Assignment 4

Q 1. Create a class 'Student' with rollno, studentName, course ,dictionary of marks(subjectName -> marks [5]). Provide following functionalities

A. initializer

B. override **str** method

C. accept student data

D. Print student data for given id.

E. Print Student who has failed in any subject.

In [2]:
```
class Student:
    def __init__(self,rollno, studentName, course, marks) -> None:
        self.rollno=rollno
        self.studentName=studentName
        self.course=course
        self.marks=marks
    def __str__(self) -> str:
        return f'Rollno: {self.rollno}, StudentName: {self.studentName}, course: {s
    def set_student(self,rollno, studentName, course):
        try:
            # self.rollno=int(input("Enter RollNo: "))
            self.rollno=rollno
            # self.studentName=input("Enter Student Name: ")
            self.studentName=studentName
            # self.course=input("Enter Course: ")
            self.course=course
            self.marks={}
            for i in range(1,6):
                subject = input(f'Enter Subject {i} Name:')
                mark=int(input(f'Enter marks for {subject}: '))
                self.marks[subject]=mark
        except ValueError:
            print("❗ Invalid input! Roll No and Marks must be numbers. Data not sav
            # Optionally reset data or raise the error
            self.rollno = 0
            self.marks = {}
    def has_failed(self, passing_mark=40):
            return any(mark < passing_mark for mark in self.marks.values())
```

In [3]:
```
class StudentManagement:
    def __init__(self):
        self.students = []

    def add_student(self, student: Student):
        self.students.append(student)

    def print_student_by_id(self, roll_id):
        """D. Prints student data for a given roll number."""
        print(f"\n--- Searching for RollNo: {roll_id} ---")
        for student in self.students:
            if student.rollno == roll_id:
```

```
                print(student)
                return
        print(f"Student with RollNo {roll_id} NOT found.")

    def get_failed_students(self, passing_mark=40):
        failed_list = [
            student for student in self.students
            if student.has_failed(passing_mark)
        ]
        print(f"\n--- Students Failed (Mark < {passing_mark}) ---")
        if not failed_list:
            print("No students failed any subject! 🥕")
        else:
            for student in failed_list:
                print(student)
        return failed_list
```

In [4]:

```
manager = StudentManagement()

# 1. Initialize Student (A, B)
s1 = Student(101, "Alice", "CS", {"Math": 85, "Physics": 78, "Chem": 62, "English":
manager.add_student(s1)
print("Initialized Student 1:")
print(s1)

# 2. Use set_student to take user input for marks (C)
# This student will pass all subjects.
s2 = Student(0, "", "", {}) # Create a dummy object first
s2.set_student(102, "Bob", "IT") # Prompts user for 5 subjects/marks
manager.add_student(s2)

# 3. Use set_student to take user input for marks (C)
# This student should fail at least one subject (e.g., mark < 40).
s3 = Student(0, "", "", {})
s3.set_student(103, "Charlie", "EE") # Prompts user for 5 subjects/marks
manager.add_student(s3)

# 4. Print student data for given id (D)
manager.print_student_by_id(101)
manager.print_student_by_id(103)
manager.print_student_by_id(999) # Non-existent ID

# 5. Print Student who has failed (E)
manager.get_failed_students()
```

```
Initialized Student 1:
Rollno: 101, StudentName: Alice, course: CS, marks: {'Math': 85, 'Physics': 78, 'Che
m': 62, 'English': 90, 'CS': 95}

--- Searching for RollNo: 101 ---
Rollno: 101, StudentName: Alice, course: CS, marks: {'Math': 85, 'Physics': 78, 'Che
m': 62, 'English': 90, 'CS': 95}

--- Searching for RollNo: 103 ---
Rollno: 103, StudentName: Charlie, course: EE, marks: {'s1': 34, 's2': 45, 's3': 67,
's4': 76, 's5': 78}

--- Searching for RollNo: 999 ---
Student with RollNo 999 NOT found.

--- Students Failed (Mark < 40) ---
Rollno: 102, StudentName: Bob, course: IT, marks: {'s1': 12, 's2': 25, 's3': 54, 's
4': 54, 's5': 55}
Rollno: 103, StudentName: Charlie, course: EE, marks: {'s1': 34, 's2': 45, 's3': 67,
's4': 76, 's5': 78}
```

Out[4]: `[<__main__.Student at 0x1ff962b7110>, <__main__.Student at 0x1ff962b7390>]`

Write menu driven program to test above functionalities.(accept records of 5 students and store those in list)

2. Write a menu driven program to maintain student information. for every student store studetid, sname, and m1,m2,m3 marks for 3 subject. also store gpa in student list, add a function in student class to return GPA of a student

- Calculate GPA() gpa=(1/3)*m1+(1/2)*m2+(1/4)*m3

Create list to store Multiple students.

1. Display All Student
2. Search by id
3. Search by name
4. Calculate GPA of a student
5. Exit

In [5]:
```python
class StudentGPA:
    """Stores studentid, sname, m1, m2, m3 marks, and can calculate GPA."""
    def __init__(self, studentid, sname, m1, m2, m3, gpa=0.0) -> None:
        self.studentid = studentid
        self.sname = sname
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3
        self.gpa = gpa # Stored initially, or updated via Calculate_GPA()

    def __str__(self) -> str:
        return f'ID: {self.studentid}, Name: {self.sname}, Marks: (M1:{self.m1}, M2

    def Calculate_GPA(self):
```

```
        """Calculates GPA based on the formula and returns the value."""
        # gpa = (1/3)*m1 + (1/2)*m2 + (1/4)*m3
        gpa_value = (1/3) * self.m1 + (1/2) * self.m2 + (1/4) * self.m3
        self.gpa = round(gpa_value, 2)
        return self.gpa

@staticmethod
def accept_gpa_student_data():
    """Static method to accept student data for the GPA class."""
    try:
        studentid = int(input("Enter Student ID: "))
        sname = input("Enter Student Name: ")

        # Function to safely get mark input
        def get_mark(mark_name):
            while True:
                try:
                    mark = int(input(f"Enter {mark_name} mark (0-100): "))
                    if 0 <= mark <= 100:
                        return mark
                    else:
                        print("❗ Mark must be between 0 and 100.")
                except ValueError:
                    print("❗ Invalid input! Mark must be a number.")

        m1 = get_mark("M1")
        m2 = get_mark("M2")
        m3 = get_mark("M3")

        # Create student object and immediately calculate GPA
        new_student = StudentGPA(studentid, sname, m1, m2, m3)
        new_student.Calculate_GPA()
        return new_student

    except ValueError:
        print("❗ Invalid input during entry. Student data not created.")
        return None
```

In [6]:

```
def menu_driven_program_gpa():
    """Menu-driven program for student GPA management."""
    student_list = []

    # Pre-populate list with a few students for testing
    s_test1 = StudentGPA(201, "Diana Prince", 80, 90, 70)
    s_test1.Calculate_GPA()
    s_test2 = StudentGPA(202, "Clark Kent", 60, 70, 80)
    s_test2.Calculate_GPA()
    student_list.extend([s_test1, s_test2])

    while True:
        print("\n==================================")
        print("      GPA Student Management")
        print("==================================")
        print("0. Add New Student Record")
        print("1. Display All Students")
        print("2. Search by ID")
```

```
        print("3. Search by Name")
        print("4. Calculate/Recalculate GPA of a student")
        print("5. Exit")

        choice = input("Enter your choice (0-5): ")

        if choice == '0':
            print("\n--- Adding New Student ---")
            new_student = StudentGPA.accept_gpa_student_data()
            if new_student:
                student_list.append(new_student)
                print(f"✅ Student {new_student.sname} added with GPA: {new_student

        elif choice == '1':
            if not student_list:
                print("List is empty. No students to display.")
                continue
            print("\n--- All Student Records ---")
            for student in student_list:
                print(student)

        elif choice == '2':
            try:
                search_id = int(input("Enter Student ID to search: "))
                found = False
                for student in student_list:
                    if student.studentid == search_id:
                        print("\n--- Student Found by ID ---")
                        print(student)
                        found = True
                        break
                if not found:
                    print(f"Student with ID {search_id} not found.")
            except ValueError:
                print("❗ Invalid input. ID must be a number.")

        elif choice == '3':
            search_name = input("Enter Student Name to search: ").strip().lower()
            found_students = [s for s in student_list if search_name in s.sname.low

            if found_students:
                print(f"\n--- Students Found for '{search_name}' ---")
                for student in found_students:
                    print(student)
            else:
                print(f"No student found with name containing '{search_name}'.")

        elif choice == '4':
            try:
                search_id = int(input("Enter Student ID to recalculate GPA: "))
                found = False
                for student in student_list:
                    if student.studentid == search_id:
                        new_gpa = student.Calculate_GPA()
                        print(f"✅ GPA for {student.sname} (ID: {student.studentid]
                        found = True
```

```
                        break
                if not found:
                    print(f"Student with ID {search_id} not found.")
            except ValueError:
                print("❗ Invalid input. ID must be a number.")

        elif choice == '5':
            print("Exiting GPA Management Program. Goodbye! 👋")
            break

        else:
            print("❗ Invalid choice. Please enter a number between 0 and 5.")

# To run the GPA program:
menu_driven_program_gpa()
```

```
==================================
      GPA Student Management
==================================
0. Add New Student Record
1. Display All Students
2. Search by ID
3. Search by Name
4. Calculate/Recalculate GPA of a student
5. Exit

--- All Student Records ---
ID: 201, Name: Diana Prince, Marks: (M1:80, M2:90, M3:70), GPA: 89.17
ID: 202, Name: Clark Kent, Marks: (M1:60, M2:70, M3:80), GPA: 75.00

==================================
      GPA Student Management
==================================
0. Add New Student Record
1. Display All Students
2. Search by ID
3. Search by Name
4. Calculate/Recalculate GPA of a student
5. Exit

--- Student Found by ID ---
ID: 201, Name: Diana Prince, Marks: (M1:80, M2:90, M3:70), GPA: 89.17

==================================
      GPA Student Management
==================================
0. Add New Student Record
1. Display All Students
2. Search by ID
3. Search by Name
4. Calculate/Recalculate GPA of a student
5. Exit
✅ GPA for Diana Prince (ID: 201) recalculated: 89.17

==================================
      GPA Student Management
==================================
0. Add New Student Record
1. Display All Students
2. Search by ID
3. Search by Name
4. Calculate/Recalculate GPA of a student
5. Exit
Exiting GPA Management Program. Goodbye! 👋
```

Missing And Repeating

Given an unsorted array arr[] of size n, containing elements from the range 1 to n, it is known that one number in this range is missing, and another number occurs twice in the array, find both the duplicate number and the missing number.

Examples:

Input: arr[] = [2, 2]
Output: [2, 1]
Explanation: Repeating number is 2 and the missing number is 1.

Input: arr[] = [1, 3, 3]
Output: [3, 2]
Explanation: Repeating number is 3 and the missing number is 2.

Input: arr[] = [4, 3, 6, 2, 1, 1]
Output: [1, 5]
Explanation: Repeating number is 1 and the missing number is 5.

In [7]:
```
def RepeatingAndMissingNumbers(arr):
    n=arr_len=len(arr)
    m=0
    r=0

    s=(n*(n+1))//2 #expected_sum
    s2=(n*(n+1)*(2*n+1))//6 #expected_sum_sq
    arrs=sum(arr) #arr_sum
    arrs2= sum(i**2 for i in arr) #arr_sum_sq
    d1=arrs-s
    d2=arrs2-s2

    m=int(((d2/d1)-d1)//2)
    r=int(((d2/d1)+d1)//2)

    return (r,m)

arr=[1,2,3,5,5,6,7,8,9]
RepeatingAndMissingNumbers(arr)
```

Out[7]: (5, 4)

Peak element

You are given an array arr[] where no two adjacent elements are same, find the index of a peak element. An element is considered to be a peak if it is greater than its adjacent elements (if they exist).

If there are multiple peak elements, Return index of any one of them. The output will be "true" if the index returned by your function is correct; otherwise, it will be "false".

Note: Consider the element before the first element and the element after the last element to be negative infinity.

Examples :
Input: arr = [1, 2, 4, 5, 7, 8, 3]

Output: true
Explanation: arr[5] = 8 is a peak element because arr[4] < arr[5] > arr[6].

Input: arr = [10, 20, 15, 2, 23, 90, 80]
Output: true
Explanation: Element 20 at index 1 is a peak since 10 < 20 > 15. Index 5 (value 90) is also a peak, but returning any one peak index is valid.

In [8]:
```
def peak(arr):
    n=len(arr)

    # For array size 1
    if n==1:
        return 0
    # if the 1st value is peak
    if arr[0]>arr[1]:
        return 0

    # if the last value is peak
    if arr[n-1]>arr[n-2]:
        return n-1

    # for array size != (1 or 2)
    # the iteration should not iterate in the polar elements
    # therefore the range(1 to n-1)
    iterr=range(1,n-1)
    for i in iterr:
        # if the element is greater than max of both,
        # it will be greater than both of them
        if arr[i] > max(arr[i-1], arr[i+1]):
            return i
```

In [12]:
```
arr=[1,2,4,5,6,8,4]
peak(arr)
```

Out[12]: 5

In [13]:
```
arr=[10, 2, 4, 5, 6, 9, 6]
peak(arr)
```

Out[13]: 0