Assignment 2: Study Reminders

Course: ACIT4420

Student: Lars Skogen Johnsen - s359056

Date: October 17, 2025

Contents

Introduction

This report presents the second assignment in problem-solving with scripting for files Input/Output (I/O), modules, and packages. Methods were implemented for handling student information, where reminders could be sent to each student based on their course and preferred study time. Relevant events like generation and sending of reminders, or errors were logged. Finally, a setup file was added to allow for local package installation through `pip`. The GitHub repository for the assignment is found in Appendix A.

Methods

Initially the given modules were created in a utilities folder under `study_reminders`:

1. `logger.py`
2. `reminder_generator.py`
3. `reminder_sender`
4. `scheduler.py`
5. `students_manager.py`

2.1 Module additions

Additions were made to the logging module, adding two new functions with the same structure to also handle logging of the initial generation of reminders and errors in reminder generation. This enables the user to check that the reminder was correctly generated without changing times or adding new students to test, or adding separate methods for testing of the reminder generation.

A function to convert time formats to a 24-hour format was made using the python `datetime` module, highlighted in Listing 1 [1]. This ensures that any valid time format input, either in 12-hour format or military time is converted to the same format, raising an error in case of an invalid input.

```
def string_to_24hr(preferred_time):
    try:
        time_in = preferred_time.strip().replace(" ","")
        time_in = datetime.strptime(time_in, "%I:%M%p")
        time_out = time_in.strftime("%H:%M")
        return time_out
    except:
        raise ValueError(f"\n\n\nTime: {preferred_time} is an invalid time format!")
```

Listing 1: 24-hour format function

The logging module additions and the time conversion was applied to the schedule reminder under `scheduler.py`, also separating the schedule function to allow for error handling as displayed in Listing 2.

```
def schedule_reminder(students_manager, reminder_generator, reminder_sender, logger):
    for student in students_manager.get_students():
        preferred_times = string_to_24hr(student["preferred_time"])

        try:
            reminder = reminder_generator(student["name"], student["course"])
            log_reminder_generation(student, reminder)

        except Exception as e:
            log_reminder_error(student)

        def student_reminder(s=student):
            try:
                reminder = reminder_generator(s["name"], s["course"])
                reminder_sender(s["email"], reminder)
                logger(s, reminder)

            except Exception as e:
                log_reminder_error(s, e)

        schedule.every().day.at(preferred_times).do(student_reminder)
```

Listing 2: Schedule reminder function

2.2 Main script

A main script was created, importing the established modules from the utilities folder. After creation of the main script the `StudentsManager` class is initialized. A while loop is then established to allow for easy script testing, allowing for the different options in:

1. Clearing the JSON file with the saved student information.
2. Add new students to the JSON file.
3. Remove students from the JSON file.
4. Display the current list of students.
5. Initialize the automatic study reminder
6. Quit the program

Options 1-4 simply call on the applicable methods from the `StudentsManager` class for manual testing to add, remove or display, where the list is saved after every action. When initializing the study reminder the `schedule_reminder` function in Listing 2 is called, where the student manager object is used as the first function argument to access the student dictionary, highlighted in Listing 3.

```
% Where manager = StudentManager()
elif mode == '5':
            print("Automatic study reminders initialized, Ctrl+C to return to menu")
            schedule_reminder(manager,
                      generate_reminder,
                      send_reminder,
                      log_reminder)
```

Listing 3: Schedule reminder function

The subsequent arguments are simply the aforementioned modules.

2.3 Pip installation setup

By adding a setup file, displayed in Listing 4, including the package name, version, author, GitHub URL, and module locations with the wheel and build packages installed, `python -m build` can be executed to build a distribution folder including the tar & gzip and wheel file.

```
setup(name='studyreminders',
      version='1.0.1',
      author='Lars Johnsen',
      author_email='s359056@oslomet.no',
      url='https://github.com/LSJohnsen/ACIT4420-study-reminders',
      packages=['study_reminders', 'study_reminders.utils'],
      description='Automated study reminder package',
      entry_points={'console_scripts': ['study-reminders = study_reminders.main:main']})
```

Listing 4: Study reminder setup.

With these files it is possible to locally install the package by calling the locally by the wheel file location in the distribution folder, where the entire package including all modules can be initialized by calling `study-reminders` in the terminal due to the determined `console_scripts` entry point, or by importing the package in a script. As the package is not uploaded to the

Python Package Index (PyPI) by using `twine`, the package is not possible to install directly by `pip install study-reminders`.

2.3.1 Installation setup

Listing 5 presents the terminal inputs to install the `study_reminders` package.

```
> python setup.py sdist
> python -m build
> pip install dist/studyreminders-1.0.1-py3-none-any.whl
```

Listing 5: Terminal package installation

Results & Discussion

After installation the package is tested by calling it by its entry point name through the console as displayed in Listing 6:

```
> study-reminders

Choose an option to:
1) clear the JSON file containing ALL student info
2) add new students to the file
3) remove students from file
4) display the current list of students
5) initialize automatic study reminders
q) quit
>
```

Listing 6: Package test

By choosing option 2 a new student can be added, with the resulting addition to the JSON dictionary seen in Listing 7, highlighting its ability to use apply the module functions when run as a package.

```
{
        "name": "Lars",
        "email": "lars@example.com",
        "course": "Math",
        "preferred_time": "4:24PM"
    }
```

Listing 7: Added student result

The presented methods returns the expected result, integrating the different modules to a main module to effectively manage student information while automatizing the delivery of personalized email study reminders. The package built from the resulting design works as expected, allowing for a local install where the script can be easily called by the user by `study-reminders` in the terminal, or by `import study_reminders` in a python script.

Further work would focus on handling more edge cases as the current report mainly highlights integrating several modules together. Additionally more options for manual testing of the different modules could be added, however the current package does currently apply the main methods.

Bibliography

[1] Python Software Foundation, “datetime documentation,” 2020. Accessed on 2025-06-10.

GitHub Repository

Assignment repository hyperlink: GitHub Repository

if broken:
`https://github.com/LSJohnsen/ACIT4420-study-reminders`