## **Student Assessment Submission and Declaration**

When submitting evidence for assessment, you must sign a declaration confirming that the work is your own.

|  |  |  |  |
| --- | --- | --- | --- |
| Student name: | Jesus Amaury Reynoso Bourdier | ESL ID No | 573907 |
| BSU ID No |  |
| Submission date: 10th February 2025 | | |  |
|  | | |  |
| Programme: |  | | |
| Module name and code: | (BSU COMP 1.2) **Software Foundations** | | |
| Title: | Mr | | |
| Assessor name: | **Khurram Asif** | | |

**Plagiarism**

Plagiarism is a form of cheating. Plagiarism must be avoided at all costs, and students who break the rules, however innocently, may be penalized. It is your responsibility to ensure that you understand correct referencing practices. As a university-level student, you are expected to use appropriate references throughout and keep carefully detailed notes of all your sources of materials for material you have used in your work, including any material downloaded from the Internet. Please consult the relevant unit lecturer or your course tutor if you need any further advice.

|  |  |  |  |
| --- | --- | --- | --- |
| **Student declaration**  I certify that the assignment submission is entirely my work. I fully understand the consequences of plagiarism. I know that making a false declaration is a form of malpractice. | | | |
| Student signature: |  | Date: | 10/02/2025 |

Table of Contents

[Student Assessment Submission and Declaration 1](#_Toc188469583)

[Development Document 3](#_Toc188469584)

[Introduction 4](#_Toc188469585)

[Solution Design 4](#_Toc188469586)

[Script Walkthrough and Technical Breakdown 4](#_Toc188469587)

[Batch Script Description 6](#_Toc188469588)

[How the Batch Script Works 6](#_Toc188469589)

[Personal Evaluation 6](#_Toc188469590)

[Improvements 7](#_Toc188469591)

[Skills to Learn 7](#_Toc188469592)

[References 7](#_Toc188469593)

[GitHub Link 7](#_Toc188469594)

# Development Document

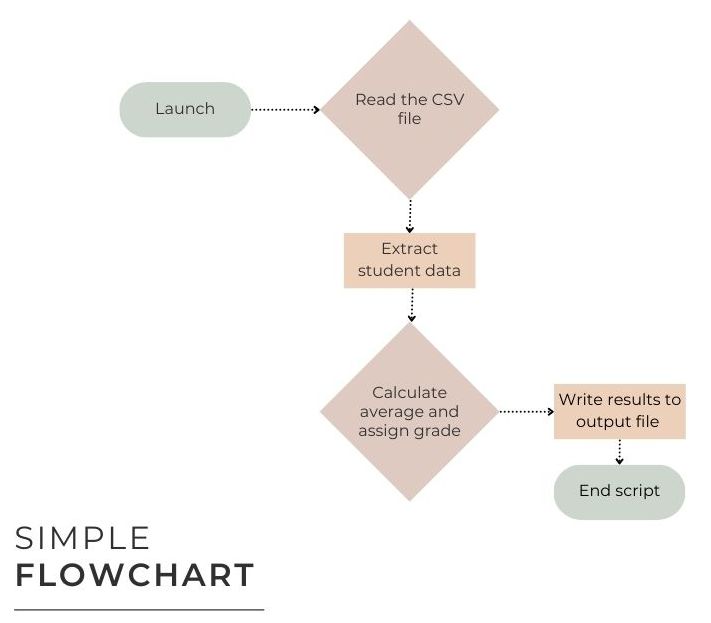
## **Introduction**

When I analysed the requirements for this project, I realized the need for a Python script to process and analyze student data from a CSV file. The task involved reading student records, calculating their performance in different subjects, and generating a new CSV file with the processed information. To accomplish this, I identified three main challenges: ensuring proper validation of the data, implementing accurate grade assignment logic, and handling errors while processing the input file. My approach was to design a modular and maintainable script that could process the data efficiently and handle unexpected situations such as missing files or invalid input data.

## **Solution Design**

The solution was designed around logical operations to read, process, and write data, ensuring clarity and robustness. Below is a Flowchart depicting the program's workflow:

A flowchart that represents the high-level flow of the Python program:



## **Script Walkthrough and Technical Breakdown**

**Initial Setup and File Reading**

* The script starts by importing the csv module for file handling. The try block ensures the program handles missing files gracefully:

try:

with open('Students.csv', 'r') as file:

reader = csv.reader(file)

header = next(reader)

except FileNotFoundError:

print("Error: 'Students.csv' file not found.")

* The header variable reads the first row to skip the column names during processing.

**Data Validation**

* Each row is checked to ensure it has the correct number of columns (six in this case):

if len(row) == 6:

name, surnames, country, math, science, english = row

else:

print(f"Skipping row due to incorrect number of columns: {row}")

* Invalid rows are logged, and processing continues to the next row.

**Processing Rows**

* Scores are converted to integers, with a try block to catch non-numeric values. Any invalid score triggers a warning, and the row is skipped:

try:

math\_score = int(math)

science\_score = int(science)

english\_score = int(english)

except ValueError:

print(f"Skipping row due to invalid score data: {row}")

**Grade Assignment**

* The **assign\_grade** function determines the grade based on predefined thresholds:

def assign\_grade(score):

if score >= 85:

return 'A'

elif score >= 75:

return 'B'

elif score >= 65:

return 'C'

elif score >= 50:

return 'D'

else:

return 'F'

* This approach ensures clarity and modularity, making the function reusable for other projects.

**Country Abbreviation**

* The **get\_country\_abbreviation** function maps country names to their abbreviations using a dictionary:

country\_abbreviations = {

'United States': 'USA',

'United Kingdom': 'UK',

'Dominican Republic': 'DO',

# ... additional mappings

}

return country\_abbreviations.get(country, 'N/A')

* This ensures that countries not listed in the dictionary are marked as 'N/A' without causing errors.

**Writing Output**

* Processed data is stored in a list and written to a new CSV file:

with open('Results.csv', 'w', newline='') as file:

writer = csv.writer(file)

writer.writerow([

'Name', 'Surnames', 'Country', 'Country Abbreviation',

'Math Score', 'Math Grade',

'Science Score', 'Science Grade',

'English Score', 'English Grade'

])

writer.writerows(students)

* The output file includes grades and country abbreviations for all valid rows.

## **Batch Script Description**

This .bat script automates the process of verifying the Python installation and running a specific Python script (**Process\_Info.py**) located in the same directory as the run\_program.bat file.

## **How the Batch Script Works**

1. **Checking Python Installation**  
   The script first checks if Python is installed and accessible via the system PATH.

* If Python is not installed or not properly set in PATH, an error message is displayed, and the script terminates.

1. **Running the Python Script**  
   If Python is available, the script proceeds to execute Process\_Info.py.

* The execution uses the root directory where **run\_program.bat** is located to ensure the correct Python script is run, regardless of the current working directory.

1. **Error Handling**  
   After running the Python script, the script handles errors as follows:

* If an error occurs (i.e., the Python script returns a non-zero exit code), an error message is displayed.
* If the Python script executes successfully, a success message is shown.

**Use of** %~dp0 **Variable**

The **%~dp0** variable dynamically retrieves the directory path of the executing batch file. This ensures that **Process\_Info.py** is correctly located and executed, regardless of where the script is run from. This feature is especially useful in environments where users have varying system configurations and working directories.

## **Personal Evaluation**

Developing this program was both a rewarding and educational experience. One of the most interesting aspects was designing functions like **assign\_grade** and **get\_country\_abbreviation**, which added reusability and modularity to the code. I also learned the importance of robust error handling, especially when working with user-provided input files. The use of **try** blocks to catch errors and ensure the program could continue processing valid data was particularly satisfying.

In hindsight, there are several areas for improvement. First, the script could benefit from additional logging to provide more detailed insights into errors and skipped rows. Integrating a library like logging instead of relying on print statements would enhance maintainability. Second, while the script handles missing files and invalid data gracefully, it could be extended to support additional file formats (e.g., Word) or to fetch data from a database. Lastly, the use of unit tests to verify the functionality of key methods like **assign\_grade** and **get\_country\_abbreviation** would make the program more robust and reliable.

To improve further, I aim to learn advanced techniques for data validation and processing, such as using the pandas library for handling larger datasets. Additionally, exploring object-oriented programming (OOP) principles could help structure the program more effectively for future scalability.

## **Improvements**

While the current program meets the requirements, several improvements could be made:

1. **User Interface:**

A graphical user interface (GUI) could be added to make the program more user-friendly for non-technical users.

1. **Input Validation:**

More comprehensive validation could be implemented to handle edge cases, such as empty rows or negative scores.

1. **Unit Testing:**

Automated unit tests could be developed to ensure that individual functions behave as expected under various conditions.

## **Skills to Learn**

To implement these improvements, I need to enhance my skills in the following areas:

* **GUI Development:**

Learning a Python library like Tkinter or PyQt would enable me to create a graphical interface for the program.

* **Testing:**

Familiarity with Python’s unittest or pytest frameworks would help me write practical unit tests and ensure code reliability.

## **References**

*Command line and environment*. (n.d.). Python Documentation. <https://docs.python.org/3/using/cmdline.html>

*Using Python on Windows*. (n.d.). Python Documentation. <https://docs.python.org/3/using/windows.html>

*An A-Z Index of Windows CMD commands - SS64.com*. (n.d.). <https://ss64.com/nt/>

*csv — CSV File Reading and Writing*. (n.d.). Python Documentation. <https://docs.python.org/3/library/csv.html>

Currie, I. (2023, January 30). *How to add Python to PATH*. <https://realpython.com/add-python-to-path/>

*UINavigation Bar custom font*. (n.d.). Stack Overflow. <https://stackoverflow.com/questions/9282479/how-to-debug-a-batch-file>

## **GitHub Link**

<https://github.com/jesusarb/project.git>