|  |  |
| --- | --- |
| 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 your own.

|  |  |  |  |
| --- | --- | --- | --- |
| Student name: | Doroteya Ivanova Rogers | ESL ID No | 123165 |
| BSU ID No | 583831 |
| Submission date: | | |  |
|  | | |  |
| Programme: | BSU Computing | | |
| Module name and code: | Software Foundations CPUF001-20 | | |
| Title: | S1: Development Project | | |
| Assessor name: | Islombek Isroilov | | |

**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 if you need any further advice.

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

**Development Document**

**Introduction**

The project consisted of creating a Python Data Processing tool, which takes data from a specified file, computes, and produces output in a new file. The tool was also executed with a bash shell script aside from execution within Python itself, together with a minimal command-line interface. The key requirements were illustrating structured programming, file operations, and commenting. A GitHub repository was also established for version control as well as publishing. The repository includes the Python file, bash file, and data files and it can be found here: <https://github.com/doroteyarog/data_processing>

**Solution Design**

The data processing problem was addressed by a clear, modular solution, in three broad logical steps: reading data in, calculation, writing data out.  
  
I made a simple flowchart in a bid to illustrate the solution. The step-by-step reasoning is as follows:  
  
With this, we start → Retrieve data from file → Find the summation and mean → Output results in file → Stop

A screenshot of a phone

AI-generated content may be incorrect.

**Reading from a File:**

The project starts by reading from a file (input\_data.csv). The native file-handling mechanism of Python is used, ensuring automatic closing through the use of With Open. The file is read line by line, stripped of whitespace, and parsed from strings into floating point numbers for arithmetic precision calculations.  
  
  
  
**Calculating Sum and Average:**Two major functions were utilized to calculate the required statistics:  
  
Both calculate\_sum() and calculate\_average() from the last version are replaced by calculate\_statistics().  
  
It takes an array of data rows which contain structured data consisting of a name, an age, and a score.  
  
The overall score is determined using Python's built-in function sum() on the score values.  
  
It calculates the average score by dividing total score by the number of valid entries, and handles the case when the input may be empty to prevent division by zero.  
  
The function encompasses the total and average calculations, which follow the organized form of the new input (input\_data.csv) and output (output\_data.csv).

**Writing Results to Output File:**  
The results, i.e., the total and average, are printed to a separate output file (output\_results.txt) by invoking another well-defined function, write\_results(). File management again employs the safe with open technique. The results are printed in human-readable form for convenience.

**Command Line Execution**For convenient execution of the Python script, two options were given:  
  
With a bash shell script, run\_program.sh: facilitates execution by predefining the names of the input file and output file for use by the Python program, making it more user friendly while reducing user errors.  
  
The script further gives instant feedback through the terminal upon successful execution, in addition to where the output data is stored.  
  
Direct Python Execution: Users in Windows or systems where bash scripts are not easily executable can execute the program through Python by running the following command: **python data\_processing.py input\_data.csv output\_data.csv**This guarantees compatibility and usability with multiple operating systems.

**Technical Explanation:**The code was organized in independent, reusable functions so that it would be more readable, maintainable, as well as easier to debug. There are explanatory comments in each function identifying its purpose and operation. The use of command-line arguments means the program can easily be modified for use with a variety of data sets without the source code being modified.  
  
The primary execution logic was structured through the use of structured programming methods, within a specifically defined main() function. Error management was achieved through correct counting of the command-line argument numbers and issuance of a friendly message in the case of improper usage.

**Reflective Evaluation**This activity was particularly fascinating as it entailed direct hands-on experience dealing with real-world data operations. Working with file operations, command-line scripts, and modular code structures helped me better comprehend good software design and maintenance.  
  
The most satisfying aspect was structuring the program into separate, manageable functions. The primary issue initially was ensuring each function interacted with the rest appropriately in terms of data transmission as well as file operations. Having overcome such issues successfully, my belief in leveraging Python's built-in functionalities was strengthened.  
  
There is always room for improvement, though. The software does, today, expect correct format from the data input, but does not handle file errors, for instance, a file not existing or non-numerical data. Proper error-handling would improve the software, by having more definite instructions in the event of a fault.  
  
Moreover, expanding the scope of the function of the application, e.g., through support for more statistical calculations (median, standard deviation) or by showing the data in simple graphics, would further increase its usability and appeal.  
  
Revisiting these upgrades, I realized where my programming skills could be further honed, i.e., in robust error handling, more in-depth data analysis tools, and maybe a combination with graphical interfaces.  
  
Overall, this assignment helped my practical skills in structured programming, scripting, and documentation immensely. It also provided constructive feedback for future enhancements, informing my future learning as well as professional growth.

**Appendix**

**Python Script:**   
  
A screen shot of a computer program

AI-generated content may be incorrect.

**Bash Script:**   
  
A screen shot of a computer program

AI-generated content may be incorrect.