```
import datetime as dt
```

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.

```
class Employee:
    def __init__(self, id: str, gender: str, salary: int, perfomance_rating: int):
        self.id, self.gender, self.salary, self.perfomance_rating = id, gender, salary, perfomance_rating

    def get_id(self):
        return self.id

    def get_gender(self):
        return self.gender

    def get_salary(self):
        return self.salary

    def get_perf_rat(self):
        return self.perfomance_rating


class JoiningDetail(Employee):
    def __init__(self, id: str, gender: str, salary: int, perfomance_rating: int, date: dt.date):
        super().__init__(id, gender, salary, perfomance_rating)
        self.date = date

    def get_DoJ(self):
        return self.date

    def __str__(self):
        return f"id {self.id} : {self.date}"
```

Start coding or generate with AI.

```
class Information:
    def __init__(self, e_list):
        self.e_list = e_list
    def readData(self):
      N = len(self.e_list)
      for i in range(N-1):
        for j in range(N-i-1):
          if self.e_list[j].get_perf_rat() > self.e_list[j+1].get_perf_rat():
            self.e_list[j], self.e_list[j+1] = self.e_list[j+1], self.e_list[j]
      for i in range(3):
        for j in range(2 - i):
          if self.e_list[j].get_DoJ() > self.e_list[j + 1].get_DoJ():
            self.e_list[j], self.e_list[j+1] = self.e_list[j+1], self.e_list[j]
      return '\n'.join(list(map(str, self.e_list[:3])))



temp = [('1', 'm', 10, 5, dt.date(2000, 10, 30)), ('2', 'f', 5, 3, dt.date(2001, 10, 12)), ('3', 'm', 100, 10, dt.date(1998, 10, 1))]

lst = Information([JoiningDetail(*i) for i in temp])
print(lst.readData())
```

```
id 3 : 1998-10-01
id 1 : 2000-10-30
id 2 : 2001-10-12
```

Q.2 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.

```
class Vehicle:
  def __init__(self, fare):
    self.fare = fare

  def Fare(self):
    return self.fare


Bus = Vehicle(100)
Car = Vehicle(80)
Train = Vehicle(50)
Truck = Vehicle(520)
Ship = Vehicle(750)


Bus.Fare()
```

```
100
```

```
Car.Fare()
```

```
80
```

```
Train.Fare()
```

```
50
```

```
Truck.Fare()
```

```
520
```

```
Ship.Fare()
```

```
750
```

```
TotalFare = (Bus.Fare() + Car.Fare() + Train.Fare() + Truck.Fare() + Ship.Fare())
print(Bus.Fare(), Car.Fare(),Train.Fare(), Truck.Fare(), Ship.Fare())
print(TotalFare)
```

```
100 80 50 520 750
1500
```

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)

```
Start coding or generate with AI.
```

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

def Max_Score(d):
  total = {}
  for key in d.keys():
    for n in d[key].keys():
      if n in total.keys():
        total[n] = total[n] + d[key][n]
      else:
        total[n] = d[key][n]

  print("Total Runs Scored by Each Players in Both Tests Match : ")
  print(total)
  ("Top player , Total Score")

  player_total = 0
  for n in total.keys():
    if total[n] > player_total:
      player_name = n
      player_total = total[n]
  return (player_name, player_total)



Output = Max_Score(M)
print(Output)
```

```
Total Runs Scored by Each Players in Both Tests Match :
{'Dhoni': 143, 'Balaji': 285}
('Balaji', 285)
```

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

def Max_Score(d):
    total = {}
    for k in d.keys():
        for n in d[k].keys():
            if n in total.keys():
                total[n] = total[n] + d[k][n]
            else:
                total[n] = d[k][n]
    print("Total Run Scored by Each Playes in 2 Tests: ")
    print(total)

    print("Player With highest score")
    maxtotal = -1
    for n in total.keys():
        if total[n] > maxtotal:
            maxname = n
            maxtotal = total[n]

    return (maxname, maxtotal)

summary = Max_Score(M)
print(summary)
```

```
Total Run Scored by Each Playes in 2 Tests:
{'Dhoni': 143, 'Balaji': 285}
Player With highest score
('Balaji', 285)
```

```
Start coding or generate with AI.
```

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.

```
from random import choice as rand_choice
```

```
class Deck:
  def __init__(self):
    self.deck = dict()
    self.suit = ("Clubs", "Diamonds", "Hearts", "Spades")
    self.card = ("Ace", 2, 3, 4, 5, 6, 7, 8, 9, 10, "Jack", "Queen", "King")
    for suit in self.suit:
      self.deck[suit] = list(self.card)
  @property
  def is_empty(self) -> bool:
    result = True
    for suit in self.suit:
      if self.deck[suit]:
        result = False
        break
    return

  def has_a_card(self, suit: str, card: str) :
    result = False
    if card in self.deck[suit]:
      result = True
    return result

  def remove_a_card(self, suit: str, card: str):
    assert isinstance(suit, str)
    #assert isinstance(card, str) or isinstance(card, int) try:
    if not self.is_empty and self.has_a_card(suit, card):
      self.deck[suit].remove(card)
      return True
    else:
      print(f'===WARNING!!!===: Randomly chosen "{card} of {suit}" was removed from the deck before!')
      return False
    #except AssertionError as e:
      print(e)
```

Q5. 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.

```
Car_0 = {}


Car_0['x_position'] = 10
Car_0['y_position'] = 72
Car_0['speed'] = 'medium'
Car_0['color'] = 'red'
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)


print(Car_0)
```

```
{'x_position': 19, 'y_position': 72, 'speed': 'medium', 'color': 'red'}
```

```
Start coding or generate with AI.
```

Q6. 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.

```
from abc import ABC,abstractmethod
class A(ABC):
     @abstractmethod
     def test1(self):
          pass
     @abstractmethod
     def test2(self):
          pass
class B(A):
     def test1(self):
          print("Hello")
class C(B):
     def test2(self):
         print("Welcome")



obj1 = A() ### There will be error messsage as there is abstract method defined in class A which cannot be instantiate.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-38-b5b6e362ea75> in <cell line: 1>()
----> 1 obj1 = A() ### There will be error messsage as there is abstract method defined in class A which cannot be instantiate.

TypeError: Can't instantiate abstract class A with abstract methods test1, test2
```

```
obj2 = B() ### The abstract method cant be instantiated with the bases abstract class.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-c78ff51f687b> in <cell line: 1>()
----> 1 obj2 = B() ### The abstract method cant be instantiated with the bases abstract class.

TypeError: Can't instantiate abstract class B with abstract method test2
```

```
obj = C() ### The bastract method can be instantiated with other class.



obj.test1()
obj.test2()
```

```
Hello
Welcome
```

Q7. Create a program in python to demonstrate Polymorphism.

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

```
# A simple Python function to demonstrate
# Polymorphism, as program giving diffrent result in different circumstances called Polymorphism.

def add(a, b, c = 0):
    return a + b + c

print(add(7, 4))
print(add(7, 4, 1))
```

```
11
12
```

```
class myClass:
  def __init__(self):
    self.x=10
    self.__y=20

obj=myClass()
print(obj.__y)

###Even though we have initialized __y variable inside the __init__(), we can not access it outside of the class using the object.
###Adding __ as a prefix makes member private.
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-43-e7d8b6fd0263> in <cell line: 7>()
      5
      6 obj=myClass()
----> 7 print(obj.__y)
      8
      9 ###Even though we have initialized __y variable inside the __init__(), we can not access it outside of the class using the
object.

AttributeError: 'myClass' object has no attribute '__y'
```

```
###Here is the program. we can rewrite to access the name mangling attribute.
class myClass:
  def __init__(self):
    self.x=10
    self.__y=20

obj=myClass()
print(obj._myClass__y)
```

```
20
```

Start coding or generate with AI.

Start coding or generate with AI.

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.

```
x = list(range(1,51))
print(x)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
```

```
def square(num):
    return num ** 2


squares = list(map(square, x)) ## Using Map( function)
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 90
```

```
# use filter() to get odd numbers , even numbers and map() to square them
y = list(range(1,51))
print(y)

squared_odds = list(map(lambda x: x**2, filter(lambda x: x%2 != 0, y)))
squared_evens = list(map(lambda x: x**2, filter(lambda x: x%2 == 0, y)))
# print the squared odd numbers
print("Squared odd numbers:", squared_odds)
print("Squared even numbers:", squared_evens)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
Squared odd numbers: [1, 9, 25, 49, 81, 121, 169, 225, 289, 361, 441, 529, 625, 729, 841, 961, 1089, 1225, 1369, 1521, 1681, 1849, 2025,
Squared even numbers: [4, 16, 36, 64, 100, 144, 196, 256, 324, 400, 484, 576, 676, 784, 900, 1024, 1156, 1296, 1444, 1600, 1764, 1936, 2
```

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

Start coding or generate with AI.

```
class Triangle(object):
     def __init__(self,angle1,angle2,angle3):
         self.angle1=angle1
         self.angle2=angle2
         self.angle3=angle3

#Q10. Create a class variable named number_of_sides and set it equal to 3.
#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 othe
#The method should print whether the angles belong to a triangle or not.
     number_of_sides=3
     def check_angles(self):
         if self.angle1+self.angle2+self.angle3 ==180:
             return True
         else:
             return False
#Q11.1  Write methods to verify if the triangle is an acute triangle or obtuse triangle.
     def type_of_triangle(self):
        if self.angle1 < 90 and self.angle2 < 90 and self.angle3 < 90:
            return "Acute"
        return "Obtuse"


#Q11.3  Create three child classes of triangle class - isosceles_triangle, right_triangle and equilateral_triangle.

class Isosceles_triangle(Triangle):
    def __init__(self, angle1, angle2, angle3):
        super().__init__(angle1, angle2, angle3)

    def check(self):
        if self.angle1 == self.angle2 or self.angle1 == self.angle3 or self.angle2 == self.angle3:
            return True
        return False



class Right_triangle(Triangle):
   def __init__(self, angle1, angle2, angle3):
        super().__init__(angle1, angle2, angle3)

   def check(self):
        if self.angle1 == 90 or self.angle2 == 90 or self.angle3 == 90:
            return True
        return False


class Equilateral_triangle(Triangle):
  def __init__(self, angle1, angle2, angle3):
        super().__init__(angle1, angle2, angle3)


  def check(self):
        if self.angle1 == self.angle2 == self.angle3:
            return True
        return False


#Q11.4  Define methods which check for their properties.

test1 = Isosceles_triangle(60, 60, 60)
print(test1.check())
test2 = Right_triangle(90, 30, 60)
print(test2.check())
test3 = Equilateral_triangle(60, 60, 60)
print(test3.check())
test1 = Isosceles_triangle(70, 55, 55)
print(test1.check())
test2 = Right_triangle(60, 60, 60)
print(test2.check())
test3 = Equilateral_triangle(60, 59, 61)
print(test3.check())
```

```
True
True
True
True
False
False
```

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

12.1 Define methods which check for their properties.

```
class isosceles_right_triangle(Isosceles_triangle, Right_triangle):
  def __init__(self, angle1, angle2, angle3):
        super().__init__(angle1, angle2, angle3)

  def check(self):
    if self.angle1 == 90 and self.angle2 == 45 and self.angle3 ==45 or self.angle2 ==90 and self.angle1 ==45 and self.angle3 == 45 or self.a
        return True

    return False
```

```
test4 = isosceles_right_triangle(90,45,45)
print(test4.check())

test4 = isosceles_right_triangle(90,45,60)
print(test4.check())
```

```
True
False
```