```python
# 1. Create a class Vehicle with a method start_engine. Create two derived classes Car and Bike that override the start_engine method. Demonstrate polymorphism by calling the method from an instance of each class.
class Vehicle:
    def start_engine(self):
        print("Vehicle Class")
class Bajaj(Vehicle):
    def start_engine(self):
        print("Class Bajaj")
class Hero(Vehicle):
    def start_engine(self):
        print("Class Hero")
b=Bajaj()
b.start_engine()
h=Hero()
h.start_engine()
# Here both class having same method start_engine which having in base class hence method is overidden by derived classes
# output:
# Class Bajaj
# Class Hero



# 2. Write a program to implement single inheritance where a base class Person contains a method show_name, and a derived class Student adds a method show_grade.
class Person:
    def __init__(self,name):
        self.name=name
    def show_name(self):
        return self.name
class Student(Person):
    def show_grade(self,grade):
        print(self.show_name(),grade)
p=Student('lavanya')
# p.show_grade(98)
# output: lavanya 98
```

```python
# 3. Implement multilevel inheritance by creating a class LivingBeing, inherited by Animal, which is further inherited by Dog. Add a method to each class and demonstrate their usage.
class LivingBeing:
    def Alive(self):
        print("I am Alive")
class Animal(LivingBeing):
    def animal(self):
        print("Animal Category")
class Dog(Animal):
    def sound(self):
        print("Dog","Bark")
# d=Dog()
# d.Alive()
# d.animal()
# d.sound()
# output:I am Alive
# Animal Category
# Dog Bark
"""
here dog class is inherited from Animal class (parent class)
and Animal class is again inherited from LivingBeing (Grand parent class)
hence a child class can access properties of parent as well as grand parent class
"""


# 4. Define a base class Shape with an abstract method area. Create two derived classes, Circle and Rectangle, that implement the area method. Use polymorphism to calculate areas of both shapes.
from abc import ABC,abstractclassmethod
class Shape(ABC):
    @abstractclassmethod
    def area(self):
        pass
class Rectangle(Shape):
    def area(self,r):
        print(3.14*r*r*2)
class Circle(Shape):
    def area(self,l,b):
        print(l*b)
# r=Rectangle()
# c=Circle()
# r.area(2)
```

```
# c.area(4,8)
# output:
# 25.12
# 32


# 5. Create a parent class Employee with attributes name and salary. Inherit it in a class Manager and add an attribute department. Demonstrate how Manager can access the Employee attributes.
class Employee:
    def __init__(self,name,salary):
        self.name=name
        self.salary=salary
class Manager(Employee):
    def Display(self):
        department="CSE"
        print(self.name,self.salary,department)
# m=Manager('Lavanya',50000)
# m.Display()
# output:Lavanya 50000 CSE


# 6. Write a program using hierarchical inheritance where a base class Animal is inherited by two classes, Bird and Fish. Each subclass should have its own unique method.
class Animal:
    def category(self):
        print("Type Animal ")
class Bird(Animal):
    def sound(self):
        print("Bird","ChivChiv")
class Fish(Animal):
    def sound(self):
        print("Fish","POOPOO")
# b=Bird()
# b.category()
# b.sound()
# b=Fish()
# b.category()
# b.sound()
# output:
# Type Animal
# Bird ChivChiv
```

```python
# Type Animal
# Fish POOPOO


# 7. Create a class Book with a method get_info. Override this method in a derived class EBook to display additional details like file size and format.
class Book:
    def __init__(self,name):
        self.name=name
    def get_info(self):
        print(self.name)
class Ebook(Book):
    def get_info(self,filesize,format):
        self.filesize=filesize
        self.format=format
        print("Name:",self.name,"Filesize:",self.filesize,"Fromat:",self.format)
e=Ebook('The Hobbit')
e.get_info('32MB',".pdf")
# output:Name: The Hobbit Filesize: 32MB Fromat: .pdf


# 8. Implement a program where a base class Instrument has a method play. Create two derived classes, Guitar and Piano, that override the play method.
class Instrument:
    def play(self):
        print("Sound")
class Guitar(Instrument):
    def play(self):
        print("lallalaa")
class Piano(Instrument):
    def play(self):
        print("Saregamapa")
# g=Guitar()
# g.play()
# g=Piano()
# g.play()
# output:
# lallalaa
# Saregamapa
```

```python
# 9. Write a program to demonstrate method resolution order (MRO) in multiple inheritance by creating classes A, B, C, and D, where D inherits from both B and C, which in turn inherit from A.
class A:
    def d(self):
        print("A")
class B(A):
    def d(self):
        print("B")
class C(A):
    def d(self):
        print("C")
class D(B,C):
    pass
d1=D()
# d1.d()
# output:B


# 10. Define a class Shape with methods for setting and getting dimensions. Create a derived class Triangle that adds a method to calculate the area.
class Shape:
    def __init__(self):
        self.__b=0
        self.__h=0
    def get(self):
        return (self.__b,self.__h)
    def set(self,b,h):
        self.__b=b
        self.__h=h

class Triangle(Shape):
    def area(self):
        t=self.get()
        print("Area",0.5*(t[0]*t[1]))
        # print(self.__b) this will through error boz b is private variable of shape
# s=Triangle()
# s.set(10,10)
# s.area()
# output: Area 50.0
```

```python
# 11. Write a Python program to show how a parent class method can be accessed using the super() function in a derived class.
class A:
   def d(self):
     print("Parent class")
class B(A):
   def d(self):
      super().d()
      print("Derived class")
# b=B()
# b.d()
# output:
# Parent class
# Derived class


# 12. Create a base class Account with methods for deposit and withdrawal. Inherit it into a class SavingsAccount and override the withdrawal method to include a minimum balance check.
class Account:
   def deposit(self):
      print("deposit")
   def withdrawal(self):
      print("withdrawal")
class SavingAccount(Account):
   def withdrawal(self):
      print("Please add minimum balance of 1000")
# s=SavingAccount()
# s.withdrawal()
# output:Please add minimum balance of 1000


# 13. Implement a class Animal with a method speak. Create two subclasses, Cat and Dog, that override speak. Demonstrate polymorphism by calling speak from a list of mixed objects.
class Animal:
   def speak(self):
      print("Animal class ")
class Cat(Animal):
   def speak(self):
      print("Cat","Meow")
class Dog(Animal):
   def speak(self):
```

```
        print("Dog","Bark")
# c=Cat()
# c.speak()
# d=Dog()
# d.speak()
# output:Cat Meow
# Dog Bark


# 14. Write a Python program to demonstrate constructor inheritance where a parent class constructor initializes common attributes, and a derived class constructor adds more attributes.
class A:
    def __init__(self,a,b):
        self.a=a
        self.b=b
class B(A):
    def __init__(self,a,b,c,d):
        self.a=a
        self.b=b
        self.c=c
        self.d=d
        print(self.a)

# b=B(10,20,30,40)


# 15. Create a program to demonstrate operator overloading by overloading the + operator for a class Vector to add two vector objects.
class Vector:
    def __init__(self,a):
        self.a=a
        print(self.a)
    def __add__(self,another):
        return self.a+another.a
# v1=Vector(10)
# v2=Vector(20)
# print("Sum:",v1+v2)
# output:Sum: 30
```