MODULE -3

1.# Python program to draw square

```
import turtle
a = turtle.Turtle()
a.goto(90,90)
for i in range(4):
        a.forward(50)
        a.right(90)

turtle.done()
```

2.Circle

```
import turtle
t = turtle.Turtle()
r=int(input("enter the radius"))
t.circle(r)
turtle.done()
```

3.Fill color square


```
import turtle
t = turtle.Turtle()
s = int(input("Enter the length of the side of the square: "))
col = input("Enter the color name or hex value of color(# RRGGBB): ")
t.fillcolor(col)
t.begin_fill()
for i in range(4):
  t.forward(s)
  t.right(90)
t.end_fill()
turtle.done()
```

4.Rectangle

```
import turtle

t = turtle.Turtle()

l = int(input("Enter the length of the Rectangle: "))
w = int(input("Enter the width of the Rectangle: "))

t.forward(l)
```

```
t.left(90)


t.forward(w)
t.left(90)
t.forward(l)
t.left(90)
t.forward(w)
t.left(90)
turtle.done()
```

5.Square

```
import turtle
t = turtle.Turtle()
s = int(input("Enter the length of the side of square: "))
for i in range(4):
    t.forward(s)
    t.left(90)

turtle.done()
```


6.star

```
import turtle
s = turtle.Turtle()

s.right(75)
s.forward(100)

for i in range(4):
    s.right(144)
    s.forward(100)
turtle.done()
```

7.Triangle fill

```
import turtle
t = turtle.Turtle()
s = int(input("Enter the length of the side of the triangle: "))
col = input("Enter the color name")
t.fillcolor(col)
t.begin_fill()
```

```
for i in range(3):
    t.forward(s)
    t.right(-120)
t.end_fill()
```

8.Inputing using simple dialog

```
import tkinter as tk
from tkinter import simpledialog

application_window = tk.Tk()

answer = simpledialog.askstring("Input", "What is your first name?",
                                parent=application_window)
if answer is not None:
    print("Your first name is ", answer)
else:
    print("You don't have a first name?")

answer = simpledialog.askinteger("Input", "What is your age?",
                                 parent=application_window,
                                 minvalue=0, maxvalue=100)
if answer is not None:
    print("Your age is ", answer)
else:
    print("You don't have an age?")

answer = simpledialog.askfloat("Input", "What is your mark?",
                               parent=application_window,
                               minvalue=0.0, maxvalue=100.0)
if answer is not None:
    print("Your mark is ", answer)
else:
    print("You don't have a salary?")
```


9.GUI

```
import tkinter as t
from tkinter import ttk

global my_counter
```

```
def create_user_interface(para_win):
    global my_counter

    my_counter = ttk.Label(para_win, text="0")
    my_counter.grid(row=0, column=0)

    increment_button = ttk.Button(para_win, text="Add 1 to counter")
    increment_button.grid(row=1, column=0)
    increment_button['command'] = increment_counter

    quit_button = ttk.Button(para_win, text="Quit")
    quit_button.grid(row=2, column=0)
    quit_button['command'] = para_win.destroy


def increment_counter():
    global my_counter
    my_counter['text'] = str(int(my_counter['text']) + 1)

win = t.Tk()
create_user_interface(win)
win.mainloop()
```


10.GUI Hello world

```
import tkinter as t
from tkinter import ttk


win = t.Tk()


my_label = ttk.Label(win, text="Hello World!")
my_label.grid(row=1, column=1)

win.mainloop()
```

11.Message Box

```
from tkinter import messagebox

messagebox.showinfo("Information","Informative message")
messagebox.showerror("Error", "Error message")
```

```
messagebox.showwarning("Warning","Warning message")
```

12.Window config

```
import turtle
sc = turtle.Screen()
sc.setup(400,400)
sc.bgcolor("blue")
```

13.Window config with position

```
import turtle
sc = turtle.Screen()
sc.setup(400, 400, startx = 50,
                starty = -200)
sc.bgcolor("blue")
```

14.TEXTBOX input


```
import tkinter as t
win = t.Tk()
win.title("TextBox Input")
win.geometry('800x200')


def printInput():
    inp = inputtxt.get(1.0, "end-1c")
    lbl.config(text = "Provided Input: "+inp)

# TextBox Creation
inputtxt = t.Text(win,height = 5,width = 20)
inputtxt.pack()
# Button Creation
printButton = t.Button(win,text = "Print",command = printInput)
printButton.pack()

# Label Creation
lbl = t.Label(win, text = "")
lbl.pack()
win.mainloop()
```

15.Image Processing concepts

a)Edge.py

```
import cv2


FILE_NAME = 'flower.jpg'
try:
    # Read image from disk.
    img = cv2.imread(FILE_NAME)

    # Canny edge detection.
    edges = cv2.Canny(img, 900, 900)

    # Write image back to disk.
    cv2.imwrite('result.jpg', edges)
except IOError:
    print ('Error while reading files !!!')
```


b) Rotate.py

```
import cv2

FILE_NAME = 'flower.jpg'
try:

    img = cv2.imread(FILE_NAME)

    (rows,cols) = img.shape[:2]

    M = cv2.getRotationMatrix2D((rows/2,cols/2),62,1)
    res = cv2.warpAffine(img, M,(rows,cols))


    cv2.imwrite('result.jpg', res)
except IOError:
    print ('Error while reading files !!!')
```

c) scaling.py

```
import cv2
import numpy as np

FILE_NAME = 'flower.jpg'
try:

    img = cv2.imread(FILE_NAME)

    (height, width) = img.shape[:2]

    res = cv2.resize(img, (int(width / 2), int(height / 2)), interpolation =
    cv2.INTER_CUBIC)

    cv2.imwrite('result.jpg', res)

except IOError:
    print ('Error while reading files !!!')
```

d)Translate.py


```
import cv2
import numpy as np

FILE_NAME = 'flower.jpg'
# Create translation matrix.
# If the shift is (x, y) then matrix would be
# M = [1 0 x]
#     [0 1 y]
# Let's shift by (100, 50).
M = np.float32([[1, 0, 100], [0, 1, 50]])

try:

    # Read image from disk.
    img = cv2.imread(FILE_NAME)
    (rows, cols) = img.shape[:2]

    # warpAffine does appropriate shifting given the
    # translation matrix.
    res = cv2.warpAffine(img, M, (cols, rows))

    # Write image back to disk.
```

```
    cv2.imwrite('result.jpg', res)

except IOError:
    print ('Error while reading files !!!')
```

e) Image basics

```
import cv2
img=cv2.imread("flower.jpg")
dimensions=img.shape
height=img.shape[0]
width=img.shape[1]
channels=img.shape[2]
print("dimensions=",dimensions)
print("height",height)
print("width",width)
print("channels",channels)
```

MODULE 4

1)DESIGN WITH CLASSES


```
class stud:

    attr1 = "Anny"
    attr2 = "Engg"


    def fun(self):
            print("I'm ", self.attr1)
            print("I'm an", self.attr2)


obj = stud()

print(obj.attr1)
obj.fun()
```

2)Constructor

```
class stud:


    def __init__(self):
```

```
        self.name = "ABC"
        self.roll = 23


    def display(self):
        print(self.name)
        print(self.roll)

obj = stud()
obj.display()
```

3)Destructor

```
class Employee:


    def __init__(self):
        print('Employee created')


    def __del__(self):
        print("Destructor called")

def Create_obj():
    print('Making Object...')
    obj = Employee()
    print('function end...')
    return obj

print('Calling Create_obj() function...')
obj = Create_obj()
print('Program End...')
```


4)Array of objects

```
class studarr:
    def __init__(self, name, roll):
        self.name = name
        self.roll = roll
```

```
list1 = []


list1.append( studarr('Akash11', 2) )
list1.append( studarr('Deependr111a', 40) )
list1.append( studarr('Reaper111', 44) )

for x in list1:
    print( x.name, x.roll, sep =' ' )
```

4)Parameterized constructor

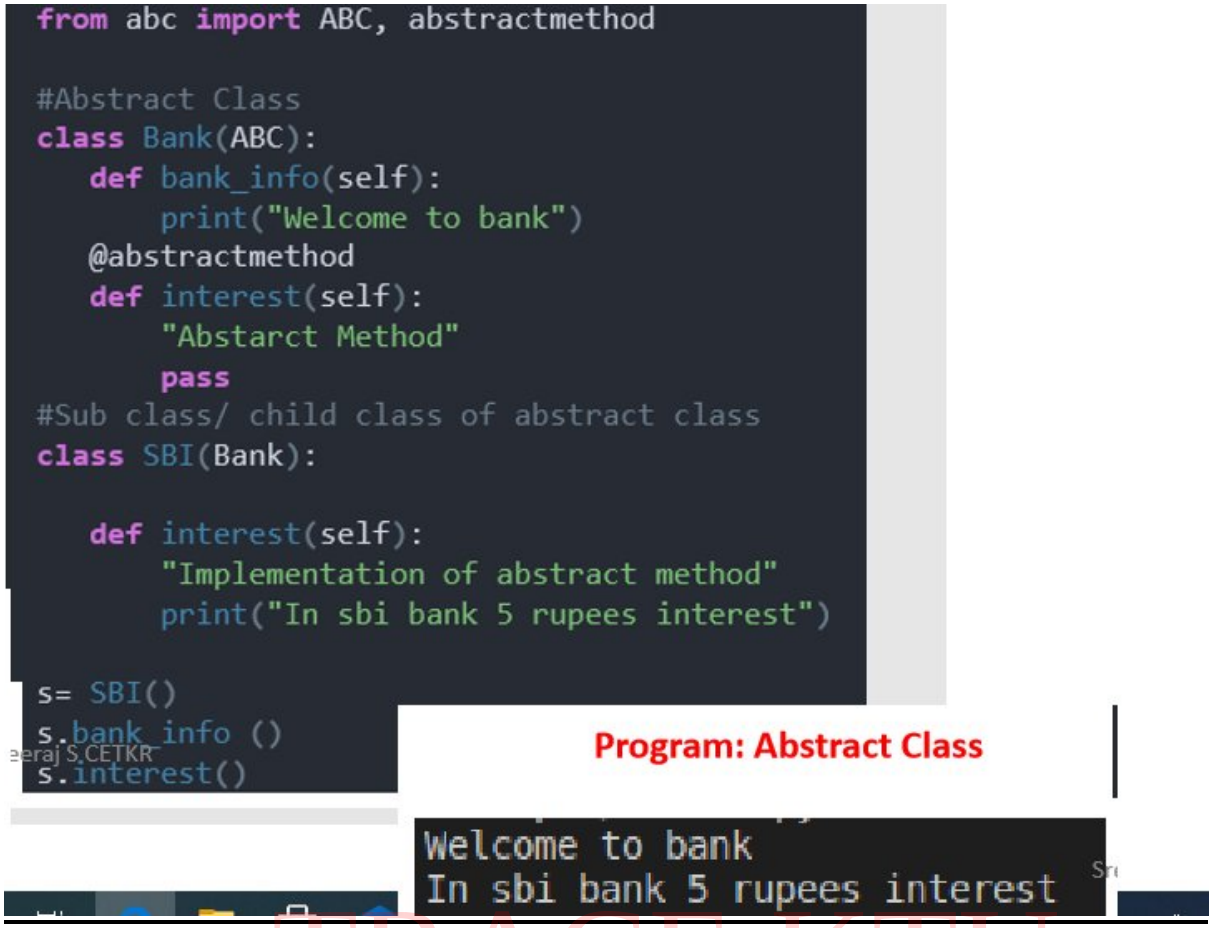
```
class Addition:
    x = 0
    y = 0
    z = 0

    # parameterized constructor
    def __init__(self, f, s):
            self.x = f
            self.y = s

    def display(self):
            print("First number = " + str(self.x))
            print("Second number = " + str(self.y))
            print("Addition of two numbers = " + str(self.z))

    def calculate(self):
            self.z = self.x + self.y

obj = Addition(1000, 2000)
obj.calculate()
obj.display()
```


5)Inheritance SEE CLASS NOTES FOR DIAGRAM

```
class Person:

  # __init__ is known as the constructor
  def __init__(self, name, idnumber):
    self.name = name
    self.idnumber = idnumber
```

```
  def display(self):
    print(self.name)
    print(self.idnumber)

  def details(self):
    print("My name is {}".format(self.name))


# child class
class Employee(Person):
  def __init__(self, name, idnumber, salary, post):
    self.salary = salary
    self.post = post
 # invoking the __init__ of the parent class
    Person.__init__(self, name, idnumber)


  def details(self):
    print("My name is {}".format(self.name))
    print("IdNumber: {}".format(self.idnumber))
    print("Salary: {}".format(self.salary))
    print("Post: {}".format(self.post))


# creation of an object variable or an instance
a = Employee('Rahul', 886012, 200000, "Intern")

# calling a function of the class Person using
a.display()
a.details()
```


6.Abstract Base Class



Abstraction classes in Python

In Python, abstraction can be achieved by using abstract classes and interfaces.

A class that consists of one or more abstract method is called the abstract class. Abstract methods do not contain their implementation. Abstract class can be inherited by the subclass and abstract method gets its definition in the subclass. Abstraction classes are meant to be the blueprint of the other class. An abstract class can be useful when we are designing large functions. An abstract class is also helpful to provide the standard interface for different implementations of components. Python provides the **abc** module to use the abstraction in the Python program. Let's see the following syntax.

Syntax

```
from abc import ABC
class ClassName(ABC):
```

Example

```
from abc import ABC, abstractmethod
class Car(ABC):
    def mileage(self):
        pass

class Tesla(Car):
    def mileage(self):
        print("The mileage is 30kmph")
class Suzuki(Car):
    def mileage(self):
        print("The mileage is 25kmph ")
class Duster(Car):
     def mileage(self):
          print("The mileage is 24kmph ")

class Renault(Car):
    def mileage(self):
           print("The mileage is 27kmph ")

     t= Tesla ()
     t.mileage()

     r = Renault()
     r.mileage()

     s = Suzuki()
     s.mileage()
     d = Duster()
     d.mileage()
```


Output

The mileage is 30kmph

The mileage is 27kmph

The mileage is 25kmph

The mileage is 24kmph

Below are the points which we should remember about the abstract base class in Python.

- An Abstract class can contain the both method normal and abstract method.
- An Abstract cannot be instantiated; we cannot create objects for the abstract class.

Abstraction is essential to hide the core functionality from the users.

Polymorphism

Polymorphism contains two words "poly" and "morphs". Poly means many, and morph means shape. By polymorphism, we understand that one task can be performed in different ways. For example - you have a class animal, and all animals speak. But they speak differently. Here, the "speak" behavior is polymorphic in a sense and depends on the animal. So, the abstract "animal" concept does not actually "speak", but specific animals (like dogs and cats) have a concrete implementation of the action "speak".

Example1

```
def add(x, y, z = 0):
        return x + y+z

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


Output:
5
9

Polymorphism with Inheritance:

This process of re-implementing a method in the child class is known as **Method Overriding**.

```
class Bird:
def intro(self):
print("There are many types of birds.")

def flight(self):
print("Most of the birds can fly but some cannot.")

class sparrow(Bird):
def flight(self):
```

```
print("Sparrows can fly.")

class ostrich(Bird):
def flight(self):
print("Ostriches cannot fly.")

obj_bird = Bird()
obj_spr = sparrow()
obj_ost = ostrich()

obj_bird.intro()
obj_bird.flight()

obj_spr.intro()
obj_spr.flight()

obj_ost.intro()
obj_ost.flight()
```

Output:

```
There are many types of birds.

Most of the birds can fly but some cannot.

There are many types of birds.

Sparrows can fly.

There are many types of birds.

Ostriches cannot fly.
```


Polymorphism with a Function and objects:

It is also possible to create a function that can take any object, allowing for polymorphism.

```
class India():
def capital(self):
print("New Delhi is the capital of India.")
def language(self):
print("Hindi is the most widely spoken language of India.")
def type(self):
print("India is a developing country.")

class USA():
def capital(self):
print("Washington, D.C. is the capital of USA.")
def language(self):
```

```
print("English is the primary language of USA.")
def type(self):
print("USA is a developed country.")

def func(obj):
        obj.capital()
        obj.language()
        obj.type()

obj_ind = India()
obj_usa = USA()

func(obj_ind)
func(obj_usa)
```

Output:

```
New Delhi is the capital of India.
Hindi is the most widely spoken language of India.
India is a developing country.
Washington, D.C. is the capital of USA.
English is the primary language of USA.
USA is a developed country.
```


7. Study Inheritance examples from lecture note

8Study exception handling and interface program from PDF notes above