Extra-A: Student Gradebook CLI

Extra-A: Student Gradebook CLI – Design and Testing Report

1. Objectives

The objective of this Extra-A task is to design and implement a simple command-line application that can manage a student gradebook. The program should let a student add, update, delete, and view course records, store them persistently in a file, and calculate weighted average scores (0–100) overall and by semester. The design aims to be easy to understand, robust for basic input errors, and convenient for future extension.

2. Data Model and Storage Design

The application stores the gradebook as a list of course dictionaries in a JSON file named gradebook.json. Each course record contains the following fields: course code (string), course name (string), number of credits (integer), semester (string, such as '2025-Fall'), and score (integer from 0 to 100). JSON is chosen because it is human-readable, supported by Python's standard library, and easy to parse and update. When the program starts, it tries to load the JSON file. If the file is missing or invalid, it starts with an empty gradebook but still allows the user to add new records.

3. Program Structure and Functions

The main file gradebook.py is organized into small, reusable functions. The load_gradebook and save_gradebook functions are responsible for reading and writing data to gradebook.json. Helper functions such as input_non_empty, input_positive_int, and input_score validate user input. They ensure that a course code is not empty, the number of credits is a positive integer, and the score is in the range 0–100. Functions add_course, update_course, delete_course, and view_gradebook implement the core CRUD operations. The function find_course_index searches for a course by its code in a case-insensitive way.

The program also includes functions calculate_overall_gpa and calculate_gpa_by_semester. Even though the name uses 'GPA', the logic calculates a weighted average score from 0 to 100. For each course, the score is multiplied by its credits to get quality points. The overall average score is computed as the sum of all quality points divided by the sum of all credits. For the per-semester view, a dictionary groups courses by semester and performs the same calculation for each group.

4. User Interface Design (CLI Menu)

The user interacts with the program through a simple text-based menu. The print_menu function shows six options: add a course, update a course, delete a course, view gradebook, show GPA summary, and exit. The main loop reads the user's choice and calls the corresponding function. This design helps keep main short and readable. Each option prints clear feedback messages, for example when the user tries to add a course with a duplicate code or update a course that does not exist. Before exiting, the application saves the current gradebook to the JSON file.

5. Usage Examples

Typical usage starts with running python gradebook.py from the terminal. A student might

choose option 1 to add three courses: CS101 with 3 credits and a score of 85, MATH201 with 4 credits and a score of 78, and ENG150 with 2 credits and a score of 92. After adding, the student can choose option 4 (View gradebook) to see a formatted table with code, name, credits, semester, and score. Option 2 (Update a course) can be used to correct a mistake, such as changing the score from 78 to 80. Option 3 (Delete a course) removes a course permanently after a confirmation step.

When the user selects option 5 (Show GPA summary), the program prints the overall weighted average score and then lists the average score for each semester. For example, the output might show an overall score of 84.37 and separate averages for '2025-Spring' and '2025-Fall'. This gives the user a quick summary of their performance across the whole degree and per semester.

6. Testing and Results

The program was tested with several scenarios: empty gradebook, adding multiple courses, updating existing courses, deleting a course, and calculating averages. Input validation was checked by entering invalid values such as negative credits, scores over 100, or non-numeric strings. In these cases, the program prints an error message and asks the user to try again, which prevents invalid data from entering the gradebook.json file. The JSON file was opened manually to confirm that the structure is correct and that all changes are saved properly across program runs.

7. Limitations and Future Improvements

The current version is intentionally simple. It only supports numeric scores from 0 to 100 and does not convert them to letter grades. It also assumes that the user runs the program in a terminal and is comfortable with typing commands. In the future, the program could be extended with support for letter grades, customizable grading scales, course categories (for example, major versus elective), and a more advanced reporting system. A graphical user interface or a web front-end could also be added on top of the same JSON-based data model.