MALAD KANDIVALI EDUCATION SOCIETY'S
NAGINDAS KHANDWALA COLLEGE OF COMMERCE,
ARTS & MANAGEMENT STUDIES & SHANTABEN NAGINDAS
KHANDWALA COLLEGE OF SCIENCE
MALAD [W], MUMBAI – 64
(AUTONOMOUS)

(Reaccredited 'A' Grade by NAAC)
(AFFILIATED TO UNIVERSITY OF MUMBAI)
(ISO 9001:2015)

CERTIFICATE

Name: Ms. AKSHAT CHUDASAMA

Roll No: 13 **Programmed**: BSC IT **Semester**: II

This is certified to be a bonfire record of practical works done by the above student in the college laboratory for the course **OBJECT ORIENTED PROGRAMMING I** (Course Code:**2022UISPR**) for the partial fulfillment of Second Semester of BSC IT during the academic year 2020-2021.

The journal work is the original study work that has been duly approved in the year 2020-2021 by the undersigned.

_____ _____

External Examiner **Subject-In-Charge**
(Mrs. Niramaye Deshpande)

Date of Examination: (College Stamp)

Name: AKSHAT CHUDASAMA **Roll No: 13**

Sr. No.	DATE	TITLE	SIGN
1.	15/01/2021	Creating Class diagram with the class and their attributes and methods	
2.	22/01/2021	Defining and using a class.	
3.	29/01/2021	Defining methods with and without attributes in a class.	
4.	05/02/2021	Creating and using constructor and Destructor	
5.	12/02/2021	Using property getters and setters.	
6.	19/02/2021	Implementing various forms of Inheritance.	
7.	26/02/2021	Implementing Polymorphism by Overloading Overriding methods.	
8.	12/03/2021	Implementing concept of Operator Overloading.	
9.	26/03/2021	Implementing abstract classes and interfaces.	
10.	09/04/2021	Implementing concept of Composition.	

PRACTICAL 1

Creating Class Diagrams With Abstract Classes, Subclasses, Their Attributes, And Methods.

WRITE UP:

CLASS:

In OOP, a class is an extensible program-code-template for creating objects, providing initial values for states (member variables) and implementations of behavior (member functions or methods)

SYNTAX:

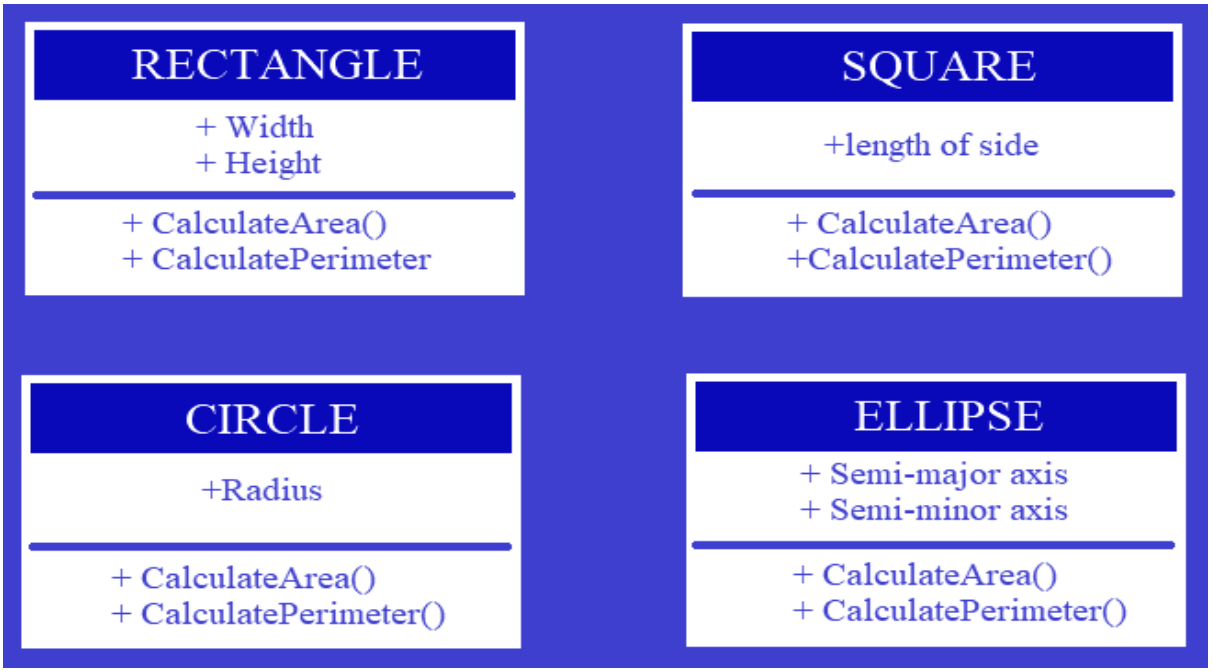
```
Class ClassName
{ Access specifier:          //can be private, public or protected
Data members;                //Variables to be used
Member Functions() {}        //Methods to access data members
};                           //Class name ends with a semicolon
```

OBJECT:

An object, in OOP, is an abstract data type created by a developer. It can include multiple properties and methods and may even contain other objects. In most programming languages, objects are defined as classes. Objects provides a structured approach to programming.

a. Aim: Draw a class diagram for calculating AREA and PERIMETER of Rectangle, Square, Circle, and Ellipse and write a program in Python for the same.

CLASS DIAGRAM



Code for area and perimeter of rectangle:

```
class Rectangle:
    def calculateArea(self):
        print("enter length: ")
        self.a = float(input())
        print("enter breadth: ")
        self.b = float(input())
        area = self.a * self.b
        print("area of rectangle is = %f
 "%(area))
    def calculatePerimeter(self):
        perimeter = 2 * (self.a * self.b)
        print("perimeter of rectangle is =
 %f" %(perimeter))

c = Rectangle()
x = c.calculateArea()
y = c.calculatePerimeter()
```

Output:

```
enter length:
20
enter breadth:
20
area of rectangle is = 400.000000
perimeter of rectangle is = 800.000000
```

Code for area and perimeter of Square:

```
class Square:
    def calculateArea(self):
        print("enter side : ")
        self.s=float(input())
        area=self.s*self.s
        print("area of square is =
 %f"%(area))
```

```
    def calculatePerimeter(self):
        perimeter=4*self.s
        print("perimeter of square is =
 %f"%(perimeter))
c=Square()
c.calculateArea()
c.calculatePerimeter()
```

Output:

```
enter side :
4
area of square is = 16.000000
perimeter of square is = 16.000000
```

Code for area and perimeter of Circle:

```
class circle:
    def calculateArea(self):
        print("enter radius: ")
        self.r=float(input())
        area=3.14*self.r*self.r
        print("area of circle is =
 %f"%(area))
    def calculatePerimeter(self):
        perimeter=2*3.14*self.r
        print("perimeter of circle is =
 %f"%(perimeter))
c= circle()
c.calculateArea()
c.calculatePerimeter()
```

Output:

```
enter radius:
5
area of circle is = 78.500000
perimeter of circle is = 31.400000
```

Code for area and perimeter of Ellipse:

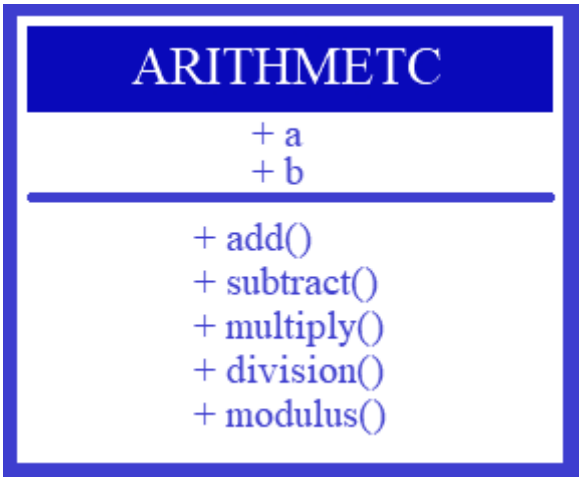
```
import math
class ellipse:
    def calculateArea(self):
        print("enter axis1: ")
        self.a=float(input())
        print("enter axis2: ")
        self.b=float(input())
        area=3.14*self.a*self.b
        print("area of ellipse is =
 %f"%(area))
    def calculatePerimeter(self):

 perimeter=2*3.14*math.sqrt((self.a*self.a+se
 lf.b*self.b)/(2))
        print("perimeter of ellipse is =
 %f"%(perimeter))
c= ellipse()
c.calculateArea()
c.calculatePerimeter()
```

Output:

```
enter axis1:
12
enter axis2:
13
area of ellipse is = 489.840000
perimeter of ellipse is = 78.562775
```

b. Aim: Draw a class diagram for performing all arithmetic operations and write a program in python that will define an arithmetic class and required methods for doing all arithmetic operations and display the result.

CLASS DIAGRAM



Code for all the arithmetic operators:

```
class Arithmetic:
    def accept(self):
        print("enter two numbers")
        self.a=int(input())
        self.b=int(input())
    def sum(self):
        x=self.a+self.b
        print("sum is",x)
    def sub(self):
        x=self.a-self.b
        print("subtraction is",x)
    def multi(self):
        x=self.a*self.b
        print("multiplication is",x)
    def div(self):
        x=self.a/self.b
        print("division is",x)
    def join(self):
        q=str(self.a)+str(self.b)
        print("concatination is",q)
```

```
a1 = Arithmetic()
a1.accept()
a1.sum()
a1.sub()
a1.multi()
a1.div()
a1.join()
```

Output:

```
enter two numbers
1
2
sum is 3
subtraction is -1
multiplication is 2
division is 0.5
concatination is 12
```

PRACTICAL 2

Defining and using a class

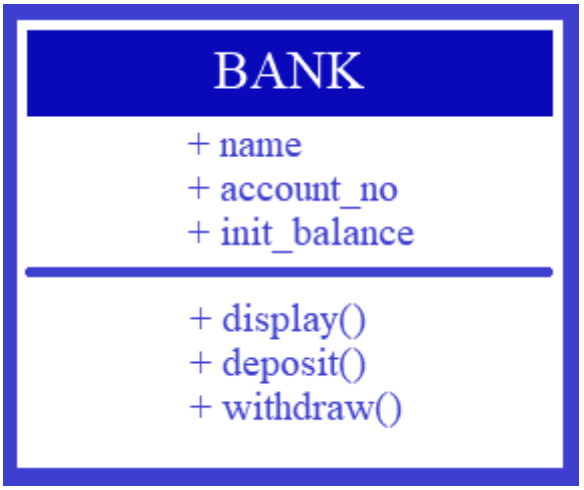
WRITE UP:

CLASS: Describes an object type by defining what data is stored in the object and what actions can the object perform. It also serves as a factory for generating instances.

SELF : The self is used to represent the instance of the class. With this keyword, you can access the attributes and methods of the class in python.

a. Aim: Draw a class diagram and define a class in Python for creating a bank account and showing basic transactions such as Deposit and Withdraw of money on a specific account and show updated balance along with customer details.

CLASS DIAGRAM



Code for bank account:

```
class bank:
    customer_name = str(input("Enter your
Name:"))
    account_no = int(input("Enter you Account
number: " ))
    balance = int(input("Enter you balance :
"))

    def deposit(self):
```

```
        deposit_amount = int(input("Enter the
amount to be deposit :"))
        self.balance = self.balance +
deposit_amount
        print("Current balance
is:",self.balance)

    def withdraw(self):
        withdraw_amount = int(input("Enter the
amount to Withdraw :"))
        self.balance = self.balance -
withdraw_amount
        print("Current balance
is:",self.balance)

c=bank()
a = int(input("Enter 1 if you want to Deposit
and 2 if you want to Withdraw : "))
if(a==1):
    c.deposit()
elif(a==2):
    c.withdraw()
```

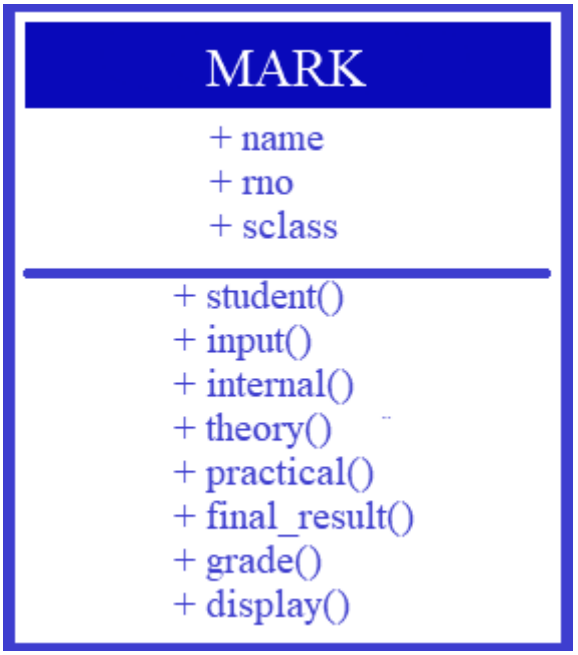
Output:

```
Enter your Name: pandu
Enter you Account number: 234545099999
Enter you balance : 2000000
Enter 1 if you want to Deposit and 2 if you want to Withdraw : 2
Enter the amount to Withdraw :30000
Current balance is: 1970000
```

b.

B : Aim: Draw a class diagram and define a class in python to accept student information such as roll no, name, class, and marks of five subjects. Calculate total and percentage and display the complete information.

CLASS DIAGRAM



Code for student information:

```
class marks:
    student_name = str(input("Enter your Name: "))
    student_rno = int(input("Enter your Roll no: "))
    sclass = input("Enter your Class: ")

    def subjects(self):
        self.dbms = int(input("Enter yout marks in DBMS: "))
        self.pp = int(input("Enter yout marks in PP: "))
        self.wp = int(input("Enter yout marks in WP: "))
```

```
        self.oop = int(input("Enter yout marks
in OOP: "))
        self.dm = int(input("Enter yout marks
in DM: "))

    def internal_mrks(self):
        print("Enter your marks out of 40 for
internals: ")
        self.subjects()
        self.int_mrks = self.dbms + self.pp +
self.wp + self.oop + self.dm

    def theory_mrks(self):
        print("Enter your marks out of 60 for
theory: ")
        self.subjects()
        self.th_mrks = self.dbms + self.pp +
self.wp + self.oop + self.dm

    def practicals(self):
        print("Enter your marks out of 50 for
practicals: ")
        self.subjects()
        self.pr_mrks = self.dbms + self.pp +
self.wp + self.oop + self.dm
        self.mrks = self.int_mrks =
self.th_mrks + self.pr_mrks
        self.total = (40 * 5) + (60 * 5) + (50
* 5)
```

```
        self.percent = (self.mrks /
self.total) * 100
        self.percentage = round(self.percent)

    def grade(self):
        if self.percentage > 90:
            print("You Pass!")
            print("Your Percentage is: ",
self.percentage)
            print("O GRADE!")
        elif self.percentage > 75:
            print("You Pass!")
            print("Your Percentage is: ",
self.percentage)
            print("A+ GRADE!")
        elif self.percentage > 60:
            print("You Pass!")
            print("Your Percentage is: ",
self.percentage)
            print("A GRADE!")
        elif self.percentage > 50:
            print("You Pass!")
            print("Your Percentage is: ",
self.percentage)
            print("B+ GRADE!")
        elif self.percentage > 45:
            print("You Pass!")
            print("Your Percentage is: ",
self.percentage)
            print("B GRADE!")
```

```
        else:
            print("You Fail!")
            print("Your Percentage is: ",
self.percentage)
    def display(self):
        print(self.student_name)
        print(self.sclass)
        print(self.student_rno)
c = marks()
c.internal_mrks()
c.theory_mrks()
c.practicals()
c.display()
c.grade()
```

Output:

```
Enter your Name: surabhi
Enter your Roll no: 50
Enter your Class: fycs
Enter your marks out of 40 for internals:
Enter yout marks in DBMS: 40
Enter yout marks in PP: 40
Enter yout marks in WP: 40
Enter yout marks in OOP: 40
Enter yout marks in DM: 40
Enter your marks out of 60 for theory:
Enter yout marks in DBMS: 59
Enter yout marks in PP: 58
Enter yout marks in WP: 59
Enter yout marks in OOP: 60
Enter yout marks in DM: 60
Enter your marks out of 50 for practicals:
Enter yout marks in DBMS: 49
Enter yout marks in PP: 50
Enter yout marks in WP: 50
Enter yout marks in OOP: 50
Enter yout marks in DM: 50
surabhi
fycs
50
You Pass!
Your Percentage is:  73
A GRADE!
```

PRACTICAL 3

Defining Methods With And Without Attributes In A Class.

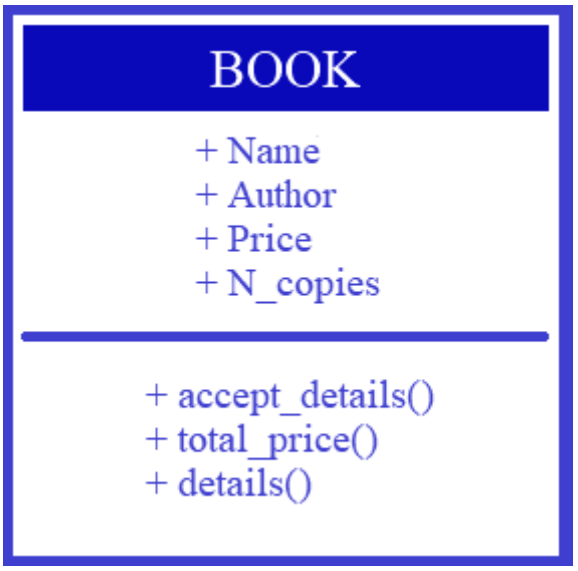
WRITE UP

ATTRIBUTES:

Class attributes are attributes which are owned by the class itself. They will be shared by all the instances of the class. Therefore they have the same value for every instance. We define class attributes outside all the methods, usually they are placed at the top, right below the class header.

a. Aim: Draw class diagram and define class book to accept name of book, author, price and number of copies to be purchased using method without parameter. Calculate total price and display all details.

CLASS DIAGRAM



Code:

```
class Book:
    def accept(self):
        self.name = str(input("Enter Name of Book: "))
        self.author = str(input("Enter Author of Book: "))
        self.price = float(input("Enter Price of Book: "))
        self.n_copies = int(input("Enter Name of Copies to be purchased of Book: "))
    def total_price(self):
```

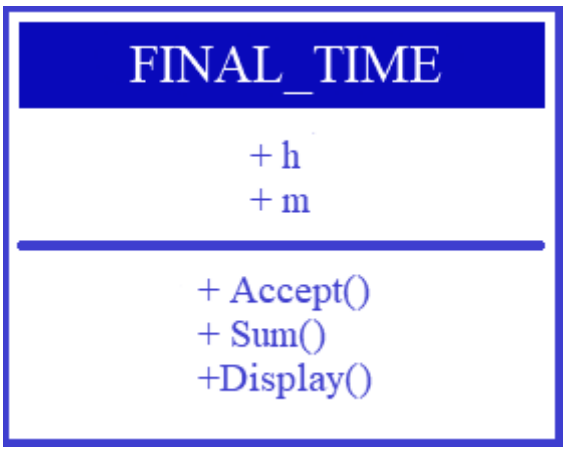
```
        t = self.price * self.n_copies
        print("Total Price: ",t)
    def details(self):
        print("Purchase Details".center(70,
"-"))
        print("Name of Book: ",self.name)
        print("Author of Book: ",self.author)
        print("Price of Book: ",self.price)
        print("Number of Copies:
",self.n_copies)
c = Book()
c.accept()
c.details()
c.total_price()
```

Output:

```
Enter Name of Book: memory
Enter Author of Book: william walker atkinson
Enter Price of Book: 150
Enter Name of Copies to be purchased of Book: 2
-------------------------Purchase Details-------------------------
Name of Book:  memory
Author of Book:  william walker atkinson
Price of Book:  150.0
Number of Copies:  2
Total Price:  300.0
```

b. Aim: Draw class diagram and define class time to accept time in hours and minutes in two different instances. Pass these instances as an argument to method to add time in hours and minutes and display total time.

CLASS DIAGRAM



Code:

```
class Time:
    h = 0;
    m = 0;

    def accept(self):
        print("Enter time in hours and mins");
        self.h = int(input())
        self.m = int(input())

    def display(self):
        print(self.h, "hours and", self.m,
"minutes")

    def sum(self,t1,t2):
        self.m = t1.m + t2.m
        self.h = self.m / 60
        self.m = self.m % 60
        self.h = self.h + t1.h + t2.h
t1_obj = Time()
t1_obj.accept()
t2_obj = Time()
```

```
t2_obj.accept()
t3 = Time()
t3.sum(t1_obj , t2_obj)
print("t1_obj= ")
t1_obj.display()
print("t2_obj= ")
t2_obj.display()
print("t3= ")
t3.display()
```

Output:

```
Enter time in hours and mins
1
30
Enter time in hours and mins
2
35
t1_obj=
1 hours and 30 minutes
t2_obj=
2 hours and 35 minutes
t3=
4.083333333333333 hours and 5 minutes
```

PRACTICAL 4

Creating and using Constructor and Destructor

WRITE UP:

CONSTRUCTOR:

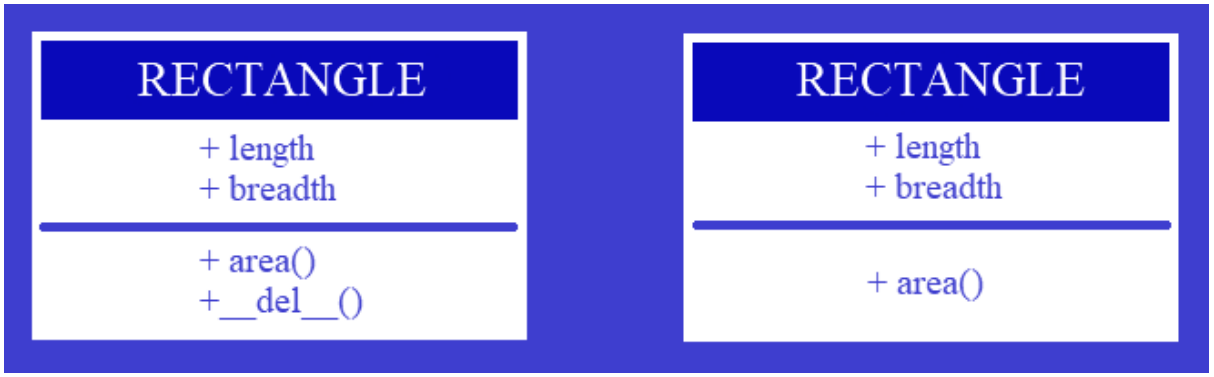
The constructor is a method that is called when an object is created. This method is defined in the class and can be used to initialize basic variables. If you create four objects, the class constructor is called four times. Every class has a constructor, but it's not required to explicitly define it.

DESRUCTOR:

Destructors are called when an object gets destroyed. In Python, destructors are not needed as much needed in C++ because Python has a garbage collector that handles memory management automatically. The __del__() method is a known as a destructor method in Python. It is called when all references to the object have been deleted i.e. when an object is garbage collected

a. Aim: Draw class diagram and write a program to calculate the area of the rectangle using constructor and destructor

CLASS DIAGRAM



Code:

```
class rectangle:
    def __init__(self, length, breadth):
        print("I'm initialising the
rectangle class")
        self.length = length
        self.breadth = breadth
    def area(self):
        self.area = self.length *
self.breadth
        print("Area is:", self.area)

    def _del_(self):
        print("Instance destroyed")
c = rectangle(15,20)
```

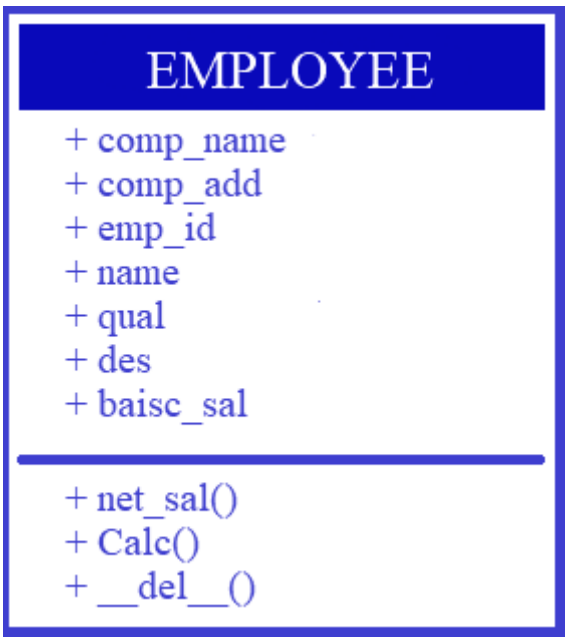
```
c.area()
c1 = rectangle(10,15)
c1.area()
c1._del_()
```

Output:

```
I'm initialising the rectangle class
Area is: 300
I'm initialising the rectangle class
Area is: 150
Instance destroyed
```

b. Aim: Define Class Employee to accept company name, address, employee ID, qualification, designation and basic salary, Calculate net salary using the following information for allowance and deduction as travelling allowance is 80% of basic salary, DA is 70% of basic salary, HRA is 25% of basic salary and employee provident fund(EPF) IS 10% of basic salary calculate net salary using the formula NET SALARY = BASIC SALARY+ TRAVELLING ALLOWANCE+ DA+ HRA-EPF. Initialize employee object using constructor and destroy it once it is of no use using destructor.

CLASS DIAGRAM



Code:

```
class Employee:
    def _init_(self):
        self.comp_name = str(input("Enter
Company Name: "))
        self.comp_add = str(input("Enter Your
Company Address: "))
        self.emp_id = int(input("Enter Your
Empoyee Id: "))
        self.name = str(input("Enter your
Name: "))
        self.qual = str(input("Enter Your
Qualification: "))
        self.des = str(input("Enter Your
Designation: "))
        self.basic_sal = int(input("Enter Your
Basic Salary: "))

    def calc(self):
        TA = 18/100 * self.basic_sal
```

```
        DA = 70/100 * self.basic_sal
        HRA = 25/100 * self.basic_sal
        EPF = 10/100 * self.basic_sal
        net_sal = self.basic_sal + TA + DA +
HRA - EPF
        print("Net Salary is: ",net_sal)

    def _del_(self):
        print("Instance Destroyed")

c = Employee()
c.calc()
c._del_()
```

Output:

```
Enter Company Name: apsara pencils
Enter Your Company Address: 510,Himalaya House, Palton road, Mumbai, Maharashtra 400001
Enter Your Empoyee Id: 12345
Enter your Name: pandu
Enter Your Qualification: graduate
Enter Your Designation: specialist
Enter Your Basic Salary: 200000
Net Salary is:  406000.0
Instance Destroyed
```

PRACTICAL 5

Using Property Getters And Setters

WRITE UP:

GETTER:

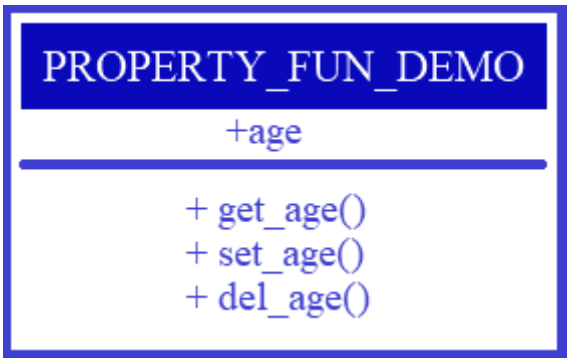
These are the methods used in Object-Oriented Programming (OOPS) which helps to access the private attributes from a class.

SETTER:

These are the methods used in OOPS feature which helps to set the value to private attributes in a class.

a. Aim: Draw Class Diagram and Write a Program to Implement Getters and Setters Property.

CLASS DIAGRAM



Code:

```
# Python Program Showing Use of Property()
Function

class property_fun_demo:
    # def _init_(self):
    # self._age = 0

    # Function to get value of _age.
    def get_age(self):
        print("Getter method called. ")
        return self._age

    # Function to set value of _age.
    def set_age(self,a):
        print("Setter method called. ")
        self._age = a

    # Function to delete _age attribute.
    def del_age(self):
        print("Delete method called. ")
        del self._age

    age = property(get_age, set_age, del_age)
```

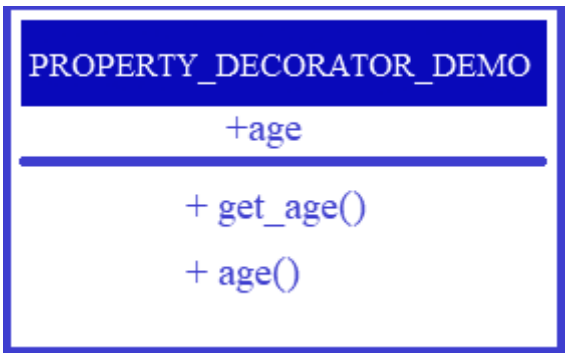
```
mark = property_fun_demo()
mark.age = 15
print(mark.age)
```

Output:

```
Setter method called.
Getter method called.
15
```

b. Aim: Write the program to demonstrate the use of the getter and setter method using a property decorator.

CLASS DIAGRAM



Code:

```
# To demonstrate the use of getter and setter
method using property decorator.
```

```
class property_decorator_demo:

    # To show use of property decorator.
    def _init_(self):
        self._age = 0

    @property
    def get_age(self):
        print("Getter method called. ")
        return self._age

    @get_age.setter
    def age(self,a):
        if(a < 18):
            raise ValueError("Sorry your age
is below eligibility criteria, ")
        print("Setter method called. ")
        self._age = a

# Creating a object of class.
mark = property_decorator_demo()

# Calling method through object.
mark.age = 23
print(mark.age)
```

Output:

```
Setter method called.
Getter method called.
18
```

PRACTICAL 6

Implementing Various Forms Of Inheritance

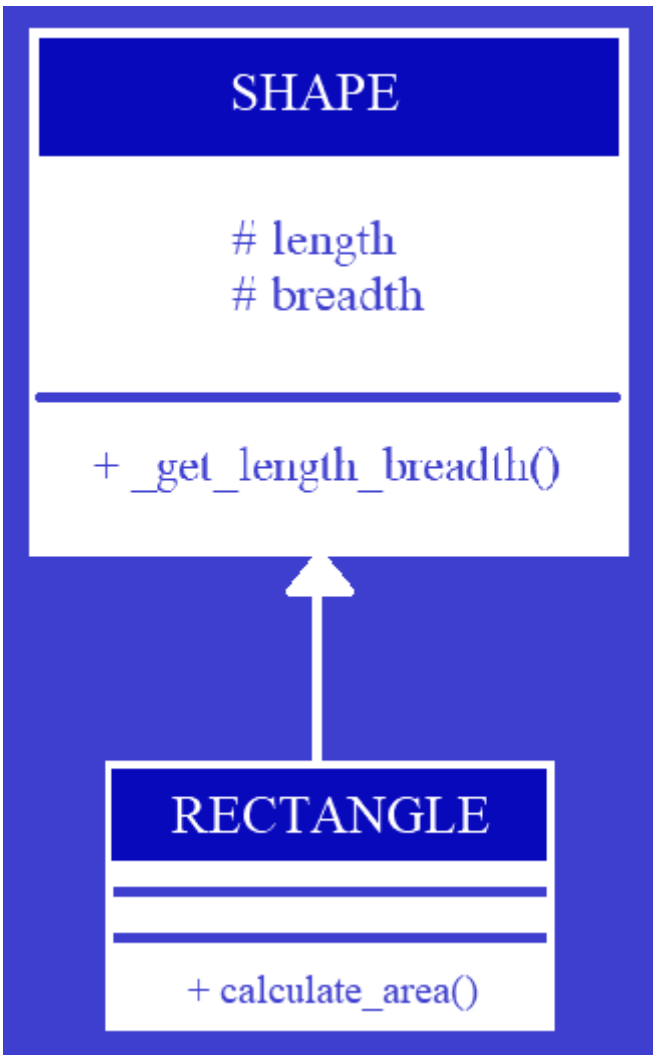
WRITEUP

Inheritance: is a process of obtaining properties and characteristics(variables and methods) of another class. In this hierarchical order, the class which inherits another class is called subclass or child class, and the other class is the parent class

MULTILEVEL INHERITANCE : in multilevel inheritance, the transfer of the properties of characteristics is done to more than one class hierarchically. To get a better visualization we can consider it as an ancestor to grandchildren relation or a root to leaf in a tree with more than one level

a. Aim: Draw class diagram and write a program to implement Single inheritance.

CLASS DIAGRAM



Code:

```
# Defining Base class 'Shape'
class Shape:

    #Function to initialize the Data Members
    def _get_Length_Breadth(self):
        self._length = int(input("Enter
Length: "))
        self._breadth = int(input("Enter
Breadth: "))

# Defining Derived class 'Rectangle'
class Rectangle(Shape):
```

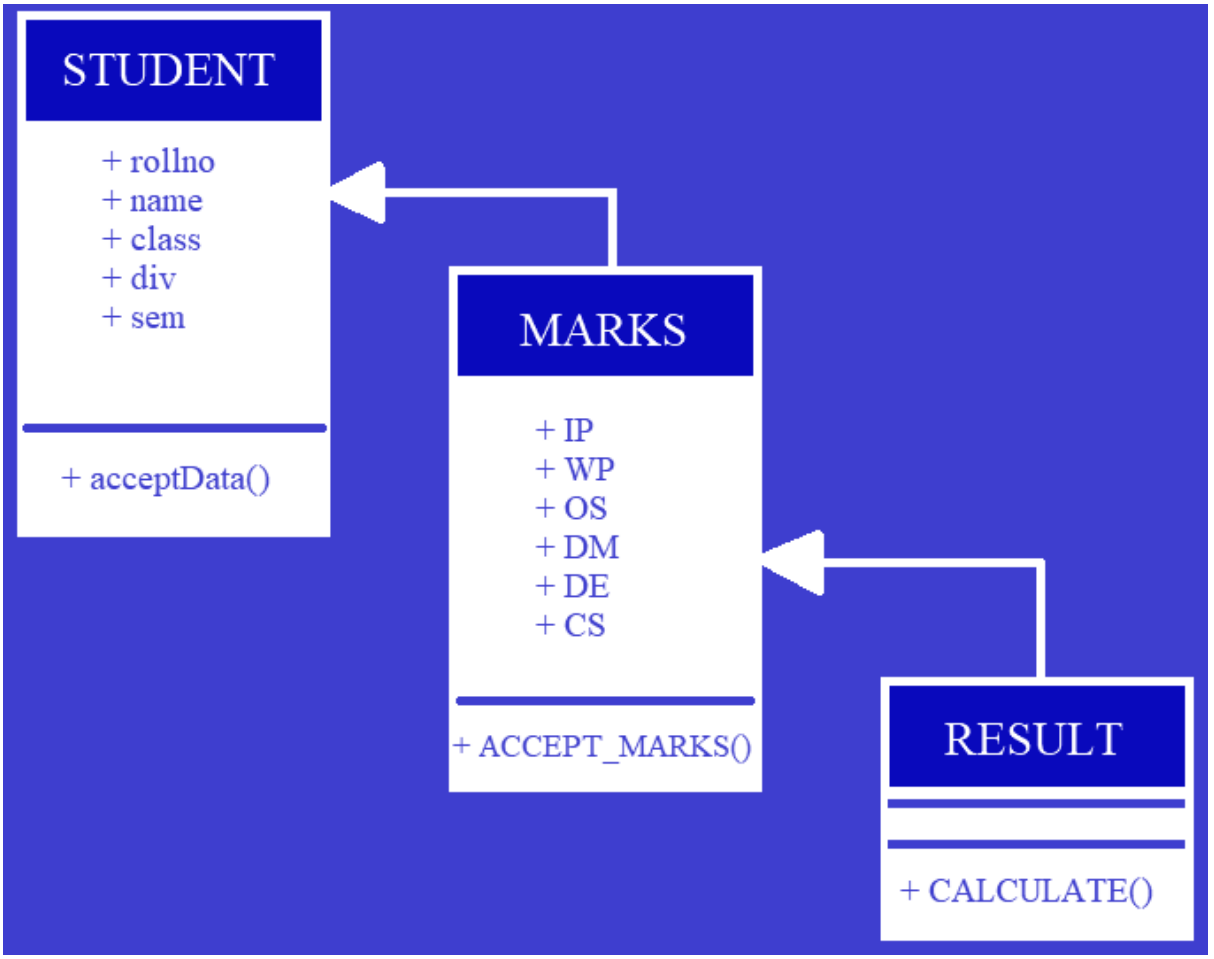
```
    # Function to calculate area of rectangle
    def calculate_area(self):
        print("Area of Rectangle is ",
self._length * self._breadth)

# Creating Object
obj = Rectangle()

# Calling Function to get input
obj._get_Length_Breadth()

# Calling Function to get the Area of
Rectangle
obj.calculate_area()
```

Output:

```
Enter Length: 30
Enter Breadth: 40
Area of Rectangle is  1200
```

b. Aim: Draw class diagram and write a program to implement multilevel inheritance and make a class diagram for it.

CLASS DIAGRAM



Code:

```
#program multilevel inherit
class Student:

    def acceptData(self):
        self.rollno = int(input("rolll no: "))
        self.name = str(input("/name: "))
        self.class_ = str(input("class: "))
        self.div = str(input("Division: "))
        self.sem = input("Semester: ")

class Marks(Student):

    def accept_marks(self):
```

```
        self.ip = int(input("imperative
programming: "))
        self.wp = int(input("web prog: "))
        self.de = int(input("digital
electronic: "))
        self.os = int(input("operating system:
"))
        self.dm = int(input("discrete maths:
"))
        self.cs = int(input("communication
skills:"))

class Result(Marks):

    def calculate(self):
         self.total = (self.ip +
                       self.wp +
                       self.de +
                       self.os +
                       self.dm +
                       self.cs)
         self.avg = self.total / 6
         self.percentage = (self.total *
100)/600
         print("\ntotal MArks: ", self.total)
         print("avg marks: ", self.avg)
         print("percentage: ",
self.percentage)
c = Result()
c.acceptData()
```

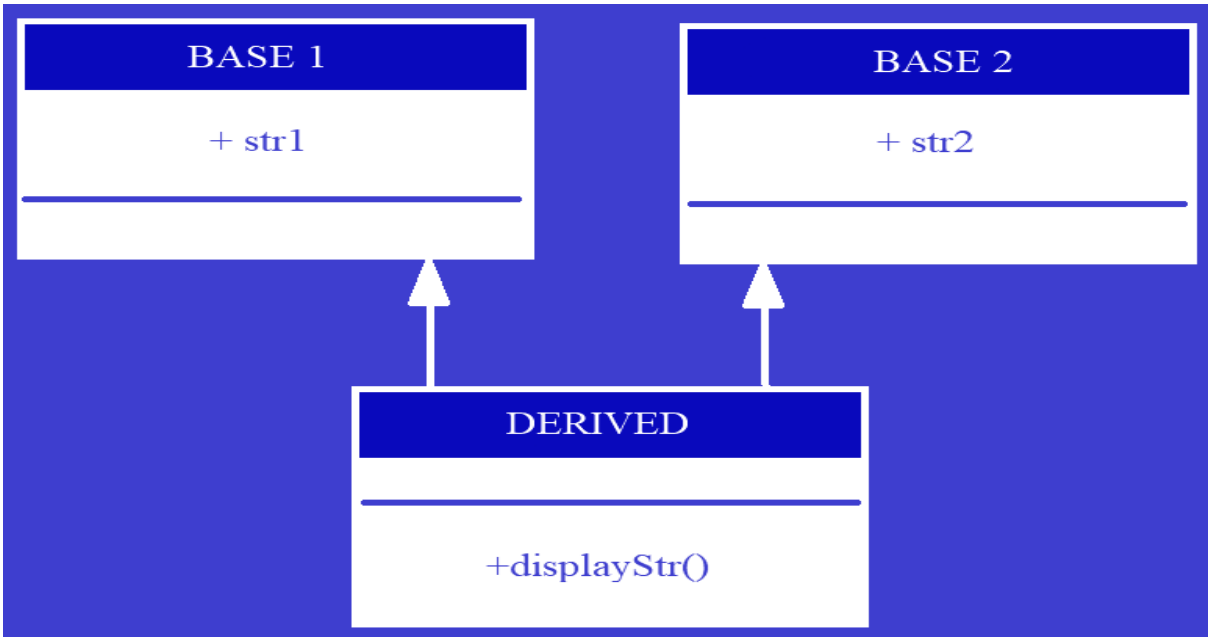
```
c.accept_marks()
c.calculate()
```

Output:

```
rolll no: 20
/name: pandu
class: fycs
Division: a
Semester: 2
imperative progrmming: 50
web prog: 50
digital electronic: 50
operating system: 50
discrete maths: 50
communication skills:50

total MArks:  300
avg marks:  50.0
percentage:  50.0
```

c. Aim: Draw class diagram Write a program to implement multiple inheritance.

CLASS DIAGRAM



Code:

```
# Program to demonstrate Multiple Inheritance


class Base1(object):
    def __init__(self):
```

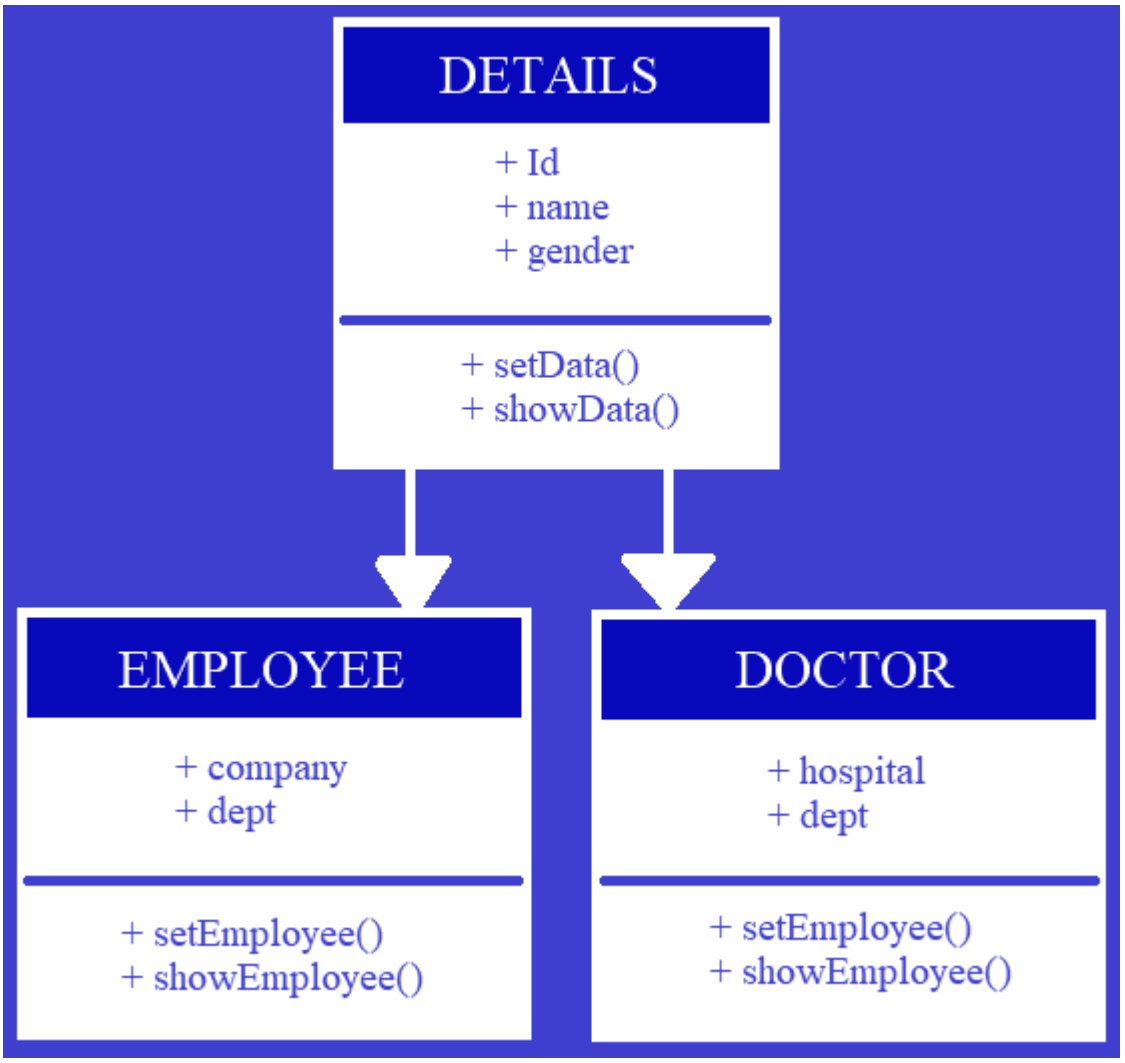
```
        print("base1: ")
        self.str1 = "heyy wassup"
        print(self.str1)

class Base2(object):
    def __init__(self):
        print("base2: ")
        self.str2 = "all good?"
        print(self.str2)


class Derived(Base1, Base2):
    def __init__(self):
        Base1.__init__(self)
        Base2.__init__(self)
        print("Derived: ")

    def displayStr(self):
         print(self.str1, self.str2)

c = Derived()
c.displayStr()
```

Output:

```
base1:
elon
base2:
musk
Derived:
elon musk
```

d. Aim: Draw class diagram and write a program to implement hierarchical inheritance.

CLASS DIAGRAM



Code:

```
#py code to demonstrate eg of
#hierarchial inheritance
class Details:
    def __init__(self):
        self.__id="<No Id>"
        self.__name="<No Name>"
        self.__gender="<No Gender>"
    def setData(self,id,name,gender):
        self.__id=id
        self.__name=name
        self.__gender=gender
```

```
    def showData(self):
        print("Id: ",self.__id)
        print("Name: ",self.__name)
        print("Gender: ",self.__gender)

class Employee(Details):  #inheritance
    def __init__(self):
        self.__company="<No comapany>"
        self.__dept="<No Dept>"
    def
setEmployee(self,id,name,gender,comp,dept):
        self.setData(id,name,gender)
        self.__company=comp
        self.__dept=dept
    def showEmployee(self):
        self.showData()
        print("Company: ",self.__company)
        print("Dept :",self.__dept)

class Doctor(Details): #inheritnce
    def __init__(self):
        self.__hospital="<No Hospital>"
        self.__dept="<No dept>"
    def
setEmployee(self,id,name,gender,hos,dept):
        self.setData(id,name,gender)
        self.__hospital=hos
        self.__dept=dept
    def showEmployee(self):
        self.showData()
```

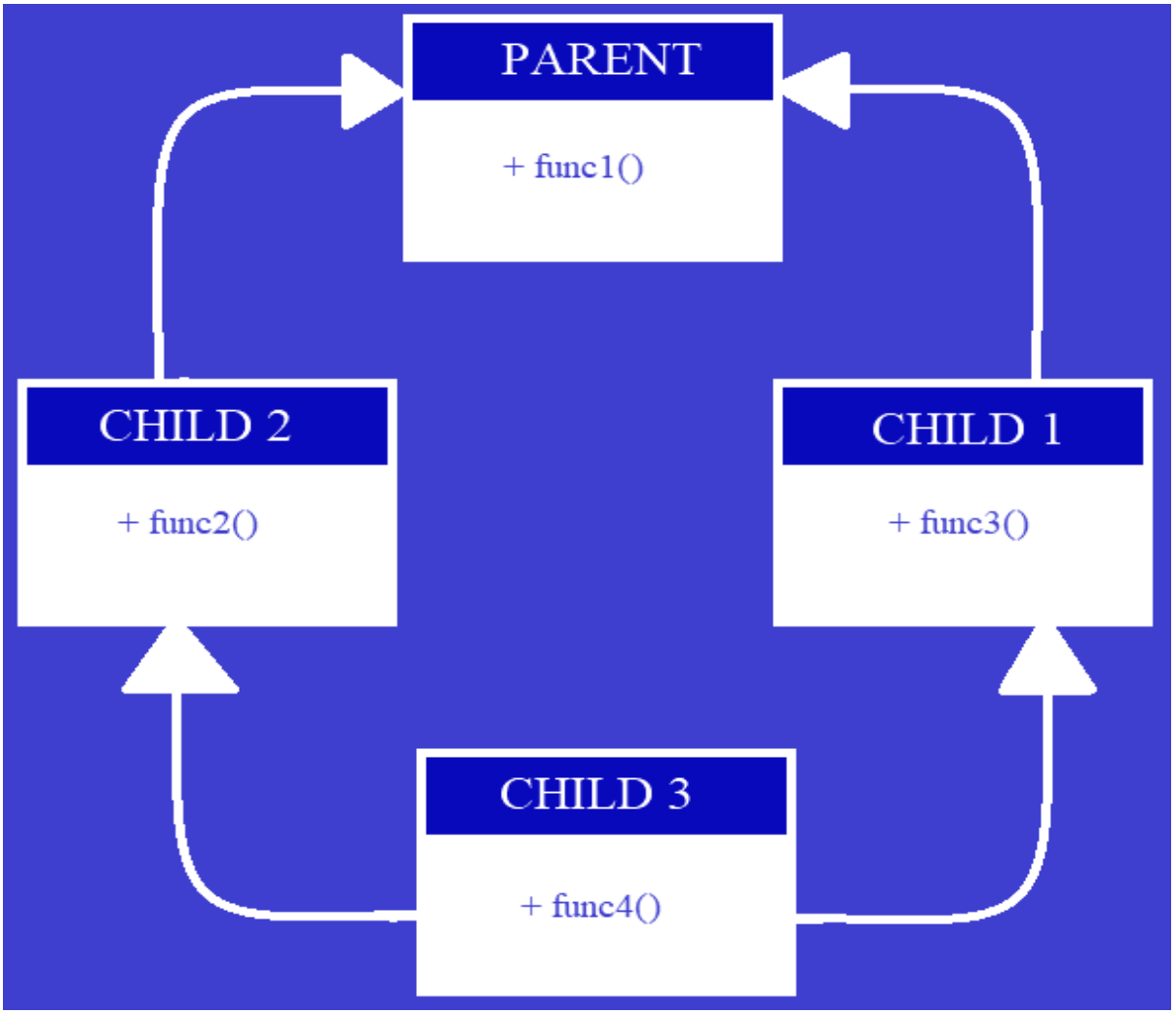
```
        print("Hospital: ",self.__hospital)
        print("Department: ",self.__dept)
def main():
    print("Employee Object")
    e=Employee()

e.setEmployee(1,"ashu","Male","gmr","excavalat
ion")
    e.showEmployee()
    print("\nDoctor Object")
    d = Doctor()

d.setEmployee(2,"tahir","male","alime","it")
    d.showEmployee()

if __name__ =="__main__":
    main()
```

Output:

```
Id:  1
Name:  ashu
Gender:  male
Company:  gmr
Dept : cs

Doctor Object
Id:  2
Name:  tahir
Gender:  male
Hospital:  alime
Department:  it
```

e. Aim: Write a program to implement hybrid inheritance and make a class diagram for it.

CLASS DIAGRAM



Code:

```
#to illustate protected access modifier in a
cls and initializing
#base clss using instance mthd


#super class
#class Protecteddemo:
   # prodata membrs
#    _name = None
 #   _roll = None
  # _branch = None
```

```
 #  def getdata(self, name, roll, branch):
  #            self._name = name
   #           self._name = roll
    #          self._name = branch

 #derivd clss
#class Student(Protecteddemo):

     #
 #    def displayDetails(self):

class Parent:
    def func1(self):
        print("This is Function One.")

class Child(Parent):
    def func2(self):
        print("This is Function Two.")

class Child1(Parent):
    def func3(self):
        print("This is Function Three.")

class Child3(Child, Child1):
    def func4(self):
        print("This is Function Four.")

ob = Child3()
ob.func1()
ob.func2()
```

```
ob.func3()
ob.func4()
```

Output:

```
This is Function One.
This is Function Two.
This is Function Three.
This is Function Four.
```

PRACTICAL 7

Implementing Polymorphism By Overloading Overriding Methods

WRITE UP:

POLYMORPHISM:

Polymorphism means the ability to take various forms. In Python, Polymorphism allows us to define methods in the child class with the same name as defined in their parent class. Two methods cannot have the same name in Python. Method overloading in Python is a feature that allows the same operator to have different meanings.

OVERLOADING:

Overloading is the ability of a function or an operator to behave in different ways based on the parameters that are passed to the function, or the operands that the operator acts on. Method Overloading in Python In Python, you can create a method that can be called in different ways. So, you 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.

a. Aim: Write a program to implement method overloading.

Code:

```
class human:
    def saHello(self,name=None):
        if name is not None:
```

```
            print("keep smiling"+name)
        else:
            print("i hope everything is going
good")
obj=human()
obj.saHello()
obj.saHello(' and doing well')
```

Output:

```
i hope everything is going good
keep smiling and doing well
```

b. Aim: Write a program to implement method overriding

Code:

```
#Method overriding

class Employee:

    def add(self, salary, incentive):
        print('Total salary in base class = ',
salary + incentive)

class Department(Employee):
    temp = "I am member of department class. "
    def add(self, salary, incentive):
        print(self.temp)
        print('Total salary in derived class =
', salary + incentive)

dept = Department()
dept.add(45000, 5000)
```

Output:

```
I am member of department class.
Total salary in derived class =  50000
```

c. Aim: Write a program to implement method overriding using super.

Code:

```
#Method overriding using(super)

class Employee:

    def add(self, salary, incentive):
        print('Total salary in base class = ',
salary + incentive)

class Department(Employee):
    temp = "I am member of department class. "
    def add(self, salary, incentive):
        print(self.temp)
        print('Total salary in derived class =
', salary + incentive)
        super(Department, self).add(salary,
incentive)

dept = Department()
dept.add(45000, 5000)
```

Output:

```
I am member of department class.
Total salary in derived class =  50000
Total salary in base class =  50000
```

d. Aim: Write a program to implement method overriding using multiple classes

Code:

```
import math

class polygons:
    def number_of_side(self):
        return 0
    def area(self):
        return 0
    def perimeter(self):
        return 0
class triangle(polygons):
    def number_of_side(self):
        return 3
    def area(self):
        base=10
        height=12
        return 1/2*base*height
    def perimeter(self):
        a=10
        b=10
        c=12
        if a+b>c:
            return a+b+c
        else:
            return "invalid input: make sure
a+b>c"

class rhombus(polygons):
    def number_of_side(self):
```

```
        return 4
    def area(self):
        p=4
        q=5
        return p*q/2
    def perimeter(self):
        a=5
        return 4*a

tri = triangle()
print("triangle area: ",tri.area())
print("triangle perimeter: ",tri.perimeter())
print("-----------------------")


rho = rhombus()
print("rhombus area: ",tri.area())
print("rhombus perimeter: ",tri.perimeter())
```

Output:

```
triangle area:  60.0
triangle perimeter:  32
----------------------
rhombus area:  60.0
rhombus perimeter:  32
```

PRACTICAL 8

Implementing The Concept Of Operator Overloading

WRITE UP

Operator Overloading:

The operator overloading assigns new functionality to existing operators so that they do what you want. Operator overloading lets objects coded with classes intercept and respond to operations that work on built-in types: addition, subtraction, multiplication, slicing, comparison and so on. Using this special method, you will be able to change the built-in behavior of the operator such as: +, -, /, or *. This special method is surrounded by double underscores (__).

a. Aim: Write a program to demonstrate the use of different types of built in methods used in operator overloading.

Code:

```
x,y=2,4
print("x = " ,x ,", y = ",y)

print("\n x + y = ",x+y)
print("x.__add__(y) = ",x.__add__(y))

print("\n x*y =",x*y)
print("x.__mul__(y) = ",x.__mul__(y))

print("\n x / y = ",x/y)
print("x.__truediv__(y) = ",x.__truediv__(y))

print("\n x ** y = ",x**y)
print("x.__pow__(y) = ",x.__pow__(y))

print("\n x % y = ",x%y)
```

```
print("x.__mod__(y) = ",x.__mod__(y))

print("\n x == y = ",x==y)
print("x.__eq__(y) = ",x.__eq__(y))

print("\n x != y = ",x!=y)
print("x.__ne__(y) = ",x.__ne__(y))

print("\n x >= y = ",x>=y)
print("x.__ge__(y) = ",x.__ge__(y))

print("\n x <= y = ",x<=y)
print("x.__le__(y) = ",x.__le__(y))

print("-----------------------")
```

Output:

```
C:\Users\surab\Desktop\STUDIES\SEM 2\Object Oriented Programming>C:
ractical 8/8A.py"
x =  2 , y =  4

 x + y =  6
x.__add__(y) =  6

 x*y = 8
x.__mul__(y) =  8

 x / y =  0.5
x.__truediv__(y) =  0.5

 x ** y =  16
x.__pow__(y) =  16

 x % y =  2
x.__mod__(y) =  2

 x == y =  False
x.__eq__(y) =  False

 x != y =  True
x.__ne__(y) =  True

 x >= y =  False
x.__ge__(y) =  False

 x <= y =  True
x.__le__(y) =  True
-----------------------
```

b. Aim: Write a program to overload Binary Plus operator

Code:

```
#python program illustrate how
#to overload an binary+oparator

class Add:
    def __init__(self,a):
        self.a = a
    def __add__(self,o):
        return self.a + o.a

obj1=Add(1)
obj2=Add(2)
obj3=Add("FY")
obj4=Add("CS")
print(obj1+obj2)
print(obj3+obj4)
```

Output:

```
3
FYCS
```

c. Aim: Write a program to overload binary plus to add two complex numbers.

Code:

```
# Python Program to perform addition
# of two complex numbers using binary
# + operator overloading.

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

```
    # adding two objects
    def __add__(self, other):
        return self.a + other.a, self.b +
other.b

    def __str__(self):
        return self.a, self.b
Ob1 = complex(1, 2)
Ob2 = complex(2, 3)
Ob3 = Ob1 + Ob2
print(Ob3)
```

Output:

```
(3, 5)
```

d. Aim: Write a program to overload comparison operator

Code:

```
#Python program to overload
# a comparison operators

class A:
    def __init__(self, a):
        self.a = a
    def __gt__(self, other):
        if(self.a>other.a):
            return True
        else:
            return False
ob1 = A(2)
ob2 = A(3)
```

```
if(ob1>ob2):
    print("ob1 is greater than ob2")
else:
    print("ob2 is greater than ob1")
```

Output:

```
ob2 is greater than ob1
```

e. Aim: Write a program to overload equality and less than operators.

Code:

```
# Python program to overload equality
# and less than operators
class A:
    def __init__(self, a):
        self.a = a
    def __lt__(self, other):
        if(self.a<other.a):
            return "ob1 is less than ob2"
        else:
            return "ob2 is less than ob1"
    def __eq__(self, other):
        if(self.a == other.a):
            return "Both are equal"
        else:
            return "Not equal"
ob1 = A(2)
ob2 = A(3)
print(ob1 < ob2)
ob3 = A(4)
ob4 = A(4)
```

```
print(ob1 == ob2)
```

Output:

```
ob1 is less than ob2
Not equal
```

PRACTICAL 9

Implementing Abstract Classes And Interfaces

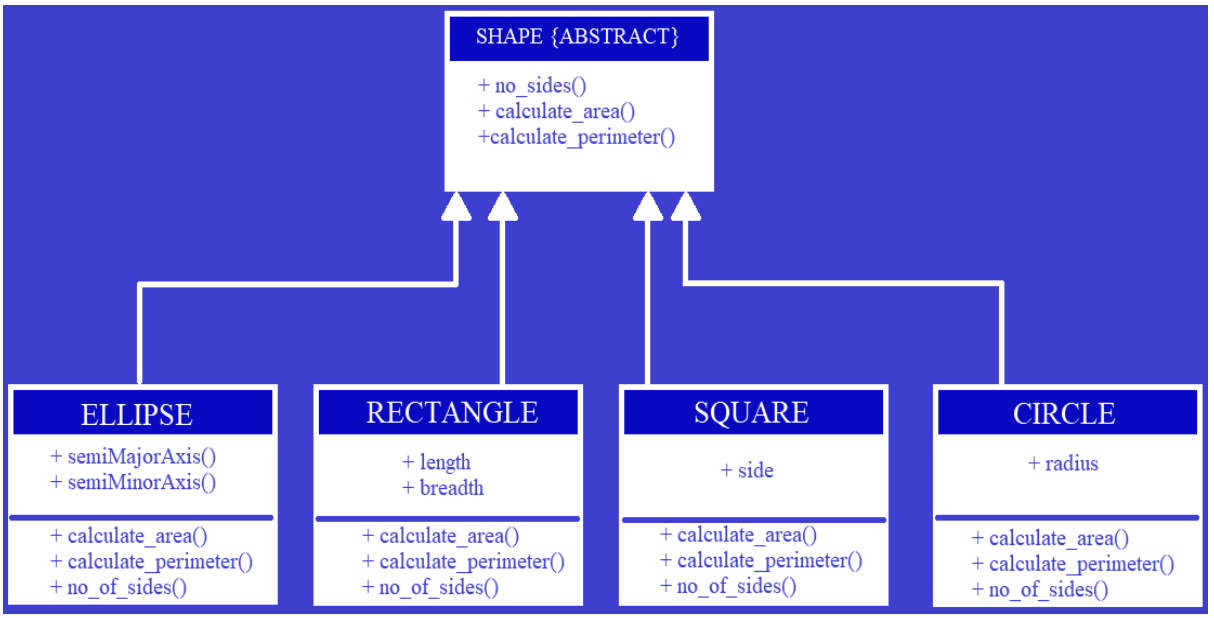
WRITE UP

ABSTRACT CLASS:

Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation. Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods. Subclasses of an abstract class in Python are not required to implement abstract methods of the parent class.

a. Aim: Write a program to calculate no of sides, area and perimeter of shapes like rectangle, square, circle and ellipse by implementing the concept of abstract classes and make a class diagram for it.

CLASS DIAGRAM



Code:

```
from abc import ABC
import math

#abstract class
class shape(ABC):
    def no_of_sides(self):
        pass
    def calculate_area(self):
        pass
    def calculate_perimeter(self):
        pass

class rectangle(shape):
    def no_of_sides(self,length,breadth):
        print("length = 40\n breadth = 30")
        self.length = length
```

```
        self.breadth = breadth

    def calculate_area(self):
        print("area of rectangle:
",self.length*self.breadth)

    def calculate_perimeter(self):
        print("perimeter of rectangle:
",2*(self.length+self.breadth),"\n")

class square(shape):
    def no_of_sides(self,side):
        print("side = 10")
        self.side=side
    def calculate_area(self):
        print("area of square: ",
self.side**2)
    def calculate_perimeter(self):
        print("perimeter of square:
",4*(self.side),"\n")

class circle(shape):
    def no_of_sides(self,radius):
        print("radius = 7")
        self.radius=radius
    def calculate_area(self):
        print("area of circle:
",(22/7)*self.radius**2)
    def calculate_perimeter(self):
```

```
        print("perimeter of circle:
",2*((22/7)*self.radius),"\n")

class ellipse(shape):
    def
no_of_sides(self,semiMajorAxis,semiMinorAxis):
        print("semi major axis = 2\n semi
minor axis = 2")
        self.semiMajorAxis = semiMajorAxis
        self.semiMinorAxis = semiMinorAxis
    def calculate_area(self):
        print("area of ellipse:
",(22/7)*self.semiMajorAxis*self.semiMinorAxis
)
    def calculate_perimeter(self):
        print("perimeter of ellipse:
",2*((22/7)**math.sqrt((self.semiMajorAxis+sel
f.semiMinorAxis)/2)))

obj = rectangle()
obj.no_of_sides(10,20)
obj.calculate_area()
obj.calculate_perimeter()

obj1 = square()
obj1.no_of_sides(20)
obj1.calculate_area()
obj1.calculate_perimeter()

obj2 = circle()
```

```
obj2.no_of_sides(10)
obj2.calculate_area()
obj2.calculate_perimeter()

obj3 = ellipse()
obj3.no_of_sides(10,20)
obj3.calculate_area()
obj3.calculate_perimeter()
```

Output:

```
length = 40
 breadth = 30
area of rectangle:  200
perimeter of rectangle:  60

side = 10
area of square:  400
perimeter of square:  80

radius = 7
area of circle:  314.2857142857143
perimeter of circle:  62.857142857142854

semi major axis = 2
 semi minor axis = 2
area of ellipse:  628.5714285714286
perimeter of ellipse:  168.71746722121068
```

PRACTICAL 10

Implementing Concept Of Composition

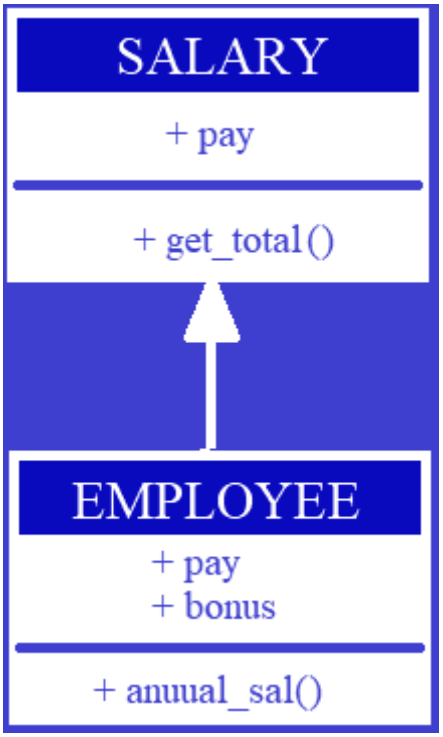
WRITE UP

COMPOSITION:

In composition, we do not inherit from the base class but establish relationships between classes through the use of instance variables that are references to other objects. In composition one of the classes is composed of one or more instances of other classes. In other words one class is container and other class is content and if you delete the container object then all of its contents objects are also deleted. Composition means that an object knows another object, and explicitly delegates some tasks to it. While inheritance is implicit, composition is explicit

a. Aim:Write a program to implement the concept of composition.

CLASS DIAGRAM



Code:

```
class salary:
    def __init__(self,pay):
        self.pay = pay
    def get_total(self):
        return (self.pay*12)
```

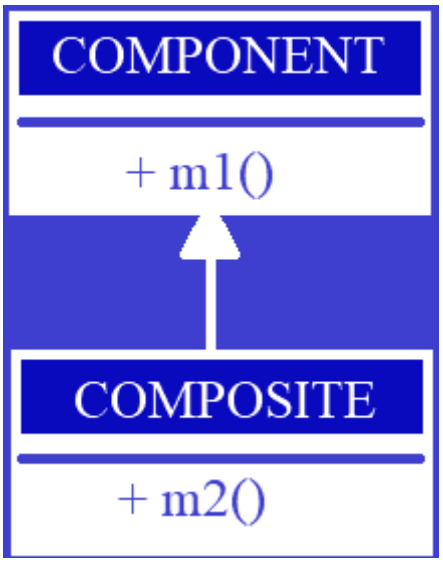
```
class employee:
    def __init__(self,pay,bonus):
        self.pay = pay
        self.bonus = bonus
    def annual_sal(self):
        return "total: "+
str(self.pay.get_total()+self.bonus)

obj_sal=salary(90000)
obj_emp = employee(obj_sal,30000)
print(obj_emp.annual_sal())
```

Output:

```
total: 1110000
```

a2.

CLASS DIAGRAM



Code:

```
class component:
    def __init__(self):
        print("component class object
created...")
```

```
    def m1(self):
        print("component class m1() method
executed...")

class composite:
    def __init__(self):
        self.obj1 = component()
        print('composite class object also
created...')
    def m2(self):
        print("composite class m2() method
executed...")
        self.obj1.m1()

obj2 = composite()
obj2.m2()
```

Output:

```
component class object created...
composite class object also created...
composite class m2() method executed...
component class m1() method executed...
```