# ACIT4420 - Assignment 2

# Automated Study Reminders for Students Using Python

Preethi Vijayakumar

## 1. Introduction

The objective of this assignment was to design and implement a Python package called ***study_reminders*** that automates the process of sending personalized study reminders to students. The system combines several scripting concepts; file handling, modular programming, scheduling, and logging; to demonstrate how automation can improve productivity and time management.

The project simulates a reminder system that manages student data, generates custom reminder messages, sends them automatically based on schedules, and logs all activities for record-keeping.

## 2. Methods

The solution was implemented as a Python package named ***study_reminders***, composed of multiple modules that each handle a specific part of the workflow. A modular design was chosen to ensure reusability, clarity, and ease of maintenance.

### Modules Overview

- ***students.py*** – Defines a Students class to manage an in-memory list of students, allowing adding, removing, and retrieving student information.
- ***students_manager.py*** – Handles persistent data storage using a JSON file (students.json). The module automatically loads and saves student data, ensuring that any additions or deletions are retained between runs.
- ***reminder_generator.py*** – Generates personalized reminder messages for each student. Each message combines the student's name and course into a clear and friendly reminder, for example:
  *"Hi Alice, remember to review Python Scripting materials before the deadline!"*
- ***reminder_sender.py*** – Simulates sending reminders to students via console output. Although no actual email is sent, this step demonstrates message delivery automation.
- ***logger.py*** – Records each sent reminder into a log file (***reminder_log.txt***) with a precise timestamp, documenting every event for accountability and debugging.
- ***scheduler.py*** – Utilizes the schedule library to automatically trigger reminders at specific times defined in the JSON file. For testing, the real-time scheduling was replaced with a 30-second simulated loop to demonstrate automated execution safely within a short runtime.
- ***main.py*** – Integrates all the above modules to execute the complete automation process sequentially.

**Testing and Validation**

To ensure robustness, a ***unit testing script (test.py)*** was implemented using Python's unittest framework. Each module was tested individually to confirm that:

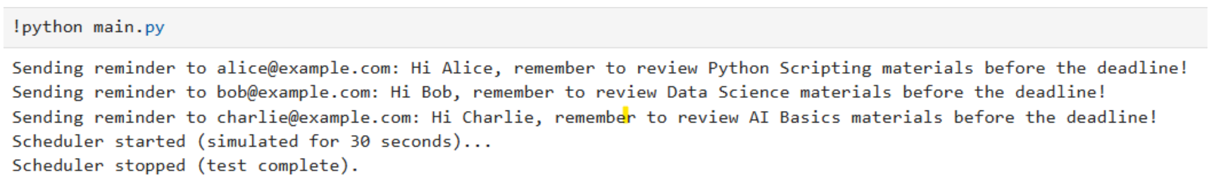
- Students could be added, removed, and retrieved correctly.
- Reminder messages were generated in the expected format.
- Logging created valid entries in reminder_log.txt.
- Scheduler invoked reminders at the expected times.

The project also included a ***README.md*** for documentation and a ***setup.py*** file to allow package installation using pip.

## 3. Results

The system successfully simulated the entire reminder workflow. When executed, main.py printed personalized reminders for each student listed in ***students.json*** and created a timestamped record of each event in ***reminder_log.txt***.

Example console output:



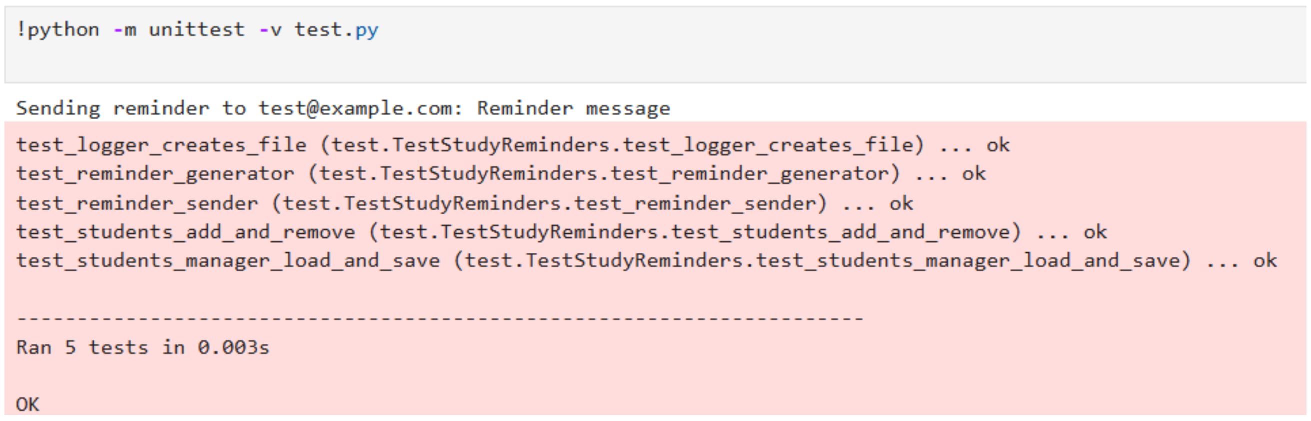
```
!python main.py

Sending reminder to alice@example.com: Hi Alice, remember to review Python Scripting materials before the deadline!
Sending reminder to bob@example.com: Hi Bob, remember to review Data Science materials before the deadline!
Sending reminder to charlie@example.com: Hi Charlie, remember to review AI Basics materials before the deadline!
Scheduler started (simulated for 30 seconds)...
Scheduler stopped (test complete).
```

The **log file** entries were recorded in the following format:

**2025-10-15 08:00:00 - Sent to Alice: Hi Alice, remember to review Python Scripting materials before the deadline!**

All **unit tests** passed successfully:



```
!python -m unittest -v test.py

Sending reminder to test@example.com: Reminder message
test_logger_creates_file (test.TestStudyReminders.test_logger_creates_file) ... ok
test_reminder_generator (test.TestStudyReminders.test_reminder_generator) ... ok
test_reminder_sender (test.TestStudyReminders.test_reminder_sender) ... ok
test_students_add_and_remove (test.TestStudyReminders.test_students_add_and_remove) ... ok
test_students_manager_load_and_save (test.TestStudyReminders.test_students_manager_load_and_save) ... ok

----------------------------------------------------------------------
Ran 5 tests in 0.003s

OK
```

These results confirm that all modules functioned correctly, with smooth integration between data handling, scheduling, and logging components.

## 4. Challenges Faced

Several challenges were encountered during development:

1. **JSON formatting errors:** Early versions of ***students.json*** caused decoding errors, resolved by ensuring proper structure and syntax.
2. **Scheduling format issues:** The schedule library required strict 24-hour time input (HH:MM), which initially led to invalid time format errors.
3. **Path and import problems:** While running in Jupyter Notebook, import paths sometimes failed due to directory differences between Windows and Python's relative imports. This was solved by carefully managing the working directory and package structure.
4. **Testing in simulated time:** Since the project couldn't run continuously, a time-limited scheduler was implemented to simulate real-time automation safely.

### Overall Reflection

This project helped me integrate key Python concepts; modules, classes, file handling, and scheduling; into one functional automation system. It demonstrated how scripting can simplify repetitive academic tasks and improve overall efficiency. I also strengthened my debugging and testing skills by resolving syntax, path, and integration issues during development.

## 5. Conclusion

The ***study_reminders*** package effectively automates the creation, scheduling, and simulated delivery of personalized study reminders. It showcases the practical use of modular programming, logging, and automation in Python. Overall, the project improved my technical understanding and illustrated how scripting can streamline academic and professional workflows.