Smart Study Scheduler and Task Prioritizer

Student ID: 25BAI10789
Course: Introduction to Problem Solving & Programming
November 2025

1. Problem Definition

Let's be real—juggling school can get overwhelming fast. Most students have to keep track of classes, homework, deadlines, revision plans, and extracurriculars, all at once. Without a simple, organized way to see and sort their to-dos, it's easy to get lost. You forget things, pick the wrong task to work on, or just miss deadlines altogether. What students really need is more than a digital notepad. They need a tool that actually helps them figure out what's next, using real factors like urgency and importance—not just a list of tasks.

2. Requirement Analysis

We broke down what this system has to do and how well it needs to work.

2.1 Functional Requirements

- Task Creation: Users can add a task with a name, subject, due date (YYYY-MM-DD), and pick a priority (HIGH, MEDIUM, LOW).
- Task Listing: The system shows all tasks that haven't been finished yet.
- Prioritization: Tasks show up sorted—first by priority (HIGH to LOW), then by which deadline is coming up next.
- Task Completion: You can mark any unfinished task as done.
- Data Persistence: The task list saves automatically to a file when you quit and loads back up when you start.

2.2 Non-Functional Requirements

- Usability: It all runs in a simple, menu-based command-line interface.
- Maintainability: The code's clean, modular, and follows Object-Oriented Programming (OOP) principles.
- Data Integrity: If something goes wrong with loading files (like if a file's missing), the program doesn't crash—it handles it smoothly.

3. Top-Down Design and Modularization

The design separates everything out, top-down style. The core logic, data handling, and file saving stuff all live in their own spaces.

- Data Model Layer: The Task class represents one task.
- Persistence Layer: Functions like load_tasks and save_tasks deal with reading and writing data to tasks.csv.

- Application Layer: The main loop and functions—add_task, view_tasks, mark_complete—handle what the user sees and interacts with.

Because of this setup, if you ever switch from CSV to, say, JSON for storage, you don't have to rewrite the main logic.

3.1 Task Class Design

The Task class is at the heart of the OOP design.
- Attributes: name, subject, due_date (as a datetime object), priority, is_completed.
- Methods: __init__ (handles creating and checking the date), __str__ (makes the task easy to read), and to_csv_row (prepares data for saving).

4. Algorithm Development: Prioritization Logic

The main algorithm lives inside the view_tasks function. It sorts tasks using two clear steps, leaning on Python's stable sorting.

4.1 Prioritization Steps

1. Filter: Only tasks that aren't finished show up.
2. Sort by Due Date: First, it sorts by which deadline is soonest.
3. Sort by Priority: Then, it uses a custom map (HIGH = 0, MEDIUM = 1, LOW = 2) to sort so HIGH comes first.

Because Python's sort is stable, sorting by date first means that tasks with the same priority stay in the right order by due date. So, if two tasks are both HIGH priority, the one due sooner comes first.

5. Implementation and Key Concepts

The whole thing runs in Python, using only standard libraries.

- OOP: The Task class keeps task objects neat and easy to work with.
- File I/O and CSV: The csv library handles saving and loading tasks.csv. Dates are saved as "YYYY-MM-DD" strings but used as datetime objects behind the scenes, so they sort correctly.
- Custom Sorting: attrgetter and a lambda function, plus stable sort, make sure tasks get prioritized exactly right.

6. Testing and Refinement

Testing focused on making sure all these features work, especially the prioritization and data saving parts.

6.1 Test Case Examples

- Test Case 1 (Date Integrity):
  - Input: Task 1—Essay, due 2025-11-20, MEDIUM. Task 2—Review, due 2025-11-05, HIGH.
  - Expected: Task 2 (Review) should come first, since HIGH beats MEDIUM, even though its date is later.
  - Actual: Works as expected. Review comes before Essay.

- Test Case 2 (Prioritization Stability):
  - Input: Task A—Meeting Prep, date and details continue…

7 Conclusion

The Smart Study Scheduler project pulls together key programming skills—Object-Oriented Design, data serialization, and stable sorting algorithms—to tackle a real problem students face: staying on top of their work. The command-line app that comes out of this isn't just a proof of concept; it's a solid, persistent, and well-prioritized task manager. It shows what's possible with the tools and ideas covered in the course. Because the design is modular, it's easy to update and expand. Down the line, you can add things like multi-user support or even a graphical interface without having to rewrite the whole thing.