**Project Report: Automated Student Feedback Generator**

**Peter Magliocco**

**7th October 2025**

**Activity 2.1 - Coding Fundamentals: Module 2**

Contents

[Introduction 3](#_Toc210766375)

[Design Choices 3](#_Toc210766376)

[Code Screenshots 4](#_Toc210766377)

[Program Execution 6](#_Toc210766378)

[Reflection and Improvements 7](#_Toc210766379)

[Project Brief 8](#_Toc210766380)

# **Introduction**

The original brief has been taken from Repo: https://github.com/chris-reeves1/python29-09-25 (report/report.md) and has been included in this report for reference.

The purpose of this project was to follow the brief and develop python code that automates a personal feedback document for students based on their performance scores.

The code was designed to interpret numeric scores ranging grom 1 to 4 and map them to predefined feedback phrases, which was repeated for all students.

# **Design Choices**

After reading the brief, I began by setting up my GitHub repository <https://github.com/MaglioccoBSI/student_feedback_plm.git> confirming I could successfully push and pull code. This ensured version control and proper tracking of my progress throughout.

I then started to breakdown the brief by first creating a student\_design.xlsx file. This spreadsheet helped me picture the overall results of what the table for each student would look like. The file included the **student’s name**, **score category**, **score,** and a key representing the **score value**, which would later be used to map to phrases in the feedback.

Next I created a sample\_wording.docx so I could visualise how the final feedback report for each student would read. This allowed me to define the structure and consistency of the generated feedback/reports.

Once I had a clear picture of the scoring and report generation process, I moved onto the code structure. I noticed that some of the data would be reused across parts of the program. Things like student names, score descriptions, and performance categories. To make these easily maintainable and extendable, I created a dedicated dictionaries subfolder containing individual .py files for each dataset.

This design decision supports reusability for future updates (e.g adding new students or categories) without changing the main program code.

I performed early testing by printing outputs from each dictionary to confirm correct data loading and structure. After validating these I then developed the main.py file as the program’s entry point. This imports the dictionary modules and guides the tutor/end user through the input process with clear printed instructions. I also included an introductory section in the terminal to ensure that the tutor/user understands each step of the feedback process, improving usability and clarity.

# **Code Screenshots**

Below are screenshots of my python code including the file structure of my codebase

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

# **Program Execution**

Below are screenshots of my python code running in Visual Studio Code.

This screen shot displays what the tutor/end user would see when following the on-screen promptsA screenshot of a computer program

AI-generated content may be incorrect.

This screenshot shows what the tutor/end user would see if they entered any incorrect scoreA screenshot of a computer program

AI-generated content may be incorrect.

This is a screenshot of the student feedback file that automatically opens for final edit/approval.

A screenshot of a document

AI-generated content may be incorrect.

# **Reflection and Improvements**

If I had more time I would focus on improving the flow to make the program more interactive and flexible for tutors. Currently, the process runs for all students, but my goal would be to allow the tutor to manually input and process each student individually. This would make the application more user friendly and adaptable to different class sizes or one off feedback sessions.

Another enhancement would be to integrate a database for storing and retrieving student information. Instead of relying solely on local files or dictionaries, using a database would make the system more scalable and efficient. Tutors could easily update student records, track historical performance, and generate reports. This approach would also support long term data storage and reduce manual file management.

# **Project Brief**

**Automated Student Feedback Generator**

The original brief has been taken from Repo: https://github.com/chris-reeves1/python29-09-25 (report/report.md)

**Overview**

---------------------------------------------------------------------------------------------------------------------------------

You are required to design and implement a Python program that generates personalised feedback for students based on a set of scores (1–4). The program should produce a feedback file for each learner using a consistent template and mapped phrases. Instead of submitting only the code, you will evidence your design and outputs in a report. The report should include screenshots, explanations, and reflections to demonstrate functionality.

**Requirements**

---------------------------------------------------------------------------------------------------------------------------------

- A list of students (at least 3 for demonstration)

- For each student, input scores (1–4) for the following attributes:

    - Overall performance

    - General understanding

    - Contribution level

    - Lab completion

    - Punctuality

    - Engagement

    - Further study level

- Use score to phrase mappings (dictionary lookups) to convert numeric values into appropriate sentences.

    (Example: performance\_map = {1: "basic", 2: "ok", 3: "good", 4: "excellent"})

- Apply the mappings inside a feedback template, for example:

    template = f"General Comments\n{student\_name} performed {performance\_map[overall]} in this module...\n\n"

    template += f"Punctuality\n{punctuality\_map[punctuality]}...\n\n"

    template += f"Further Learning\n{further\_study\_map[further\_study]}..."

- Save one feedback file per student in a folder called feedback/.

- File name format: feedback/<student\_name>\_feedback.txt.

- Automatically open the file in an editor for final manual edits/approval.

**Additional Requirements**

---------------------------------------------------------------------------------------------------------------------------------

- Code quality:

    - Clear naming conventions.

    - Readable structure with comments and docstrings.

    - Modular design (functions for mapping, file writing, etc.).

- Stretch goal:

    - Format CLI outputs with colours/spaces.

**Deliverables**

---------------------------------------------------------------------------------------------------------------------------------

- Report (Word or PDF) containing:

    - Explanation of your design choices.

    - Screenshots of your Python code.

    - Screenshots of the program running in the terminal (showing coloured output if implemented).

    - Screenshots of the generated feedback/ directory.

    - Screenshot of one feedback file open in an editor.

    - Reflection section explaining:

        - How your mapping design supports reusability.

        - What you would improve if given more time.

    note: you can include full code in an appendix.

You have a maximum of 3 hours to complete both the program and the report.

---------------------------------------------------------------------------------------------------------------------------------

**Example completed feedback:**

General comments

Mark did well in this module. He demonstrated a good understanding of concepts we covered and showed he has a broad knowledgebase.

He contributed to discussions and asked relevant questions.

Learner Punctuality and engagement

Mark was punctual throughout the module and engaged well through Webex.

Recommendations on further learning

Continue to practice the basics of containerisation, Kubernetes and pipelines.

Marking Rubric:

Program Functionality: 25%

Template Design & Mapping 15%

Code Quality 20%

Report Quality 20%

Reflection & Understanding 10%

Stretch CLI Formatting 10%