



**MAPÚA UNIVERSITY**

**SCHOOL OF ELECTRICAL, ELECTRONICS, AND COMPUTER ENGINEERING**

## **Experiment 2: Strings, Lists, Tuples, and Dictionaries**

CPE106L (Software Design Laboratory)

**Brian Matthew E. Clemente**

**Maria Angelica Acantilado**

**Chalzea Fransen C. Dytianquin**

**Vance David G. Samia**

Group No.: **4**

Section: **FOPI01**



## PreLab

---

### Readings, Insights, and Reflection

**Lambert, K. A. (2019). *Fundamentals of Python: First Programs* (3rd ed.). Cengage Learning US. ISBN: 9780357881132.**

**METIS Book, Chapter 5, Pages 126–127.**

Kenneth A. Lambert's *Fundamentals of Python: First Programs* (3rd Edition, 2019) included a detailed explanation of how to define and use functions in Python in Chapter 5. The structure of functional definitions, which are composed of a header and a body, was one of the most important lessons learned. The function name, parameters, and the `def` keyword are all contained in the header, whereas the statements that are executed when the function is called are found in the body. It is essential to comprehend function syntax in order to write flexible and reusable code.

In addition, the chapter presented parameters and arguments, highlighting the fact that the former are the actual values supplied when the function is called, while the latter operate as placeholders within function definitions. This distinction is essential for guaranteeing correct function execution and preventing mistakes. The `return` statement, which expressly returns a value from a function, was another topic we studied. Python instantly evaluates the expression and returns control to the caller when it comes across a `return` statement. Python returns `None` by default if an empty `return` statement is provided.

We learned how to write organized and effective programs by learning how to define and call functions. Functions improve code organization and reusability, which facilitates the management and debugging of complicated programs. This chapter reaffirmed Python's focus on readability and simplicity while emphasizing the value of well-specified functions in software development and problem-solving.

**METIS Book, Chapter 4, Pages 87–88.**

Kenneth A. Lambert's *Fundamentals of Python: First Programs* (3rd Edition, 2019) examined encryption and string manipulation strategies in Chapter 4, emphasizing both basic programming ideas and practical uses. The chapter started with a more thorough examination of string structures, going into how to extract and work with substrings in Python. Text processing, a frequent activity in software development, requires this comprehension.

The chapter placed a lot of emphasis on data security, especially the risks associated with network data transmission. It discussed the Caesar cipher as one of the earliest encryption methods and introduced encryption as a way to safeguard sensitive data. The Caesar cipher illustrates the fundamentals of cryptography by obscuring plaintext messages by shifting

characters in the alphabet by a predetermined amount. Python scripts were made available to perform encryption and decryption using this technique to demonstrate how programming may be used to address security issues.

The chapter covered the Caesar cipher's shortcomings, especially its vulnerability to brute-force attacks, even if it serves as an example of the encryption principle. As a result, more sophisticated encryption techniques were introduced, like block ciphers, which boost security by utilizing mathematical structures like invertible matrices.

### **Clemente (Chapter 4 and Chapter 5)**

Python's versatility as a programming language lies in its robust data structures and efficient data handling capabilities. Chapters 4 and 5 of *Fundamentals of Python: First Programs* by Kenneth A. Lambert introduce essential concepts related to strings, text, files, lists, and dictionaries. Foremost, Chapter 4 emphasizes the importance of strings as data structures in Python. Their immutability ensures data integrity, preventing accidental modifications, which is crucial during textual data handling, such as parsing user inputs or reading from files, where unintended changes could lead to errors. String manipulation techniques, such as slicing and indexing, provide flexibility in data processing, making it easier to extract relevant information. The chapter also highlights the significance of text files in programming. The ability to store and retrieve data from files extends the functionality of programs beyond temporary memory, making persistence possible. Methods like `read()`, `readLine()`, and `write()` simplify the interactions with files, while using loops to iterate over file lines enhances coding efficiency.

Chapter 5 introduces lists and dictionaries — two versatile data structures that allow efficient storage and manipulation of data. Unlike strings, lists are mutable, providing dynamic control over their contents. The ability to modify, sort, and slice lists makes them ideal for handling collections of related data. However, mutability also introduces potential issues, such as aliasing, which can lead to unintended side effects when multiple variables reference the same object. Dictionaries, on the other hand, offer a structured way to associate keys with values, making data retrieval more intuitive. Unlike lists, which rely on positional access, dictionaries use key-value mapping, making them particularly useful in scenarios requiring quick lookups, such as storing user profiles or configuration settings. The chapter also discussed functions and testing methods, such as bottom-up and top-down testing strategies, on how such practices are crucial for writing reliable and maintainable code. Overall, the knowledge gained from these chapters lays a strong foundation for writing well-structured and data-driven Python programs.

### **Acantilado (Chapter 4 and 5)**

Chapters 4 and 5 of *Fundamentals of Python: First Programs* by Kenneth A. Lambert provide essential knowledge on handling text, files, and data structures in Python. These chapters emphasize the importance of strings, text files, lists, and dictionaries in programming, highlighting their roles in data storage, retrieval, and manipulation. Understanding these fundamental concepts allows programmers to develop efficient and structured applications. Additionally, the chapters introduce testing strategies, reinforcing the need for reliable and

maintainable code. Mastering these concepts equips programmers with the necessary tools to handle various data-driven tasks effectively.

Foremost, Chapter 4 focuses on strings and text file operations, emphasizing the immutability of strings and their significance in preserving data integrity. String manipulation techniques, such as indexing, slicing, and concatenation, enable efficient data processing. Additionally, file handling methods like `read()`, `readline()`, and `write()` are introduced, allowing programs to store and retrieve data persistently. The underscored chapter underscores the importance of text files in extending a program's functionality beyond temporary memory, making data storage more reliable and accessible.

Moreover, Chapter 5 introduces lists and dictionaries, two versatile data structures that enhance data organization and retrieval. Lists, being mutable, provide flexibility in modifying data, though they require careful handling to avoid unintended consequences like aliasing. Dictionaries, with their key-value mapping, enable efficient lookups and structured data storage, making them useful for applications such as database management. The chapter discusses top-down and bottom-up testing approaches, emphasizing their role in ensuring the reliability of Python programs. By mastering these concepts, programmers can write more efficient and structured code, ultimately improving software performance and maintainability.

### **Dytianquin (Chapter 4)**

Chapter 4 of Kenneth A. Lambert's *Fundamentals of Python: First Programs* (3rd Edition, 2019) explores key programming concepts, including string manipulation, data structures, and error handling, which are fundamental to writing efficient Python programs. Strings play a crucial role in capturing and processing user input, often requiring methods like `.split()` to extract numerical values by breaking down strings into manageable components. This method is particularly useful when handling multiple inputs at once, converting them into numeric data types for further computations. Lists serve as the primary data structure for storing these values, offering flexibility due to their dynamic nature, which allows for efficient data organization and modification.

Built-in list methods such as `.count()` help determine the frequency of specific elements, making them essential for statistical calculations like finding the mode. Additionally, Python's robust error-handling mechanisms, particularly `try` and `except` blocks, ensure smooth execution by catching invalid inputs, preventing unexpected crashes, and providing meaningful feedback to users. These techniques not only improve program reliability but also contribute to a more seamless and user-friendly experience, making them vital for developing well-structured, data-driven applications.

### **Samia (Chapter 5)**

Loop and repetition, which are crucial for automating repeated programming activities, were the subject of Chapter 5. The chapter covered the two fundamental Python looping structures:

loops, which carry on running as long as a certain condition is true, and loops, which iterate over sequences like lists and strings. We looked at the significance of loop control techniques, such as `break` and `continue`, which let programmers adjust the behavior of loops by skipping iterations or departing them early. We also looked at how counters and accumulators facilitate effective data manipulation in iterative processes. Writing scalable and effective programs requires an understanding of loops since they enhance code organization and eliminate redundancy.

### Answers to Questions

1. b. 20
2. b. [20, 30]
3. a. 1
4. b. [10, 20, 30, 40, 50]
5. b. [10, 5, 30]
6. c. [10, 15, 20, 30]
7. b. ["name", "age"]
8. b. None
9. b. pop
10. b. strings and tuples

# PostLab

## Programming Problems

This post-lab activity consists of three programming tasks designed to further enhance our problem-solving skills using Python. Each task focuses on different aspects of programming, including statistical computation, file handling, and sentence generation. Given that this is a group laboratory activity, each programming problem has been distributed among the members, with their names denoted as comments within the code files to distinguish between the respective contributors of each problem. Much like the previous lab report, Figure 1.1 and Figure 1.2 reflect the contributions of each group member in completing the assigned programming tasks. These figures provide a visual representation of the GitHub repository's commit history, illustrating the collaborative effort and individual contributions made throughout the development process.

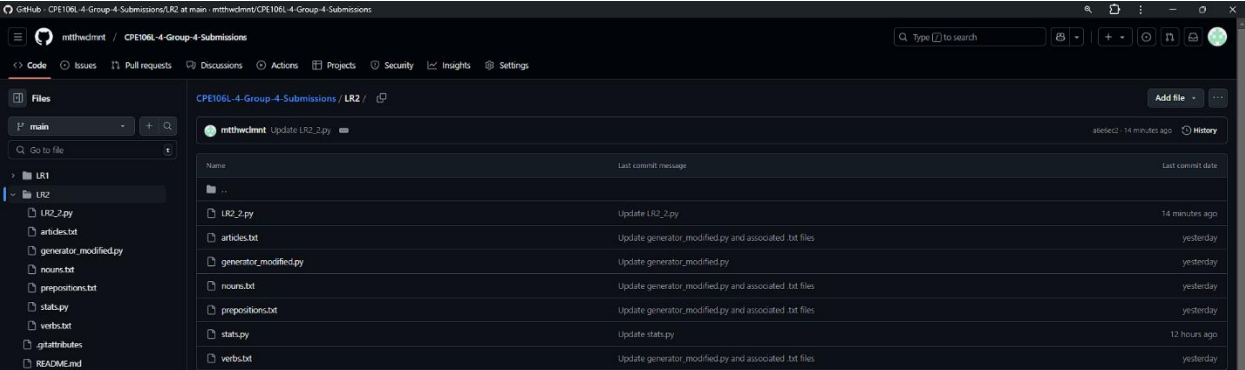


Figure 1.1. Group 4 Lab Report 2 Folder GitHub Repository Preview

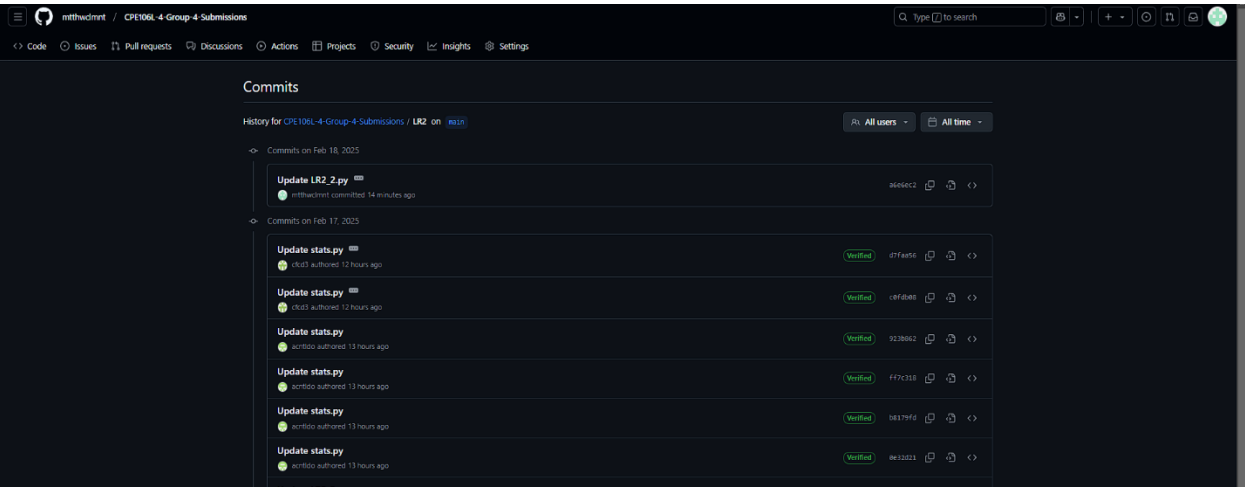


Figure 1.2. Lab Report 2 GitHub Repository Commits History Log (Screenshot 1)

Programming Problem #1

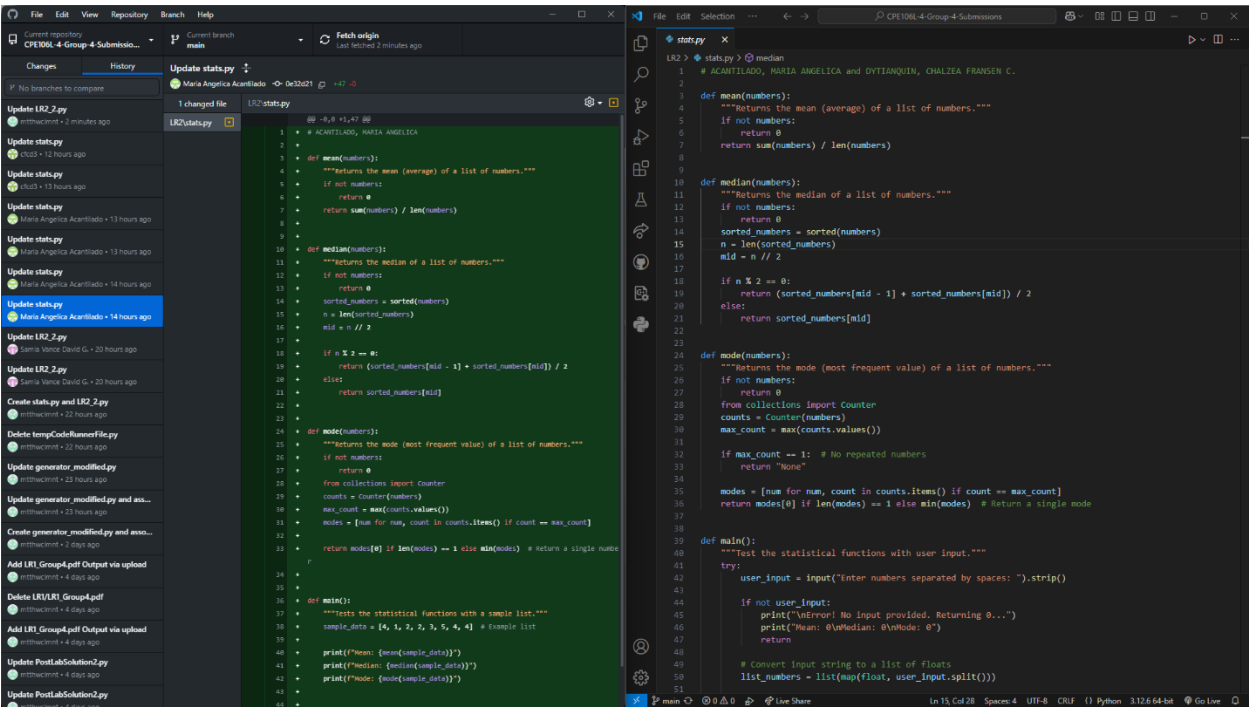
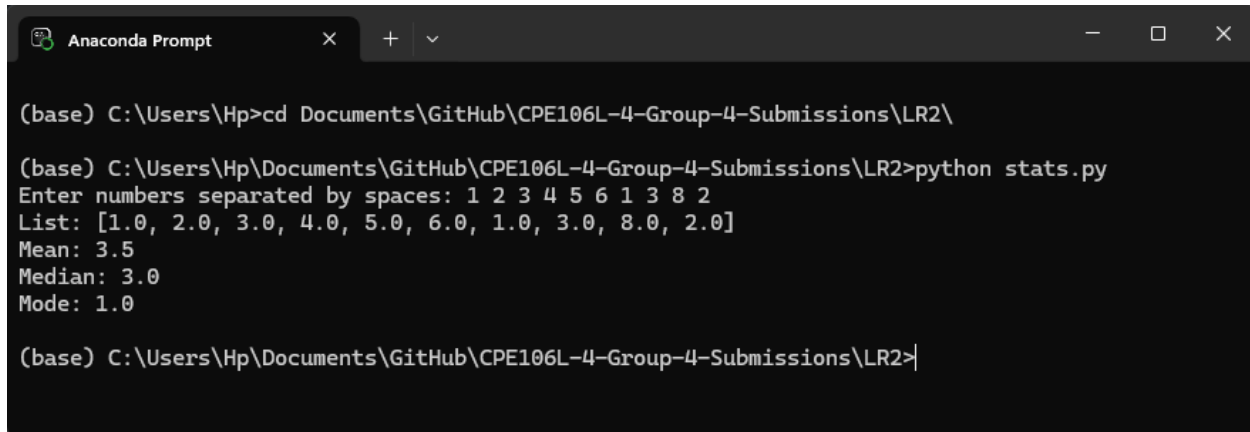


Figure 2.1 Screenshot of GitHub Desktop and VS Code open during the Programming Problem #1 Python File Commit and Push to Origin

Figure 2.1 illustrates the development process for *stats.py*, showcasing GitHub Desktop and VS Code open during the commit and push operation for the underscored problem. It required the creation of a Python module containing functions to compute the mean, median, and mode of a given dataset. The implementation involved defining separate functions for each statistical measure, ensuring they correctly handle both populated and empty lists. The screenshot highlights the collaborative workflow, with version control tracking modifications and ensuring smooth integration of contributions.

Particularly, the structure of *stats.py* follows a modular design, with each statistical function implemented separately to ensure clarity and maintainability. The script begins by defining the necessary functions—*mean()*, *median()*, and *mode()*—each handling its computation efficiently while accounting for edge cases, such as empty lists. The program utilizes sorting techniques for median calculation and dictionary-based frequency counting for mode determination, optimizing performance for larger datasets. Additionally, a *main()* function integrates these statistical operations, allowing for streamlined testing by processing sample inputs and displaying results. This structured approach not only improves readability but also enables seamless debugging and future enhancements, making the module adaptable for more advanced statistical analysis.

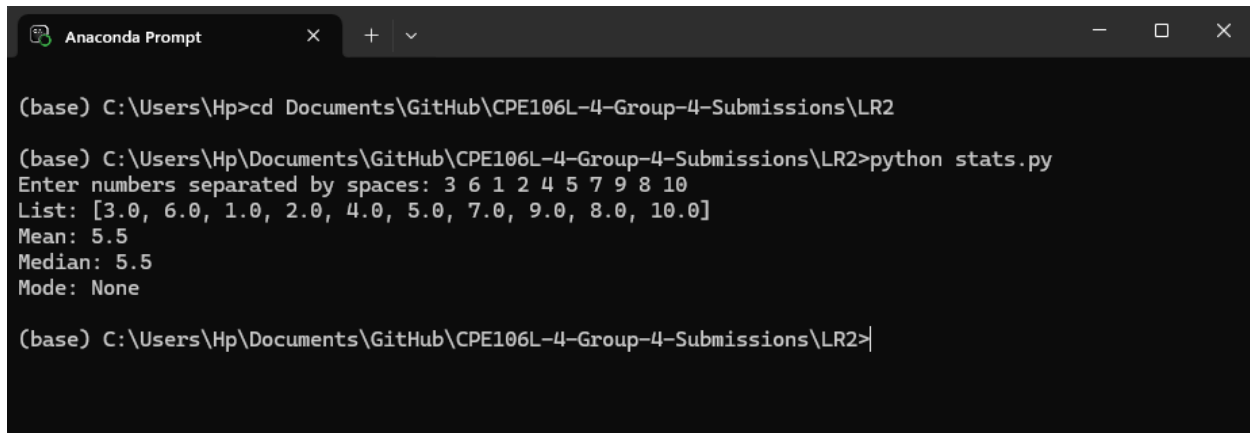


A screenshot of the Anaconda Prompt terminal window. The window title is "Anaconda Prompt". The terminal shows the following commands and output:

```
(base) C:\Users\Hp>cd Documents\GitHub\CPE106L-4-Group-4-Submissions\LR2\
(base) C:\Users\Hp\Documents\GitHub\CPE106L-4-Group-4-Submissions\LR2>python stats.py
Enter numbers separated by spaces: 1 2 3 4 5 6 1 3 8 2
List: [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 1.0, 3.0, 8.0, 2.0]
Mean: 3.5
Median: 3.0
Mode: 1.0
(base) C:\Users\Hp\Documents\GitHub\CPE106L-4-Group-4-Submissions\LR2>
```

Figure 2.2 Problem #1 Showcasing a Data Set of 10 Elements in Anaconda Prompt

Figure 2.2 demonstrates the execution of `stats.py` in Anaconda Prompt, showcasing its ability to process a dataset of  $n$  elements and compute the mean, median, and mode. The user inputs a series of numbers, which the program converts into a list for statistical computation. It is to underscore that such an interactive nature of the program was intended to allow users to input different datasets dynamically, ensuring flexibility in statistical analysis. This output validation reinforces the effectiveness of the program in handling numerical data while maintaining accuracy and efficiency.

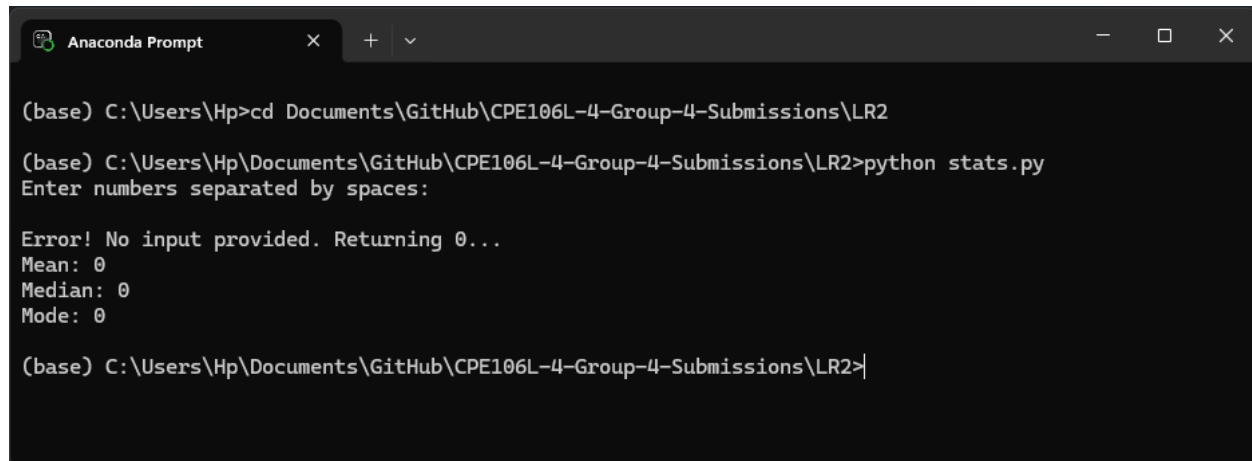
A screenshot of the Anaconda Prompt terminal window. The window title is "Anaconda Prompt". The terminal shows the following commands and output:

```
(base) C:\Users\Hp>cd Documents\GitHub\CPE106L-4-Group-4-Submissions\LR2
(base) C:\Users\Hp\Documents\GitHub\CPE106L-4-Group-4-Submissions\LR2>python stats.py
Enter numbers separated by spaces: 3 6 1 2 4 5 7 9 8 10
List: [3.0, 6.0, 1.0, 2.0, 4.0, 5.0, 7.0, 9.0, 8.0, 10.0]
Mean: 5.5
Median: 5.5
Mode: None
(base) C:\Users\Hp\Documents\GitHub\CPE106L-4-Group-4-Submissions\LR2>
```

Figure 2.3 Problem #1 Showcasing a Data Set of 10 Elements with No Mode in Anaconda Prompt

Figure 2.3 shows the output generated by the program after processing the input data. It displays the calculated mode, along with any relevant messages that the program produces, such as handling cases where no mode exists or when the data set is empty. This output is critical for demonstrating the program's effectiveness in performing statistical analysis. By showcasing the results, this figure allows readers to see the practical application of the code and understand how the program responds to different input scenarios.





```
(base) C:\Users\Hp>cd Documents\GitHub\CPE106L-4-Group-4-Submissions\LR2

(base) C:\Users\Hp\Documents\GitHub\CPE106L-4-Group-4-Submissions\LR2>python stats.py
Enter numbers separated by spaces:

Error! No input provided. Returning 0...
Mean: 0
Median: 0
Mode: 0

(base) C:\Users\Hp\Documents\GitHub\CPE106L-4-Group-4-Submissions\LR2>
```

*Figure 2.4 Problem #1 Showcasing an Empty Data Set in Anaconda Prompt*

Figure 2.4 shows a scenario in which the program encounters an invalid input, an empty data set. It shows the error message generated by the program, highlighting the error handling mechanisms implemented to manage such situations gracefully. This visual emphasizes the robustness of the code, demonstrating how it provides user-friendly feedback instead of crashing. By addressing potential errors, this figure underscores the importance of error handling in creating reliable and user-centric software.

Programming Problem #2

The second program is designed to provide users with an interactive way to navigate the contents of a text file by accessing specific lines based on user input. Upon execution, the program first prompts the user to enter a filename and then reads the file’s contents into a list where each element represents a line from the file. The program then enters a loop, displaying the total number of lines and requesting a line number from the user. If the input corresponds to a valid line number, the program retrieves and prints the corresponding text. Entering zero allows the user to exit the program, ensuring a controlled and user-friendly navigation experience.

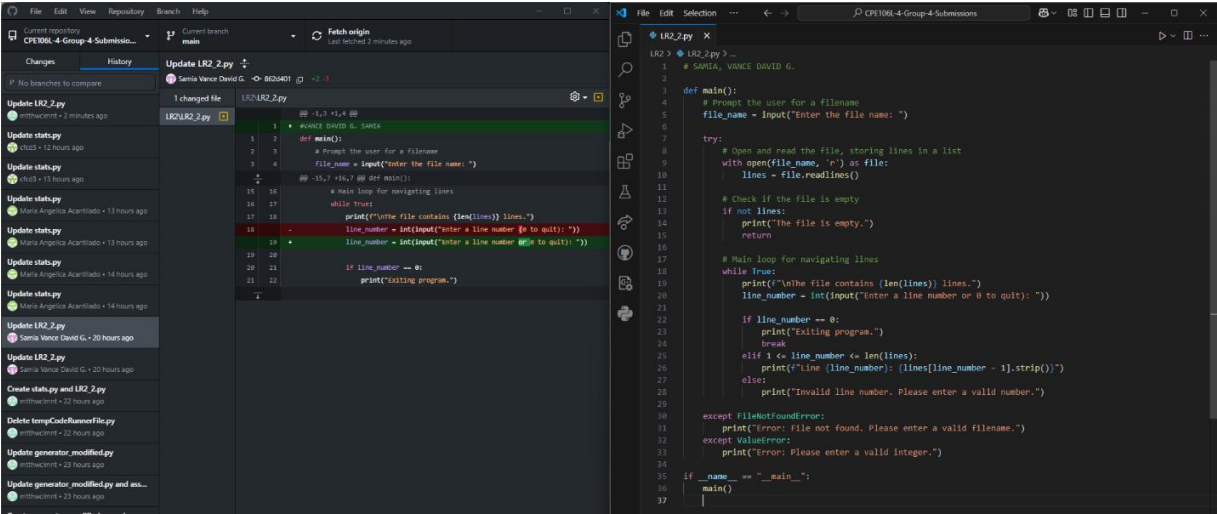
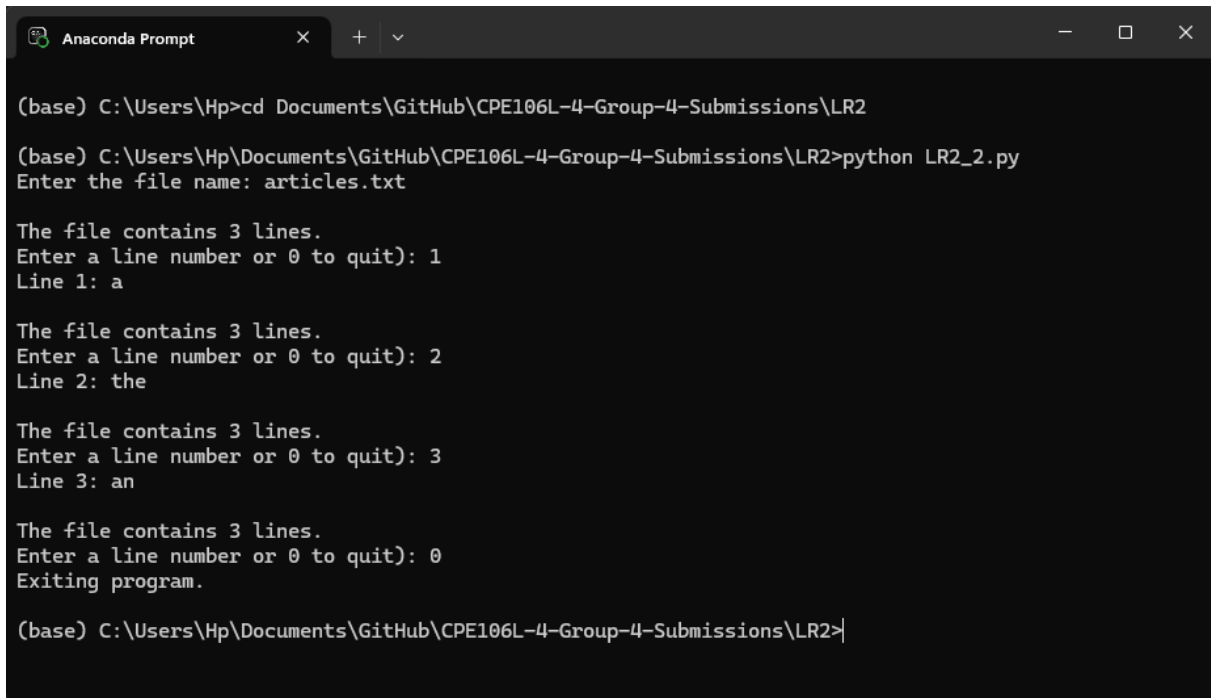


Figure 3.1 Screenshot of GitHub Desktop and VS Code open during the Programming Problem #2 Python File Commit and Push to Origin

Figure 3.1 shows a side-by-side comparison of GitHub Desktop and Visual Studio Code with the main source code for the second programming problem. The picture shows GitHub Desktop with commit messages describing the changes, including the creation and updates of the Python file. The changed Python script is displayed in VS Code, which is open concurrently, emphasizing how version control and active coding are integrated. The script follows a structured approach, beginning with file reading and storing lines in a list, followed by a loop that enables user navigation through the file’s contents. Input validation ensures users can only select valid line numbers, preventing errors. This configuration guarantees quick change tracking, seamless communication, and a well-organized development workflow while maintaining code clarity and efficiency.



```
(base) C:\Users\Hp>cd Documents\GitHub\CPE106L-4-Group-4-Submissions\LR2

(base) C:\Users\Hp\Documents\GitHub\CPE106L-4-Group-4-Submissions\LR2>python LR2_2.py
Enter the file name: articles.txt

The file contains 3 lines.
Enter a line number or 0 to quit): 1
Line 1: a

The file contains 3 lines.
Enter a line number or 0 to quit): 2
Line 2: the

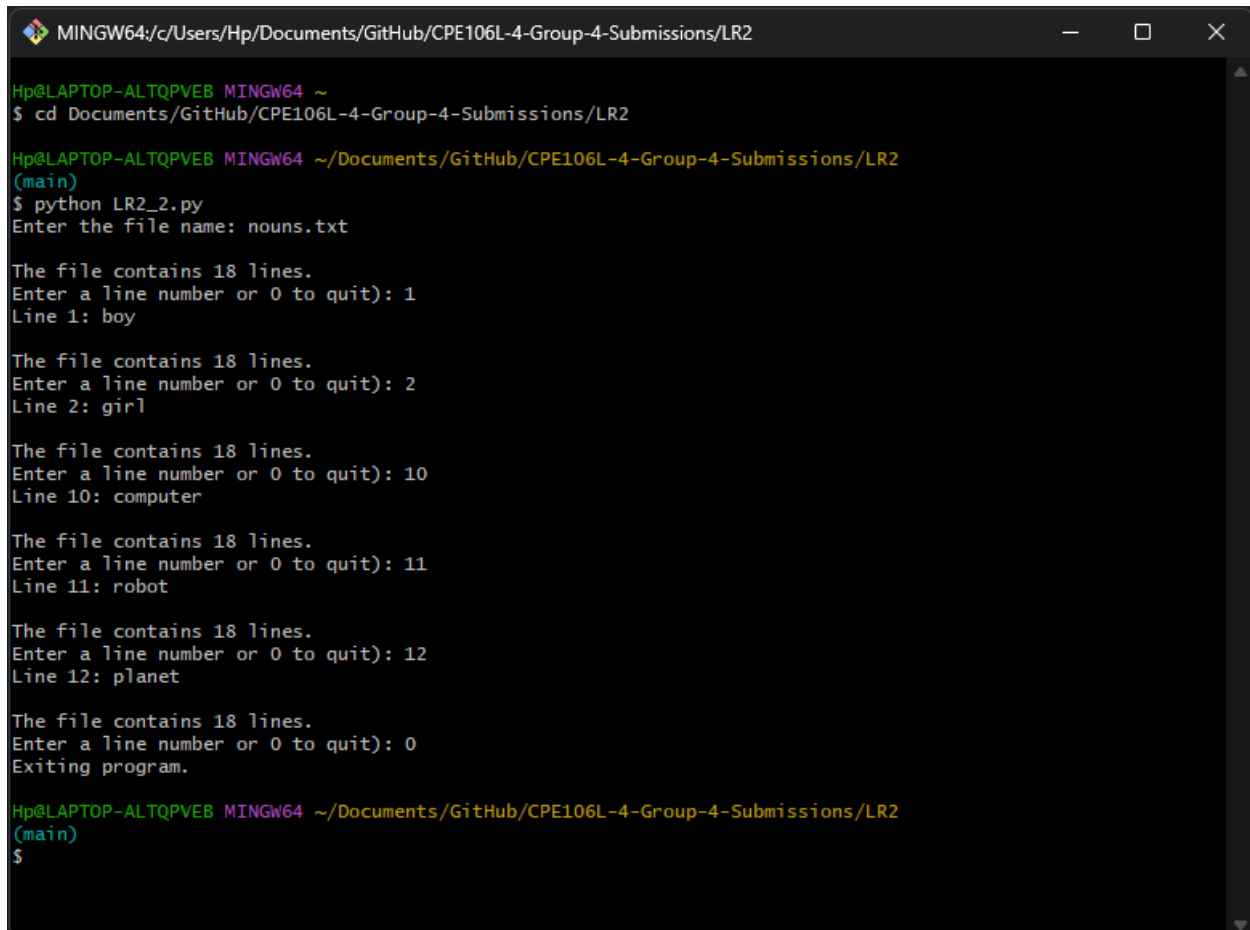
The file contains 3 lines.
Enter a line number or 0 to quit): 3
Line 3: an

The file contains 3 lines.
Enter a line number or 0 to quit): 0
Exiting program.

(base) C:\Users\Hp\Documents\GitHub\CPE106L-4-Group-4-Submissions\LR2>
```

*Figure 3.2 Programming Problem #2 Python Script and Associated Text Files executed in Anaconda Prompt*

Figure 3.2 showcases the execution of the *LR2\_2.py* Python script along with its associated text files within the Anaconda Prompt as part of Programming Problem #2. The screenshot captures the program's interactive nature, displaying the console output where the user is prompted to enter a line number, and the corresponding text is retrieved from the file. The structured implementation of the script ensures that user input is validated, allowing only valid line selections while providing an option to exit by entering 0. This setup highlights how the program efficiently processes text file data and demonstrates fundamental file-handling techniques in Python. Running the script within Anaconda Prompt ensures compatibility with necessary Python packages and provides a controlled execution environment, minimizing dependency-related issues.



```
MINGW64/c/Users/Hp/Documents/GitHub/CPE106L-4-Group-4-Submissions/LR2
Hp@LAPTOP-ALTQPVEB MINGW64 ~
$ cd Documents/GitHub/CPE106L-4-Group-4-Submissions/LR2
Hp@LAPTOP-ALTQPVEB MINGW64 ~/Documents/GitHub/CPE106L-4-Group-4-Submissions/LR2
(main)
$ python LR2_2.py
Enter the file name: nouns.txt

The file contains 18 lines.
Enter a line number or 0 to quit): 1
Line 1: boy

The file contains 18 lines.
Enter a line number or 0 to quit): 2
Line 2: girl

The file contains 18 lines.
Enter a line number or 0 to quit): 10
Line 10: computer

The file contains 18 lines.
Enter a line number or 0 to quit): 11
Line 11: robot

The file contains 18 lines.
Enter a line number or 0 to quit): 12
Line 12: planet

The file contains 18 lines.
Enter a line number or 0 to quit): 0
Exiting program.

Hp@LAPTOP-ALTQPVEB MINGW64 ~/Documents/GitHub/CPE106L-4-Group-4-Submissions/LR2
(main)
$
```

*Figure 3.3 Programming Problem #2 Python Script and Associated Text Files executed in GitBash*

Ultimately, the second programming problem involves executing the Python script alongside its associated text files within the GitBash terminal, as illustrated in Figure 3.3. The screenshot captures the interactive functionality of the script, displaying user inputs and corresponding outputs within the command line interface. By running the program in GitBash, users can navigate the contents of a text file through sequential line selection, reinforcing key concepts in file handling and user-driven input processing. This configuration also highlights the advantages of using GitBash as a portable and efficient script execution environment, particularly in workflows that integrate version control systems like Git.

## Programming Problem #3

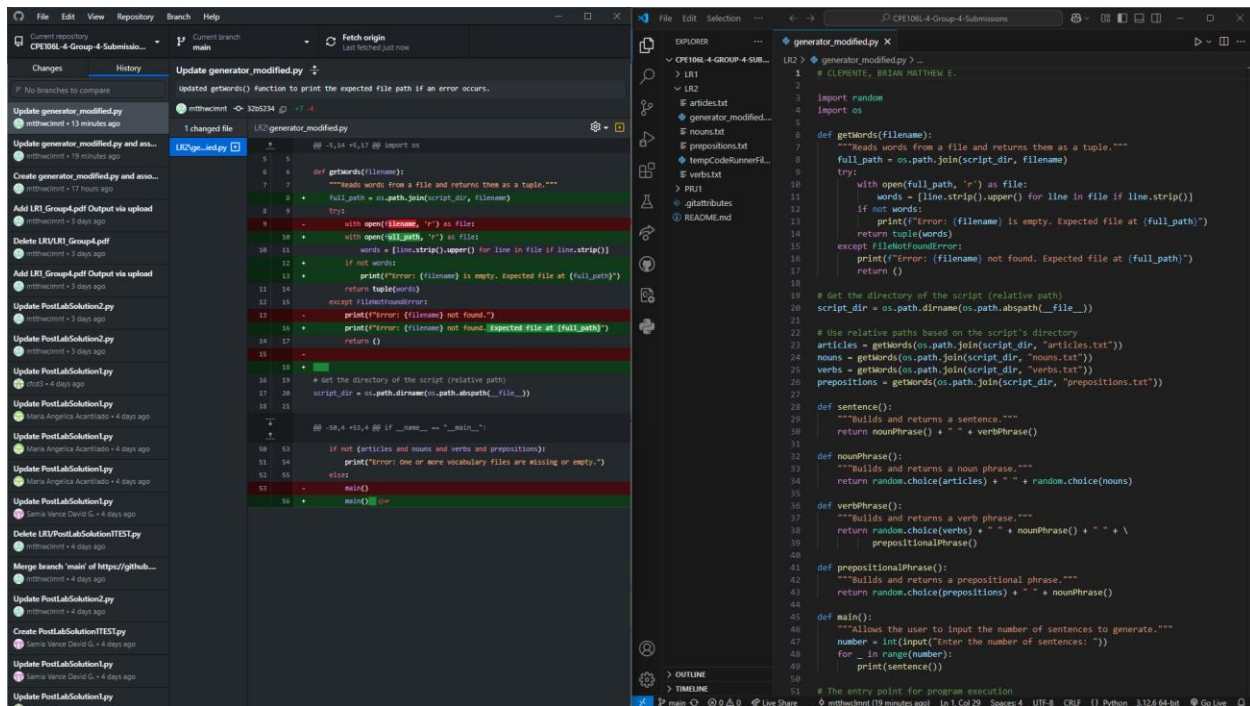


Figure 4.1 Screenshot of GitHub Desktop and VS Code open during the Programming Problem #3 Python File Commit and Push to Origin

Figure 4.1 displays a screenshot capturing the use of GitHub and Visual Studio Code during the commit and push process for `generator_modified.py` file and its associated text files for the group's Programming Problem #3 output. In the image, GitHub Desktop shows the creation and updates of the python file. Meanwhile, VS Code is open, displaying the modified code involved in the updates, illustrating the workflow of version control integration with GitHub and active coding in VS Code.

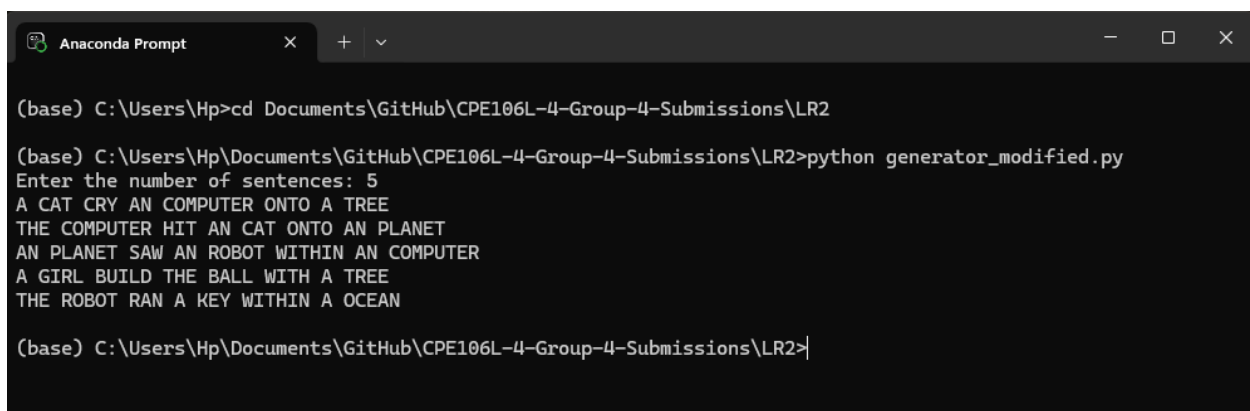
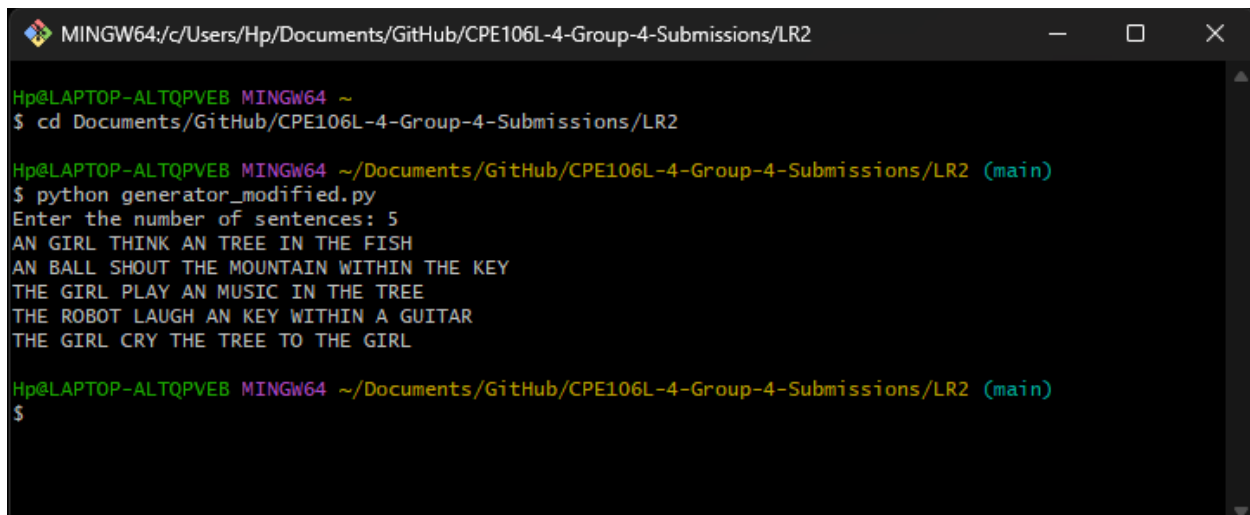


Figure 4.2 Programming Problem #3 Python Script and Associated Text Files executed in Anaconda Prompt



```
MINGW64:/c/Users/Hp/Documents/GitHub/CPE106L-4-Group-4-Submissions/LR2
Hp@LAPTOP-ALTQPVEB MINGW64 ~
$ cd Documents/GitHub/CPE106L-4-Group-4-Submissions/LR2

Hp@LAPTOP-ALTQPVEB MINGW64 ~/Documents/GitHub/CPE106L-4-Group-4-Submissions/LR2 (main)
$ python generator_modified.py
Enter the number of sentences: 5
AN GIRL THINK AN TREE IN THE FISH
AN BALL SHOUT THE MOUNTAIN WITHIN THE KEY
THE GIRL PLAY AN MUSIC IN THE TREE
THE ROBOT LAUGH AN KEY WITHIN A GUITAR
THE GIRL CRY THE TREE TO THE GIRL

Hp@LAPTOP-ALTQPVEB MINGW64 ~/Documents/GitHub/CPE106L-4-Group-4-Submissions/LR2 (main)
$
```

*Figure 4.3 Programming Problem #3 Python Script and Associated Text Files executed in GitBash*

Figures 4.2 and 4.3 present the execution of `generator_modified.py` along with its associated text files using both the Anaconda Prompt and GitBash, respectively. The figures showcase how the script interacts with the text files, demonstrating its functionality and compatibility within the Anaconda and GitBash environments.