

Student Marks Management System

Objective: Build a Python application for managing student records, including the ability to calculate marks, handle exceptions, process data from files, and perform various operations using functions, OOP, and lists.

Requirements:

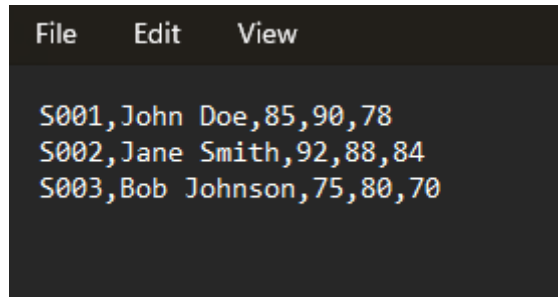
1. Class Definition (OOP)

- **Class: Student**
 - Attributes:
 - student_id (string)
 - name (string)
 - marks (list of integers)
 - Methods:
 - `__init__(self, student_id, name)`: Initializes the student's ID, name, and an empty list of marks.
 - `add_marks(self, marks)`: Adds a marks to the marks list.
 - `calculate_average(self)`: Returns the average of the marks list.
 - `__str__(self)`: Returns a formatted string containing the student's name, ID, and average marks.

2. Functions

Function 1: `read_students_from_file(filename)`

- Reads student data from a file and returns a list of Student objects.
- The file format is:



```
File Edit View
S001,John Doe,85,90,78
S002,Jane Smith,92,88,84
S003,Bob Johnson,75,80,70
```

Function 2: `show_summary_of_student(filename, students)`

- show the summary of each student (student ID, name, current marks ,average marks)

Function 3: `add_new_student(filename, student_id, name)`

- Adds a new student with the given student_id and name to the file. Initially, the student will have an empty list of marks.

Function 4: `update_student_marks(filename , student_id , new_marks)`

- Updates a student's marks in the file by student_id. If the student exists, append the new marks to their record.

Function 5: `calculate_class_average(students)`

- Takes a list of Student objects and calculates the overall average marks of the class.

Function 6: `find_top_performing_student(students)`

- Finds the student with the highest average marks in the class and returns their details.

Function 7: `validate_marks_input(mark)`

- Takes a marks input and validates if it's an integer between 0 and 100. If the marks is invalid, it raises a ValueError.

3. Main Program

- **Menu System:**

- Prompt the user with a menu of options:
 - 1. Add new student
 - 2. Add marks to an existing student
 - 3. View student summary
 - 4. Calculate class average
 - 5. Find top performing student
 - 6. Exit
- Implement a loop that keeps showing the menu until the user chooses to exit.

- **Student Operations:**

- When the user selects option 1, prompt them for the student ID and name and add the student to the file.
- When the user selects option 2, prompt for the student ID, the marks to add, and update the student's marks.
- When the user selects option 3, display the student's name, ID, marks, and average marks.
- When the user selects option 4, call `calculate_class_average()` and display the average marks for the class.
- When the user selects option 5, call `find_top_performing_student()` and display the details of the highest performing student.

4. Exception Handling

- Handle the following exceptions:
 - **FileNotFoundError**: If the file is not found, prompt the user to check the filename.
 - **ValueError**: If marks are invalid (non-integer or out of range), display an error message.
 - **IndexError**: Handle situations where there's no such student in the file.

5. String and Arithmetic Operations

- Use string manipulation for parsing input (e.g., split by commas for file data).
- Use arithmetic operations to calculate the average, highest, and lowest marks.

6. File Processing

- Read and write to a file containing student data.
- Each student record is represented as:
student_id,name,marks1,marks2,...
- When marks are updated, they should replace the old marks in the file, not append them.

1. Test Case: Add New Student

- **Description:** Verifies that a new student is successfully added to the file.
- **Test Input:**
 - Student ID: "S123"
 - Student Name: "Alice"

2. Test Case: Add Marks to an Existing Student

- **Description:** Verifies that marks can be added to an existing student's record.
- **Test Input:**
 - Student ID: "S123"
 - New Marks: [85, 90, 92]

3. Test Case: View Student Summary

- **Description:** Verifies that the summary of a student's marks is correctly displayed.

4. Test Case: Invalid Marks Input

- **Description:** Verifies that an invalid marks input (e.g., non-integer or out of range) raises a ValueError.
- **Test Input:**
 - Marks: "abc"

5. Test Case: Calculate Class Average

- **Description:** Verifies that the class average is calculated correctly.

6. Test Case: Find Top Performing Student

- **Description:** Verifies that the top-performing student is identified correctly based on average marks.

7. Test Case: Handle File Not Found Error

- **Description:** Verifies that a FileNotFoundError is raised when attempting to read from a non-existent file.

- **Test Input:**
 - Filename: "non_existent_file.txt"

8. Test Case: Update Student Marks and Replace Old Marks

- **Description:** Verifies that the marks are replaced (not appended) when updating a student's marks.
- **Test Input:**
 - Student ID: "S123"
 - New Marks: [85, 90, 92]

Submission Instructions

Please submit the following materials as part of your assignment:

1. **Word Document:** Create a single Word document that includes:
 - **Code Text:** The code for the test cases you have written.
 - **Screenshots:** Include screenshots showing the execution results of the test cases with the output clearly visible.
2. **Code Files:** Along with the Word document, please submit the actual **code files** (e.g., .py,) containing the test cases and any associated scripts.