Development Project – Software Foundation (CPUF001)

GitHub Repository Link: https://github.com/yourusername/data-processing-project

# 1. Introduction

This project presents a data processing application developed using Python. The program reads numerical values from an input text file, performs statistical calculations (average and standard deviation), and writes the output to a results file. It is designed to be executed via a command-line interface (CLI), which supports basic automation and user control. A script file is included to run the program with minimal user interaction. This project demonstrates key programming concepts such as modularity, file handling, and exception management, aligning with foundational software development skills. All source files and documentation are hosted in the GitHub repository linked above.

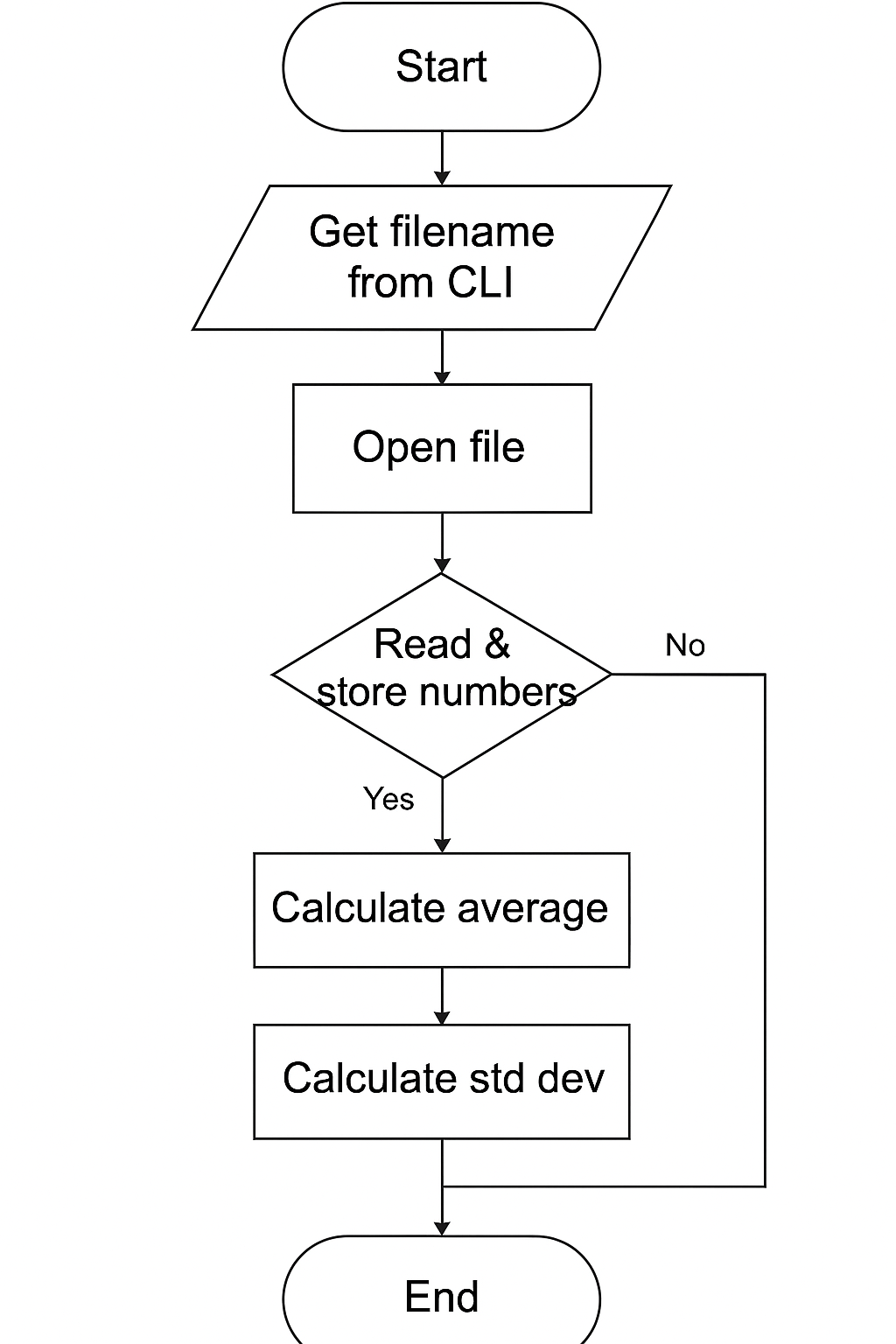
# 2. Solution Design

## 2.1 Pseudocode

Algorithm: ProcessDataFile  
1. Start  
2. Read file path from command line  
3. Open the input file  
4. Create an empty list to store numbers  
5. For each line in the file:  
 a. Convert line to float  
 b. Append to the list  
6. Calculate average:  
 a. Sum of list / Length of list  
7. Calculate standard deviation:  
 a. Subtract average from each number, square the result  
 b. Sum the squared differences  
 c. Divide by count, take square root  
8. Write results to output file  
9. Display completion message  
10. End

## 2.2 Flowchart

The following flowchart illustrates the logic of the data processing program:



## 2.3 Code Structure and Explanation

The program is contained in a single Python file named main.py, and uses the following key components:  
- read\_data(file\_path): Reads data from the input file and returns a list of floats.  
- calculate\_average(data): Returns the average of the dataset.  
- calculate\_std\_dev(data, mean): Computes the standard deviation.  
- write\_results(output\_path, avg, std\_dev): Saves results to an output text file.  
- main(): The entry point, parsing command line arguments and orchestrating the process.  
  
The code is fully commented to explain logic and encourage readability. It includes exception handling for file I/O errors and empty input data.

# 3. Reflective Evaluation

This project deepened my understanding of core Python programming concepts, particularly regarding functions, file operations, and CLI argument handling. The task of reading and validating input from an external file taught me the importance of error handling and robust code structure. Implementing standard deviation was particularly rewarding, as it required understanding and applying mathematical operations to dynamic lists in Python.  
  
One challenge I faced was managing invalid or empty data entries, which initially caused runtime errors. After some debugging, I resolved this with conditional checks and list comprehensions. Another difficulty was structuring the output in a user-friendly format, which I addressed using formatted strings and clear file output naming.  
  
In future projects, I would consider building a graphical interface (GUI) to improve usability and add flexibility for selecting files. Additionally, integrating unit testing using unittest or pytest would improve reliability and test coverage.  
  
Overall, this assignment improved my technical and problem-solving skills and gave me hands-on experience with essential development tools such as GitHub and CLI scripting.

# 4. Appendix: Code

Include your full Python code and script file in this section:  
- main.py  
- run\_script.bat  
- Sample data\_input.txt  
  
You can copy and paste the full code here or attach it as separate files.

# Appendix: Full Code

## main.py

# main.py  
  
import sys  
  
def read\_data(file\_path):  
 with open(file\_path, 'r') as f:  
 return [float(line.strip()) for line in f if line.strip()]  
  
def calculate\_average(data):  
 return sum(data) / len(data)  
  
def calculate\_std\_dev(data, mean):  
 return (sum((x - mean) \*\* 2 for x in data) / len(data)) \*\* 0.5  
  
def write\_results(output\_path, average, std\_dev):  
 with open(output\_path, 'w') as f:  
 f.write(f"Average: {average:.2f}\n")  
 f.write(f"Standard Deviation: {std\_dev:.2f}\n")  
  
def main():  
 if len(sys.argv) < 2:  
 print("Usage: python main.py <input\_file>")  
 return  
  
 input\_file = sys.argv[1]  
 output\_file = "output.txt"  
  
 try:  
 data = read\_data(input\_file)  
 average = calculate\_average(data)  
 std\_dev = calculate\_std\_dev(data, average)  
 write\_results(output\_file, average, std\_dev)  
 print("Processing complete. Results saved in output.txt")  
 except Exception as e:  
 print(f"An error occurred: {e}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

## run\_script.bat

:: run\_script.bat  
@echo off  
REM Batch script to run the Python data processor  
REM Usage: run\_script.bat data\_input.txt  
  
python main.py %1  
pause