SIKSHA 'O' ANUSANDHAN
DEEMED TO BE UNIVERSITY

Admission Batch: 2019 **Session:** 2021-22

Laboratory Record

Programming in Python (CSE 3142)

Submitted by

Name: Saswat Mohanty

Registration No.: 1941012407

Branch: Computer Science and Engineering

Semester: 5th Section: 'D'

Department of Computer Science & Engineering

Faculty of Engineering & Technology (ITER)

Jagamohan Nagar, Jagamara, Bhubaneswar, Odisha - 751030

INDEX

Sl. No	Name of Program	Page No	Remarks
1.	Minor Assignment - 1	1-8	good 06/12/21
2.	Minor Assignment - 2	9-21	good 06/12/21
3.	Minor Assignment - 3	22-39	good 06/12/21
4.	Minor Assignment - 5	40-45	good 21/12/21
5.	Minor Assignment - 6	46-57	good 21/12/21
6.	Minor Assignment - 7	58-68	good 21/12/21
7.	Minor Assignment - 8	69-77.	good 04/01/22
8.	Minor Assignment - 9	78-88	
9.	Minor Assignment - 10	89-97	
10	Minor Assignment - 11	98-112	

Minor Assignment - 11

Classes II

Q1) Define a class complex Numbers, Write operations for addition, subtraction, and multiplication, using the notion of operator overloading.

Program :-

```
class complex:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __add__(self, other):
        return self.a + other.a, self.b + other.b

    def __sub__(self, other):
        return self.a - other.a, self.b - other.b

    def __mul__(self, other):
        return self.a * other.a, self.b * other.b

ob1 = complex(1,2)
ob2 = complex(2,3)
ob3 = ob1 + ob2
ob4 = ob1 - ob2
ob5 = ob1 * ob2
print(ob3)
print(ob4)
print(ob5)
```

Output :-

(3, 5)
(-1, -1)
(2, 6)

Q2) Define function overloading. How function overloading works in Python. Explain. What will be the output of the following Python script define all the functioning of the script.

(Ans) Function overloading is the ability to have multiple functions with the same name but with different signatures / implementations.

Python does not support function overloading. When we define multiple functions with the same name, the latter one always overrides the former and thus, in the namespace, there will always be a single entry against each function name.

Program :-

```
class compute:
  def area(self, x = None, y = None):
    if x != None and y != None:
      return x * y
    elif x != None:
      return x + x
    else:
      return 0

obj = compute()
print('Area Value: ', obj.area())
print('Area Value: ', obj.area(4))
print('Area Value: ', obj.area(3,5))
```

Output :-

Area Value : 0

Area Value : 16

Area Value : 15

Q3) Write short note on the following with examples :

a) Data Hiding (Include advantages & disadvantages of data hiding)

b) Encapsulation

c) Modifier and Accessor Methods

d) Static Method

e) Inheritance .

Ans) a) Data hiding is a concept which underlines the hiding of data or information from the user. It is one of the key aspects of object-oriented programming strategies. It includes object details such as data members, internal work. Data hiding excludes full data entry to class members and defends object integrity by preventing unintended changes. Data hiding also minimizes system complexity for increase robustness by limiting interdependencies between software components. Data hiding is also known as information hiding. In class, if we declare the data members as private so that no other class can access the data members, then it is a process of hiding data.

Advantages of data hiding :

- It helps to prevent damage or misuse of volatile data by hiding it from the public. The class objects are disconnected from the irrelevant data.
- It isolates objects as the basic concept of OOP.
- It increases the security against hackers that are unable

to access important data.

Disadvantages of data hiding:

- It enables programmers to write lengthy code to hide important data from common clients
- The linkage between the visible and invisible data makes the objects work faster, but data hiding prevents this linkage.

b) Encapsulation enables us to group together related data and its associated functions under one name.

c) The methods in the class definition that modify the value of one or more arguments are known as modifiers. For example, the methods setAddress and setName in the class Person are modifiers since they change the values of the data attributes address and name respectively associated with parameter self. However methods such as getAddress and getName of the class MyDate only access (do not modify) the data attributes address and name of the object self. Such methods are known as accessors.

d) Static Method used for modifying class data members and do not require passing by object as the first parameter

e) Inheritance is an important property of object oriented programming that imparts ability to a class to inherit properties and behaviour of another class.

Q4) Write a class Point having x and y coordinates as data members. Write another class LineSegment that derives the class Point. Also, list appropriate methods.

Program :-

```
import math
from numpy import mat
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
class LineSegment(Point):
    def __init__(self, x, y):
        super().__init__(x,y)
    def length(self, other):
        return math.sqrt(((self.x-other.x)**2) +
                         (self.y-other.y)**2))

ob1 = LineSegment(10, 20)
ob2 = LineSegment(5, 10)
print(LineSegment.length(ob1, ob2))
```

Output :-

11.180339887498949

Q5) Why might a class need to manually call the __init__ method in superclass?

(Ans) It because one needs to define something that is NOT done in the base-class '__init__', and the only possibility to obtain that is to put its execution in a derived class '__init__' function.

Q6) Examine the following class Shape :

```
class Shape:
    def __init__(self, shapeType, x, y):
        self.shapeType = shapeType
```

```
        self.length = x
        self.width = y
    def computeArea():
        pass
```

The class Shape should be inherited by the classes Rectangle and Triangle. Both the derived classes should invoke the method __init__ to initialize data members. Note that length and width correspond to height and base for a triangle. The derived classes should override the method computeArea of the superclass Shape.

Program :-

```
class Shape:
    def __init__(self, shapeType, x, y):
        self.shapeType = shapeType
        self.length = x
        self.width = y
    def computeArea():
        pass

class Rectangle(Shape):
    def __init__(self, shapeType, x, y):
        super().__init__(shapeType, x, y)
    def computeArea(self):
        return self.length * self.width

class Triangle(Shape):
    def __init__(self, shapeType, x, y):
        super().__init__(shapeType, x, y)
```

```
def computeArea(self):
    return self.length * self.width *0.5

ob1 = Rectangle("Rectangle",10,20)
print("Area of ",ob1.shapeType, ob1.computeArea())
ob2 = Triangle("Triangle", 10,20)
print("Area of "ob2.shapeType, ob2.computeArea())
```

Output :-

```
Area of Rectangle 200
Area of Triangle 100.0
```

07) Define a base class vehicle, having attributes registration number, make, model, and colour. Also, define classes PassengerVehicle that derive from the class Vehicle. The PassengerVehicle class should have additional attribute for maximum passenger capacity. The CommercialVehicle class should have an additional attribute for maximum load capacity. Define __init__ method for all these classes. Also, define get and set methods to retrieve & set the value of the data attributes.

Program :-

```
class Vehicle:
  def __init__(self, regnum, make, model, color):
    self.regnum = regnum
    self.make = make
    self.model = model
    self.color = color

class PassengerVehicle(Vehicle):
  def __init__(self, regnum, make, model, color, pasCap):
    super().__init__(regnum, make, model, color)
```

```
        self.fasCap = fasCap.

    def set(self):
        setattr(self, input("what you want to change"), input("
                "Enter the value"))

    def get(self):
        print(self.regnum,'', self.make,'', self.model,'', self.color,
                '', self.fasCap)

class commercialVehicle(Vehicle):
    def __init__(self, regnum, make, model, color, loadCap):
        super().__init__(regnum, make, model, color)
        self.loadcap = loadCap

    def set(self):
        setattr(self, input("what you want to change"), input("E-
                -nter the value"))

    def get(self):
        print(self.regnum,'', self.make,'', self.model,'', self.co-
                -lor,'', self.loadcap)
```

Q8) Define classes car, Autorickshaw, and Bus which derive from the PassengerVehicle class mentioned in the previous question. The car and Bus should have attributes for storing information about the number of doors, not shared by Autorickshaw. The Bus should have Boolean attribute doubledecker not shared by car and Autorickshaw, define __init__ method for all these classes. Also, define get and set methods to determine and set the values of the data attributes.

Program :-

```
from A1107 import PassengerVehicle
class car(PassengerVehicle):
    def __init__(self, regnum, make, model, color, fasCap, numdoor):
```

```
        super().__init__(regnum, make, model, color, pascap)
        self.numdoor = numdoor

    def set(self):
        setattr(self, input("what you want to change"), input("E
            -nter the value")).

    def get(self):
        print(self.regnum, ' ', self.make, ' ', self.model, ' ', self.color,
            ' ', self.pascap, ' ', self.numdoor)

class Autorickshaw(PassengerVehicle):
    def __init__(self, regnum, make, model, color, pascap):
        super().__init__(regnum, make, model, color, pascap)

    def set(self):
        setattr(self, input("what you want to change"), input(
            "Enter the value"))

    def get(self):
        print(self.regnum, ' ', self.make, ' ', self.model, ' ', self.color,
            ' ', self.pascap)

class Bus(PassengerVehicle):
    def __init__(self, regnum, make, model, color, pascap, numdoor,
                 dD):
        super().__init__(regnum, make, model, color, pascap)
        self.numdoor = numdoor
        self.dD = dD

    def set(self):
        setattr(self, input("what you want to change"), input("
            Enter the value"))
```

```
def get(self):
    if (self.dD == True):
        print (self.regnum,' ', self.make,' ', self.model,' ',
               self.color,' ', self.pascap,' ', self.numdoor,' ',
               'double decker')
    else:
        print (self.regnum,' ', self.make,' ',self.model,' ', self.
               color,' ', self.pascap,' ', self.numdoor,' ','Not
               Double decker').
```

09) Define a class Account, having attributes account holders name, account number, account type, the amount deposited and minimum deposit amount. Define two classes, namely Savings and current. The Savings class should have a property interest. Define __init__ method for all these classes. Also, define get and set methods to determine & set the value of the data attributes.

```
class Account:
    def __init__(self, name, accnum, type, depo, mindepo):
        self.name = name
        self.accnum = accnum
        self.type = type
        self.depo = depo
        self.mindepo = mindepo.

class Saving(Account):
    def __init__(self, name, accnum, type, depo, mindepo):
        super().__init__(name, accnum, type, depo, mindepo)
    def interest(self, rate):
        total = self.mindepo + self.depo
```

```
        interest = (total * rate * 1)/100
        return interest.
    def set(self):
        setattr(self, input("what you want to change"), input
                ("Enter the value"))

    def get(self):
        print(self.name, self.accnum, self.type, self.depo,
              self.mindepo)
class current(account):
    def __init__(self, name, accnum, type, depo, mindepo):
        super().__init__(name, accnum, type, depo, mindepo)

    def set(self):
        setattr(self, input("what you want to change"),
                input("Enter the value"))

    def get(self):
        print(self.name, self.accnum, self.depo, self.mindepo)
```

Q10) Using Python built-in functions, write the statements to

a) determine whether A is a subclass of B

b) determine whether attribute attr exists in the namespace of object

c) assign the value 70 to attribute attr of object ob of class A.

d) delete an attribute attr of object ob of class A.

Program :-

```
class A:
    attr = 0
class B(A):
    pass
```

```
print (issubclass(A, B))
print (hasattr (A, 'attr'))
A.attr = 70
print (A.attr)
delattr (A, 'attr')
print (hasattr (A, 'attr'))
```

Output :-

```
False
True
70
False
```

Q11) Define a base MyDate and MyTime. Base class MyTime having attributes and methods setHours, setMinutes and setSeconds. Also, define class Appointment derive from the class MyTime and MyDate. The appointment class contains attributes description the Appointment class should have method __str__ which call __str__ function of MyDate, MyTime and description passed as argument. Define __init__ method for all these classes. Also, define get and set methods to retrieve and set the value of the data attributes.

Program :-

```
class MyDate:
  def __init__ (self, day = 1, month = 1, year = 2000):
    if not (type (day) == int and type (month) == int and type (year) == int):
      print ('Invalid data provided for date !')
      sys.exit()
    if month > 0 and month <= 12:
      self.month = month
    else:
      print ('Invalid data for month !')
      sys.exit()
```

```
        if year > 1990:
            self.year = year
        else:
            print ('Invalid data for year and year should be greater
                    than 1990.')
            sys.exit()
        self.day = self.checkday(day)

    def checkday (self, day):
        if self.year % 400 == 0 or (self.year % 100 != 0 and self.year % 4 == 0):
            currentYear = [31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
        else:
            currentYear = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
        if (day > 0 and day <= currentYear[self.month - 1]):
            return day
        else:
            print ('Invalid value for day')
            sys.exit()
    def __str__(self):
        if self.day <= 9:
            day = '0' + str(self.day)
        else:
            day = str(self.day)
        if self.month <= 9:
            month = '0' + str(self.month)
        else:
            month = str(self.month)
        return day + '-' + month + '-' + str(self.year)

class MyTime:
    def __init__ (self, hours = 0, minutes = 0, seconds = 0):
        self.sethours(hours)
        self.setminutes(minutes)
```

```
        self.setseconds(seconds)
    def sethours(self, hours):
        if hours >=0 and hours <=23:
            self.hours = hours
        else:
            print('Invalid value for hours')
            sys.exit()
    def setminutes(self, minutes):
        if minutes >=0 and minutes <= 59:
            self.minutes = minutes
        else:
            print('Invalid value for minutes')
            sys.exit()

    def setseconds(self, seconds):
        if seconds >=0 and seconds <=59
            self.seconds = seconds
        else:
            print('Invalid value for seconds')
            sys.exit()

    def __str__(self):
        if self.hours <=9
            hours = '0' + str(self.hours)
        else:
            hours = str(self.hours)

        if (self.minutes <=9):
            minutes = '0' + str(self.minutes)
        else:
            minutes = str(self.minutes)
```

```
    if self.seconds <= 9:
        seconds = '0' + str(self.seconds)
    else:
        seconds = str(self.seconds)

    return hours + ':' + minutes + ':' + seconds

class Appointment(MyDate, MyTime):
    def __init__(self, day, month, year, hours, minutes, seconds,
                 description):
        MyDate.__init__(self, day, month, year)
        MyTime.__init__(self, hours, minutes, seconds)
        self.description = description

    def __str__(self):
        return MyDate.__str__(self) + ', ' + MyTime.__init__(self)
                                    + '\n' + self.description

print(Appointment(15, 1, 2022, 10, 0, 0, 'meeting regarding
                                          covid-19'))
```

Output :-

15-01-2022, 10:00:00
meeting regarding covid-19.