

2)GUI Applications: Threading is often used in graphical user interface (GUI) applications to keep the user interface responsive while performing other tasks in the background. By running time-consuming operations in separate threads, you can prevent the GUI from freezing or becoming unresponsive.

3)Network Operations: Threading is useful for handling network-related tasks such as making HTTP requests, downloading files, or running a server. By using threads, you can initiate multiple network operations simultaneously and process their results concurrently.

4)Parallel Processing: Threading allows you to perform parallel processing, especially on multi-core or multi-processor systems. You can split a large computation-intensive task into smaller parts and execute them concurrently in separate threads to take advantage of the available computing resources.

5)Asynchronous Operations: Threading is often used in conjunction with asynchronous programming frameworks such as concurrent futures to perform non-blocking I/O operations. This allows you to initiate multiple I/O operations concurrently and continue executing other tasks while waiting for their completion.

6)Producer-Consumer Situations: Threading is commonly used in situations where multiple threads need to communicate or exchange data. For instance, in producer-consumer patterns, one thread produces data while another thread consumes it. Threading provides synchronization mechanisms such as locks, semaphores, or queues to ensure proper communication and coordination between threads.

7)Background Tasks: Threading can be used to perform background tasks while the main program continues its execution. This is useful for scenarios where you need to perform periodic tasks, maintenance operations, or data processing tasks in the background without blocking the main execution flow.

steps for creating simple thread in python :-

1)Import the required module:

Syntax :- import threading

2)**Define a function** that represents the task you want to perform in a separate thread. This function will be executed concurrently:

Syntax :- def your_task(): # Code for the task you want to perform

3)**Create a thread object**, passing the function as the target and any arguments as needed:

Syntax :- thread = threading.Thread(target=your_task)

4) **Start the thread** by calling the start() method:

Syntax :- thread.start()

5)**If needed**, you can **join the thread** to wait for its completion using the join() method:

Syntax thread.join()

Example:-1)

```
import threading
def task():
   print("Execute task")
thread=threading.Thread(target=task)
thread.start()
thread.join()
print("main thread continues executing")
```

In this example, the your_task() function represents the task you want to perform in a separate thread. The threading.Thread() class is used to create a thread object, passing your_task as the target function. The start() method starts the thread, and the join() method is called to wait for the thread's completion. Finally, the main thread continues executing after the join.

Ex.2) Creating Thread by using **thread class** with implement **run method** only

```
import time
import threading
class A(threading.Thread):
      def run(self):
            print("thread1")
class B(threading.Thread):
       def run(self):
            print("thread2")
obj1=A()
obj2=B()
obj1.start()
```

```
time.sleep(2)
obj2.start()
```

Methods of thread class :-

1)start() :-start a thread by calling start method

Ex.

```
 import threading
 def show(a,b):
       c=a+b
       print(c)
t=threading.Thread(target=show,args=(10,20))
t.start()
```

2)Run() :- Entry point for a thread.

Ex.

```
import threading
def show(a):
    print("value of a in thread=",a)
t=threading.Thread(target=show,args=(10,))
t.run()
```

3) Sleep() :-suspend thread for a specified time.

Ex.1)

```
import time
import threading
def show(a):
    print('value of an in thread1=',a)
def display(a) :
    print('value of an in thread2=',a)
t1=threading.Thread(target=show,args=(10,))
t2=threading.Thread(target=display,args=(20,))
t1.run()
time.sleep(5)
t2.run()
```

Ex. 2)

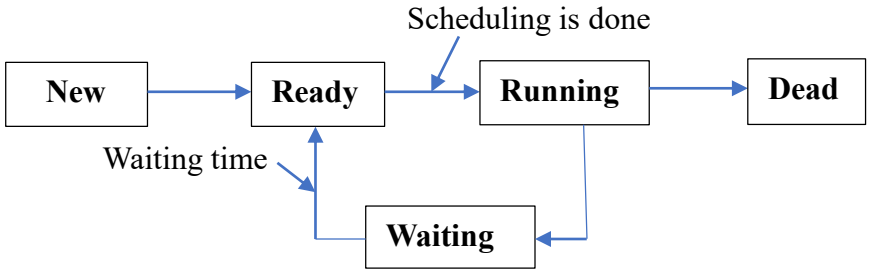
```
import threading
def show(a,b):
    print("value of a in thread=",a)
    print("value of b in thread=",b)
x=threading.Thread(target=show,args=(10,20))
y=threading.Thread(target=show,args=(30,40))
z=threading.Thread(target=show,args=(50,60))
x.run()
y.run()
z.run()
```

Thread life cycle:-

From **thread creation to thread termination** the thread goes different states are known as thread life cycle.

The life cycle has following step :-

1) New
2) Ready
3) Running
4)Waiting
5) Dead



1) New :- When we create the thread that is we create the object of that class that time it is in new state.

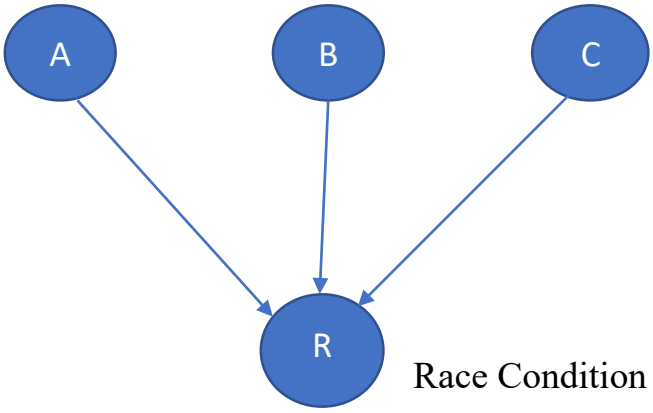
2) Ready :- After creation of thread we call the start method on thread then it is in ready state.

3) Running :-After scheduling is done it is sent to running state and in running state the actual execution is started.

4)Waiting :- During the execution of thread if any interruption is occurred i.e. if we call the sleep or join method on thread then it is waiting state.

5) Dead :-After successfully execution of thread is terminated i.e. the natural death of thread it is in dead state.

Thread Syncronization :-



Race Condition Occurs

In this mechanism the more than one thread has allow to access the common data or common resources simultaneously, at that time there will be chance of race condition. The race condition we need to implement thread synchronization.

Synchronization :-it is a mechanism which allow only one thread thread at a time access the command data or common resource i.e. execute the critical section.

In python for the thread synchronization they have provided lock class, the lock class provides following two methods.

1)**Acquire** -trade must requires the lock by using acquire method before executing it's critical section i.e. accessing the common data.

2)**Release**-the thread must need to release the lock by using release method after execution of its a critical section.

Ex.

1)

```
import threading
a=100
b=50
lock=threading. Lock()
def show():
    lock.acquire()
    c=a+b
```

```
    print("Addition of thread1=",c)
    lock.release()
def display():
    lock.acquire()
    c=a-b
    print("subtraction of thread2=",b)
    lock.release()
t1=threading.Thread(target=show)
t2=threading.Thread(target=display)
t1.start()
t2.start()
```

Exception Handling :

Before understand exception handling we have to understand about exception.

Exception: An exception is a **condition which occurs when program encounters an abnormal error.**When these exceptions occur, the Python interpreter stops the current process and passes it to the calling process until it is handled.
If these Exceptions not handled, the program will crash or stops immediately.Exceptions are of two types,

a) Built-in Exceptions b) User defined Exception

a) Built-in Exceptions: The exceptions which are already defined by python are called as built-inexceptions.
Following table shows built-in exceptions

Sr.No.	Exception Name & Description
1	Exception
	Base class for all exceptions
2	StopIteration
	Raised when the next() method of an iterator does not point to any object.

3	SystemExit
	Raised by the sys.exit() function.
4	StandardError
	Base class for all built-in exceptions except StopIteration and SystemExit.
5	ArithmeticError
	Base class for all errors that occur for numeric calculation.
6	OverflowError
	Raised when a calculation exceeds maximum limit for a numeric type.
7	FloatingPointError
	Raised when a floating point calculation fails.
8	ZeroDivisionError
	Raised when division or modulo by zero takes place for all numeric types.
9	AssertionError
	Raised in case of failure of the Assert statement.
10	AttributeError
	Raised in case of failure of attribute reference or assignment.
11	EOFError
	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
12	ImportError
	Raised when an import statement fails.
13	KeyboardInterrupt

	Raised when the user interrupts program execution, usually by pressing Ctrl+c.
14	LookupError
	Base class for all lookup errors.
15	IndexError
	Raised when an index is not found in a sequence.
16	KeyError
	Raised when the specified key is not found in the dictionary.
17	NameError
	Raised when an identifier is not found in the local or global namespace.
18	UnboundLocalError
	Raised when trying to access a local variable in a function or method but no value has been assigned to it.
19	EnvironmentError
	Base class for all exceptions that occur outside the Python environment.
20	IOError
	Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
21	IOError
	Raised for operating system-related errors.
22	SyntaxError
	Raised when there is an error in Python syntax.
23	IndentationError
	Raised when indentation is not specified properly.
24	SystemError

	Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
25	SystemExit
	Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
26	TypeError
	Raised when an operation or function is attempted that is invalid for the specified data type.
27	ValueError
	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
28	RuntimeError
	Raised when a generated error does not fall into any category.
29	NotImplementedError
	Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

b) User defined Exception : The exception defined by user is called as user defined exception.

Syntax to create user defined exception

```
raise exception
```

where raise- is a keyword which

raises exceptionexception-

any user defined exception

Exception Handling : Exception handling is a **technique to handle exception which maintains normal flow of a program .**

For exception handling following blocks are used

1) try block/clause :

In Python, exceptions can be handled using a try statement.

The critical operation which can raise an exception is placed inside the try block.The try block diverts flow of program towards next block called **except block.**

Note: In single block **only one** try block will be used.

Syntax of try block

```
try:
--------

        ---Logic---
```

2) except block/clause:

The except block let us handle the exception by which our program don't get terminated .For one try block we can use single or multiple except blocks.

Syntax of except block

```
except exceptiontype:
        body of except block
```

In above syntax **exceptiontype** is optional

3) finally block/clause :

The finally block lets us execute code, regardless of the result of the try- and except blocks.Finally block is an optional block.

Syntax of finally block

```
finally:
        body of finally block
```

A simple program of exception handling

```
import sys
try:
    a=10
    b=0
    c=a/b
    print("Division is",c)
except:
    print("Can not divide by zero")
finally:
    print("finally block executed")
```

Output:

Can not divide by zero

finally block executed

In above program the code is enclosed in **try** block which raise built-in exception called ZeroDivisionError because we cannot divide by zero(a/b where a=10 , b=0).The raised exception is diverted towards **except** block which will execute.After that **finally** block gets executed.

Another example of exception handling (IndexError)

```
import sys
try:
    a=[10,20,30]
    print("Division is",a[5])
except :
    print("Invalid index number")
```

Output:

```
Invalid index number
```

In above program our code kept in try block from which we are trying to access index number a[5]Which is not available.
So the built-in exception called IndexError is raised and passed towards except block where theexception is handled

User defined Exception and User defined Exception Handling / Custom Exception Handling We already seen about user defined exception and how to raise an user defined exception fromtry block.

Now we are going to handle user defined exception in

```
class myexception(Exception):
    pass
def show(age):
    try:
        if age<18:
            raise myexception
        else:
            print("yes")
    except:
            print("not")
show(10)
```

Python File Operation

File: File is a place on secondary storage where data can be stored permanently.

Since Random Access Memory (RAM) is volatile (which loses its data when the computer is turnedoff), we use files for future use of the data by permanently storing them.
In python Files are of two types ,

1) Text Files : Text file stores data in the form of Characters.

2) Binary files : Binary files stores data in the form of bytes i.e. group of 8 bits each

When we want to read from or write to a file(Both files Text and Binary), we need to open it first.When we are done, it needs to be closed .

Hence, in Python, a file operation takes place in the following order:

1) Open a file
2) Read or write (perform operation)
3) Close the file

1) Opening Files in Python :-

Python has a built-in open() function to open a file. This function returns a file object it is used toread, write or modify the file .

The open() function takes two parameters; *filename*, and file opening *mode*.

```
f = open("test.txt")    # open file in current directory
f = open("C:/Python38/README.txt")  # specifying full path
f = open("test.txt",'w')  # write in text mode
```

We can specify the mode while opening a file. In mode, we specify whether we want to read **r**,write **w** or append **a** to the file. We can also specify if we want to open the file in text mode orbinary mode.

The default mode is reading in text mode. In this mode, we get strings when reading from the file.On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like images or executable files.

File Opening Modes :-

Mode	Description
r	Opens a file for reading. (default)
w	Opens a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
x	Opens a file for exclusive creation. If the file already exists, the operation fails.
a	Opens a file for appending at the end of the file without truncating it. Creates a new file if it does not exist.
t	Opens in text mode. (default)
b	Opens in binary mode.
+	Opens a file for updating (reading and writing)

2) **Read or write (perform operation) :** We will reading or writing text and binary files but beforethat we are going to understand closing operation

3) **Close the file :** After reading or writing a file we need to close the file as ,

```
f.close()
```

Python File Methods :

There are various methods available with the file object. Here is the complete list of methods intext mode with a brief description:

Method	Description
close()	Closes an opened file. It has no effect if the fileis already closed.
detach()	Separates the underlying binary buffer fromthe TextIOBase and returns it.
fileno()	Returns an integer number (file descriptor) ofthe file.
flush()	Flushes the write buffer of the file stream.
isatty()	Returns True if the file stream is interactive.
read(n)	Reads at most n characters from the file. Readstill end of file if it is negative or None.
readable()	Returns True if the file stream can be readfrom.
readline(n=-1)	Reads and returns one line from the file. Readsin at most n bytes if specified.
readlines(n=-1)	Reads and returns a list of lines from the file. Reads in at most n bytes/characters if specified.
seek(offset,from=SEEK_SET)	Changes the file position to offset bytes, inreference to from (start, current, end).
seekable()	Returns True if the file stream supports randomaccess.
tell()	Returns the current file location.
truncate(size=None)	Resizes the file stream to size bytes. If size isnot specified, resizes to current location.
writable()	Returns True if the file stream can be writtento.
write(s)	Writes the string s to the file and returns thenumber of characters written.
writelines(lines)	Writes a list of lines to the file.

Now let's see Writing and reading operation on **Text** file first,

Text File Writing :

In order to write into a file in Python, we need to open it in write **w**, append **a** or exclusivecreation **x** mode.

We need to be careful with the w mode, as it will overwrite into the file if it already exists. Due tothis, all the previous data are erased.

Program to write data in Text file(w)

```
f=open('test.txt','w')
f.write("Hi")
f.close()
```

-In above program the file is opened by open() method. The open method contains two argumentsfirst one is 'test.txt'(it us a file name which is stored in its default location i.e.python folder present in C drive) and second argument is 'w' file opening mode as write mode.

-After that we write a string **Hi** by using write() method.And after that we closed our file by close() method.

Text File Reading(r) : To read a file in Python, we must open the file in reading **r** mode.There are various methods available for this purpose.

We can use the read(size) method to read in the size number of data.

If the size parameter is not specified, it reads and returns up to the end of the file.

```
f = open("Test.txt", "r")
m=f.read()
f.close()
print(m)
```

Reading Multiline Text File Line by Line:

we can read multiline text file line by Line forthat readlines() method is used.

```
f = open("Test.txt", "r")
m=f.readlines()
for x in m:
        print(x)
f.close()
```

Text File Appending(a/a+)

In order to append a new line to the existing file, open the file in append mode, by

usingeither 'a' or 'a+' as the access mode. The definition of these access modes are as follows:

-**Append Only ('a'):** Open the file for writing. The file is created if it does not exist. The handle ispositioned at the end of the file. The data being written will be inserted at the end, after the existing data.

-**Append and Read ('a+'):** Open the file for reading and writing. The file is created if it does notexist. The handle is positioned at the end of the file. The data being written will be inserted atthe end, after the existing data.

When the file is opened in append mode, the handle is positioned at the end of the file. The databeing written will be inserted at the end, after the existing data. Let's see the below example to clarify the difference between write mode and append mode.

```
f=open('test.txt','a')
f.write("How are You?")
f.close()
```

Output :Hi How are you

In above program we opened our previous file test.txt (which contain a string 'Hi') in appendmode.
After that the string How are you is append on file using write() function.The write function appends How are you at the end of string Hi .

Binary File writing:

Before writing binary file first we have to open a file by same way but the file opening modeshould be wb .
After that we can write a binary file by write() or writelines() method.
Program to write Binary file

```
f=open("binfile.bin","wb")
num=[5, 10, 15, 20, 25]
arr=bytearray(num)  #converting to bytearray type
f.write(arr)
f.close()
```

Binary File reading:

Before reading binary file first we have to open a file with file opening mode **rb** .After that we can read a binary file by read() or readlines() method.

```
f=open("binfile.bin","rb")
p=f.read()                          #reading binary file
num=list(p)              # conversion of binary data to list type
print (num)
f.close()
```

seek() and tell() methods:

seek() method :

In Python, seek() function/method is used to **change the position of the File Handle** to a given specific position. File handle is like a cursor, which defines from where the data has to be read orwritten in the file.

Syntax:

```
f.seek(offset, from_what)
```

where f is file pointer

Parameters:

Offset: Number of postions to move forward from_what: It defines point of reference.

The reference point is selected by the from_what argument.

It accepts three values:

0: sets the reference point at the beginning of the file

1: sets the reference point at the current file position

2: sets the reference point at the end of the file

By default from_what argument is set to 0.

tell() method :

The tell() method returns the file handlers current position in a file stream.

Syntax

```
f.tell()
```

where f is file pointer

Example of seek() and tell() method

Note: test.txt file is already present in secondary storage in that file Sangola(data) is present.

```
f=open("F:/test.txt",'r')
f.seek(3)
v=f.tell()
print('file pointer is on',v)
str=f.read()
print(str)
f.close()
```

Output: file pointer is on 3

gola

Program of file with exception handling

```
try:
   f=open('test.txt','r')
   s=f.read()
   f.close()
   print(s)

except:
   print('can not perform operation on file')
```

Python Directory and Files Management :

If there are a large number of files to handle in our Python program, we can arrange our code within different directories to make things more manageable.

A directory or folder is a collection of files and subdirectories. Python has the os module thatprovides us with many useful methods to work with directories (and files as well).

Get Current Directory :-

We can get the present working directory using the getcwd() method of the os module. This method returns the current working directory in the form of a string. We can also usethe getcwdb() method to get it as bytes object.

```
import os

os.getcwd()
os.getcwdb()
```

Making a New Directory :-

We can make a new directory using the mkdir() method.

This method takes in the path of the new directory. If the full path is not specified, the newdirectory is created in the current working directory.

```
>>> os.mkdir('test')

>>> os.listdir()
['test']
```

Renaming a Directory or a File :-

The rename() method can rename a directory or a file.

For renaming any directory or file, the rename() method takes in two basic arguments: the oldname as the first argument and the new name as the second argument.

```
import os
os.rename('test','sangola')
```

Removing Directory or File :-

A file can be removed (deleted) using the remove() method. Similarly, the rmdir() method removes an empty directory.

Ex. 1) import os
os.rmdir('test')

Ex. 1) import os
os.remove('E:/abc/b.txt')

Note: The rmdir() method can only remove empty directories.

In order **to remove a non-empty directory**, we can use the **rmtree()** method inside the **shutil** module.

Ex.1) import os
os.listdir()
['a']
os.rmdir('E:/abc')

Ex.2) import os
import shutil
shutil.rmtree('E:/abc')
os.listdir()

Unit-3 Python GUI And Database Connectivity

GUI (Graphical User Interface) :-

A graphical user interface (GUI) is a digital interface in which a user interacts with graphical components such as icons, buttons, and menus. In a GUI, the visuals displayed in the user interface convey information relevant to the user, as well as actions that they can take.

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, **tkinter** is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter is the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task.

To create a tkinter app:

1. Importing the module – tkinter
2. Create the main window (container)
3. Add any number of widgets to the main window
4. Apply the event Trigger on the widgets.(**Widget is an element of Graphical User Interface (GUI) that displays/illustrates information or gives a way for the user to interact with the OS**. In Tkinter , Widgets are objects , instances of classes that represent buttons, frames, and so on. Each separate widget is a Python object.)

Importing tkinter is same as importing any other module in the Python code. Note that the name of the module in Python 2.x is 'Tkinter' and in Python 3.x it is 'tkinter'.

```
import tkinter
```

Tkinter is a Python library that can be used to construct basic graphical user interface (GUI) applications

There are **two main methods** used which the user needs to remember while creating the Python application with GUI.

1. **Tk(screenName=None, baseName=None, className='Tk', useTk=1):**
   To create a main window, tkinter offers a method 'Tk(screenName=None, baseName=None, className='Tk', useTk=1)'. To change the name of the window, you can **change the className** to the desired one.
   The basic code used to create the main window of the application is:

   ```
   m=tkinter.Tk()
   ```

   where ,**m** is the name of the **main window object**

   - **screenName**: This parameter is used to specify the display name.
   - **baseName:** This parameter can be used to set the base name of the application.
   - **className:** We can change the name of the window by setting this parameter to the desired name.
   - **useTk:** This parameter indicates whether to use Tk or not.

2)mainloop(): There is a method known by the name mainloop() is used when your application is ready to **run. mainloop()** is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
m.mainloop()
```

EX.

```
import tkinter
m = tkinter.Tk(screenName=None,  baseName=None,  className='mywindow',  useTk=1)
'''
widgets are added here
'''
m.mainloop()
```

tkinter also offers access to the geometric configuration of the widgets which can organize the widgets in the parent windows.

Geometry Managers :-

- **Geometry Managers** method used **to set the dimensions of the Tkinter window** and is used to set the position of the main window on the user's desktop.
- Geometry managers are used **for adding child widgets to parent widgets.**

  Ex.

```
import tkinter as tk
root = tk.Tk()
root.geometry("400x500")  # Set window size to 800x600 pixels
root.minsize(400, 300)    # Set minimum window size to 400x300 pixels
root.mainloop()
```

There are mainly **three geometry manager classes**

1. **place() method:-** It organizes the widgets **by placing** them on **specific positions directed by the programmer.**
2. **pack() method:-** It organizes the widgets in **blocks before placing in the parent widget.**
3. **grid() method:-** It organizes the widgets in **grid (table-like structure) before placing in the parent widget.**

1)Python - Tkinter place() Method :-

This geometry manager organizes widgets **by placing them in a specific position** in the parent widget.

Syntax

```
widget.place( place_options ),
widget.place(relX = 0.5, relY = 0.5, anchor = CENTER)
```

Here is the list of possible options −

- **anchor** − The exact spot of widget other options refer to: may be N, E, S, W, NE, NW, SE, or SW, compass directions indicating the corners and sides of widget; default is NW (the upper left corner of widget)
- **bordermode** − INSIDE (the default) to indicate that other options refer to the parent's inside (ignoring the parent's border); OUTSIDE otherwise.
- **height, width** − Height and width in pixels.

- **relheight, relwidth** − Height and width as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.
- **relx, rely** − Horizontal and vertical offset as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.
- **x, y** − Horizontal and vertical offset in pixels.

Example 1)

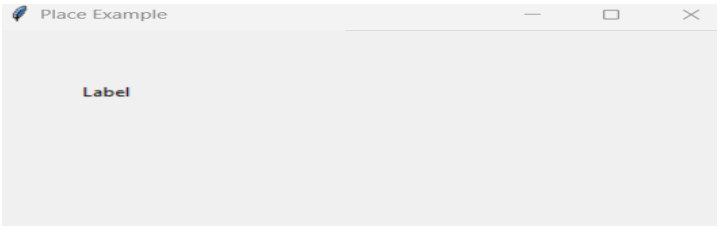
Try the following example by moving cursor on different buttons –

```
from tkinter import *
from tkinter import messagebox
import tkinter
top = tkinter.Tk()

def helloCallBack():
   messagebox.showinfo( "Hello Python", "Hello World")

B = tkinter.Button(top, text ="Hello", command = helloCallBack)

B.pack()
B.place(bordermode=OUTSIDE, height=100, width=100)
top.mainloop()
```

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


```
Ex. 2)
import tkinter as tk
root = tk.Tk()
root.title("Place Example")
# Create a label
label = tk.Label(root, text="Label")
# Place the label at specific coordinates
label.place(x=50, y=50)

root.mainloop()
```

Output



2)Python - Tkinter pack() Method :-

This geometry manager organizes widgets in blocks before placing them in the parent widget.

Syntax

widget.pack(pack_options)

Here is the list of possible options −

- **expand** − When set to true, widget expands to fill any space not otherwise used in widget's parent.
- **fill** − Determines whether widget fills any extra space allocated to it by the packer, or keeps its own minimal dimensions: NONE (default), X (fill only horizontally), Y (fill only vertically), or BOTH (fill both horizontally and vertically).
- **side** − Determines which side of the parent widget packs against: TOP (default), BOTTOM, LEFT, or RIGHT.

Example1.

Try the following example by moving cursor on different buttons −

```
from tkinter import *

root = Tk()
frame = Frame(root)
frame.pack()

bottomframe = Frame(root)
bottomframe.pack( side = BOTTOM )

redbutton = Button(frame, text="Red", fg="red")
redbutton.pack( side = LEFT)

greenbutton = Button(frame, text="green", fg="green")
greenbutton.pack( side = LEFT )

bluebutton = Button(frame, text="Blue", fg="blue")
bluebutton.pack( side = LEFT )

blackbutton = Button(bottomframe, text="Black", fg="black")
blackbutton.pack( side = BOTTOM)

root.mainloop()
```

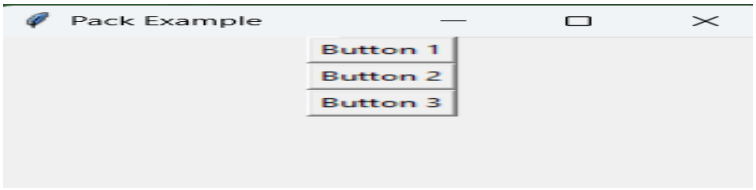
When the above code is executed, it produces the following result −

Ex.2)

```
import tkinter as tk
root = tk.Tk()
root.title("Pack Example")
# Create three buttons
button1 = tk.Button(root, text="Button 1" )
button2 = tk.Button(root, text="Button 2")
button3 = tk.Button(root, text="Button 3")
```

```
# Pack the buttons vertically
button1.pack()
button2.pack()
button3.pack()
root.mainloop()
```

Output :-



3)Python - Tkinter grid() Method :-

This geometry manager organizes widgets in a table-like structure in the parent widget.

Syntax

widget.grid(grid_options)

Here is the list of possible options −

- **column** − The column to put widget in; default 0 (leftmost column).
- **columnspan** − How many columns widgetoccupies; default 1.
- **ipadx, ipady** − How many pixels to pad widget, horizontally and vertically, inside widget's borders.
- **padx, pady** − How many pixels to pad widget, horizontally and vertically, outside v's borders.
- **row** − The row to put widget in; default the first row that is still empty.
- **rowspan** − How many rowswidget occupies; default 1.
- **sticky** − What to do if the cell is larger than widget. By default, with sticky='', widget is centered in its cell. sticky may be the string concatenation of zero or more of N, E, S, W, NE, NW, SE, and SW, compass directions indicating the sides and corners of the cell to which widget sticks.

Example

Try the following example by moving cursor on different buttons −

```
import tkinter
root = tkinter.Tk()
for r in range(3):
   for c in range(4):
      tkinter.Label(root, text='R%s/C%s'%(r,c),
         borderwidth=10).grid(row=r,column=c)
root.mainloop( )
```

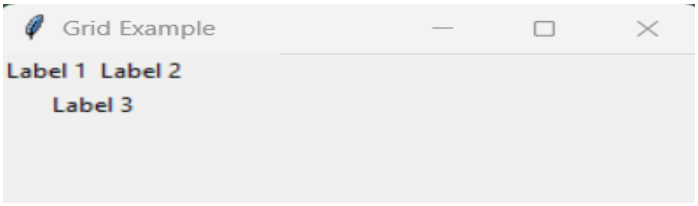
This would produce the following result displaying 12 labels arrayed in a 3 × 4 grid −

Ex.2)

```
import tkinter as tk
root = tk.Tk()
root.title("Grid Example")
# Create three labels
```

```
label1 = tk.Label(root, text="Label 1")
label2 = tk.Label(root, text="Label 2")
label3 = tk.Label(root, text="Label 3")

# Grid the labels in a 2x2 grid
label1.grid(row=0, column=0)
label2.grid(row=0, column=1)
label3.grid(row=1, column=0, columnspan=2)
root.mainloop()
```

Output :-



Widgets :-

widget is a graphical control element, like a button, label, or text entry field, that allows users to interact with a program's interface.

There are a number of widgets which you can put in your tkinter application. Some of the major widgets are explained below:

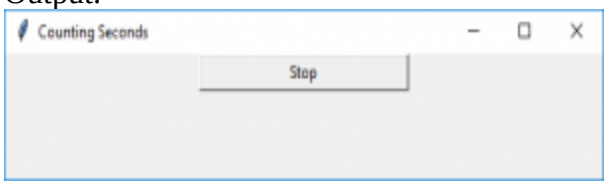
1. **Button**:To add a button in your application, this widget is used.
   The general syntax is:
   w=Button(master, option=value)

   master is the parameter used to represent the parent window.
   There are number of options which are used to change the format of the Buttons. Number of options can be passed as parameters separated by commas. Some of them are listed below.

   - **activebackground**: to set the background color when button is under the cursor.
   - **activeforeground**: to set the foreground color when button is under the cursor.
   - **bg**: to set he normal background color.
   - **command**: to call a function.
   - **font**: to set the font on the button label.
   - **image**: to set the image on the button.
   - **width**: to set the width of the button.
   - **height**: to set the height of the button.

```
import tkinter as tk
r = tk.Tk()
r.title('Counting Seconds')
button = tk.Button(r, text='Stop', width=25, command=r.destroy)
button.pack()
r.mainloop()
```

Output:



2. **Canvas:** It is used to **draw pictures and other complex layout like graphics, text and widgets.**
The general syntax is:
w = Canvas(master, option=value)

   master is the parameter used to represent the parent window.

   There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

   - **bd**: to set the border width in pixels.
   - **bg**: to set the normal background color.
   - **cursor**: to set the cursor used in the canvas.
   - **highlightcolor**: to set the color shown in the focus highlight.
   - **width**: to set the width of the widget.
   - **height**: to set the height of the widget.

```
from tkinter import *
master = Tk()
w = Canvas(master, width=500, height=300)
w.pack()
w.create_line(108, 120, 320, 40, fill="green")
mainloop()
```

   Output:

   ***creating different shapes :-**

   First, we need to **initialize** the Tkinter and canvas class. Tkinter Canvas class contains every method for creating different shapes.

```
root = Tk()
canvas = Canvas()
root.mainloop()
```

   After initialization of the Tkinter and canvas class, we start with the drawing of different shapes

   ***Line :-**

```
from tkinter import *

root = Tk()

C = Canvas(root, bg="yellow", height=300, width=400)

line = C.create_line(108, 120, 320, 40, fill="green")

C.pack()

mainloop()
```

   ***Oval :-**

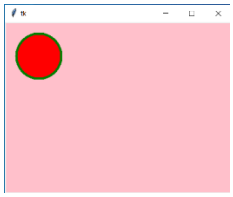
```
from tkinter import *

root = Tk()

C = Canvas(root, bg="yellow", height=400, width=400)

oval = C.create_oval(100, 50, 240,250,fill="Red")

C.pack()
```

```
mainloop()
```

***Circle:-**

```
from tkinter import *
root = Tk()
C = Canvas(root, bg="pink", height=300, width=400)
line = C.create_oval(10, 10, 80, 80, outline = "green", fill = "red",width = 4)
C.pack()
mainloop()
```

O/p-



***Reactangle or Square:-**

```
from tkinter import *
root = Tk()
C = Canvas(root, bg="pink", height=300, width=400)
line = C.create_rectangle(10,10,110,110,outline ="black",fill ="white",width = 4)
C.pack()
mainloop()
```

O/p-

***Polygon :**

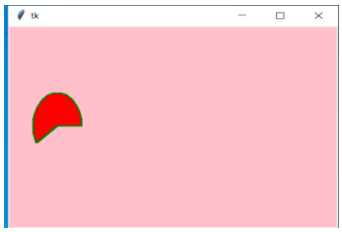
```
from tkinter import *
root = Tk()
C = Canvas(root, bg="pink", height=300, width=400)
points = [150, 100, 200, 120, 240, 180, 210, 200, 150, 150, 100, 200]
line = C.create_polygon(points, outline = "blue", fill = "orange", width = 2)
C.pack()
mainloop()
```

***Arc :-**

```
from tkinter import *
```

```
root = Tk()
C = Canvas(root, bg="pink", height=300, width=400)
line = C.create_arc(30,200,90,100,extent =210,outline ="green",fill ="red", width =2)
C.pack()
mainloop()
```

o/p :-



3. **CheckButton:** To select any number of options by displaying a number of options to a user as toggle buttons. The general syntax is:
   w = CheckButton(master, option=value)

   There are number of options which are used to change the format of this widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

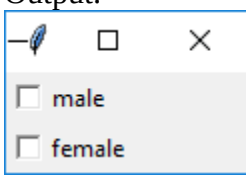
   - **Title**: To set the title of the widget.
   - **activebackground**: to set the background color when widget is under the cursor.
   - **activeforeground**: to set the foreground color when widget is under the cursor.
   - **bg**: to set he normal backgrouSteganography
     Break

     Secret Code:

     Attach a File:nd color.
   - **command**: to call a function.
   - **font**: to set the font on the button label.
   - **image**: to set the image on the widget.

```
from tkinter import *
master = Tk()
var1 = IntVar()
Checkbutton(master, text='male', variable=var1).grid(row=0, sticky=W)
var2 = IntVar()
Checkbutton(master, text='female', variable=var2).grid(row=1, sticky=W)
mainloop()
```

Output:



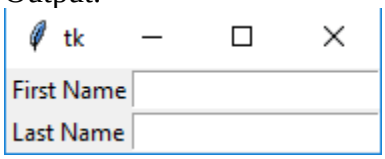
4. **Entry:**It is used to input the single line text entry from the user.. For multi-line text input, Text widget is used.
   The general syntax is:
   w=Entry(master, option=value)

master is the parameter used to represent the parent window.
There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **bd**: to set the border width in pixels.
- **bg**: to set the normal background color.
- **cursor**: to set the cursor used.
- **command**: to call a function.
- **highlightcolor**: to set the color shown in the focus highlight.
- **width**: to set the width of the button.
- **height**: to set the height of the button.

```
from tkinter import *
master = Tk()
Label(master, text='First Name').grid(row=0)
Label(master, text='Last Name').grid(row=1)
e1 = Entry(master)
e2 = Entry(master)
e1.grid(row=0, column=1)
e2.grid(row=1, column=1)
mainloop()
```

Output:



5. **Frame:** It acts as a container to hold the widgets. It is used for grouping and organizing the widgets. The general syntax is:
w = Frame(master, option=value)

   master is the parameter used to represent the parent window.

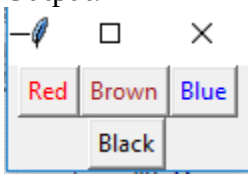
   There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

   - **highlightcolor**: To set the color of the focus highlight when widget has to be focused.
   - **bd**: to set the border width in pixels.
   - **bg**: to set the normal background color.
   - **cursor**: to set the cursor used.
   - **width**: to set the width of the widget.
   - **height**: to set the height of the widget.

```
from tkinter import *

root = Tk()
frame = Frame(root)
frame.pack()
bottomframe = Frame(root)
bottomframe.pack( side = BOTTOM )
redbutton = Button(frame, text = 'Red', fg ='red')
redbutton.pack( side = LEFT)
greenbutton = Button(frame, text = 'Brown', fg='brown')
greenbutton.pack( side = LEFT )
```

```
bluebutton = Button(frame, text ='Blue', fg ='blue')
bluebutton.pack( side = LEFT )
blackbutton = Button(bottomframe, text ='Black', fg ='black')
blackbutton.pack( side = BOTTOM)
root.mainloop()
```

Output:



6. **Label**: It refers to the display box where you can put any text or image which can be updated any time as per the code.
The general syntax is:
w=Label(master, option=value)

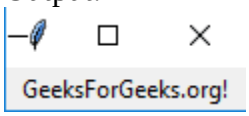
   master is the parameter used to represent the parent window.

   There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

   - **bg**: to set he normal background color.
   - **bg** to set he normal background color.
   - **command**: to call a function.
   - **font**: to set the font on the button label.
   - **image**: to set the image on the button.
   - **width**: to set the width of the button.
   - **height**" to set the height of the button.

```
from tkinter import *
root = Tk()
w = Label(root, text='GeeksForGeeks.org!')
w.pack()
root.mainloop()
```

Output:



7. **Listbox**: It offers a list to the user from which the user can accept any number of options.
The general syntax is:
w = Listbox(master, option=value)

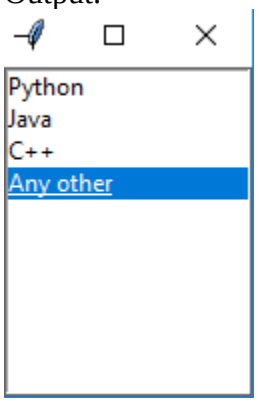
   master is the parameter used to represent the parent window.

   There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

   - **highlightcolor**: To set the color of the focus highlight when widget has to be focused.
   - **bg**: to set he normal background color.
   - **bd**: to set the border width in pixels.
   - **font**: to set the font on the button label.
   - **image**: to set the image on the widget.
   - **width**: to set the width of the widget.
   - **height**: to set the height of the widget.

```
from tkinter import *
top = Tk()
Lb = Listbox(top)
Lb.insert(1, 'Python')
Lb.insert(2, 'Java')
Lb.insert(3, 'C++')
Lb.insert(4, 'Any other')
Lb.pack()
top.mainloop()
```

Output:



8. **MenuButton**: It is a part of top-down menu which stays on the window all the time. Every menubutton has its own functionality. The general syntax is:

w = MenuButton(master, option=value)

master is the parameter used to represent the parent window.

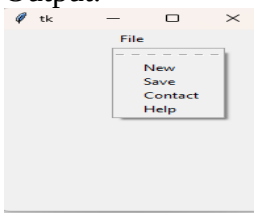
There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **activebackground**: To set the background when mouse is over the widget.
- **activeforeground**: To set the foreground when mouse is over the widget.
- **bg**: to set he normal background color.
- **bd**: to set the size of border around the indicator.
- **cursor**: To appear the cursor when the mouse over the menubutton.
- **image**: to set the image on the widget.
- **width**: to set the width of the widget.
- **height**: to set the height of the widget.
- **highlightcolor**: To set the color of the focus highlight when widget has to be focused.

```
from tkinter import *
top = Tk()
top.geometry("200x250")
menubutton = Menubutton(top, text = "File", relief = FLAT)
menubutton.grid()
menubutton.menu = Menu(menubutton)
menubutton["menu"]=menubutton.menu
menubutton.menu.add_checkbutton(label = "New", variable=IntVar())
menubutton.menu.add_checkbutton(label = "Save", variable = IntVar())
menubutton.menu.add_checkbutton(label = "Contact", variable = IntVar())
menubutton.menu.add_checkbutton(label = "Help", variable = IntVar())
menubutton.pack()
```

```
top.mainloop()
```

Output:



9. **Menu**: It is used to create all kinds of menus used by the application.
   The general syntax is:

w = Menu(master, option=value)

master is the parameter used to represent the parent window.

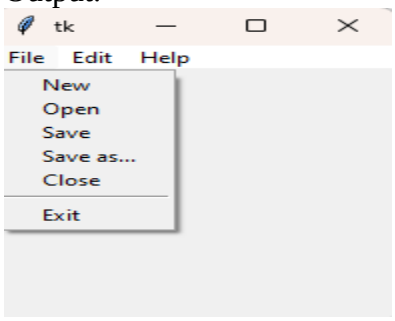
There are number of options which are used to change the format of this widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **title**: To set the title of the widget.
- **activebackground**: to set the background color when widget is under the cursor.
- **activeforeground**: to set the foreground color when widget is under the cursor.
- **bg**: to set he normal background color.
- **command**: to call a function.
- **font**: to set the font on the button label.
- **image**: to set the image on the widget.

```
from tkinter import Toplevel, Button, Tk, Menu
top = Tk()
menubar = Menu(top)
file = Menu(menubar, tearoff=0)
file.add_command(label="New")
file.add_command(label="Open")
file.add_command(label="Save")
file.add_command(label="Save as...")
file.add_command(label="Close")
file.add_separator()
file.add_command(label="Exit", command=top.quit)

menubar.add_cascade(label="File", menu=file)
edit = Menu(menubar, tearoff=0)
edit.add_command(label="Undo")

edit.add_separator()

edit.add_command(label="Cut")
edit.add_command(label="Copy")
edit.add_command(label="Paste")
edit.add_command(label="Delete")
edit.add_command(label="Select All")

menubar.add_cascade(label="Edit", menu=edit)
```

```
help = Menu(menubar, tearoff=0)
help.add_command(label="About")
menubar.add_cascade(label="Help", menu=help)

top.config(menu=menubar)
top.mainloop()
```

Output:



10. **Message**: It refers to the multi-line and non-editable text. It works same as that of Label.
    The general syntax is:
    w = Message(master, option=value)

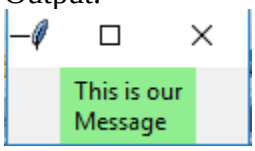
    master is the parameter used to represent the parent window.

    There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

    - **bd**: to set the border around the indicator.
    - **bg**: to set he normal background color.
    - **font**: to set the font on the button label.
    - **image**: to set the image on the widget.
    - **width**: to set the width of the widget.
    - **height**: to set the height of the widget.

```
from tkinter import *
main = Tk()
ourMessage ='This is our Message'
messageVar = Message(main, text = ourMessage)
messageVar.config(bg='lightgreen')
messageVar.pack( )
main.mainloop( )
```

Output:



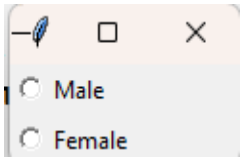
11. **RadioButton:** It is used to offer multi-choice option to the user. It offers several options to the user and the user has to choose one option.
    The general syntax is:
    w = RadioButton(master, option=value)

    There are number of options which are used to change the format of this widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **activebackground**: to set the background color when widget is under the cursor.
- **activeforeground**: to set the foreground color when widget is under the cursor.
- **bg**: to set he normal background color.
- **command**: to call a function.
- **font**: to set the font on the button label.
- **image**: to set the image on the widget.
- **width**: to set the width of the label in characters.
- **height**: to set the height of the label in characters.

```
from tkinter import *
root = Tk()
v = IntVar()
Radiobutton(root, text='Male', variable=v, value=1).pack(anchor=W)
Radiobutton(root, text='Female', variable=v, value=2).pack(anchor=W)
mainloop()
```

Output:



12. **Scale:** It is used to provide a graphical slider that allows to select any value from that scale. The general syntax is:
w = Scale(master, option=value)

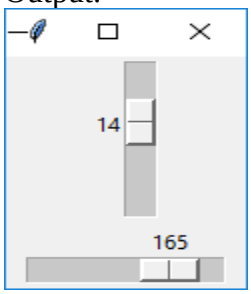
    master is the parameter used to represent the parent window.

    There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

    - **cursor**: To change the cursor pattern when the mouse is over the widget.
    - **activebackground**: To set the background of the widget when mouse is over the widget.
    - **bg**: to set he normal background color.
    - **orient**: Set it to HORIZONTAL or VERTICAL according to the requirement.
    - **from_**: To set the value of one end of the scale range.
    - **to**: To set the value of the other end of the scale range.
    - **image**: to set the image on the widget.
    - **width**: to set the width of the widget.

```
from tkinter import *
master = Tk()
w = Scale(master, from_=0, to=42)
w.pack()
w = Scale(master, from_=0, to=200, orient=HORIZONTAL)
w.pack()
mainloop()
```

Output:



13. **Scrollbar**: It refers to the slide controller which will be used to implement listed widgets.
The general syntax is:
w = Scrollbar(master, option=value)

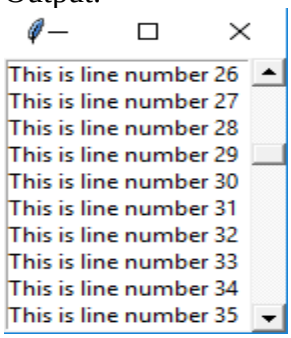
master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **width**: to set the width of the widget.
- **activebackground**: To set the background when mouse is over the widget.
- **bg**: to set he normal background color.
- **bd**: to set the size of border around the indicator.
- **cursor**: To appear the cursor when the mouse over the menubutton.

```
from tkinter import *
root = Tk()
scrollbar = Scrollbar(root)
scrollbar.pack( side = RIGHT, fill = Y )
mylist = Listbox(root, yscrollcommand = scrollbar.set )
for line in range(100):
   mylist.insert(END, 'This is line number' + str(line))
mylist.pack( side = LEFT, fill = BOTH )
scrollbar.config( command = mylist.yview )
mainloop()
```

Output:



14. **Text:** To edit a multi-line text and format the way it has to be displayed.
The general syntax is:
w =Text(master, option=value)

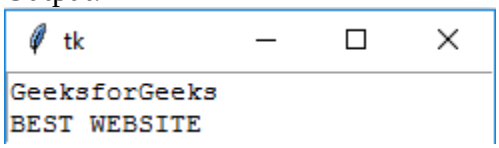
There are number of options which are used to change the format of the text. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **highlightcolor**: To set the color of the focus highlight when widget has to be focused.
- **insertbackground**: To set the background of the widget.
- **bg**: to set he normal background color.
- **font**: to set the font on the button label.
- **image**: to set the image on the widget.
- **width**: to set the width of the widget.
- **height**: to set the height of the widget.

```
from tkinter import *
root = Tk()
T = Text(root, height=2, width=30)
```

```
T.pack()
T.insert(END, 'GeeksforGeeks\nBEST WEBSITE\n')
mainloop()
```

Output:



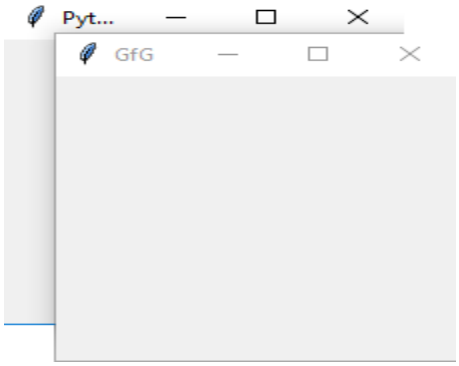
15. **TopLevel:** This widget is directly controlled by the window manager. It don't need any parent window to work on.The general syntax is:
w = TopLevel(master, option=value)

    There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

    - **bg**: to set he normal background color.
    - **bd**: to set the size of border around the indicator.
    - **cursor**: To appear the cursor when the mouse over the menubutton.
    - **width**: to set the width of the widget.
    - **height**: to set the height of the widget.

```
from tkinter import *
root = Tk()
root.title('GfG')
top = Toplevel()
top.title('Python')
top.mainloop()
```

Output:



16. **SpinBox:** It is an entry of 'Entry' widget. Here, value can be input by selecting a fixed value of numbers.The general syntax is:
w = SpinBox(master, option=value)

    There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

    - **bg**: to set he normal background color.
    - **bd**: to set the size of border around the indicator.
    - **cursor**: To appear the cursor when the mouse over the menubutton.
    - **command**: To call a function.
    - **width**: to set the width of the widget.

- **activebackground**: To set the background when mouse is over the widget.
- **disabledbackground**: To disable the background when mouse is over the widget.
- **from_**: To set the value of one end of the range.
- **to**: To set the value of the other end of the range.

```
from tkinter import *
master = Tk()
w = Spinbox(master, from_ = 0, to = 10)
w.pack()
mainloop()
```

Output:

17. **PannedWindow**It is a container widget which is used to handle number of panes arranged in it. The general syntax is:

w = PannedWindow(master, option=value)

master is the parameter used to represent the parent window.
There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **bg**: to set he normal background color.
- **bd**: to set the size of border around the indicator.
- **cursor**: To appear the cursor when the mouse over the menubutton.
- **width**: to set the width of the widget.
- **height**: to set the height of the widget.

```
from tkinter import *
m1 = PanedWindow()
m1.pack(fill = BOTH, expand = 1)
left = Entry(m1, bd = 5)
m1.add(left)
m2 = PanedWindow(m1, orient = VERTICAL)
m1.add(m2)
top = Scale( m2, orient = HORIZONTAL)
m2.add(top)
mainloop()
```

Output:

18. MessageBox Widget

Python Tkinter – MessageBox Widget is used to display the message boxes in the python applications. This module is used to display a message using provides a number of functions.

Syntax:

messagebox.Function_Name(title, message [, options])

Parameters:
There are various parameters :

```
from tkinter import *
from tkinter import messagebox
root = Tk()
root.geometry("300x200")
w = Label(root, text ='GeeksForGeeks', font = "50")
w.pack()
messagebox.showinfo("showinfo", "Information")
messagebox.showwarning("showwarning", "Warning")
messagebox.showerror("showerror", "Error")
messagebox.askquestion("askquestion", "Are you sure?")
messagebox.askokcancel("askokcancel", "Want to continue?")
messagebox.askyesno("askyesno", "Find the value?")
messagebox.askretrycancel("askretrycancel", "Try again?")
root.mainloop()
```

Example-2

- **Function_Name:** This parameter is used to represents an appropriate message box function.
- **title:** This parameter is a string which is shown as a title of a message box.
- **message:** This parameter is the string to be displayed as a message on the message box.
- **options:** There are two options that can be used are:
  1. **default:** This option is used to specify the default button like ABORT, RETRY, or IGNORE in the message box.
  2. **parent:** This option is used to specify the window on top of which the message box is to be displayed.

Function_Name:
There are functions or methods available in the messagebox widget.

1. **showinfo():** Show some relevant information to the user.
2. **showwarning():** Display the warning to the user.
3. **showerror():** Display the error message to the user.
4. **askquestion():** Ask question and user has to answered in yes or no.
5. **askokcancel():** Confirm the user's action regarding some application activity.
6. **askyesno():** User can answer in yes or no for some action.
7. **askretrycancel():** Ask the user about doing a particular task again or not.

Example:1

```
from tkinter import *
from tkinter import messagebox
import tkinter

top = tkinter.Tk()

def helloCallBack():
   messagebox.showinfo( "Hello Python", "Hello World")

B = tkinter.Button(top, text ="Hello", command = helloCallBack)

B.pack()
B.place(bordermode=OUTSIDE, height=100, width=100)
top.mainloop()
```

Root window :-

To display the graphical output, we need space on the screen. This space that is initially allocated to every GUI program is called 'top level window' or 'root window' .

Example 1

```
#import all components from tkinter
from tkinter import *
#create the root window
root = Tk()
#wait and watch for any events that may take place
#in the root window
root.mainloop()
```

Introduction to Tkinter mainloop

Tkinter is defined as a module inside the Python standard library that helps to serve as an interface to an easy Toolkit Tk. This has a variety of GUI to build an interface like buttons and widgets. The method mainloop plays a vital role in Tkinter as it is a core application that waits for events and helps in updating the GUI or in simple terms, we can say it is event-driven programming. If no mainloop() is used then nothing will appear on the window Screen. This method takes all the objects that were created and have interactions response.

Syntax

To get started

```
from tkinter import Tk    // Tkinter Library
root = Tk()                 //Creates root master with the TK() constructor
app = App(root)
app.pack()
root.mainloop             //main event loop.
```

How the mainloop work in Tkinter?

Root. mainloop() is simply a method in the main window that executes what we wish to execute in an application (lets Tkinter to start running the application). As the name implies it will loop forever until the user exits the window or waits for any events from the user. The mainloop automatically receives events from the window system and deliver them to the application widgets. This gets quit when we click on the close button of the title bar. So, any code after this mainloop() method will not run.

When a GUI is gets started with the mainloop() a method call, Tkinter python automatically initiates an infinite loop named as an event loop. So, after this article, we will be able to use all the widgets to develop an application in Python. Different widgets are Combo box, button, label, Check button, Message Box, Images, and Icons.

Constructor

Well, a **root window** is created by calling Tkinter Constructor TK() as it creates a widget.

```
Root=tk.TK() // Constructor
```

This line of code automatically creates a GUI window with a title bar, close button.

```
Creating window TK()
```

```
Window =TK()
……
Window.mainloop()
```

Methods

Tkinter provides the following methods to specify a widget to display on a window.

pack(): This method helps in Placing the widgets within its master. With options like side, fill, expand.

```
Widget.pack();
```

grid(): This method helps to place a widget in a grid format or tabular form. We can specify in the method call with the number of rows and columns. It takes possible options like column, row, ipadx, and Y and sticky.

```
Widget.grid();
```

Place(): This method arranges the widgets by placing them in specific positions instructed by the programmer. It organizes with respective to X and Y coordinates.

```
Widget.place();
```

Grid Sample code

```
from tkinter import *
master = Tk()
Label(master, text='Full Name').grid(row=1)
Label(master, text='Email-ID').grid(row=2)
A = Entry(master)
B = Entry(master)
A.grid(row=1, column=1)
B.grid(row=2, column=1)
mainloop()
```

Examples

Here are the following examples mention below

Example 1

```
import tkinter as tk
```

```
res = tk.Tk()
res.title('Incrementing the Process')
button = tk.Button(res, text='Pause', width=30, command=res.destroy)
button.pack()
res.mainloop()
```

Explanation: The above code is a simple application that creates a button widget, when we click on it we could see a tiny pop-up window with the text ‘Pause’. The output is shown in the below screenshot.

Example 2

Using pack() method

Code:

```
import tkinter
from tkinter import *
parent = Tk()
blbtn= Button(parent, text = "Blue", fg = "Blue")
blbtn.pack( side = TOP)
Brbtn = Button(parent, text = "Brown", fg = "Brown")
Brbtn.pack( side = LEFT)
Aqbtn = Button(parent, text = "Aqua", fg = "Aqua")
Aqbtn .pack( side = BOTTOM )
orgbtn = Button(parent, text = "Orange", fg = "Orange")
orgbtn.pack( side = RIGHT)
parent.mainloop()
```

Example 3

Using Label widgets on the GUI window.

Code:

```
import tkinter as tk
window = tk.Tk()
frame = tk.Frame(master=window, width=100, height=100)
frame.pack()
lab1 = tk.Label(master=frame, text="Points at (5, 5)", bg="Aqua")
lab1.place(x=5, y=5)
lab2 = tk.Label(master=frame, text="Points at (65, 65)", bg="Pink")
lab2.place(x=65, y=65)
window.mainloop()
```

Explanation: Here the code produces a window like this. We have imported a Tkinter package and a window is defined. Next, a label is defined to show the output with the points on the window. The scale point is given for each label. Finally, the mainloop is executed to touch the event.

Example 4

Using Check box -mainloop() is executed

Code:

```
from tkinter import *
root = Tk()
root.geometry("400x300")
aa = Label(root, text ='EDUCBA-Online ', font = "60")
aa.pack()
Chckbtn1 = IntVar()
Chckbtn2 = IntVar()
Chckbtn3 = IntVar()
Btn1 = Checkbutton(root, text = "Courses",
variable = Chckbtn1,
onvalue = 2,
offvalue = 0,
```

```
height = 3,
width = 12)
Btn2 = Checkbutton(root, text = "Free-Trial",
variable = Chckbtn2,
onvalue = 3,
offvalue = 0,
height = 3,
width = 12)
Btn3 = Checkbutton(root, text = "Paid",
variable = Chckbtn3,
onvalue = 1,
offvalue = 0,
height = 2,
width = 10)
Btn1.pack()
Btn2.pack()
Btn3.pack()
mainloop()
```

Explanation:

When we execute the above code, we could see a window pop-up with a title and some label and button clicks. We have used a Class Tk to create the main window and a pack() method is used to position inside the parent window. And the command keyword is used here to specify the function in handling the click events and here it is a check box.

Example 5

Code:

```
import tkinter as tk
window = tk.Tk()
frA = tk.Frame()
labA = tk.Label(master=frA, text="This is A Frame")
labA.pack()
frb = tk.Frame()
labB = tk.Label(master=frb, text="This is B Frame")
labB.pack()
frb.pack()
frA.pack()
window.mainloop()
```

Example 6

Using place() method :-

Code:

```
from tkinter import *
```

```
top = Tk()
top.geometry("300x150")
LoginID = Label(top, text = "LoginID").place(x = 20,y = 40)
email = Label(top, text = "Email").place(x = 20, y = 80)
Phone = Label(top, text = "Phone").place(x = 20, y = 110)
a1 = Entry(top).place(x = 70, y = 40)
a2 = Entry(top).place(x = 70, y = 80)
a3 = Entry(top).place(x = 85, y = 110)
```

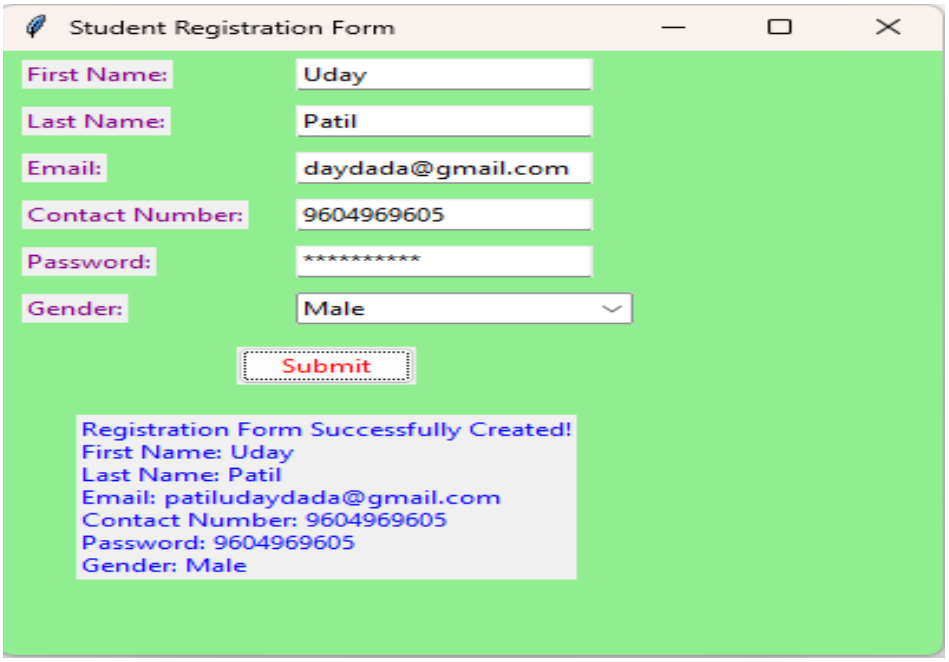
```
top.mainloop()
```

Implementation of Python Student Registration Form :-

```
import tkinter as tk
from tkinter import ttk
def submit_form():
  # Retrieve values from the form
  first_name = entry_first_name.get()
  last_name = entry_last_name.get()
  email = entry_email.get()
  contact_number = entry_contact_number.get()
  password = entry_password.get()
  gender = gender_var.get()
  # Display the submitted information in blue color
  result_label.config(text=f"Registration Form Successfully Created!\n"
                      f"First Name: {first_name}\n"
                      f"Last Name: {last_name}\n"
                      f"Email: {email}\n"
                      f"Contact Number: {contact_number}\n"
                      f"Password: {password}\n"
                      f"Gender: {gender}", foreground="blue")
# Create main window
root = tk.Tk()
root.title("Flood-Registration Form")
root.geometry("400x400")
root.configure(bg="lightgreen")
# Create labels
label_first_name = ttk.Label(root, text="First Name:", foreground="purple")
```

```
label_last_name = ttk.Label(root, text="Last Name:", foreground="purple")

label_email = ttk.Label(root, text="Email:", foreground="purple")

label_contact_number = ttk.Label(root, text="Contact Number:", foreground="purple")

label_password = ttk.Label(root, text="Password:", foreground="purple")

label_gender = ttk.Label(root, text="Gender:", foreground="purple")

# Create entry widgets

entry_first_name = ttk.Entry(root)

entry_last_name = ttk.Entry(root)

entry_email = ttk.Entry(root)

entry_contact_number = ttk.Entry(root)

entry_password = ttk.Entry(root, show="*")

# Create a Combobox for gender

gender_var = tk.StringVar()

gender_combobox = ttk.Combobox(root, textvariable=gender_var, values=["Male", "Female"],
state="readonly")

gender_combobox.set("Male")  # Default value

# Create submit button

submit_button = ttk.Button(root, text="Submit", command=submit_form, style="TButton")

# Create label for displaying the result

result_label = ttk.Label(root, text="", foreground="blue")

# Place widgets on the grid

label_first_name.grid(row=0, column=0, padx=10, pady=5, sticky="w")

label_last_name.grid(row=1, column=0, padx=10, pady=5, sticky="w")

label_email.grid(row=2, column=0, padx=10, pady=5, sticky="w")

label_contact_number.grid(row=3, column=0, padx=10, pady=5, sticky="w")

label_password.grid(row=4, column=0, padx=10, pady=5, sticky="w")

label_gender.grid(row=5, column=0, padx=10, pady=5, sticky="w")
```

```
entry_first_name.grid(row=0, column=1, padx=10, pady=5, sticky="w")

entry_last_name.grid(row=1, column=1, padx=10, pady=5, sticky="w")

entry_email.grid(row=2, column=1, padx=10, pady=5, sticky="w")

entry_contact_number.grid(row=3, column=1, padx=10, pady=5, sticky="w")

entry_password.grid(row=4, column=1, padx=10, pady=5, sticky="w")

gender_combobox.grid(row=5, column=1, padx=10, pady=5, sticky="w")


submit_button.grid(row=6, column=0, columnspan=2, pady=10)

result_label.grid(row=7, column=0, columnspan=2, pady=10)

# Configure style for the submit button

style = ttk.Style()

style.configure("TButton", foreground="red")

# Run the Tkinter main loop

root.mainloop()
```



Database Connectivity :-

Python Connectivity Code:-

How to connect MySQL database in Python

Let's see how to connect the MySQL database in Python using the 'MySQL Connector Python' module.

Arguments required to connect

You need to know the following detail of the MySQL server to perform the connection from Python.

Argument	Description
Username	The username that you use to work with MySQL Server. The default username for the MySQL database is a **root**.
Password	Password is given by the user at the time of installing the MySQL server. If you are using root then you won't need the password.
Host name	The server name or Ip address on which MySQL is running. if you are running on localhost, then you can use **localhost** or its IP 127.0.0.0
Database name	The name of the database to which you want to connect and perform the operations.

How to Connect to MySQL Database in Python

1. **Install MySQL connector module**

   1. Use the pip command to install MySQL connector Python.
   `pip install mysql-connector-python` **or** `python -m pip install mysql-connector-python`

2. **Import MySQL connector module**

   Import using a `import mysql.connector` statement so you can use this module's methods to communicate with the MySQL database.

3. **Use the connect() method**

Use the `connect()` method of the MySQL Connector class with the required arguments to connect MySQL. It would return a `MySQLConnection` object if the connection established successfully

4. **Use the cursor() method**

   Use the cursor() method of a `MySQLConnection` object to create a cursor object to perform various SQL operations.

5. **Use the execute() method**

   The execute() methods run the SQL query and return the result.

6. **Extract result using fetchall()**

   Use `cursor.fetchall()` or `fetchone()` or `fetchmany()` to read query result.

7. **Close cursor and connection objects**

   use `cursor.clsoe()` and `connection.clsoe()` method to close open connections after your work completes

Example to connect to MySQL Database in Python :-

1)Create Database :-

Create database mydatabase;

Connect mydatabase;

2)Creating the connection:-

```
import mysql.connector

#Create the connection object

myconn = mysql.connector.connect(host = "localhost", user = "root",password = "root", database = "mydatabase")

#printing the connection object

print(myconn)

#creating the cursor object

cur = myconn.cursor()

  print(cur)
```

3)Create Table Custormer :-

Create table student (RollNo int,Name varchar(255), MobileNo int);

a)For insert:-

To fill a table in MySQL, use the "INSERT INTO" statement.

```
import mysql.connector

 mydb = mysql.connector.connect(  host="localhost",  user="root",  password="root",
  database="mydatabase")
cur = mydb.cursor()

 dbs = cur.execute("insert into mydb.student values(124,'Ajay',12345)")
mydb.commit();

print("save data successfully…..")
mydb.close()
```

User Input :-

```
import mysql.connector

mydb = mysql.connector.connect(host = "localhost", user = "root",password = "root", database
= " mydatabase ")

cur = mydb.cursor()
rno=input("inter your Roll Number:")
rnm=input("inter your Name:")

sql="insert into student(rollno,name)values(%s,%s)"
val=(rno,rnm)

cur.execute(sql,val)
mydb.commit()

print(cur.rowcount,"record inserted")
```

b) Select and display From a Table-

To select from a table in MySQL, use the "SELECT" statement:

Select all records from the "student" table, and display the result:-

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="root",
  database=" mydatabase "
)
```

```
mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

c) Update :-

You can update existing records in a table by using the "UPDATE" statement:

Overwrite the Name column from "Ajay" to "Vijay":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="root",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "UPDATE customers SET Name = 'Vijay' WHERE Name = 'Ajay'"

mycursor.execute(sql)

mydb.commit()

print(mycursor.rowcount, "record(s) Updated")
```

b)For delete:-

You can delete records from an existing table by using the "DELETE FROM" statement:

```
import mysql.connector

  mydb = mysql.connector.connect(  host="localhost",  user="root",
  password="root",  database=" mydatabase ")
cur = mydb.cursor()

  dbs = cur.execute("delete from student where name='Ajay'")
mydb.commit();

  print("Delete data successfully…")
mydb.close()
```