OsloMet - Oslo Metropolitan University, Department of Computer Science, Norway
ACIT4420-1 25H Problem-solving with Scripting, Master in ACIT, 2025

(Assignment II: Files I/O, Modules, and Packages)

Shailendra Bhandari*

September 26, 2025

The aim of this assignment is to develop a Python package that automates the generation and distribution of personalized study reminders for students. The task requires creating modular components to manage student information, generate reminders, simulate delivery, log operations, and schedule automated reminders.

Assignment description

You are tasked with creating a Python package named `study_reminders` to automate sending personalized study reminders. The package must include multiple modules, each handling a specific aspect of the task, integrated to form a cohesive automation tool.

1 Automation task: Sending personalized study reminders

The automation process comprises the following steps:

1. **Manage a list of students:** Develop a module to manage student information, including names, contact details, course names, and reminder preferences. (use dummy info like Alice, alice@mail.com, Python scripting etc..)
2. **Generate personalized reminders:** Create a module to produce customized study reminders, including course-specific tasks and deadlines.
3. **Simulate sending reminders:** Implement a module to simulate sending reminders to students.
4. **Log the reminders:** Develop a module to record details of sent reminders, including timestamps.
5. **Schedule reminder delivery:** Integrate a scheduling mechanism to automate reminder delivery based on student preferences.

2 Requirements

2.1 Package structure

- Create a package named `study_reminders`.
- Include at least five modules, each responsible for a distinct step in the automation process.
- Ensure modules are independently importable and reusable.

*shailendra.bhandari@oslomet.no

2.2 Module implementation

1. **students.py:** Create a module to manage the list of students, supporting operations to add, remove, and retrieve student information (name, contact details, course, and preferred reminder time).

```python
# students.py
class Students:
    """Class to manage a list of students' information."""
    def __init__(self):
        self.students = []

    def add_student(self, name, contact_info, course, preferred_time="08:00 AM"):
        """Add a student with name, contact info, course, and preferred reminder time."""
        student = {
            'name': name,
            'contact_info': contact_info,
            'course': course,
            'preferred_time': preferred_time
        }
        self.students.append(student)

    def remove_student(self, name):
        """Remove a student by name."""
        self.students = [s for s in self.students if s['name'] != name]

    def get_students(self):
        """Retrieve the list of students."""
        return self.students
```

2. **reminder_generator.py:** Develop a module to generate personalized study reminders, such as "Hi [Name], remember to review [Course] materials before the deadline!"

```python
# reminder_generator.py
def generate_reminder(name, course):
    """Generate a personalized study reminder for the given name and course."""
    return f"Hi {name}, remember to review {course} materials before the deadline!"
```

3. **reminder_sender.py:** Implement a module to simulate sending reminders by printing them to the console with a note indicating delivery.

```python
# reminder_sender.py
def send_reminder(email, reminder):
    """Simulate sending a reminder to the specified email."""
    if not email:
        raise ValueError("Email address is missing")
    print(f"Sending reminder to {email}: {reminder}")
```

4. **logger.py:** Create a module to log sent reminders, including timestamps, to a text file.

```python
# logger.py
import datetime

def log_reminder(student, reminder):
    """Log a sent reminder with a timestamp to a file."""
    with open("reminder_log.txt", "a") as log_file:
        log_file.write(f"{datetime.datetime.now()} - Sent to {student['name']}:
    {reminder}\n")
```

5. **students_manager.py:** Develop a module to manage student data (names, emails, courses, and reminder times) in a structured format, with support for reading from a JSON file.

```python
# students_manager.py
import json

class StudentsManager:
    """Class to manage student data with JSON storage."""
    def __init__(self, file_path="students.json"):
        self.file_path = file_path
        self.students = self.load_students()

    def load_students(self):
        """Load student data from a JSON file."""
        try:
            with open(self.file_path, "r") as file:
                return json.load(file)
        except FileNotFoundError:
            return [
                {"name": "Alice", "email": "alice@example.com", "course": "Computer Science", "preferred_time": "08:00 AM"},
                {"name": "Bob", "email": "bob@example.com", "course": "Mathematics", "preferred_time": "09:00 AM"},
                {"name": "Charlie", "email": "charlie@example.com", "course": "Physics", "preferred_time": "07:30 AM"}
            ]

    def add_student(self, name, email, course, preferred_time="08:00 AM"):
        """Add a student and save to the JSON file."""
        student = {
            'name': name,
            'email': email,
            'course': course,
            'preferred_time': preferred_time
        }
        self.students.append(student)
        self.save_students()

    def remove_student(self, name):
        """Remove a student by name and update the JSON file."""
        self.students = [s for s in self.students if s['name'] != name]
        self.save_students()

    def save_students(self):
        """Save student data to the JSON file."""
        with open(self.file_path, "w") as file:
            json.dump(self.students, file, indent=4)

    def get_students(self):
        """Retrieve the list of students."""
        return self.students

    def list_students(self):
        """Print all students."""
        for student in self.students:
            print(f"Name: {student['name']}, Email: {student['email']}, Course: {student['course']}, Preferred Time: {student['preferred_time']}")
```

6. **scheduler.py:** Implement a module to schedule reminder delivery using the `schedule` library.

```python
# scheduler.py
```

```python
import schedule
import time

def schedule_reminders(students_manager, reminder_generator, reminder_sender, logger):
    """Schedule reminder delivery for each student at their preferred time."""
    for student in students_manager.get_students():
        reminder = reminder_generator(student['name'], student['course'])
        schedule.every().day.at(student['preferred_time']).do(
            lambda s=student, r=reminder: (reminder_sender(s['email'], r), logger(s, r)))

    while True:
        schedule.run_pending()
        time.sleep(60)  # Check every minute
```

2.3 Main script

- Develop a main script (`main.py`) that integrates all modules to perform the automation process: manage student data, generate personalized reminders, simulate sending, log operations, and schedule daily execution.
- Simulate daily automation using the `scheduler.py` module, allowing manual triggering for testing.

2.4 Logging

- Implement logging to track key events, such as reminder generation, sending, and scheduling, in the automation process.

2.5 Documentation

- Include docstrings and comments in each module to explain functionality.
- Provide a `README.md` file detailing installation and usage instructions for the `study_reminders` package.

2.6 Package installation

- Create a `setup.py` file to enable installation via `pip`.
- Demonstrate how to install the package locally and execute the automation task.

2.7 Testing

(Optional: Extra credit)

- Write unit tests for each module to verify functionality.
- Include a `test.py` script to run all tests.

3 Submission

- Submit the `study_reminders` package as a compressed zip file or via a GitHub repository link, including all source code and supporting files.
- Ensure the package includes all necessary files (`README.md`, `setup.py`, `main.py`, etc.) to enable installation and execution of the automation task.
- Provide a detailed report in a scientific format, describing the working mechanism of the `study_reminders` package, your development experience, and any challenges encountered during implementation.

4 Why this assignment?

- **Automated academic notifications:** The tool schedules and delivers personalized reminders to students about study tasks or deadlines, demonstrating how automation can enhance time management in educational settings, such as sending alerts for assignment due dates or exam preparations.
- **Task automation and logging**: It automates repetitive tasks by reading student data from a JSON file and logging actions to a text file, illustrating techniques for managing workflows and maintaining records, applicable in fields like system administration or project management.
- **Learning modular programming:** The program serves as a practical example to learn modular design, file handling, and error management in Python, preparing them for developing scalable automation solutions in academic or professional environments.

Good luck!