Python OOPs Assignment

Q1. Write a Python program to demonstrate multiple inheritance.

1. Employee class has 3 data members EmployeeID, Gender (String), Salary and PerformanceRating(Out of 5) of type int. It has a get() function to get these details from the user.

2. JoiningDetail class has a data member DateOfJoining of type Date and a function getDoJ to get the Date of joining of employees.

3. Information Class uses the marks from Employee class and the DateOfJoining date from the JoiningDetail class to calculate the top 3 Employees based on their Ratings and then Display, using readData, all the details on these employees in Ascending order of their Date Of Joining.

CODE:

```
class Employee:
    def __init__(self):
        self.EmployeeID = ""
        self.Gender = ""
        self.Salary = 0
        self.PerformanceRating = 0

    def get(self):
        self.EmployeeID = input("Enter Employee ID: ")
        self.Gender = input("Enter Gender: ")
        self.Salary = int(input("Enter Salary: "))
        self.PerformanceRating = int(input("Enter Performance Rating (out of 5): "))


class JoiningDetail:
    def __init__(self):
        self.DateOfJoining = ""

    def getDoJ(self):
        self.DateOfJoining = input("Enter Date of Joining (YYYY-MM-DD): ")
```

```
class Information(Employee, JoiningDetail):
    def __init__(self):
        Employee.__init__(self)
        JoiningDetail.__init__(self)

    @staticmethod
    def readData(employees):
        sorted_employees = sorted(employees, key=lambda x: x.DateOfJoining)
        for emp in sorted_employees[:3]:
            print(f"Employee ID: {emp.EmployeeID}, Gender: {emp.Gender}, Salary: {emp.Salary}, "
                  f"Performance Rating: {emp.PerformanceRating}, Date of Joining: {emp.DateOfJoining}")
```

<u>Q2.</u> Write a Python program to demonstrate Polymorphism.

1. Class Vehicle with a parameterized function Fare, that takes input value as fare and returns it to calling Objects.

2. Create five separate variables Bus, Car, Train, Truck and Ship that call the Fare function.

3. Use a third variable TotalFare to store the sum of fare for each Vehicle Type.

4. Print the TotalFare.

<u>CODE:</u>

```
class Vehicle:
    def Fare(self, fare):
        return fare


vehicles = {
    "Bus": 50,
    "Car": 20,
    "Train": 70,
    "Truck": 40,
    "Ship": 100

}


TotalFare = sum(Vehicle().Fare(fare) for fare in vehicles.values())

print(f"Total Fare: {TotalFare}")
```

Q3. Consider an ongoing test cricket series. Following are the names of the players and their scores in the test1 and 2. Test Match 1 : Dhoni : 56 , Balaji : 94 Test Match 2 : Balaji : 80 , Dravid : 105 Calculate the highest number of runs scored by an individual cricketer in both of the matches. Create a python function Max_Score (M) that reads a dictionary M that recognizes the player with the highest total score. This function will return (Top player , Total Score) . You can consider the Top player as String who is the highest scorer and Top score as Integer . Input : Max_Score({'test1':{'Dhoni':56, 'Balaji : 85}, 'test2':{'Dhoni' 87, 'Balaji ':200}}) Output : ('Balaji ' , 200)

CODE:

```
def Max_Score(M):

    scores = {}

    for test, players in M.items():

        for player, score in players.items():

            scores[player] = scores.get(player, 0) + score



    top_player = max(scores, key=scores.get)

    return top_player, scores[top_player]



M = {'test1': {'Dhoni': 56, 'Balaji': 85}, 'test2': {'Dhoni': 87, 'Balaji': 200}}

print(Max_Score(M))
```

Q4. Create a simple Card game in which there are 8 cards which are randomly chosen from a deck. The first card is shown face up. The game asks the player to predict whether the next card in the selection will have a higher or lower value than the currently showing card. For example, say the card that's shown is a 3. The player chooses "higher," and the next card is shown. If that card has a higher value, the player is correct. In this example, if the player had chosen "lower," they would have been incorrect. If the player guesses correctly, they get 20 points. If they choose incorrectly, they lose 15 points. If the next card to be turned over has the same value as the previous card, the player is incorrect.

CODE:

```
import random


def card_game():
    deck = list(range(1, 14)) * 4
    random.shuffle(deck)
    points = 0

    current_card = deck.pop()
    print(f"First card: {current_card}")

    while deck:
        guess = input("Will the next card be higher or lower? (h/l): ").strip().lower()
        next_card = deck.pop()
        print(f"Next card: {next_card}")

        if (guess == 'h' and next_card > current_card) or (guess == 'l' and next_card < current_card):
            points += 20
            print("Correct! +20 points")
        else:
            points -= 15
            print("Incorrect. -15 points")

        current_card = next_card

    print(f"Game over. Total points: {points}")

card_game()
```

<u>Q5.</u> Create an empty dictionary called Car_0 . Then fill the dictionary with Keys : color , speed , X_position and Y_position. car_0 = {'x_position': 10, 'y_position': 72, 'speed': 'medium'} .

a) If the speed is slow the coordinates of the X_pos get incremented by 2.

b) If the speed is Medium the coordinates of the X_pos gets incremented by 9 c) Now if the speed is Fast the coordinates of the X_pos gets incremented by 22. Print the modified dictionary.

<u>CODE:</u>

```
car_0 = {'x_position': 10, 'y_position': 72, 'speed': 'medium'}


if car_0['speed'] == 'slow':
    car_0['x_position'] += 2
elif car_0['speed'] == 'medium':
    car_0['x_position'] += 9
elif car_0['speed'] == 'fast':
    car_0['x_position'] += 22


print(car_0)
```

<u>Q6.</u> Show a basic implementation of abstraction in python using the abstract classes.

1. Create an abstract class in python.

2. Implement abstraction with the other classes and base class as abstract class.

<u>CODE:</u>

```
from abc import ABC, abstractmethod


class Shape(ABC):
    @abstractmethod
    def area(self):
        pass


    @abstractmethod
```

```
 def perimeter(self):
    pass


class Rectangle(Shape):
 def __init__(self, length, width):
    self.length = length
    self.width = width


 def area(self):
    return self.length * self.width
 def perimeter(self):
    return 2 * (self.length + self.width)
rect = Rectangle(10, 5)
print(f"Area: {rect.area()}, Perimeter: {rect.perimeter()}")
```

Q7. Create a program in python to demonstrate Polymorphism.

1. Make use of private and protected members using python name mangling techniques.
   CODE:

```
class Animal:
  def __init__(self, name):
    self.__name = name  # Private member
    self._type = "Animal"  # Protected member

  def make_sound(self):
    pass

class Dog(Animal):
  def __init__(self, name):
    super().__init__(name)

  def make_sound(self):
    return "Bark"

class Cat(Animal):
  def __init__(self, name):
    super().__init__(name)
```

```
        def make_sound(self):
            return "Meow"

    animals = [Dog("Doggo"), Cat("Kitty")]
    for animal in animals:
        print(animal.make_sound())
```

Q8. Given a list of 50 natural numbers from 1-50. Create a function that will take every element from the list and return the square of each element. Use the python map and filter methods to implement the function on the given list.

CODE:

```
numbers = list(range(1, 51))

squares = list(map(lambda x: x ** 2, numbers))

print(squares)
```

Q9. Create a class, Triangle. Its init() method should take self, angle1, angle2, and angle3 as arguments.

CODE:

```
class Triangle:

    number_of_sides = 3


    def __init__(self, angle1, angle2, angle3):

        self.angle1 = angle1

        self.angle2 = angle2

        self.angle3 = angle3


    def check_angles(self):

        if self.angle1 + self.angle2 + self.angle3 == 180:

            print("Valid triangle")

            return True

        else:

            print("Invalid triangle")
            return False

    def is_acute(self):
        return all(angle < 90 for angle in [self.angle1, self.angle2, self.angle3])

    def is_obtuse(self):
        return any(angle > 90 for angle in [self.angle1, self.angle2, self.angle3])

triangle = Triangle(60, 60, 60)
triangle.check_angles()
print(f"Is acute? {triangle.is_acute()}, Is obtuse? {triangle.is_obtuse()}")
```

Q10. Create a class variable named number_of_sides and set it equal to 3.

CODE:

```
class Triangle:
    number_of_sides = 3  # Class variable

    def __init__(self, angle1, angle2, angle3):
        self.angle1 = angle1
        self.angle2 = angle2
        self.angle3 = angle3

    def check_angles(self):
        return self.angle1 + self.angle2 + self.angle3 == 180

# Example
triangle = Triangle(60, 60, 60)
print(f"Number of sides: {Triangle.number_of_sides}")
print(f"Is triangle valid? {triangle.check_angles()}")
```

Q11. Create a method named check_angles. The sum of a triangle's three angles should return True if the sum is equal to 180, and False otherwise. The method should print whether the angles belong to a triangle or not.

11.1 Write methods to verify if the triangle is an acute triangle or obtuse triangle.

11.2 Create an instance of the triangle class and call all the defined methods.

11.3 Create three child classes of triangle class- isosceles_triangle, right_triangle and equilateral_triangle.

11.4 Define methods which check for their properties.

CODE:

```
class Triangle:

  number_of_sides = 3


  def __init__(self, angle1, angle2, angle3):

    self.angle1 = angle1

    self.angle2 = angle2

    self.angle3 = angle3


  def check_angles(self):

    is_valid = self.angle1 + self.angle2 + self.angle3 == 180

    print("Valid Triangle" if is_valid else "Invalid Triangle")

    return is_valid


  def is_acute(self):

    return all(angle < 90 for angle in [self.angle1, self.angle2, self.angle3])


  def is_obtuse(self):

    return any(angle > 90 for angle in [self.angle1, self.angle2, self.angle3])
```

Q12. Create a class isosceles_right_triangle which inherits from isosceles_triangle and right_triangle.

12.1 Define methods which check for their properties.

CODE:

```
class EquilateralTriangle(Triangle):

    def is_equilateral(self):

        return self.angle1 == self.angle2 == self.angle3


class IsoscelesRightTriangle(IsoscelesTriangle, RightTriangle):

    def is_isosceles_right(self):

        return self.is_isosceles() and self.is_right()
```