**Design Rationale**

1. **Choice of Data Structures**

* **Dictionary (for Books):**

Each book in the library has a unique ISBN, which serves perfectly as a dictionary key. This allows fast lookups, updates, and deletions by ISBN without having to iterate through a list. For example, if a user wants to borrow a book, the program can directly check books[isbn] instead of searching through all books.

* **List of Dictionaries (for Members):**

Members are stored in a list because the order is not critical, but we may need to iterate through all members for operations like checking who borrowed a particular book. Each member is represented as a dictionary to keep their details organized (member\_id, name, email, borrowed\_books). This combination provides both structure and flexibility.

* **Tuple (for Genres):**

Genres are fixed categories and do not change during program execution. Using a tuple ensures immutability, preventing accidental modifications and maintaining data integrity. It also makes validation easy — we can simply check if the user’s genre input exists in the tuple.

1. **Function Design**

Each operation is implemented as a separate function (CRUD + borrow/return). This design follows the Single Responsibility Principle, where each function handles one clear task. For example:

• add\_book() only adds books, checking for uniqueness and valid genre.

• borrow\_book() only handles borrowing logic, enforcing limits and updating counts.

This modularity makes the code easier to maintain, debug, and test. New features, such as tracking overdue books, could be added without changing the existing functions.

1. **Error Handling**

Functions return clear, human-readable messages like "Book added successfully!" or "No copies left!" instead of raising exceptions. This approach is beginner-friendly, helps users understand what happened, and prevents the program from crashing during normal usage. For instance, if a member tries to borrow more than 3 books, the system simply returns "Borrow limit reached!", making it easy to handle in the demo.

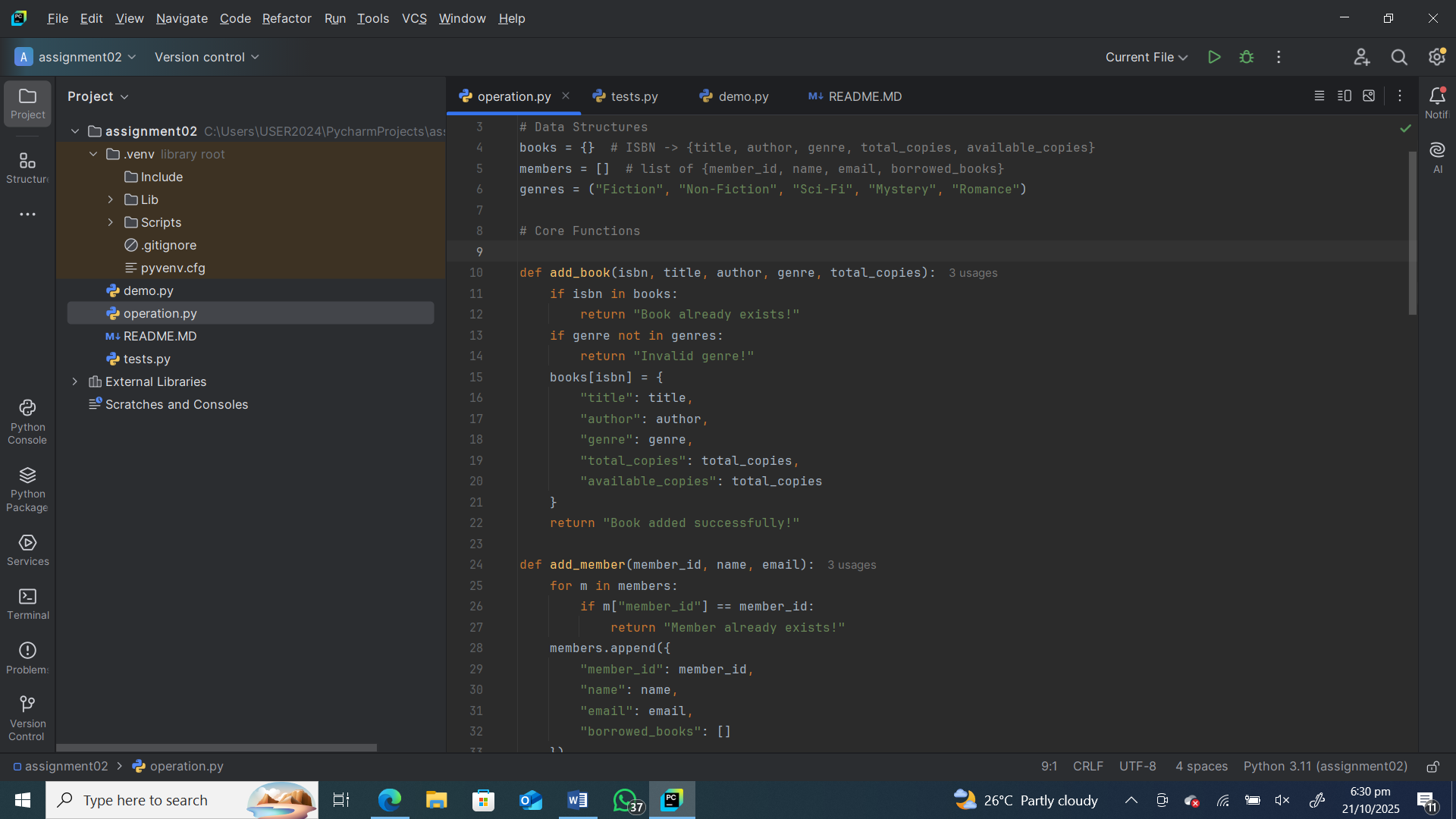
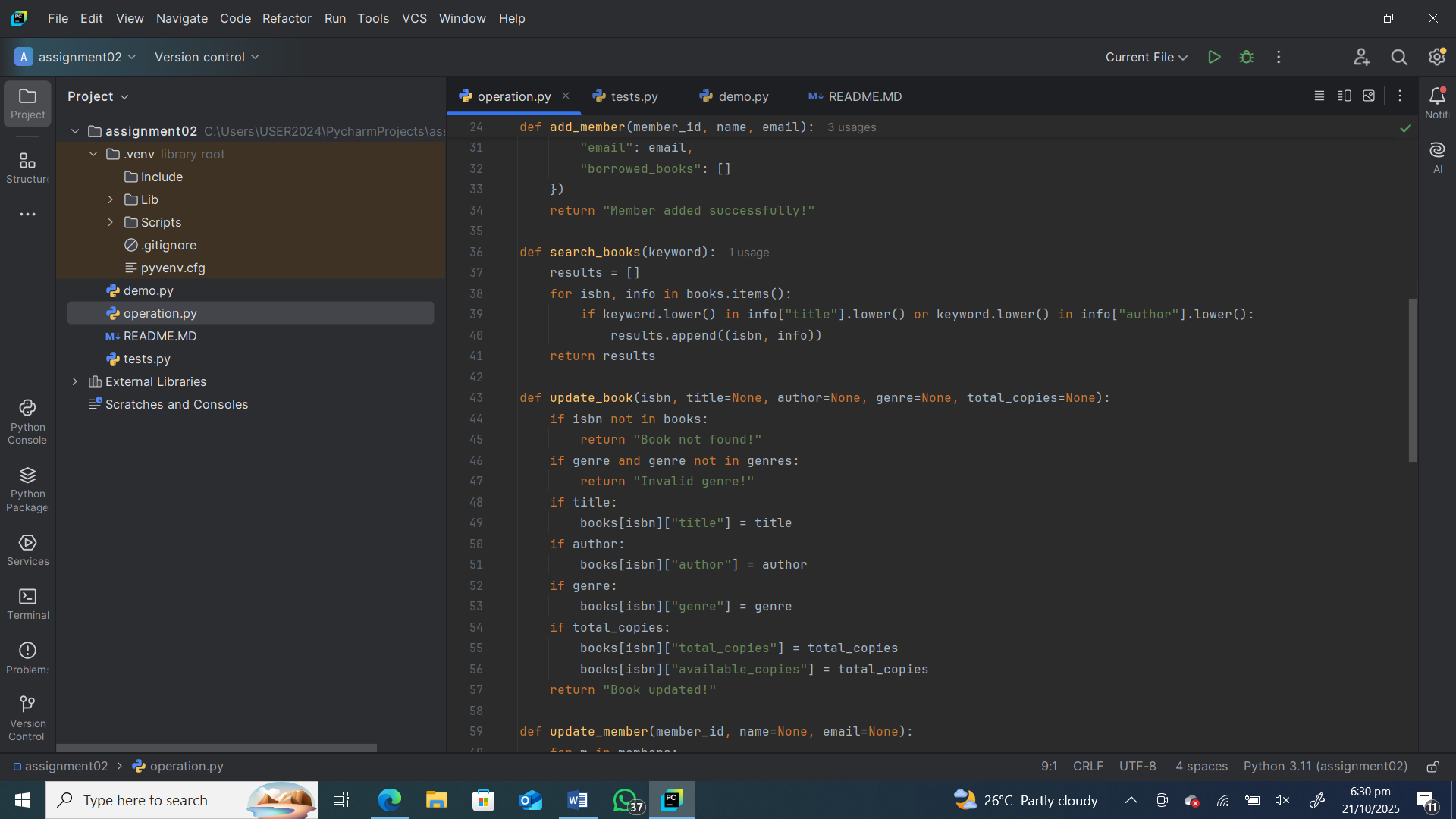
1. **Why These Choices Suit the Project**

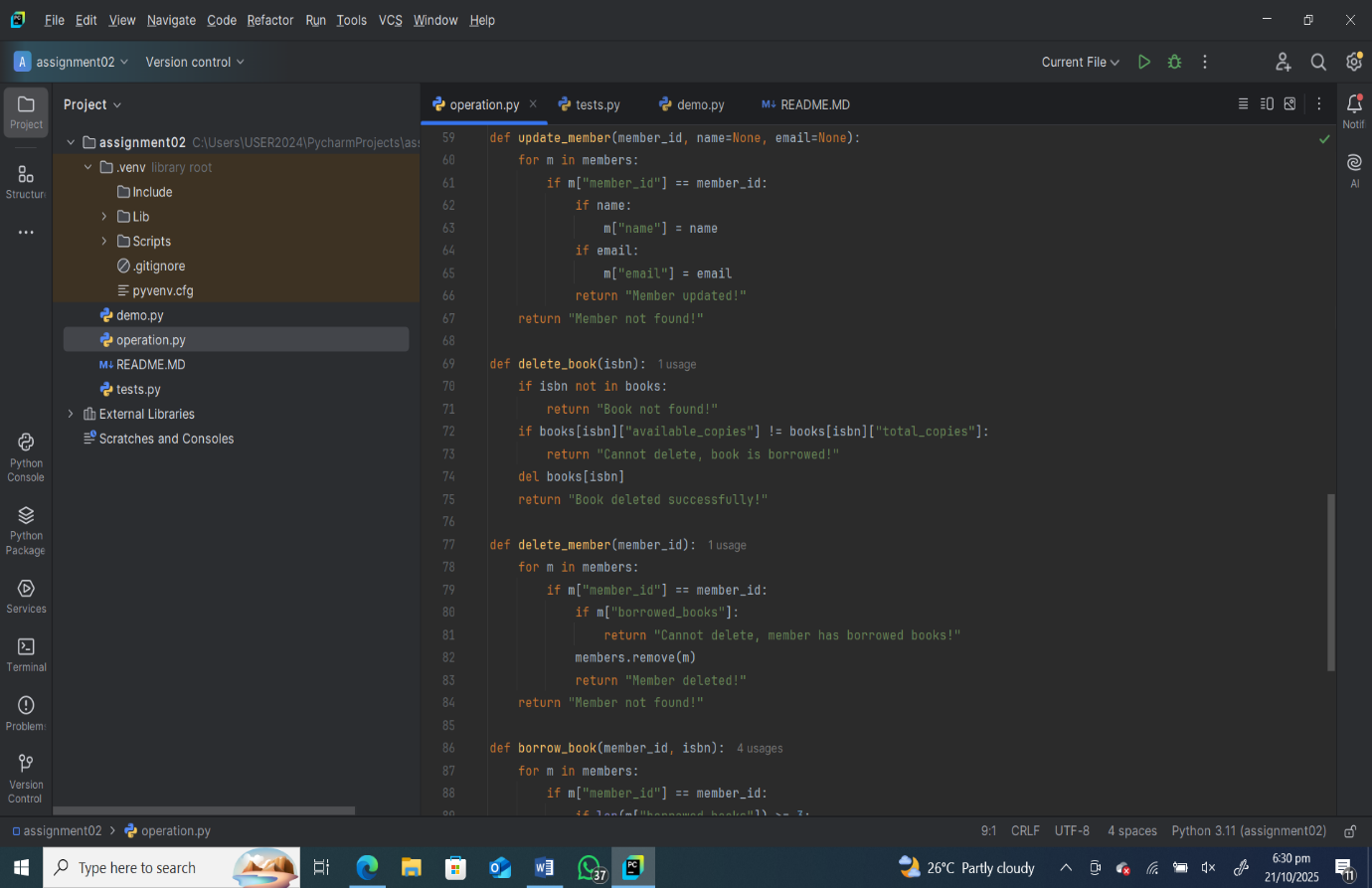
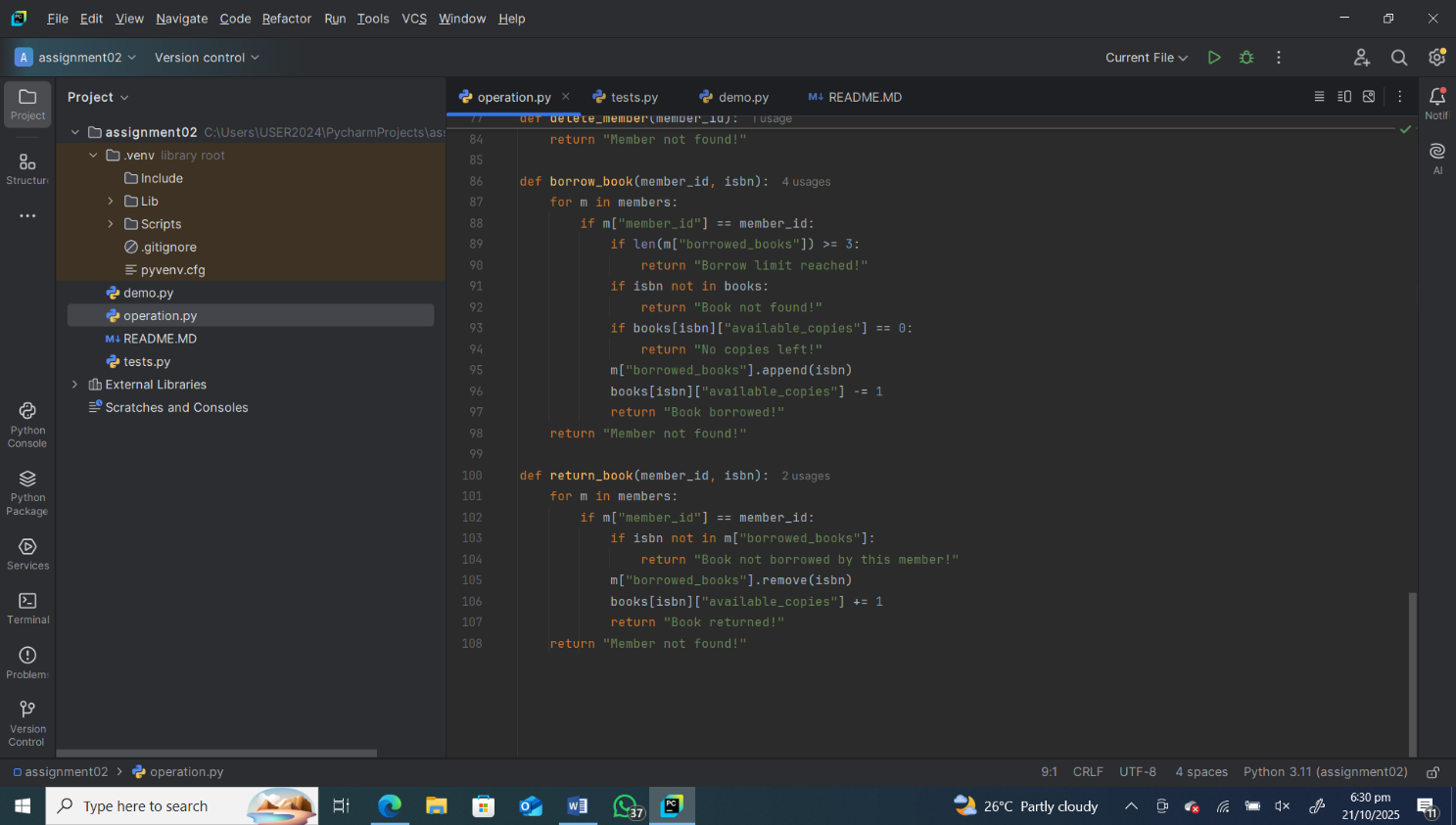
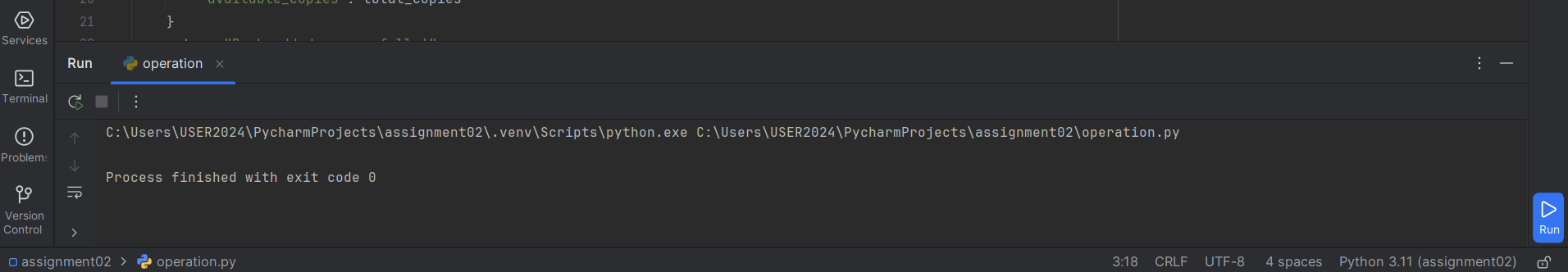
The system is small-scale, so simple structures like dictionaries and lists are sufficient and efficient.

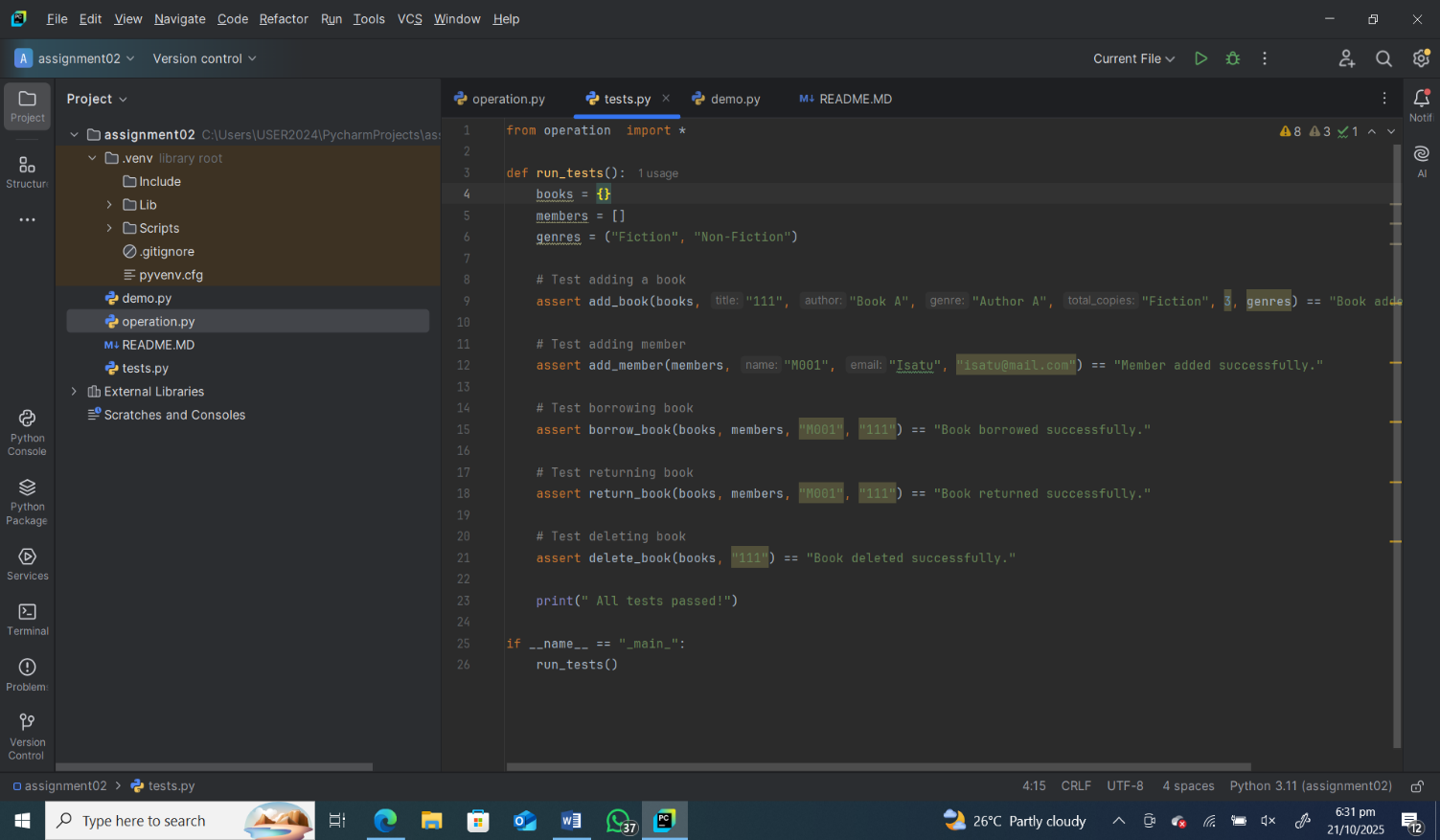
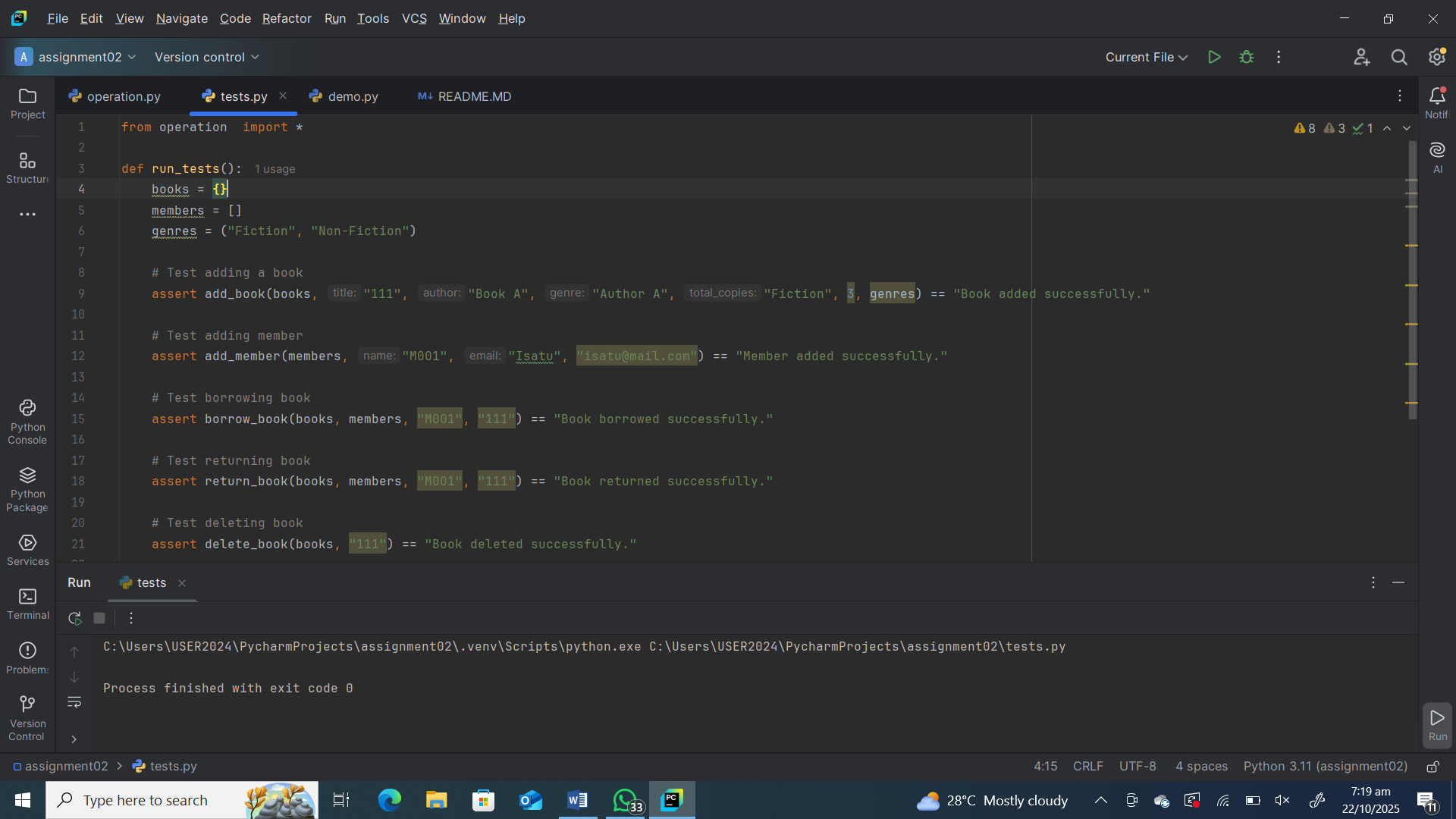
* Using functions with clear messages keeps the system interactive and readable, which is perfect for classroom assignments or demos.
* The combination of dictionaries, lists, and tuples ensures data integrity, fast access, and code clarity, which are key goals for a Library Management System.

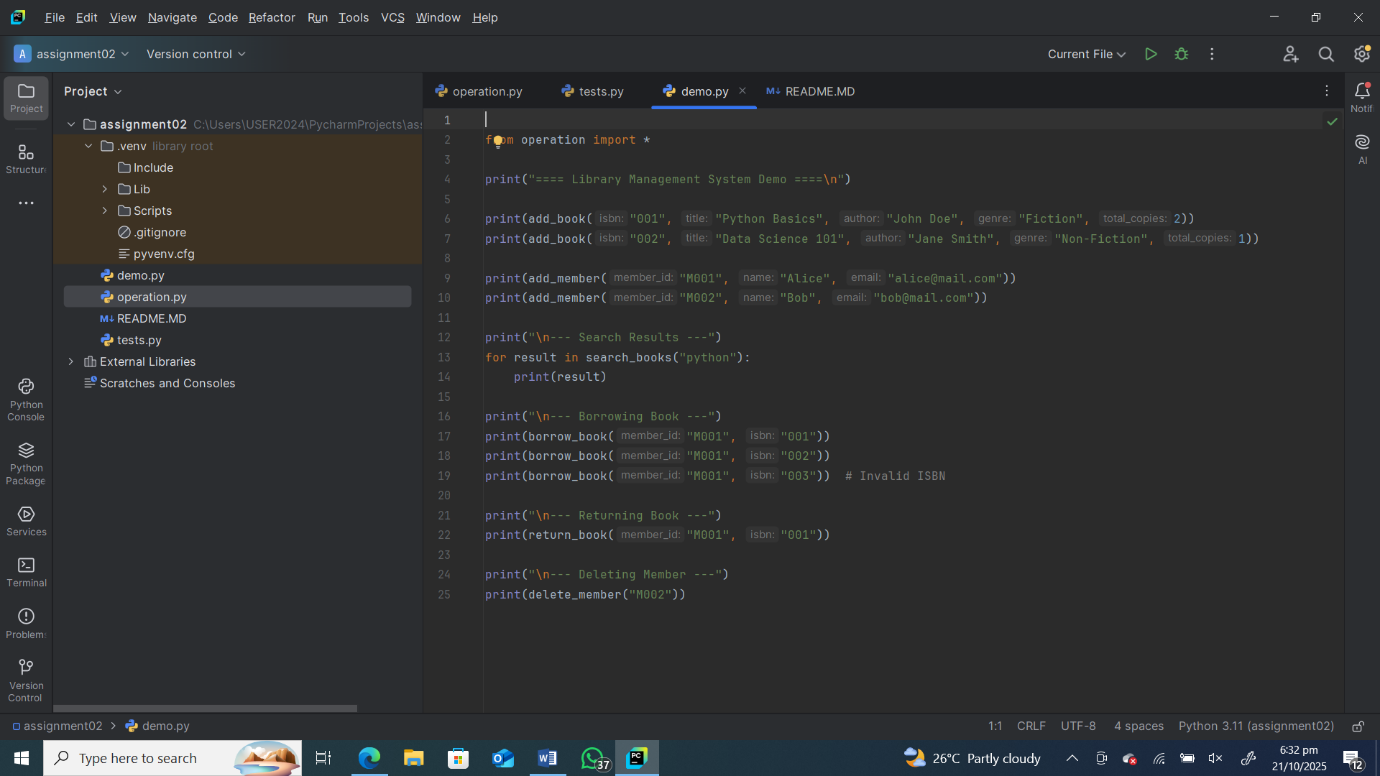
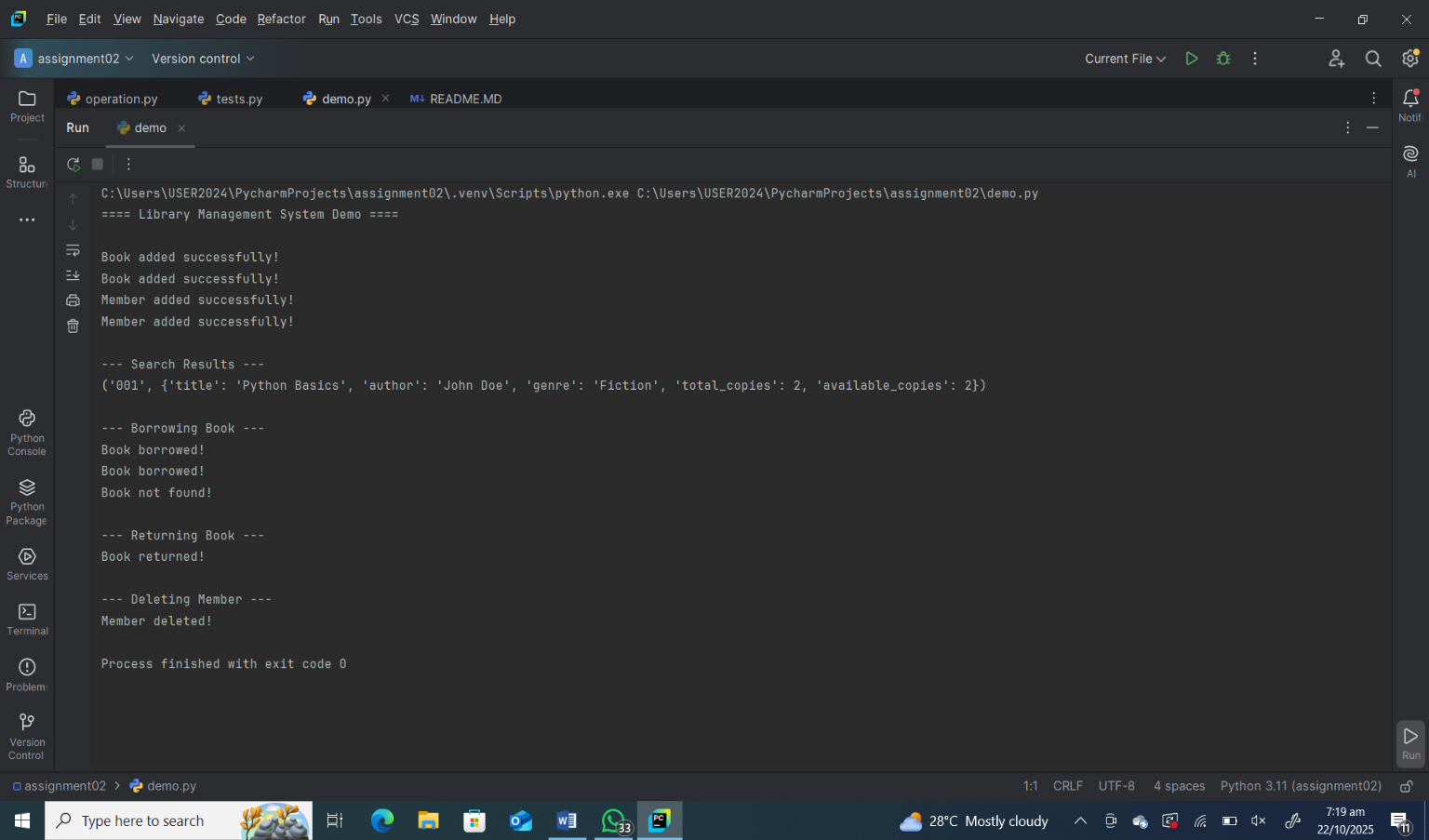
1. **Future Extensions**

* This design allows easy future improvements, such as:
* Adding a database backend for persistence.
* Implementing more complex search filters.
* Adding user authentication or borrowing history.









Conclusion

The Library Management System was successfully developed using core Python data structures such as dictionaries, lists, and tuples, combined with well-structured functions to manage books and members. The system demonstrates essential programming principles like modularity, reusability, and simplicity.

Through this project, I learned how to design and organize data effectively, perform CRUD operations, and apply logical conditions to handle real-world tasks such as borrowing and returning books. Using GitHub for version control and collaboration also provided valuable experience in managing code repositories professionally.

Overall, this project highlights how even simple Python tools can be combined to create a practical, functional system. It lays a strong foundation for building more advanced systems in the future, such as integrating databases, user interfaces, or online access for a complete digital library experience.

**References**

1. GeeksforGeeks. (2024). Basic Components of a Computer System. Retrieved from <https://www.geeksforgeeks.org>
2. Intel Corporation. (2024). Understanding Computer Hardware Components. Retrieved from <https://www.intel.com>
3. PCMag. (2024). How to Build a Gaming PC: Step-by-Step Guide. Retrieved from <https://www.pcmag.com>
4. TechRadar. (2024). Best Gaming PC Components and How They Work Together. Retrieved from <https://www.techradar.com>
5. Wikipedia. (2024). Computer Hardware. Retrieved from <https://en.wikipedia.org/wiki/Computer_hardware>