VIT®

BHOPAL

www.vitbhopal.ac.in

Name : Eshita Parihar

Registration Number : 25BAI10698

Submitted to : Dr. Vinesh Kumar

Slot : E21 + A24 + F22

Project Report

Student Manager — A CLI-based Student Performance System

Abstract

This project implements a simple, robust command-line Student Manager written in Python. It lets a teacher or administrator add students, update grades and attendance, calculate weighted GPAs, and flag at-risk students (attendance < 75%). Data persistence is implemented using a CSV file with JSON-encoded fields for complex values (grades and attendance). The system emphasizes input validation, readable reports, and easy extensibility.

Table of Content

Introduction

Managing student records — grades and attendance — is a common administrative task. This Student Manager provides a minimal, easy-to-use, text-based interface to store student details, update their scores and attendance, generate formatted reports, and identify students at risk due to low attendance. The program stores persistent data in a CSV file while keeping structured fields (grades and attendance) JSON-encoded inside the CSV.

Objectives

‣ Provide simple CLI operations to add students, update grades and attendance, and display reports.

‣ Persist student records across runs using a CSV file.

‣ Calculate weighted GPA from multiple graded assignments.

‣ Calculate attendance percentage and flag students with attendance below a threshold (75%).

‣ Validate user input robustly to avoid corrupted or invalid data.

‣ Keep code modular, readable, and easy to extend.

Tools and Technologies

- **Programming Language:** Python 3.x
- **Standard Libraries Used:** csv, json, os
- **Storage Format:** CSV file (student_data.csv) with JSON-encoded grades and attendance fields
- **Execution Environment:** Command-line / Terminal

System Overview

The Student Manager is a CLI (command-line interface) application. When started, it loads existing data from student_data.csv (if present), decoding the grades and attendance JSON strings into Python objects. The user navigates a simple menu to perform operations. After changes, data is serialized and saved back to the CSV file.

High-level flow:

1. Load data from disk (if available).
2. Show menu with choices: add student, update grade, update attendance, display report, flag at-risk students, exit.
3. Execute chosen action, validate inputs, persist changes (where appropriate).
4. Repeat until exit.

Data Model and File Format

```
student = {
    'id': 'student_id_string',
    'name': 'Student Name',
    'grades': {
        'assignment_name': {'score': float, 'weight': float},
        ...
    },
    'attendance': {'attended': int, 'total': int}
}
```

CSV Storage (student_data.csv) — columns:

- Id — student ID (string)
- name — student name (string)
- grades — JSON string of the grades object
- attendance — JSON string of the attendance object

Functional Requirements & Features

- **Load existing records** from student_data.csv on startup.
- **Add student**: add a unique student ID and name; initialize empty grades and zeroed attendance.
- **Update grade**: add/update an assignment (score and weight). Score must be 0–100, weight must be 0–1.
- **Update attendance**: update attended and total classes. Attendance values validated (non-negative and total ≥ attended).
- **Display report**: list all students with weighted GPA (0–100) and attendance percentage in a formatted table.
- **Flag at-risk students**: list students whose attendance percentage is below the threshold (75%).
- **Save data**: persist the STUDENTS list to CSV after changes.

Non-functional Requirements

- **Usability:** Simple CLI menu with clear prompts.
- **Robustness:** Input validation to guard against invalid numeric values.
- **Maintainability:** Modular functions with clear responsibilities (load_data, save_data, add_student, etc.).
- **Portability:** Pure Python standard-library implementation — runs anywhere Python 3 is available.
- **Performance:** Sufficient for small-to-medium class sizes (tens to hundreds of students). CSV file read/write ensures fast startup and shutdown.

Detailed Module Description

load_data()

- Checks if student_data.csv exists.
- Uses csv.DictReader to read rows.
- Deserializes grades and attendance fields using json.loads.
- Populates global STUDENTS list.

save_data()

- Serializes grades and attendance via json.dumps.
- Writes a CSV header and rows using csv.DictWriter.
- Skip saving if STUDENTS is empty (to avoid creating empty files unnecessarily).

get_input_validated(prompt, validator, error_msg)

- Generic input/validation loop.
- Accepts a validator function that returns the validated value or raises on invalid input.
- Provides a reusable mechanism used for numeric parsing and validation.

find_student(student_id)

- Returns matching student dict by id or None.

add_student()

- Prompts for ID and name.
- Checks uniqueness of ID before adding.

update_grade()

- Prompts for student ID, assignment name, score (0–100), and weight (0–1).
- Stores assignment in student['grades'].

update_attendance()

- Prompts for student ID and updates attended and total classes (validated).

calculate_gpa(student)

- Weighted average = (sum of score * weight) / (sum of weights)
- If no grades exist or total weight is 0 → returns 0.0.

get_attendance_pct(student)

- Returns 0.0 when total is 0 to avoid division by zero.
- Otherwise returns attended / total * 100.

display_report() and flag_at_risk_students()

- Format and print tables sorted by student ID.

Input Validation & Error Handling

● Input parsing uses get_input_validated to transform user input to the required type and range.

● Score validation: numeric (float) and within 0–100.

● Weight validation: numeric (float) within 0–1.

● Attendance validation: integers, attended ≥ 0 and total ≥ attended.

● File I/O wrapped in try/except to gracefully handle missing files or read/write errors.

● find_student() guards operations that require an existing student; user notified when ID not found.

Algorithms & Calculations

Weighted GPA

- For each assignment: multiply score by weight.
- Sum weighted scores; sum weights.
- Weighted GPA = total_weighted_score / total_weight if total_weight > 0, else 0.0.
- This produces a GPA value on a 0–100 scale (not a 4.0-scale GPA).

Attendance Percentage

- If total == 0: return 0.0 (no classes recorded).
- Else: attendance_pct = attended / total * 100.

At-Risk Determination

- A student is at risk when attendance_pct < 75.0.

Sample and Data Output

Example: Adding a student

```
Enter new student ID: S001
Enter new student name: John Doe
Student added successfully.
```

Example: Update grade

```
Enter student ID to update grade: S001
Enter assignment name: hw1
Enter score (0-100): 85
Enter weight (0-1): 0.3
Grade for John Doe updated.
```

Example Report Display

```
--- Full Student Report ---
ID         | Name                 | Weighted GPA (%)     | Attendance (%)
----------------------------------------------------------------------
S001       | John Doe             | 85.00                | 90.00
----------------------------------------------------------------------
```

Example At-Risk Output

```
--- At-Risk Students (Attendance < 75%) ---
ID         | Name                 | Attendance (%)
-------------------------------------------------------
No students are currently at-risk.
-------------------------------------------------------
```

Testing Strategy & Test Cases

Unit / Functional Test Cases

1. **Load with missing file**: Ensure program starts with empty STUDENTS and prints a helpful message.
2. **Add student**: Add student with unique ID — verify STUDENTS updated and file saved.
3. **Duplicate ID**: Try adding same ID twice — program should reject the second attempt.
4. **Update grade validation**: Input non-numeric or out-of-range scores/weights → validator should prompt error and retry.
5. **Update attendance validation**: Negative numbers or total < attended should be rejected.
6. **GPA calculation**: Multiple assignments with various weights sum correctly; total weight 0 returns 0.0.
7. **Attendance edge cases**: total = 0 returns 0% without crashing.
8. **Persistence**: After save, reopen the app and confirm data loads identically.

Manual / Integration Tests

- Add multiple students, provide several graded assignments and attendance values, then check display_report() sorting and formatting.
- Corrupt CSV row intentionally to check load_data() error handling.

Limitations

● **Single-file storage:** CSV with JSON fields is simple but not optimal for concurrent access, large datasets, or complex queries.

● **CLI-only:** No GUI; not ideal for non-technical users.

● **No authentication or multi-user support.**

● **No versioning/backup** of stored data; accidental overwrites could cause data loss.

● **GPA scale fixed to 0–100** — not adaptable to different grading schemes out-of-the-box.

● **No constraint enforcement across weights:** weights for different assignments are independent (no requirement that weights sum to 1).

Future Enhancements

● Add a small GUI (Tkinter or web-based with Flask) to make interface friendlier.

● Migrate storage to SQLite for safer persistence, queries, and concurrency handling.

● Add import/export features (Excel, JSON).

● Implement user authentication and role-based access (teacher/admin).

● Add bulk operations (import students in batch).

● Add analytics: class averages, grade distributions, trend graphs.

● Add automated backups and versioning on save.

● Enforce or validate cumulative assignment weights (optional normalization).

● Add unit tests and CI (pytest) for automated testing.

Conclusion

The Student Manager provides a compact, practical solution to store and manage student grades and attendance. It demonstrates correct use of Python file I/O, JSON serialization within CSV, input validation, and modular design. While suited for small classes and educational purposes, the design is intentionally simple to keep the code readable and easily extensible.

Appendix — Sample CSV Format & Example

Header row:

id, name, grades, attendance

Example row (CSV cell values shown conceptually):

- id: S001
- name: John Doe
- grades: {"hw1": {"score": 85.0, "weight": 0.3}, "midterm": {"score": 72.0, "weight": 0.7}}
- attendance: {"attended": 18, "total": 20}

Stored line (escaped JSON inside CSV):

S001,John Doe,"{\"hw1\": {\"score\": 85.0, \"weight\": 0.3}, \"midterm\": {\"score\": 72.0, \"weight\": 0.7}}","{\"attended\": 18, \"total\": 20}"