



MAPÚA UNIVERSITY

SCHOOL OF ELECTRICAL, ELECTRONICS, AND COMPUTER ENGINEERING

Experiment 3:

Object Oriented Design and Implementation

CPE106L (Software Design Laboratory)

Brian Matthew E. Clemente

Maria Angelica Acantilado

Chalzea Fransen C. Dytianquin

Vance David G. Samia

Group No.: **4**

Section: **FOPI01**



Readings, Insights, and Reflection

METIS #1:

Alan Dennis, Barbara Haley Wixom, David Tegarden. *Systems Analysis and Design: An Object-Oriented Approach with UML (Edition 5)*. ISBN: 9781119561217

Wiley Global Education U (2015), Chapter 4, pp 121 - 130 (Use case Diagram), Chapter 5, pp 176 - 185 (Class Diagram)

The group's reflection on *Systems Analysis and Design: An Object-Oriented Approach with UML* emphasizes the critical role of structured modeling in software development. Chapter 4, which discusses use case diagrams, highlights their importance in bridging user requirements with system functionality. By visually representing how users interact with a system, these diagrams provide clarity in defining system processes and identifying key user roles. This structured approach ensures that requirements are well understood before development begins, reducing miscommunication between stakeholders. Through using case diagrams, teams can effectively analyze user needs, streamline workflows, and prevent potential issues.

Similarly, Chapter 5's focus on class diagrams underscores the importance of defining relationships between objects to create a well-structured and maintainable system. Class diagrams serve as blueprints that guide system architecture, ensuring that data and behaviors are properly organized. Understanding these relationships is essential for developing scalable and efficient software designs. The group recognizes that mastering these modeling techniques enhances the design process by minimizing errors and promoting consistency. By applying these concepts, development teams can build systems that accurately reflect user needs while maintaining flexibility for future modifications.

METIS #2:

Kenneth A. Lambert. *Fundamentals of Python: First Programs (Edition 2)*. ISBN: 9781337671019

Cengage Learning US (2019), Chapter 9: Design with Classes

The group's reflection on *Fundamentals of Python: First Programs* emphasizes the significance of designing with classes as discussed in Chapter 9. Object-oriented programming (OOP) plays a crucial role in structuring code efficiently, and the chapter highlights how classes serve as blueprints for creating objects. By encapsulating data and behaviors within classes, developers can create modular, reusable, and scalable programs, allowing for better organization and maintainability. Additionally, the discussion explores how proper class design contributes to logical workflows, making it easier to manage complex systems and ensuring that different components interact seamlessly. The group recognizes that understanding class structures is

not only beneficial for individual coding projects but also essential for collaborative development, as it promotes consistency and clarity across a team's codebase.

The discussion also underscores the importance of key OOP concepts such as inheritance, encapsulation, and polymorphism. These principles allow for the creation of hierarchical relationships between classes, improving code efficiency and reducing redundancy. The group acknowledges that mastering these concepts leads to better software design, as it enables developers to create flexible and extensible programs. By applying structured class design, programming teams can streamline development processes, minimize errors, and improve collaboration through well-organized code. The chapter reinforces the idea that a strong foundation in OOP is essential for writing efficient and maintainable Python programs.

METIS Book #1

Clemente (Chapter 4 and Chapter 5)

Chapters 4 and 5 of *Systems Analysis and Design: An Object-Oriented Approach with UML* by Alan Dennis, Barbara Wixom, and David Tegarden emphasize the importance of visual modeling in system development. Use case diagrams and class diagrams serve as essential tools for capturing system requirements and structuring system components, respectively. A use case diagram provides a high-level view of system interactions, defining the various ways users (actors) engage with the system. By focusing on user needs and system functionality, it helps developers ensure that all critical processes are accounted for before implementation, which promotes clarity in requirements gathering and facilitates communication among stakeholders.

On the other hand, a class diagram delves into the system's internal structure by defining classes, attributes, methods, and their relationships. It acts as a blueprint for system design, ensuring consistency and reusability. The relationships between classes, such as inheritance, association, and aggregation, demonstrate how different components interact, making the system more modular and scalable. Together, the aforementioned diagrams provide a structured approach to software development, bridging the gap between user requirements and technical implementation. Mastering their use allows for more efficient system analysis and design, ultimately leading to more reliable and maintainable software solutions.

Acantilado (Chapter 4 and 5)

Understanding the significance of class and using case diagrams in system modeling deepens my appreciation for structured software development. Chapter 4 highlights the importance of using case-driven development in capturing user-system interactions, ensuring that all functional requirements are identified before moving to more complex design stages. Use cases provide a high-level view of how the system should behave, allowing analysts to refine and validate requirements early in the process. The distinction between as-is and to-be systems further reinforces that system development is an evolving process where existing workflows must be carefully analyzed and improved.

Likewise, Chapter 5 shifts the focus to structural modeling, where class diagrams play a crucial role in defining how objects and their relationships are structured within a system. While functional models describe what a system should accomplish, structural models serve as a blueprint for organizing information and business logic, ensuring clarity and efficiency in system design. By visually representing key system components and their interactions, class diagrams help developers establish a solid foundation for scalability and maintainability. The concept of analysis classes and their associations is essential in structuring data effectively, allowing for reusability and adaptability as system requirements evolve. Furthermore, the iterative nature of structural modeling aligns with real-world software engineering practices, enabling continuous refinement and flexibility in design- creating a cohesive system that meets business needs and long-term sustainability.

Dytianquin (Chapter 4 and 5)

Use case diagrams and their critical function in the early phases of system analysis were thoroughly introduced in Chapter 4. The chapter illustrates how these diagrams capture interactions between external actors and the system, ensuring that all functional requirements are identified and documented. The significance of matching system functionality with user wants is emphasized by the emphasis on iterative refinement and the comparison of the current state with the intended future state. This structured approach facilitates clear communication among stakeholders and provides a strong foundation for subsequent design phases.

Likewise, Chapter 5 delved into the intricacies of class diagrams, providing a detailed blueprint of a system's internal architecture. A strong and modular design framework that supports scalability and maintainability can be developed through the chapter's examination of classes, attributes, methods, and their interactions. The chapter emphasized the significance of a well-structured technological framework that is in line with both business logic and operational requirements by successfully tying conceptual design to real-world implementation.

Samia (Chapter 4 and Chapter 5)

The comprehension of the essential function of class diagrams and use case diagrams in system modeling enhances my admiration for organized software development. The focus on the use of case-driven development in Chapter 4 emphasizes the importance of functional modeling in capturing user-system interactions. Before proceeding to more intricate design stages, analysts make sure that all user requirements are taken into consideration by establishing use cases, which provide a high-level depiction of the system's behavior. It is further supported by the contrast between the as-is and to-be systems that system development is a dynamic process in which current workflows need to be thoroughly examined. Furthermore, the introduction of activity diagrams as an additional tool for process modeling makes it clear how processes can be organized visually to improve comprehension.

However, Chapter 5 turns its attention to structural modeling, where class diagrams are essential for specifying how objects supporting business activities are organized. Class diagrams

show how information is organized within a system, whereas functional models outline what the system should be able to perform. The foundation for creating scalable and maintainable systems is the idea of analysis classes and their connections. From conceptual design to implementation, the iterative process of creating structural models is consistent with practical software engineering techniques. Developers can build a solid system that satisfies business requirements by making sure that such models are consistent with one another.

After giving these ideas some thought, I can see how both class diagrams and use case diagrams support a clear system design. Class diagrams give implementation its structural foundation, while use cases aid in comprehending user interactions. In actuality, creating systems that are not just useful but also effective and maintainable requires striking a balance between the two viewpoints. The significance of object-oriented analysis and design (OOAD) as a technique that connects technical solutions with user needs is emphasized by this. As I continue to study system modeling, I see how important it is to become proficient in these methods to create software that is both structurally robust and user centric.

METIS Book #2

Clemente (Chapter 9)

Object-oriented programming (OOP) is a fundamental paradigm in Python that enhances code organization, reusability, and maintainability. Chapter 9 of *Fundamentals of Python: First Programs* by Kenneth A. Lambert introduces key concepts of OOP, including class design, instance variables, method definitions, and advanced techniques like operator overloading and exception handling. These concepts provide a structured approach to software development, making programs more scalable and easier to manage.

One of the most valuable takeaways from this chapter is the role of encapsulation, which ensures that an object's internal data is protected and only modified through well-defined methods. This promotes reliability and prevents unintended modifications. Inheritance further strengthens OOP by allowing new classes to extend existing ones, reducing redundancy and encouraging code reuse. Additionally, polymorphism simplifies interactions between different object types by enabling methods to share the same name while behaving appropriately depending on the context.

Beyond these core principles, the chapter also covers practical aspects like pickling, which allows objects to be saved and loaded for persistent storage, and exception handling, which ensures robustness by managing unexpected errors gracefully. The model/view design pattern is particularly insightful, as it highlights the separation between data management and user interaction, a crucial approach for building maintainable applications. Overall, Chapter 9 underscores the importance of designing classes with clarity and efficiency. By mastering these OOP principles, programmers can develop modular, reusable, and scalable software, aligning with best practices in modern development.

Acantilado (Chapter 9)

Chapter 9 explores object-oriented programming in Python, highlighting its importance in enhancing code organization, reusability, and maintainability. It introduces fundamental concepts such as class design, instance variables, and method definitions, which provide a structured approach to software development. Advanced techniques like operator overloading and exception handling further demonstrate how OOP simplifies complex programming tasks. Encapsulation plays a key role in protecting an object's internal data, ensuring modifications occur only through well-defined methods, thereby improving reliability. Inheritance reduces redundancy by enabling new classes to build upon existing ones, while polymorphism enhances flexibility by allowing methods to share the same name but function depending on the context.

Beyond these core principles, the chapter delves into practical applications that make OOP even more powerful in real-world programming. Pickling, for example, enables object persistence by allowing data to be efficiently stored and retrieved, which is particularly useful for applications requiring long-term data management. Exception handling is another crucial topic, ensuring that programs remain robust and can gracefully handle unexpected errors without crashing. Additionally, the chapter introduces the model/view design pattern, which emphasizes the separation of data management from user interaction, leading to more maintainable and scalable applications. This design approach is especially beneficial in large software projects where keeping concerns separate improves code readability and maintainability. By understanding and applying these principles, developers can write modular, scalable, and efficient programs that adhere to modern engineering practices.

Dytianquin (Chapter 9)

Lists are among the most flexible and widely used data structures in Python, allowing efficient storage and manipulation of data. Unlike stacks and queues, which follow strict order-based rules such as last-in, first-out or first-in, first-out, lists provide more freedom by enabling modifications based on position, content, or index. This adaptability makes them useful in a variety of applications, from simple data organization to complex dynamic data processing. A key distinction between lists and arrays lies in their structure and usage—lists are an abstract data type that can be implemented using either an array or a linked structure, while arrays are low-level data structures that require a continuous block of memory. Both use indices for element access, but lists offer greater flexibility, making them well-suited for scenarios where data needs to be frequently modified or expanded.

Because lists allow direct access and modification of elements based on their position, they are highly effective for managing ordered data. However, inserting or removing elements in the middle of a list can be inefficient, especially in array-based implementations, as it often requires shifting other elements to maintain order. Modifications based on content rather than position provide an alternative way to manage data, making lists easier to use when frequent updates, searches, or reorganizations are needed. This dynamic nature allows lists to efficiently handle changing data requirements, making them an essential tool in many programming tasks, from simple data storage to more advanced algorithmic applications.

Samia (Chapter 9)

With a large range of functions that enable effective data processing, lists are among Python's most popular and adaptable data structures. Lists offer more flexibility by enabling index-based, content-based, and position-based operations, in contrast to stacks and queues, which adhere to order limitations (LIFO and FIFO, respectively). The distinction between lists and arrays is among the most important lessons to be learned from this reading. A list is an abstract data type that can be implemented using either an array or a linked structure, while an array is a low-level data structure that needs contiguous memory allocation. Both employ indices to access elements. Because of their adaptability, lists can be used for a variety of tasks, including sophisticated algorithms and dynamic data storage.

Elements can be directly accessed and modified via their positions thanks to index-based operations. Lists are an effective tool for managing ordered data because of methods like *.insert(i, item)* and *.pop(i)* that make it possible to insert and remove elements quickly. However, because it necessitates shifting components to make room for the change, adding entries in the middle of a list can be expensive in terms of temporal complexity, particularly in array-based implementations. By using values rather than positions, the content-based procedures present an alternative viewpoint. Lists can be easily modified dynamically with functions like *add(item)*, *remove(item)*, and *index(item)*, especially in situations where elements are constantly altered and searched. In applications that need dynamic data retrieval and storage, these procedures improve lists' usefulness.

Answers to Questions

1. b. can allow the user to enter inputs in any order
2. b. EasyFrame
3. c. command
4. All of the above
5. b. text area
6. c. scroll bar
7. a. controls the alignment of a window component in its grid cell
8. b. radio button
9. b. panel
10. a. (0, 0)

PostLab

Programming Problems

This post-lab activity consists of three programming tasks that reinforce key concepts in object-oriented programming, class design, and sorting techniques. The tasks focus on implementing object comparisons, manipulating lists of objects, and ensuring an ordered string representation of data. These exercises highlight essential principles such as encapsulation, inheritance, and structured modeling, which contribute to writing modular and maintainable code. Each programming problem was assigned to different group members, with individual contributions documented within the code files. The development process emphasized clarity and organization, utilizing structured modeling techniques and version control tools to track modifications and ensure collaboration. Furthermore, executing the tasks in different environments, such as Anaconda Prompt and GitBash, validated the functionality and adaptability of the implemented methods, demonstrating a practical application of the discussed programming concepts.

Programming Problem #1

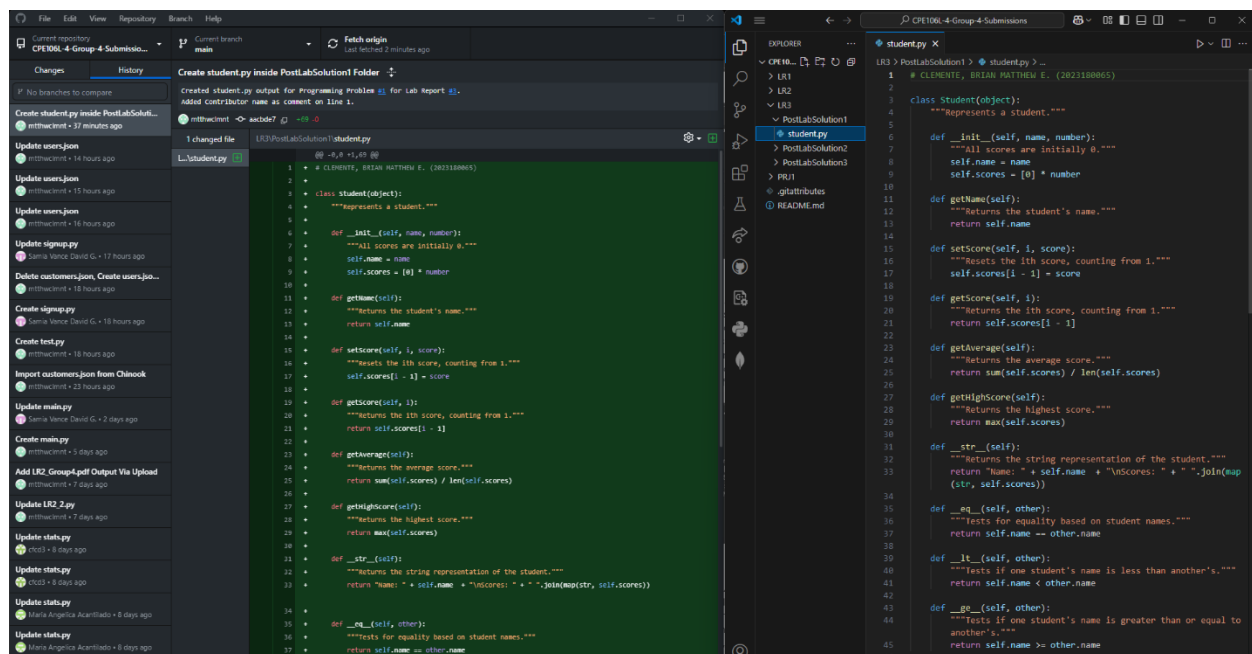


Figure 1.1 Screenshot of GitHub Desktop and VS Code open after the Programming Problem #1 Python File Commit and Push to Origin

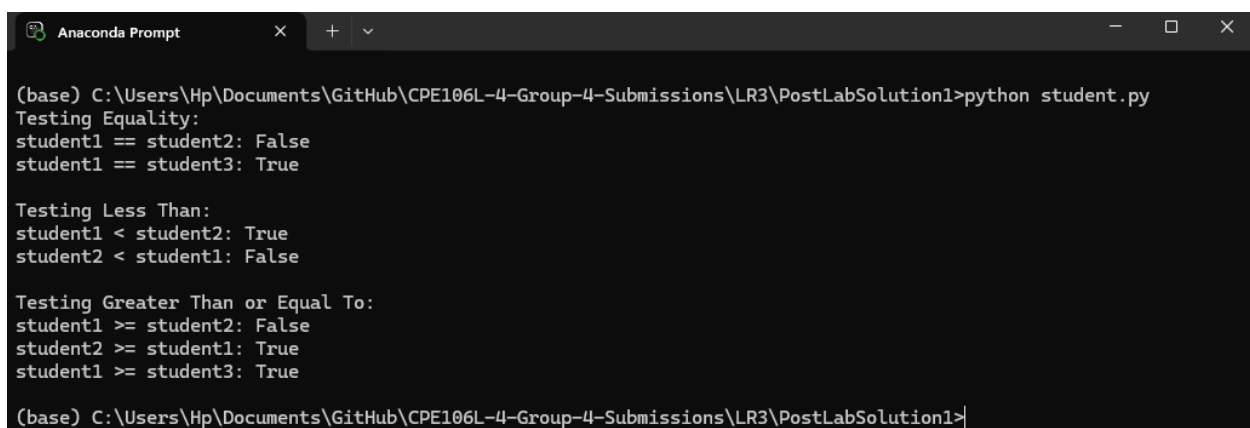
Figure 1.1 displays a screenshot capturing the use of GitHub and Visual Studio Code after the commit and push process for the *student.py* file, which contains the group's output for

Programming Problem #1. The image illustrates how GitHub Desktop tracks changes made to the Python file, showing the creation and subsequent updates reflecting the modifications in the code. Meanwhile, Visual Studio Code is open, displaying the edited *student.py* file with the newly implemented comparison methods for the *Student* class. This highlights the active coding process and version control workflow, ensuring that all modifications are properly documented and synchronized across the team. The integration of GitHub allows for efficient collaboration, version tracking, and error management, reinforcing the best practices for software development.

```
48 def main():
49     """Tests the comparison operators."""
50     student1 = Student("Charlie", 3) # lowercase letters will be considered *greater* than uppercase (e.g. ethan vs Ethan)
51     student2 = Student("David", 3)
52     student3 = Student("Charlie", 3)
53
54     print("Testing Equality:")
55     print(f"student1 == student2: {student1 == student2}") # False (Charlie != David)
56     print(f"student1 == student3: {student1 == student3}") # True (Charlie == Charlie)
57
58     print("\nTesting Less Than:")
59     print(f"student1 < student2: {student1 < student2}") # True (Charlie comes BEFORE David, alphabetically)
60     print(f"student2 < student1: {student2 < student1}") # False (Charlie comes AFTER David, alphabetically)
61
62     print("\nTesting Greater Than or Equal To:")
63     print(f"student1 >= student2: {student1 >= student2}") # False (Charlie comes BEFORE David)
64     print(f"student2 >= student1: {student2 >= student1}") # True (David comes AFTER Charlie)
65     print(f"student1 >= student3: {student1 >= student3}") # True (Charlie is IDENTICAL to Charlie)
```

Figure 1.2 Programming Problem #1 main() Function that Tests all the Required Comparison Operators

Additionally, Figure 1.2 demonstrates the use of multiple print() functions, which play a crucial role in displaying and testing the comparison operators within the *student.py* file. By printing the results of various test cases, the group ensures that the implemented methods function correctly and produce the expected outputs. This systematic approach to testing helps in debugging potential errors and refining the logic before finalizing the program. The combination of version control in GitHub and real-time coding in Visual Studio Code demonstrates an efficient workflow, promoting collaboration, accuracy, and maintainability in software development.



```
(base) C:\Users\Hp\Documents\GitHub\CPE106L-4-Group-4-Submissions\LR3\PostLabSolution1>python student.py
Testing Equality:
student1 == student2: False
student1 == student3: True

Testing Less Than:
student1 < student2: True
student2 < student1: False

Testing Greater Than or Equal To:
student1 >= student2: False
student2 >= student1: True
student1 >= student3: True

(base) C:\Users\Hp\Documents\GitHub\CPE106L-4-Group-4-Submissions\LR3\PostLabSolution1>
```

Figure 1.3 Programming Problem #1 Python Script executed in Anaconda Prompt

```
MINGW64/c/Users/Hp/Documents/GitHub/CPE106L-4-Group-4-Submissions/LR3/PostLabSolution1

Hp@LAPTOP-ALTQPVEB MINGW64 ~/Documents/GitHub/CPE106L-4-Group-4-Submissions/LR3/PostLabSolution1 (main)
$ python student.py
Testing Equality:
student1 == student2: False
student1 == student3: True

Testing Less Than:
student1 < student2: True
student2 < student1: False

Testing Greater Than or Equal To:
student1 >= student2: False
student2 >= student1: True
student1 >= student3: True

Hp@LAPTOP-ALTQPVEB MINGW64 ~/Documents/GitHub/CPE106L-4-Group-4-Submissions/LR3/PostLabSolution1 (main)
$
```

Figure 1.4 Programming Problem #1 Python Script executed in GitBash

Furthermore, the confines of both *Figure 1.3* and *Figure 1.4* present the execution of *student.py* using both Anaconda Prompt and GitBash, respectively. These figures showcase how the script correctly outputs the required comparison operators within the *main()* function, verifying its functionality across different execution environments. By running the script in both terminals, the group ensures that the implemented methods behave consistently and produce accurate results regardless of the execution platform. This demonstrates the script's compatibility with both Anaconda and GitBash, reinforcing its reliability and effectiveness in handling object comparisons while maintaining cross-platform functionality.

Programming Problem #2

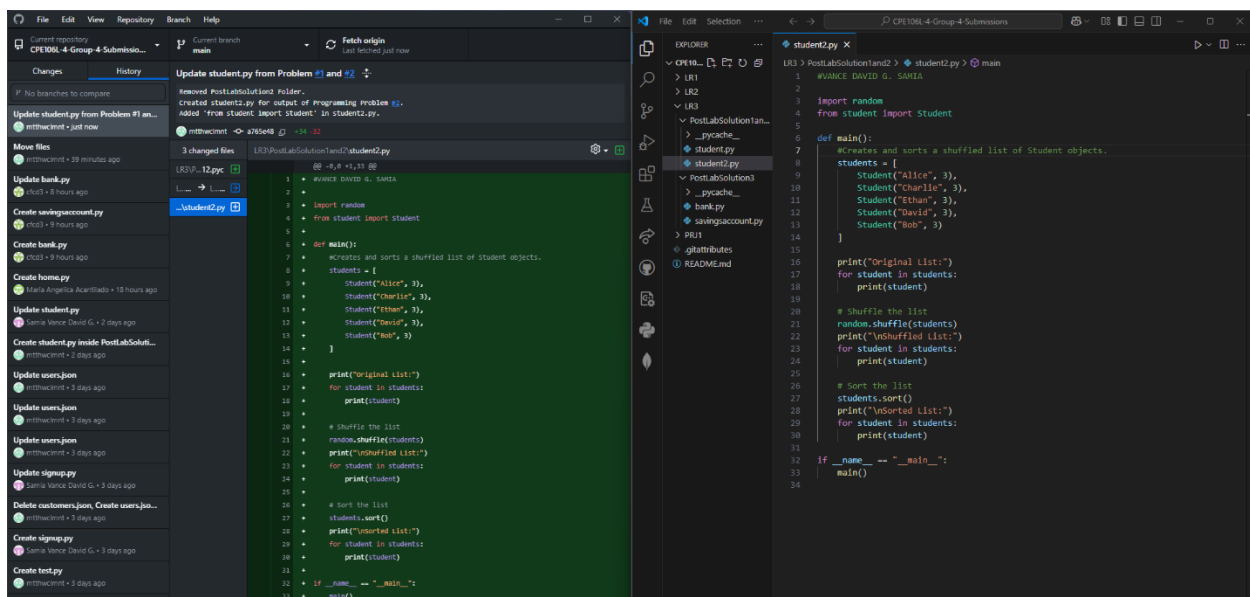
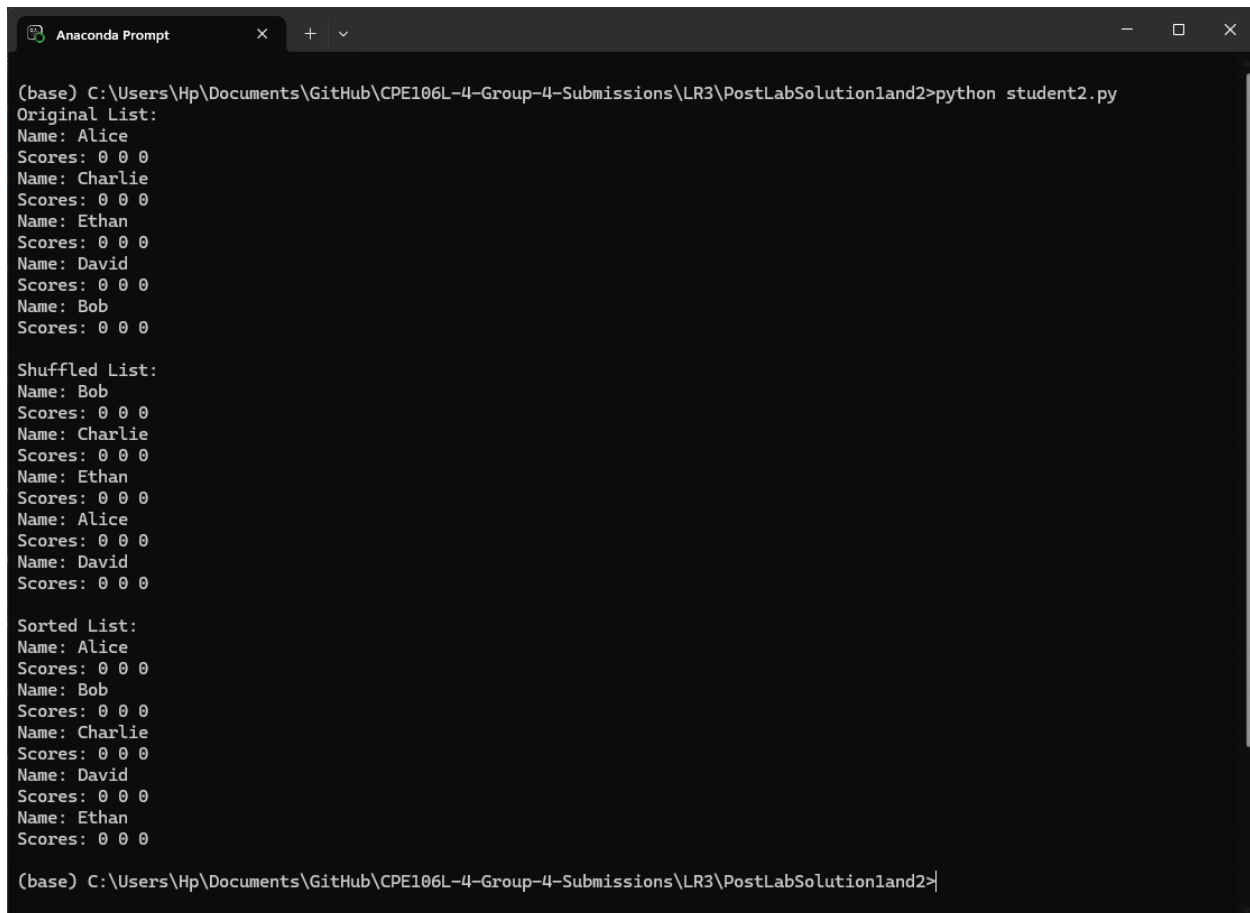


Figure 2.1 Screenshot of GitHub Desktop and VS Code open after the Programming Problem #2 Python File Commit and Update

A screenshot of GitHub Desktop and Visual Studio Code is shown in *Figure 2.1* following the commit and push of the Python script improvements for Programming Problem #2. In this image, GitHub Desktop records the version control procedure, tracking changes made to the file, while VS Code remains active, displaying the modified code related to sorting objects within a list. This demonstrates how GitHub enables a smooth transition between active coding and version control, ensuring that all modifications are properly documented and synchronized. The integration of these tools highlights the efficiency of collaborative programming, allowing the team to refine and test the implementation of sorting methods while maintaining an organized workflow.



```
(base) C:\Users\Hp\Documents\GitHub\CPE106L-4-Group-4-Submissions\LR3\PostLabSolution1and2>python student2.py
Original List:
Name: Alice
Scores: 0 0 0
Name: Charlie
Scores: 0 0 0
Name: Ethan
Scores: 0 0 0
Name: David
Scores: 0 0 0
Name: Bob
Scores: 0 0 0

Shuffled List:
Name: Bob
Scores: 0 0 0
Name: Charlie
Scores: 0 0 0
Name: Ethan
Scores: 0 0 0
Name: Alice
Scores: 0 0 0
Name: David
Scores: 0 0 0

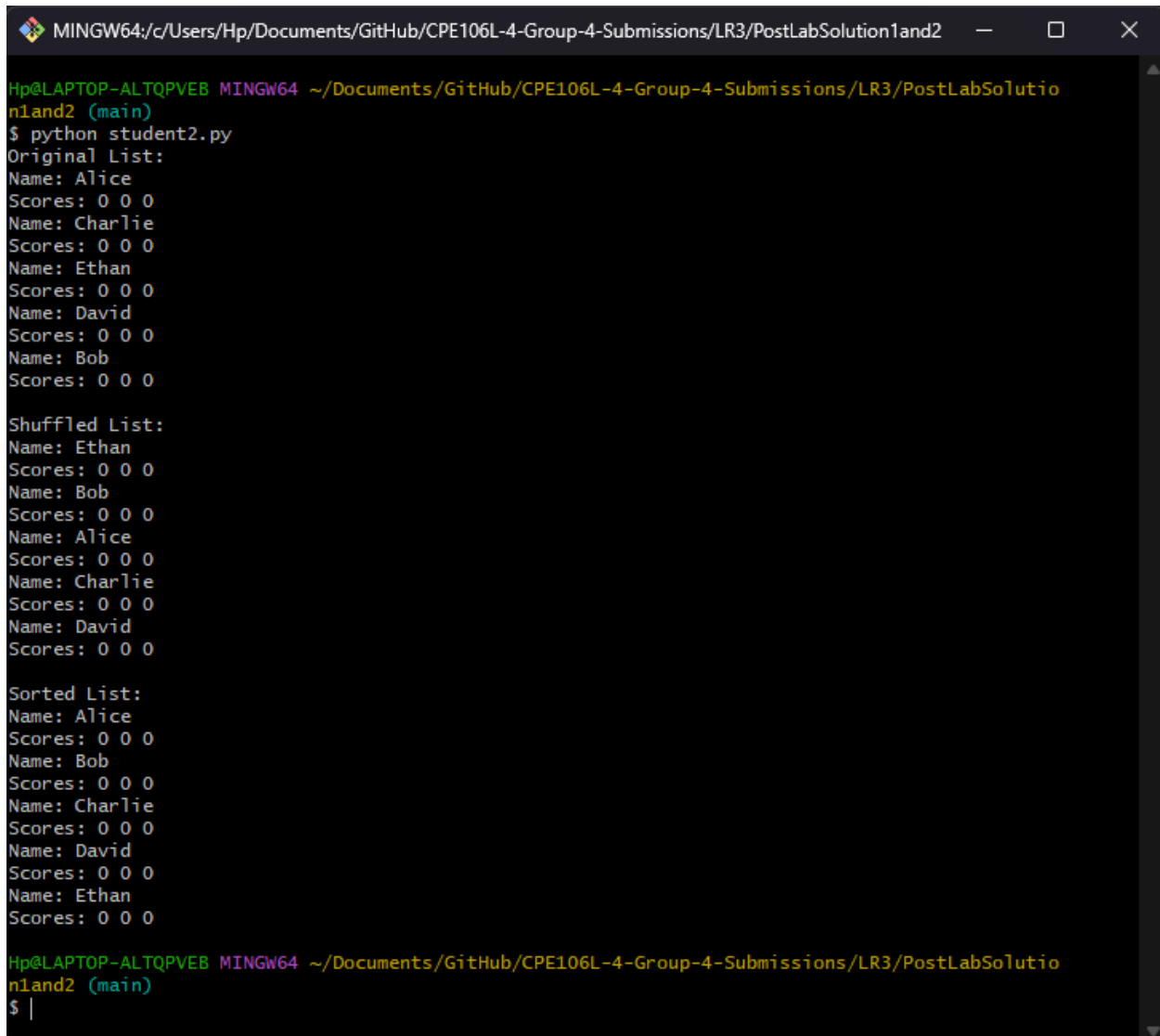
Sorted List:
Name: Alice
Scores: 0 0 0
Name: Bob
Scores: 0 0 0
Name: Charlie
Scores: 0 0 0
Name: David
Scores: 0 0 0
Name: Ethan
Scores: 0 0 0

(base) C:\Users\Hp\Documents\GitHub\CPE106L-4-Group-4-Submissions\LR3\PostLabSolution1and2>
```

Figure 2.2 Programming Problem #2 Python Script executed in GitBash

The execution of the modified Python script for Programming Problem #2 in the GitBash environment is shown in *Figure 2.2*. The output verifies that the list of student objects is correctly shuffled before being sorted by name, demonstrating the proper functionality of the implemented sorting algorithm. This confirms that the script successfully applies comparison techniques within the class to arrange the objects in the desired order. The results validate the correctness and efficiency of the sorting logic, ensuring that the program produces consistent and expected outcomes across different executions. Additionally, running the script in GitBash highlights its

compatibility with various execution environments, reinforcing the robustness and reliability of the implemented methods in handling data with accuracy, efficiency, and stability across multiple test runs, regardless of initial input conditions or execution variations.



```
MINGW64:/c/Users/Hp/Documents/GitHub/CPE106L-4-Group-4-Submissions/LR3/PostLabSolution1and2
Hp@LAPTOP-ALTQPVEB MINGW64 ~/Documents/GitHub/CPE106L-4-Group-4-Submissions/LR3/PostLabSolution1and2 (main)
$ python student2.py
Original List:
Name: Alice
Scores: 0 0 0
Name: Charlie
Scores: 0 0 0
Name: Ethan
Scores: 0 0 0
Name: David
Scores: 0 0 0
Name: Bob
Scores: 0 0 0

Shuffled List:
Name: Ethan
Scores: 0 0 0
Name: Bob
Scores: 0 0 0
Name: Alice
Scores: 0 0 0
Name: Charlie
Scores: 0 0 0
Name: David
Scores: 0 0 0

Sorted List:
Name: Alice
Scores: 0 0 0
Name: Bob
Scores: 0 0 0
Name: Charlie
Scores: 0 0 0
Name: David
Scores: 0 0 0
Name: Ethan
Scores: 0 0 0

Hp@LAPTOP-ALTQPVEB MINGW64 ~/Documents/GitHub/CPE106L-4-Group-4-Submissions/LR3/PostLabSolution1and2 (main)
$ |
```

Figure 2.3 Programming Problem #2 Python Script executed in GitBash

Another execution of the script in GitBash further confirms the consistency of the output, as reflected in Figure 2.3. The final sorted list verifies that the program accurately arranges student items in alphabetical order, while the shuffled list highlights the randomness introduced before sorting. This repeated execution ensures that the implemented sorting algorithm functions correctly across different runs, demonstrating its accuracy and reliability. The results reinforce the effectiveness of the comparison logic and the stability of the sorting method, validating its dependability in handling ordered data within the program. Additionally, this successful execution emphasizes the importance of rigorous testing in different environments to ensure that the

sorting algorithm consistently produces the expected results, regardless of initial list order or execution conditions. This process highlights the role of systematic debugging and iterative refinement in software development, ensuring that the algorithm remains efficient, adaptable, and free from unexpected errors in various runtime scenarios.

Programming Problem #3

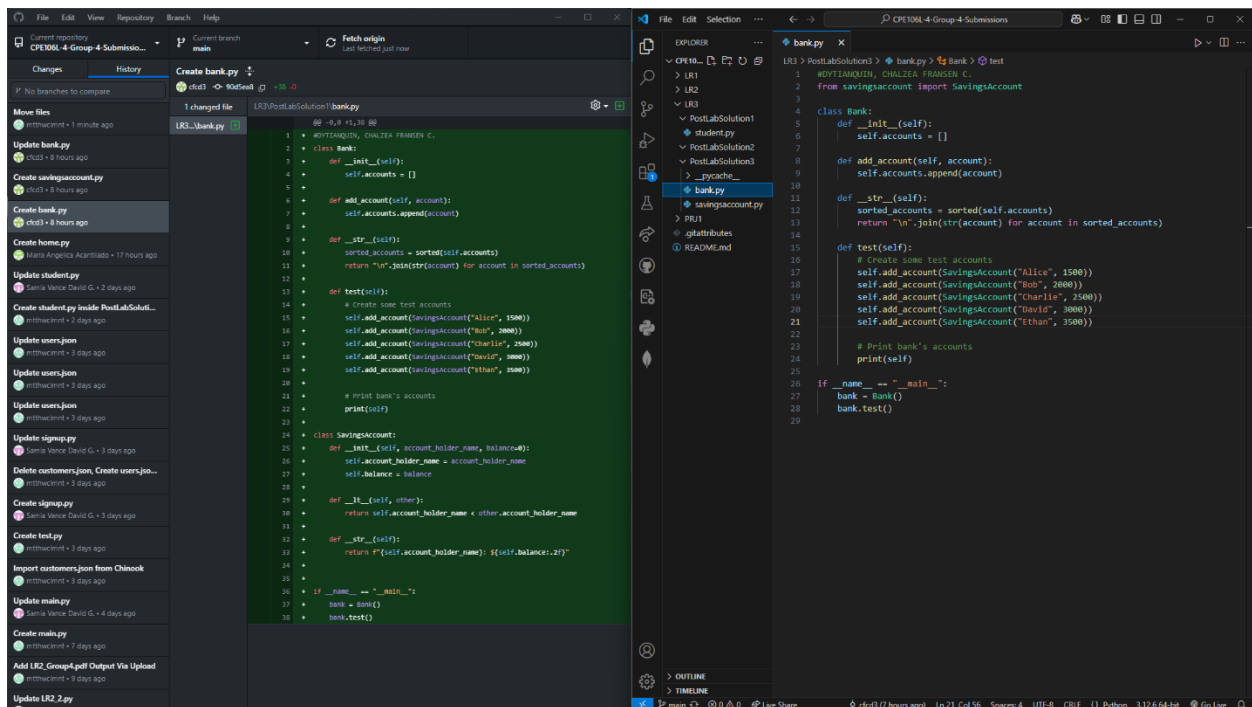


Figure 3.1 Screenshot of GitHub Desktop and VS Code open after the Programming Problem #3 Python File Commit and Push to Origin

Figure 3.1 showcases the process of committing and pushing updates to the group's version control system after implementing modifications to improve data organization and presentation. The changes made ensure that information is displayed in a structured and systematic manner rather than appearing in an arbitrary sequence, enhancing readability and usability. The screenshot highlights the seamless integration of coding and version control, demonstrating an efficient workflow in refining the program's functionality while maintaining accuracy. By leveraging collaborative tools, the group ensures proper documentation of revisions, effectively tracks modifications, and facilitates seamless teamwork. This approach reinforces the importance of structured development practices in enhancing both program reliability and team efficiency, ensuring that updates are consistently tested, reviewed, and deployed with precision.

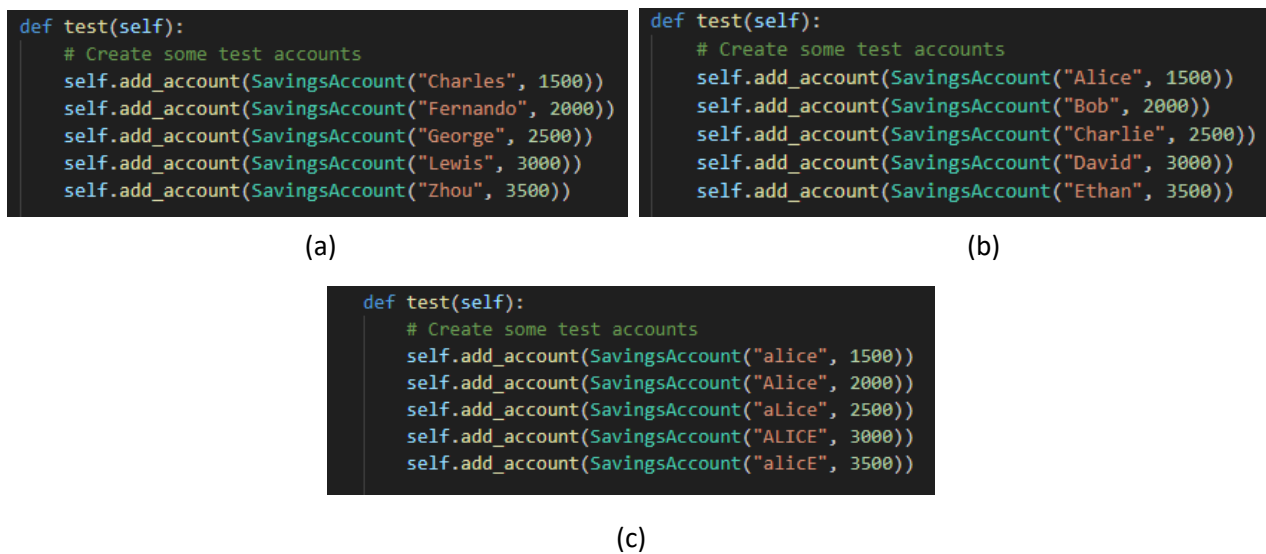


Figure 3.2 Screenshot of Test Accounts for Programming Problem #3.

Figure 3.2 presents multiple test cases to verify the accuracy and consistency of the implemented modifications. In (a), the test accounts appear in a randomized sequence, demonstrating the initial unstructured state of the data before any sorting is applied. This highlights the need for an effective sorting mechanism to ensure that the accounts are systematically organized. In (b), the accounts are successfully arranged in alphabetical order, confirming that the implemented sorting logic functions correctly and organizes the data as expected. The transition from (a) to (b) showcases the effectiveness of the applied modifications in structuring unordered data.

Meanwhile, (c) displays test accounts with the same name, "Alice," but with varying letter cases, testing the program's ability to handle case sensitivity properly, as reflected under Figure 3.2. This scenario ensures that the sorting algorithm remains consistent regardless of differences in capitalization, reinforcing the program's robustness.

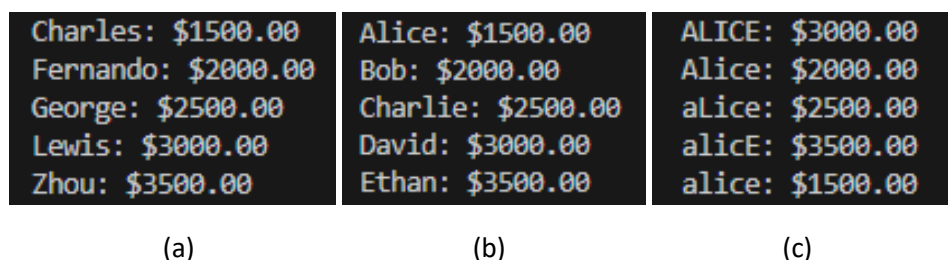
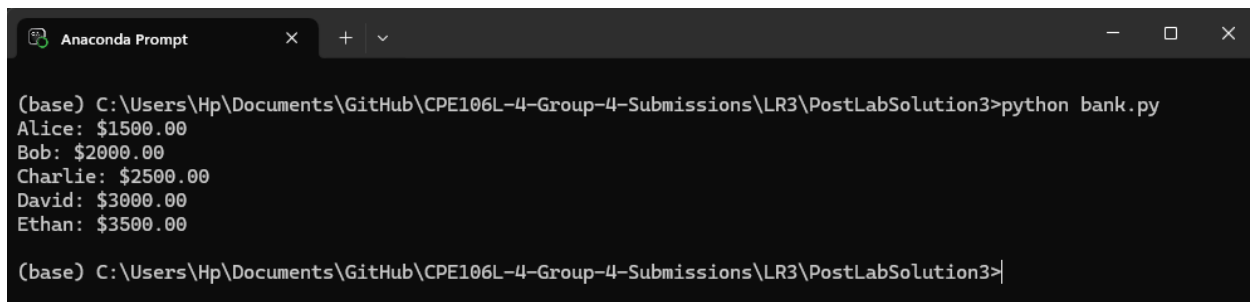


Figure 3.3 Multiple Screenshots of the Test Account Outputs from Figure 3.2

Figure 3.3 presents multiple test cases to validate the accuracy of the sorting implementation. In (a), the test accounts are successfully arranged in alphabetical order, demonstrating the program's ability to correctly organize unordered data. This confirms that the implemented sorting

mechanism effectively processes and arranges account names based on the desired criteria. In contrast, (b) displays test accounts that remain unchanged, as they were already in alphabetical order before execution. This scenario verifies that the program does not apply unnecessary modifications to already sorted data, ensuring efficiency and preserving the original structure when no adjustments are required.

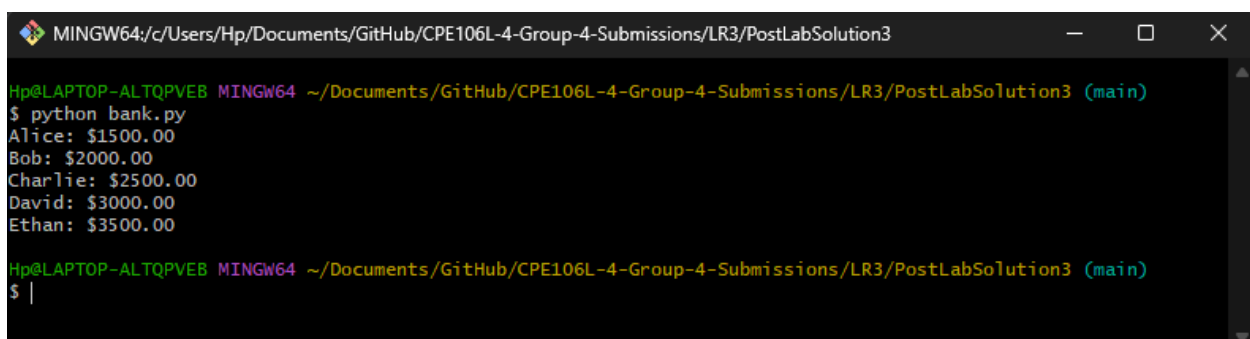
Likewise, (c) illustrates the test accounts sorted according to letter cases, with capital letters appearing before lowercase ones. This case sensitivity in ordering highlights how the sorting algorithm differentiates between uppercase and lowercase characters, following a specific hierarchy in character comparison. These test cases collectively reinforce the functionality and reliability of the sorting method by demonstrating its effectiveness in handling various input conditions. By testing different scenarios, the group ensures that the program consistently produces the expected results, maintaining accuracy and precision across multiple executions.



```
(base) C:\Users\Hp\Documents\GitHub\CPE106L-4-Group-4-Submissions\LR3\PostLabSolution3>python bank.py
Alice: $1500.00
Bob: $2000.00
Charlie: $2500.00
David: $3000.00
Ethan: $3500.00

(base) C:\Users\Hp\Documents\GitHub\CPE106L-4-Group-4-Submissions\LR3\PostLabSolution3>|
```

Figure 3.4 Programming Problem #3 Python Script executed in GitBash



```
MINGW64/c/Users/Hp/Documents/GitHub/CPE106L-4-Group-4-Submissions/LR3/PostLabSolution3
Hp@LAPTOP-ALTQPVEB MINGW64 ~/Documents/GitHub/CPE106L-4-Group-4-Submissions/LR3/PostLabSolution3 (main)
$ python bank.py
Alice: $1500.00
Bob: $2000.00
Charlie: $2500.00
David: $3000.00
Ethan: $3500.00

Hp@LAPTOP-ALTQPVEB MINGW64 ~/Documents/GitHub/CPE106L-4-Group-4-Submissions/LR3/PostLabSolution3 (main)
$ |
```

Figure 3.5 Programming Problem #3 Python Script executed in GitBash

Lastly, *Figures 3.4 and 3.5* display the execution of the Python script for Programming Problem #3 in different command-line environments, demonstrating its consistency and accuracy. In both Anaconda Prompt and GitBash, the script successfully processes and displays the test account data, verifying that the sorting and formatting functions operate as expected. The uniform output across these platforms confirms the program's reliability in handling account information, ensuring that the implementation remains functional regardless of the execution environment.