**Test 3**

1. Create an Abstract class Vehicle with method wheels(), implement car, bike, bus

from abc import ABC, abstractmethod

class Vehicle(ABC):

    @abstractmethod

    def wheels(self):

        pass

class Car(Vehicle):

    def wheels(self):

        return "Car has 4 wheels"

class Bike(Vehicle):

    def wheels(self):

        return "Bike has 2 wheels"

class Bus(Vehicle):

    def wheels(self):

        return "Bus has 6 wheels"

vehicles = [Car(), Bike(), Bus()]

for v in vehicles:

    print(v.wheels())

1. Can we create object for abstract class , prove it using code

from abc import ABC, abstractmethod

class Test(ABC):

    @abstractmethod

    def show(self):

        pass

# Attempt to create object/ so we used error handling here

try:

    obj = Test()   # ❌ Not allowed

except TypeError as e:

    print("Error:", e)

1. Write a class BankAccount that keeps balance private, provide methods to deposit, withdraw and check\_balance

class BankAccount:

    def \_\_init\_\_(self, balance=0):

        self.\_\_balance = balance   # private variable

    def deposit(self, amount):

        self.\_\_balance += amount

        print(f"Deposited {amount}, Balance: {self.\_\_balance}")

    def withdraw(self, amount):

        if amount <= self.\_\_balance:

            self.\_\_balance -= amount

            print(f"Withdrawn {amount}, Balance: {self.\_\_balance}")

        else:

            print("Insufficient funds")

    def check\_balance(self):

        return self.\_\_balance

acc = BankAccount(1000)

acc.deposit(500)

acc.withdraw(300)

acc.withdraw(2000)

print("Final Balance:", acc.check\_balance())

1. Write a program for inheritance to showcase one\_parent, multiple childerns

class Human:   # Parent class

    def speak(self):

        print("Humans can communicate")

class Teacher(Human):   # Child 1

    def teach(self):

        print("Teacher teaches students")

class Doctor(Human):   # Child 2

    def treat(self):

        print("Doctor treats patients")

class Engineer(Human):   # Child 3

    def build(self):

        print("Engineer builds solutions")

t = Teacher()

d = Doctor()

e = Engineer()

print("Teacher Object:")

t.speak()

t.teach()

print("\nDoctor Object:")

d.speak()

d.treat()

print("\nEngineer Object:")

e.speak()

e.build()