|  |  |
| --- | --- |
| Blue letters on a white background  Description automatically generated | Text  Description automatically generated with medium confidence |

**Student Assessment Submission and Declaration**

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

|  |  |  |  |
| --- | --- | --- | --- |
| Student name: | Joel Florian Perez | ESL ID No | 2409-123122 |
| BSU ID No | 583950 |
| Submission date: 16/5/2025 | | | **19th May 2025, 23:59** |
|  | | |  |
| Programme: |  | | |
| Module name and code: | **Software Foundation -** CPUF001 | | |
| Title: | S1. Development Project | | |
| Assessor name: | Lahiru Thomas Sooriyabandara  Sohail Ilyas | | |

**Plagiarism**

Plagiarism is a form of cheating. Plagiarism must be avoided at all costs, and students who break the rules, however innocently, may be penalised. 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 for further advice.

|  |  |  |  |
| --- | --- | --- | --- |
| **Student declaration**  I certify that the assignment submission is entirely my work. I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice. | | | |
| Student signature: | Joel Florian | Date: | 16/05/2025 |

**Contents**

[**Introduction:** 3](#_Toc197173891)

[**Solution Design** 3](#_Toc197173892)

[**Flowchart Overview** 3](#_Toc197173893)

[**Program Walkthrough and Technical Explanation** 4](#_Toc197173894)

[**Reflective Evaluation:** 6](#_Toc197173895)

[**Conclusion:** 7](#_Toc197173896)

[**References** 9](#_Toc197173897)

[**APPENDIX** 10](#_Toc197173898)

# **Introduction:**

I created a Python-based data processing software that efficiently handles data input, validation, transformation, and output to meet the brief. I started by examining the specifications to identify the main programming ideas to illustrate, including functions, control structures, file handling, and error management. A modular architecture is reflected in the software structure to improve readability and maintainability (Grus, J.2015). Finding common data handling problems and making sure the solution was flexible and scalable were the main goals of the analysis. The GitHub repository has the project files, including the documentation and main script: [GitHub Repository Link].

# **Solution Design**

## **Flowchart Overview**

**Start**

↓

**Accept input file from the command line.**

↓

**Check if the file exists and is readable.**

**↓**

**Read data values from the file.**

**↓**

**Validate data (e.g., numeric format, length)**

**↓**

**Perform Calculation 1 (e.g., Sum, Average, etc.)**

**↓**

**Perform Calculation 2 (e.g., Max, Min, or Conversion)**

**↓**

**Write results to the output file.**

**↓**

**Display a success message or log output.**

**↓**

**End**

## **Program Walkthrough and Technical Explanation**

The Python Data Processing program was developed with modularity, clarity, and extensibility in mind. The program starts by accepting an input file via the command line using the argparse module (McKinney, W. 2012). This approach ensures flexibility and enables batch processing through command-line automation.

**1. Command-Line Input Handling:**

Initialising the script and enabling the user to specify an input file straight from the terminal are the responsibilities of the command-line input processing portion. The application records the filename supplied as an argument by using Python's argparse module. The script includes a validation step that uses os.path.isfile() to determine whether the supplied file exists and is accessible to guard against runtime problems and guarantee robustness. The application can gently alert the user and stop running if the file is missing or invalid.

**2. Reading and Validating Data:**

Accurate and tidy input processing is guaranteed by the data reading and validation portion. Line by line, the script reads the contents of the provided file after opening it. After removing the leading and trailing whitespace from each line, the content is examined to make sure it contains a legitimate number. Converted to floats, valid items are then saved in a list for later processing. Basic validation is included in this step to avoid corrupted or non-numeric entries, preventing mistakes and improving the program's general stability and dependability (Lutz, M. 2013).

**3. Calculations:**

The calculation section performs two key operations on the validated dataset: computing the average and identifying the maximum value. These calculations provide useful insights into the dataset’s overall distribution and peak values. Both operations are implemented within separate, well-defined functions, which improves code modularity and clarity. Encapsulating the logic in functions not only enhances readability but also promotes reusability and simplifies testing. This approach supports future scalability, allowing additional computations to be easily integrated without affecting the core logic.

**4. Writing Output:**

Based on the processed data, the write\_output() function is in charge of producing an understandable and organised summary report. The findings are written to an output text file along with the maximum value, the computed average, and the total number of valid entries. To improve user comprehension, the data is prepared for readability with consistent layout and precision (McKinney, W., 2012). This function serves as a record of successful processing for future reference or auditing, and it guarantees that the program's output is well-structured and understandable.

**5. Script Execution & Error Handling:**

The program includes error handling for:

* Missing files
* Empty input
* ZeroDivisionError for averaging zero items

This ensures graceful failure and clear user feedback, enhancing usability.\

**Advanced Features**

To demonstrate a deeper understanding of Python scripting and file handling, additional features were added:

* A **log file** that tracks the success or failure of operations.
* Use of **functions and modules** to separate logic (e.g., main.py for core logic and utils.py for helpers).
* **Commenting and docstrings** to explain the purpose and usage of each function (Lutz, M. 2013).
* **Cross-platform script** using shell/batch scripts to launch Python with arguments.
* Use of try...except blocks to catch I/O errors, especially in scenarios involving invalid or locked files.

# **Reflective Evaluation:**

It was instructive and beneficial to use Python to create this data processing software. Learning how to incorporate command-line capabilities with the argparse module was one of the most fascinating parts of the project. This added a professional touch to the software and made it easy to use in real-world situations. Building reusable functions for computations like average and maximum also captivated me because it strengthened my knowledge of modular programming and clean code techniques. I obtained practical experience with data validation, error checking, and file management during the process—skills that are essential in data-driven applications. I became more aware of the significance of guaranteeing data integrity before processing after putting basic validation into practice to weed out corrupted or non-numeric inputs. I also became more aware of the value of organised reporting in programming projects after writing the results to a summary file.

But I think some things could be done better. For example, the application can only take a single input format at this time and is not flexible enough to accept more complicated data structures like CSV or JSON files. I want to improve the software in subsequent versions by adding third-party libraries, such as pandas, for more sophisticated data processing. Additionally, adding logging features and enhancing the user interface with better error warnings would strengthen the software.

I want to get better at data structures, exception handling, and modular programming in order to make these gains. In order to guarantee dependability and maintainability, I also intend to investigate automated testing.

# **Conclusion:**

In conclusion, this project enhanced my understanding of Python programming, data validation, and file handling. It provided practical experience in developing structured, functional scripts. While the program meets core requirements, there is clear potential for future improvements, encouraging ongoing learning in advanced data processing and software development practices (Pimenidis, E., el 2009).

# **References**

* Grus, J. (2015). *Data science from scratch: first principles with Python*. Sebastopol, CA: O’Reilly Media.
* Lutz, M. (2013). *Learning Python*. ‘O’Reilly Media, Inc.’
* McKinney, W. (2012). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. [online] *Google Books*. ‘O’Reilly Media, Inc.’ Available at: <https://books.google.com/books?hl=en&lr=&id=v3n4_AK8vu0C&oi=fnd&pg=PR3&dq=Python+for+Data+Analysis:+Data+Wrangling+with+Pandas>
* Pimenidis, E., Hill, P. and Poxon, J. (2009). Clean Code: A Handbook of Agile Software Craftsmanship \* The Mechanical Mind in History \* The IT Value Stack: A Boardroom Guide to IT Leadership. *ITNOW*, 51(1), pp.31–31. Doi: <https://doi.org/10.1093/itnow/bwp019>.

‌

‌\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

‌

# **APPENDIX**

**1. Command-Line Input Handling:**

*import argparse*

*import os*

*parser = argparse.ArgumentParser(description="Process input data file.")*

*parser.add\_argument("input\_file", help="Path to the input data file")*

*args = parser.parse\_args()*

**2. Reading and Validating Data:**

*def read\_data(file\_path):*

*with open(file\_path, 'r') as file:*

*lines = file.readlines()*

*data = [float(line.strip()) for line in lines if line.strip().isdigit()]*

*return data*

**3. Calculations:**

*def calculate\_average(data):*

*return sum(data) / len(data)*

*def calculate\_maximum(data):*

*return max(data)*

**4. Writing Output:**

*def write\_output(data, average, maximum, output\_file='output.txt'):*

*with open(output\_file, 'w') as file:*

*file.write(f"Processed {len(data)} entries\n")*

*file.write(f"Average: {average:.2f}\n")*

*file.write(f"Maximum: {maximum:.2f}\n")*

**5. Script Execution & Error Handling:**

*data = read\_data(args.input\_file)*

*if not data:*

*raise ValueError("No valid data found.")*

*avg = calculate\_average(data)*

*max\_val = calculate\_maximum(data)*

*write\_output(data, avg, max\_val)*

*print("Processing complete. Results written to output.txt")*

*except Exception as e:*

*print(f"Error: {e}")*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_