```python
# 1. Write a Python program to create an abstract class for a shape with an abstract method to calculate area. Implement this class for a rectangle and a circle.
from abc import ABC,abstractmethod
class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class rectangle(Shape):
    def area(self,l,b):
        return l*b
class circle(Shape):
    def area(self,r):
        return 3.14*2*r
r=rectangle()
# print(r.area(10,20))
c=circle()
# print(c.area(2))
# output:200,12.56


# 2. Create a class in Python with encapsulation, demonstrating private and public variables and methods.
class A:
    def __init__(self):
        self.__a='I am a'
        self.b=2
    def get_a(self):
        return self.__a
c=A()
# print(c.b)   # b is public varible it can be accessed outside of class with help of objects
# print(c.__a) # this will throw an attribute error boz we are accessing private varible of class a
            # to access that varible we have to use getter and setter function within a class
# print(c.get_a())
# output:
# 2
# I am a


# 3. Design a Python program that uses polymorphism by creating a base class for animals and subclasses for different types of animals with a method speak.
class animals:
```

```
    def __init__(self):
        print("Welcome to Zoo")
    def speak(self):
        print("Sound")
class Tiger(animals):

    def __init__(self,name):
        self.name=name
        super().__init__() # here we can avoid method overriding with help of super() method

    def speak(self):
        print(self.name,"Sounds Roar")# here method overridding is a same name method speak is also present in class animals hence it overwrriten by recent child class method of speak

# t=Tiger("Tiger")
# t.speak()
# output:
# Welcome to Zoo
# Tiger Sounds Roar


# 4. Implement a program in Python to demonstrate single inheritance with a parent class Vehicle and a child class Car.
class Vehicle:
    def __init__(self):
        print("welcome to Vehicle Class")
class Car(Vehicle):
    def type(self,name):
        self.name=name
        print(self.name)
# c=Car()
# c.type("Toyoto")
# output:
# welcome to Vehicle Class
# Toyoto


# 5. Write a Python program to demonstrate multiple inheritance with two parent classes and one child class, showing how attributes are shared.
class mom:
    def d(self,name):
        self.name=name
```

```
        print(self.name,"I am from Mom class")
class dad:
    def d(self,name):
        self.name=name
        print(self.name,"I am From Dad Class")
class child(dad,mom):
    pass
# c=child()
# c.d('Nagesh')
# output:Nagesh I am From Dad Class
# if there are 2 methods or attributes from parent class are same then python will gives priority to the fist class which inherited from left hence here dad class d method is excuted


# 6. Create a program to implement multilevel inheritance, where the grandchild class inherits attributes and methods from the grandparent and parent classes.
class grandparent:
    def home(self):
        print("Grandparents home")
class parent(grandparent):
    def phome(self):
        print("Parents home")
class child(parent):
    pass
# c=child()
# c.home()
# c.phome()
# output:Grandparents home
# Parents home
# here child can get both futures from grandparent and parent class


# 7. Write a Python program to demonstrate method overloading using default arguments in a class.
class A:
    def __init__(self,city='solapur'):
        print(city)
# a=A()
# a=A('Pune')
# output:
# solapur
# pune
```

```python
# 8. Write a Python program to demonstrate method overriding by creating a base class and a derived class with the same method.
class A:
    def d(self):
        print("i am class A")
class B(A):
    def d(self):
        print("i am updated now as B")
# a=B()
# a.d()
# output:
# i am updated now as B


# 9. Design a Python class with both public and private methods. Demonstrate how to call each from within and outside the class.
class A:
    def public(self):
        print("I am public method")
    def __private(self):
        print("I am private method")
        # if you want to access private methods of class then you have to acces then with help of geteer or seeter methods

    def g(self):
        self.__private()
# a=A()
# a.public()
# a.g()
# output:
# I am public method
# I am private method


# 10. Write a Python program to create a class with a private variable and provide getter and setter methods to access and update its value.
class A:
    def __init__(self):
        self.__a='Lava@2203'
    def get_a(self):
        return self.__a
```

```
    def set_a(self,new_a):
        self.__a=new_a
# a=A()
# print(a.get_a())
# a.set_a('Sara@2203')
# print('updated private varible',a.get_a())
# output:
# Lava@2203
# updated private varible Sara@2203


# 11. Create a Python program to demonstrate how polymorphism works with functions or methods having the same name but different implementations in different classes.
class A:
    def d(self):
        print("i am class A")
class B(A):
    def d(self):
        print("i am updated now as B")
b=B()
b.d()
# here object b having same method in both class but NotImplementation is different the method will overrite with recent method of class


# 12. Write a Python program to demonstrate the use of protected access specifiers and show how they can be accessed in derived classes.
class A:
    def __init__(self):
        self._a='I will only give access to my anccestors'
class B(A):
    pass
b=B()
print(b._a)
# output:
# I will only give  access to my anccestors


# 13. Create a Python program to demonstrate abstraction using an interface with multiple classes implementing it.
from abc import ABC,abstractmethod
class Shape(ABC):
```

```
    @abstractmethod
    def area(self):
        pass

class rectangle(Shape):
    def area(self,l,b):
        print(l*b)
class circle(Shape):
   pass
r=rectangle()
r.area(2,4)
# c=circle() # we must have to create area method in circle class if we are inheriting the shape class boz shape class having abstract method
#Can't instantiate abstract class circle without an implementation for abstract method 'area'


# 14. Write a Python program to demonstrate hierarchical inheritance, where multiple child classes inherit from a single parent class.
class A:
    def __init__(self):
        print("Parent class")
class B(A):
    pass
class C(A):
    pass
b=B()
c=C()
# output:
# Parent class
# Parent class

# 15. Implement a Python program to demonstrate the use of `@property` for encapsulation by creating a class to manage student grades with controlled access to the data.
class A:
    def __init__(self):
        self.__a='I am a'
        self.b=2
    def get_a(self):
        return self.__a
    v=property(get_a)
c=A()
print(c.v)
```

```
# output:I am a
```