**Mini Library Management System**

1. **Introduction**

**Purpose of the project**

The purpose of this project is to design a simple Mini Library Management system using python. It aims to show how basic programming concepts like functions, loops and data structures can be used to build system that manages library books and members. This project helps in understanding how to apply theoretical knowledge from class to Real World examples.

1. **Data Structure**

**Dictionary for books**

Each book is uniquely identified by its ISBN, which serves as a key and allows fast lookup, insertion and deletion in **O(1)** for book details.

**List for members**

We created a list for members because it holds records which are stored as dictionaries within the list. Searching or updating member information is simple and readable.

**Tuple for Genres**

Tuples are used to define pre-set genres because they are immutable, ensuring genre categories remain fixed and cannot be altered during runtime**.**

1. **Real-World Library Operations Modeled**

- add \_ book (): Simulates registering a new book into the system.

- add \_ member (): Models signing up a new member.

- borrow \_ book (): Mimics checking out a book. It checks if the book is available and if the member hasn't exceeded the borrowing limit.

- return \_ book (): Reflects returning a book and updating its availability.

- search \_ books (): Allows users to find books by title, author, or genre, simulating a library search feature.

- delete \_ book (): Removes catalog entry and deletes a book if it’s not currently borrowed.

- delete \_ member (): Deregistering a patron and deleting a member if they have no borrowed books.

1. **Ensuring Data Integrity**

Unique ISBNs:

Before adding a book, the system checks if the ISBN already exists in the dictionary, preventing duplicates.

Loan Limits:

Members are restricted (e.g., 3 books max) to avoid over-borrowing. This simulates the typical library policy.

Validation Checks:

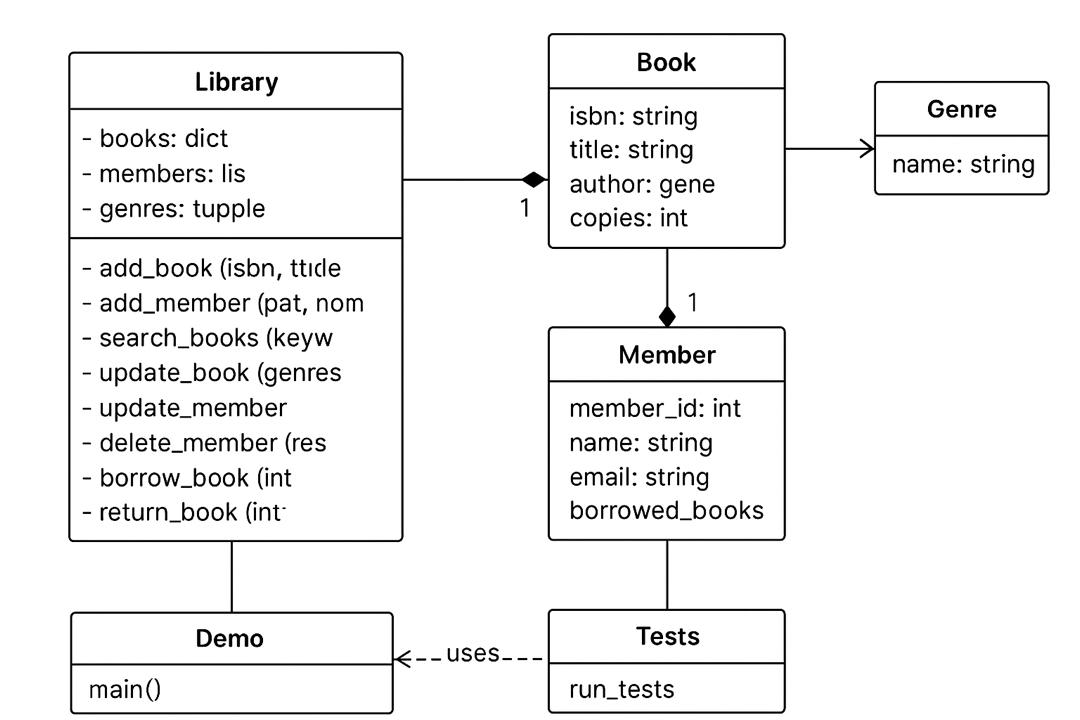
The system validates all inputs: empty fields, incorrect data types, invalid genres, or nonexistent records before processing requests.

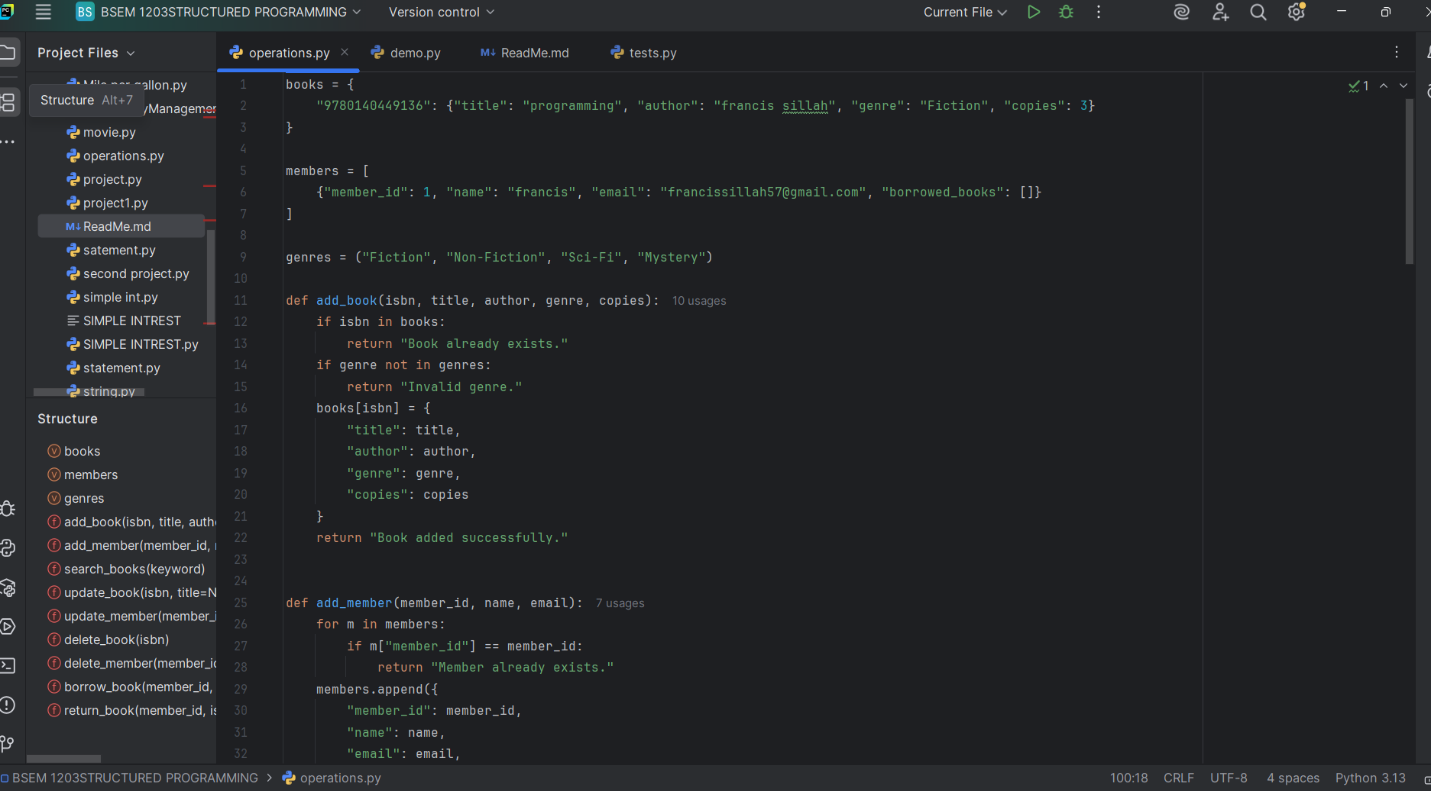
**System Overview**

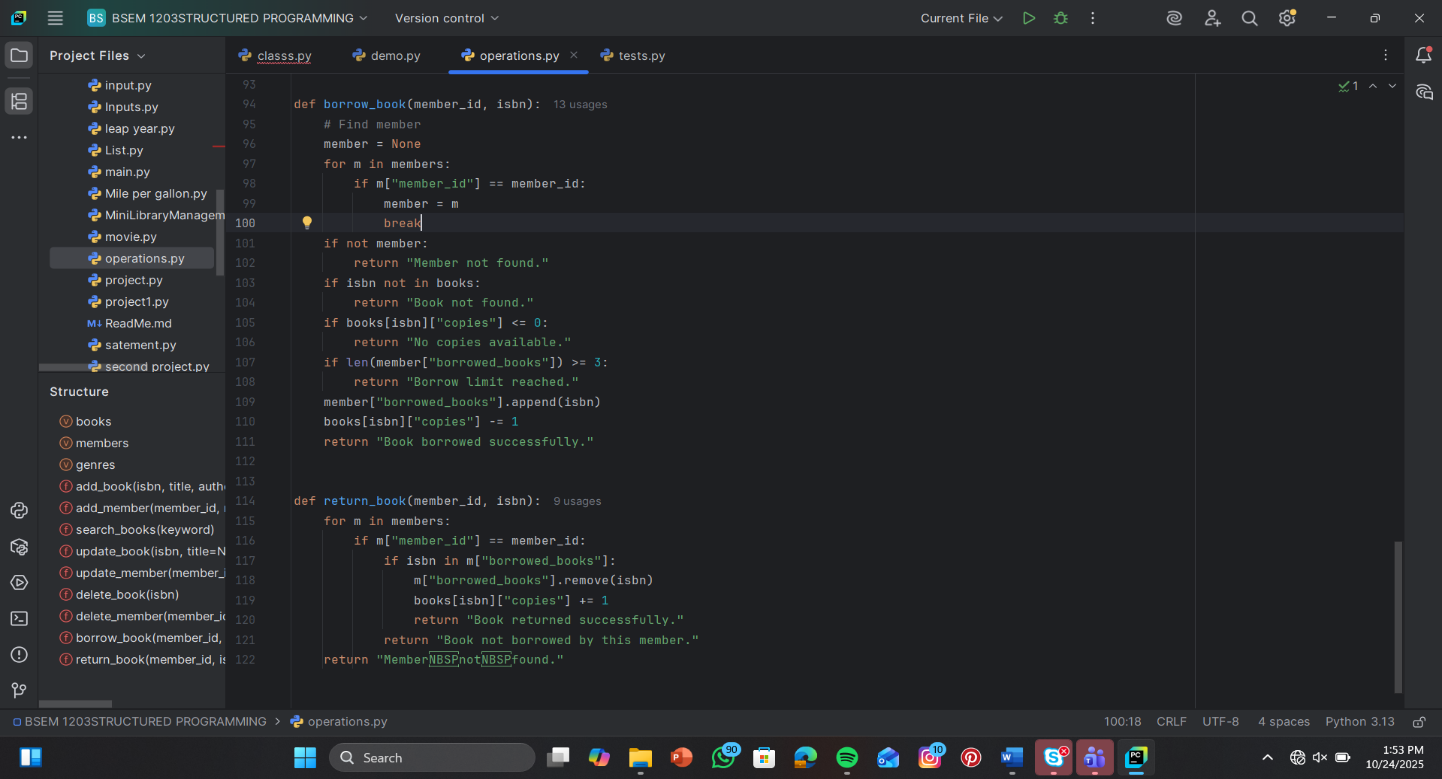
**How the system works**

The Library Management System runs through a simple text menu that guides users to perform different actions. Whereas users can add books, remove members, borrow or return books and can view all the stored information made available from the menu which triggers a function that performs a specific operation. The system stores book details in a dictionary, members and details in a list, and uses a tuple to hold fixed genres.

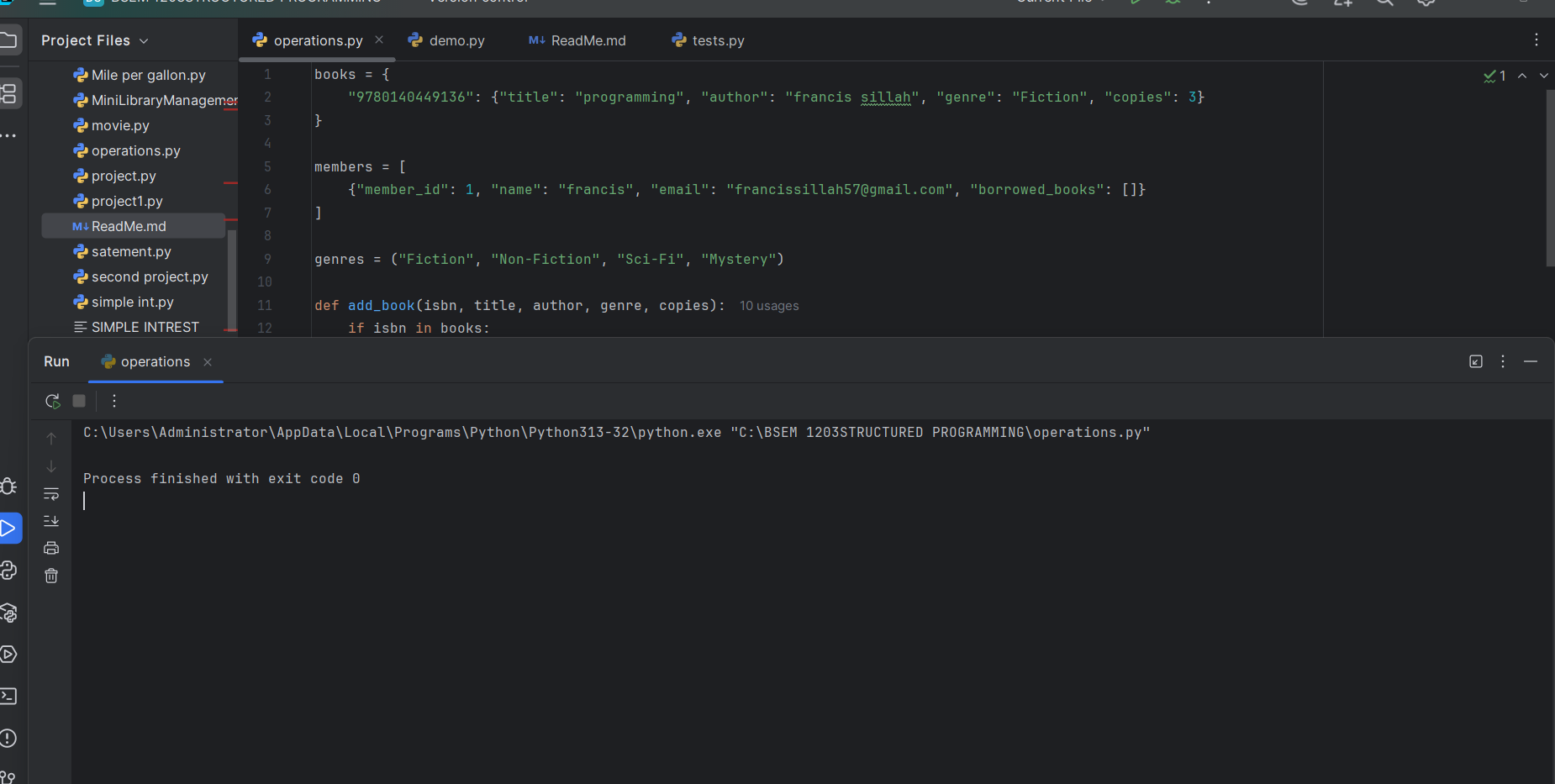
**UML diagram**

****

**Operations.py**

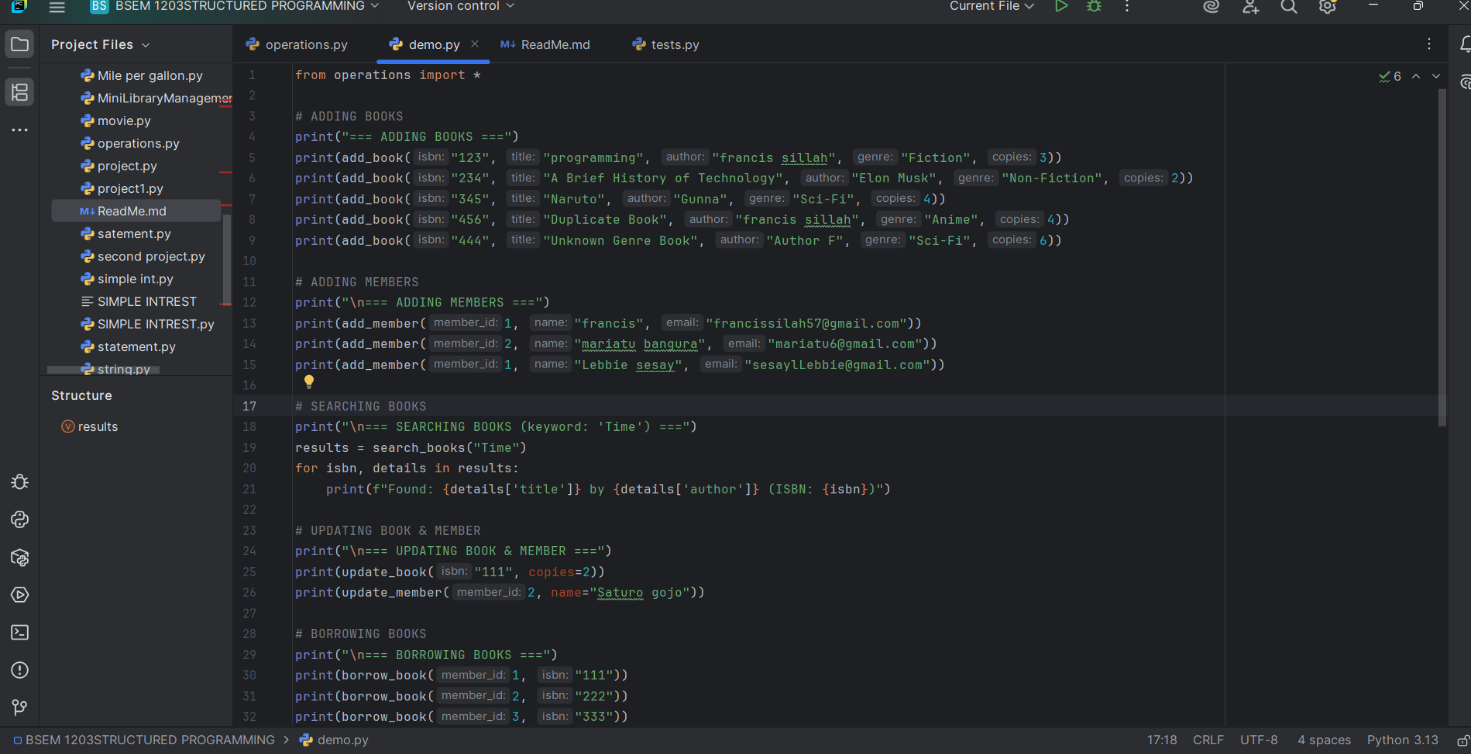
****

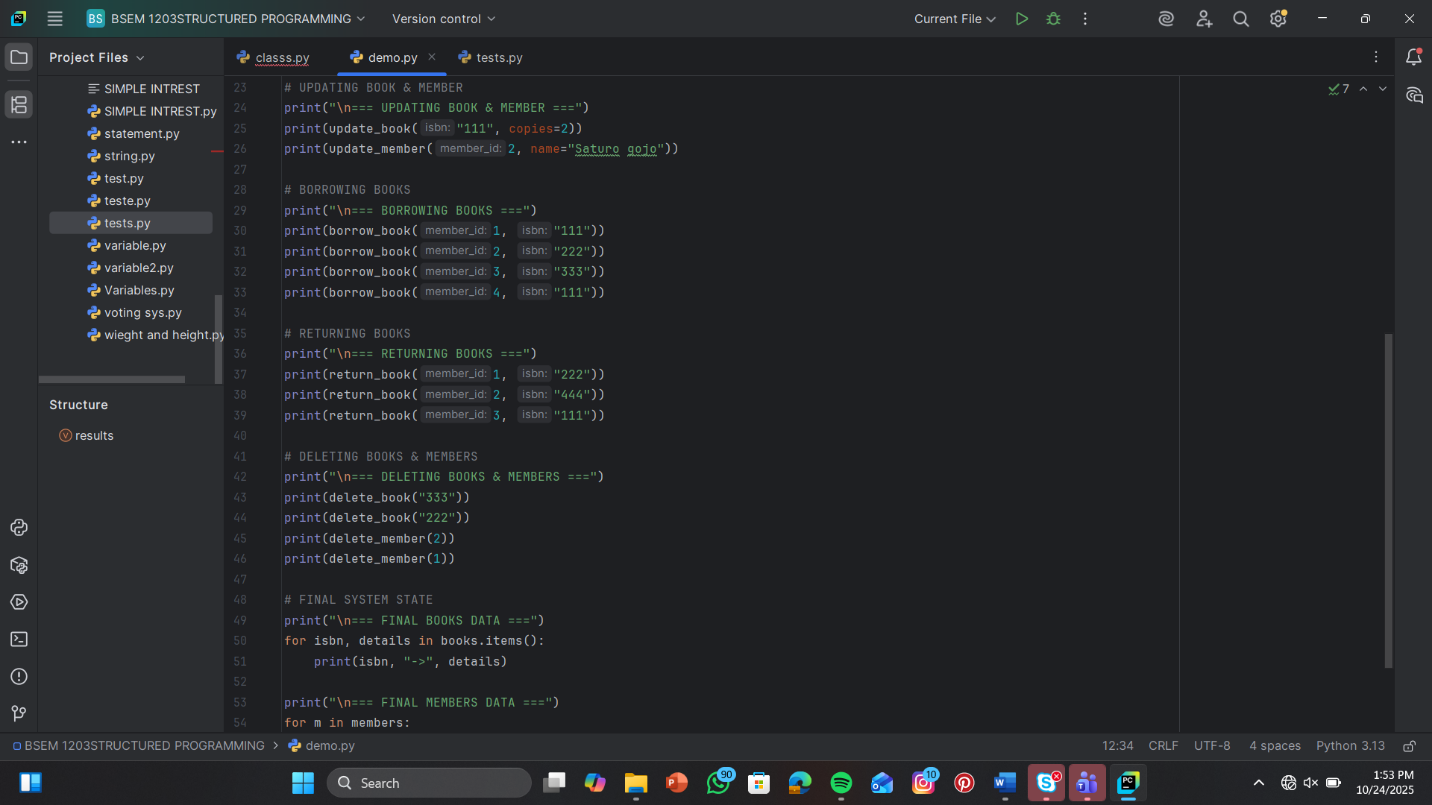
**Output**

****

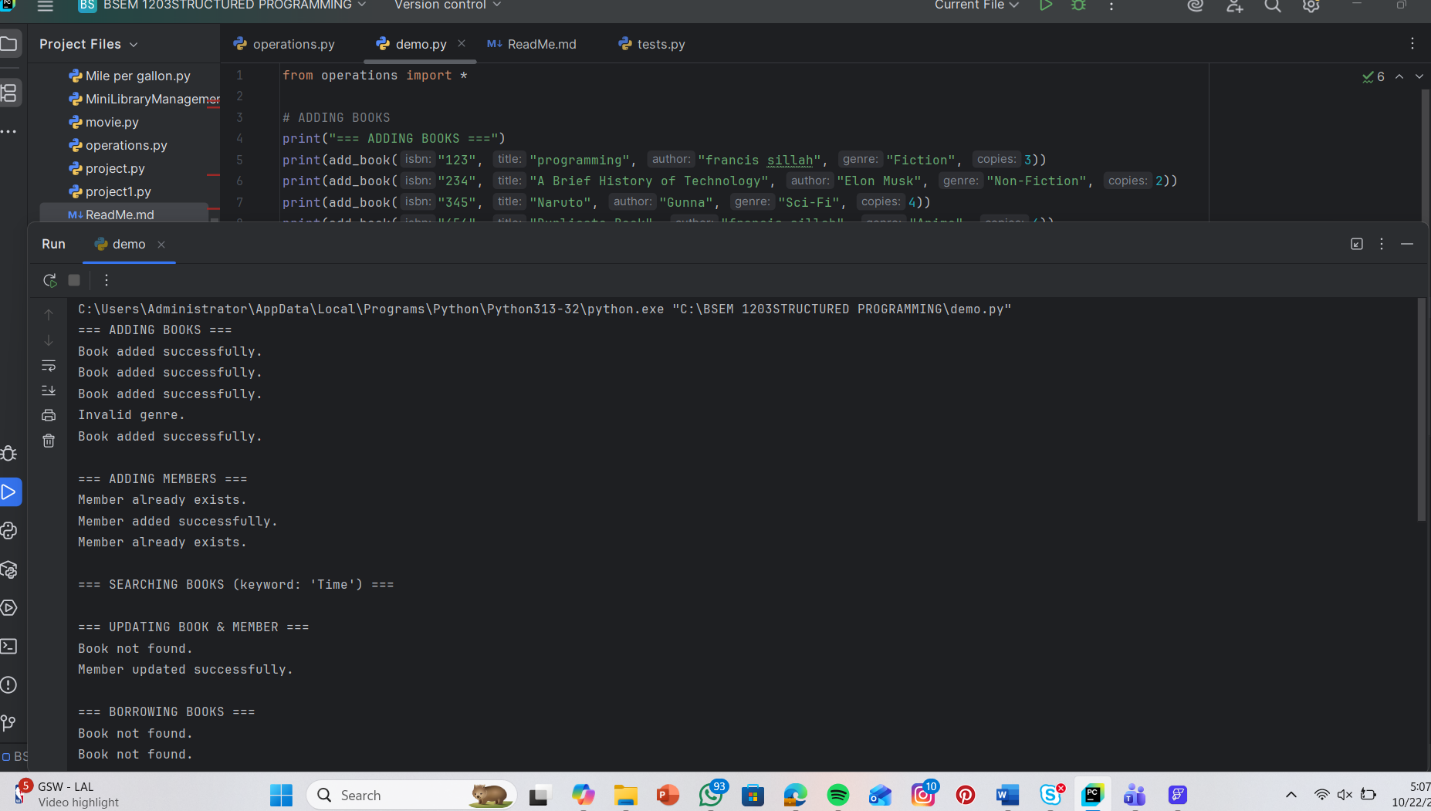
Purpose: This code contains the main functions that power your system — like adding books, borrowing, updating members, etc.

* Why it matters: It separates business logic from user interaction, making your code modular and easier to test or expand.
* Typical contents: All the functions you shared earlier (e.g., add book, borrow book, delete member).

**Demo.py**

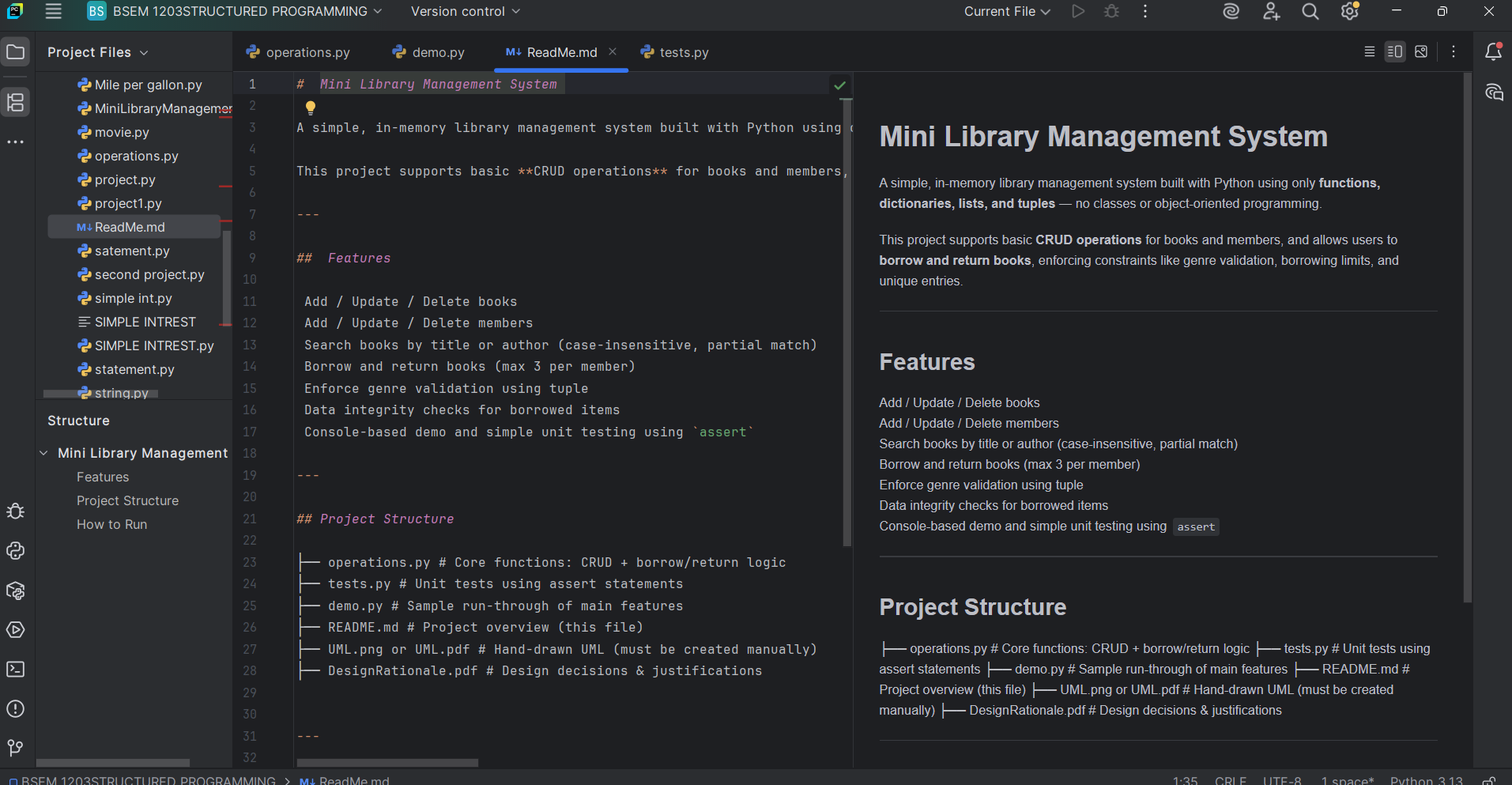
****

**Output**

****

**Purpose**: Acts as a sample run or interface to demonstrate how the system works.

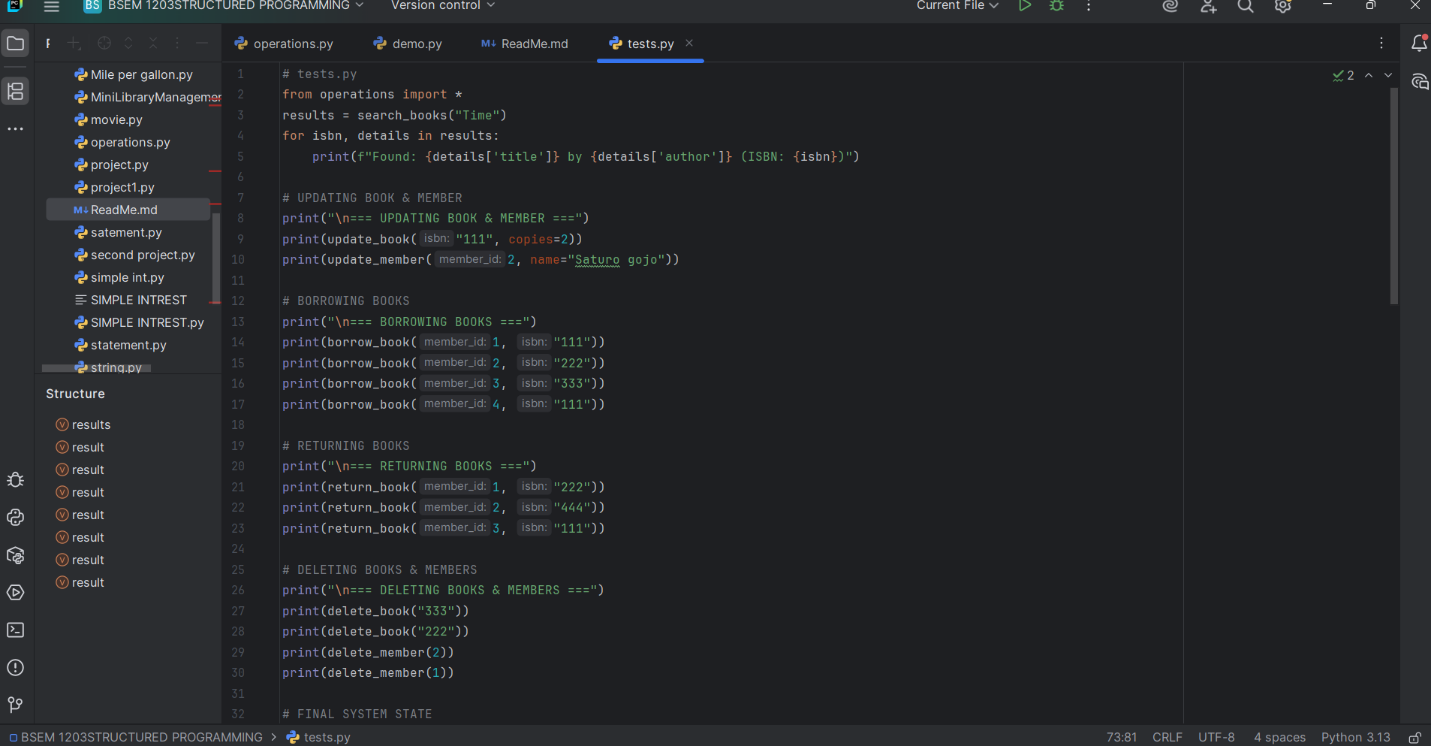
* Why it matters: Useful for testing manually or showing others how to use the system without diving into the code.
* Typical contents: Calls to functions from operations.py, maybe with input () prompts or hardcoded examples.

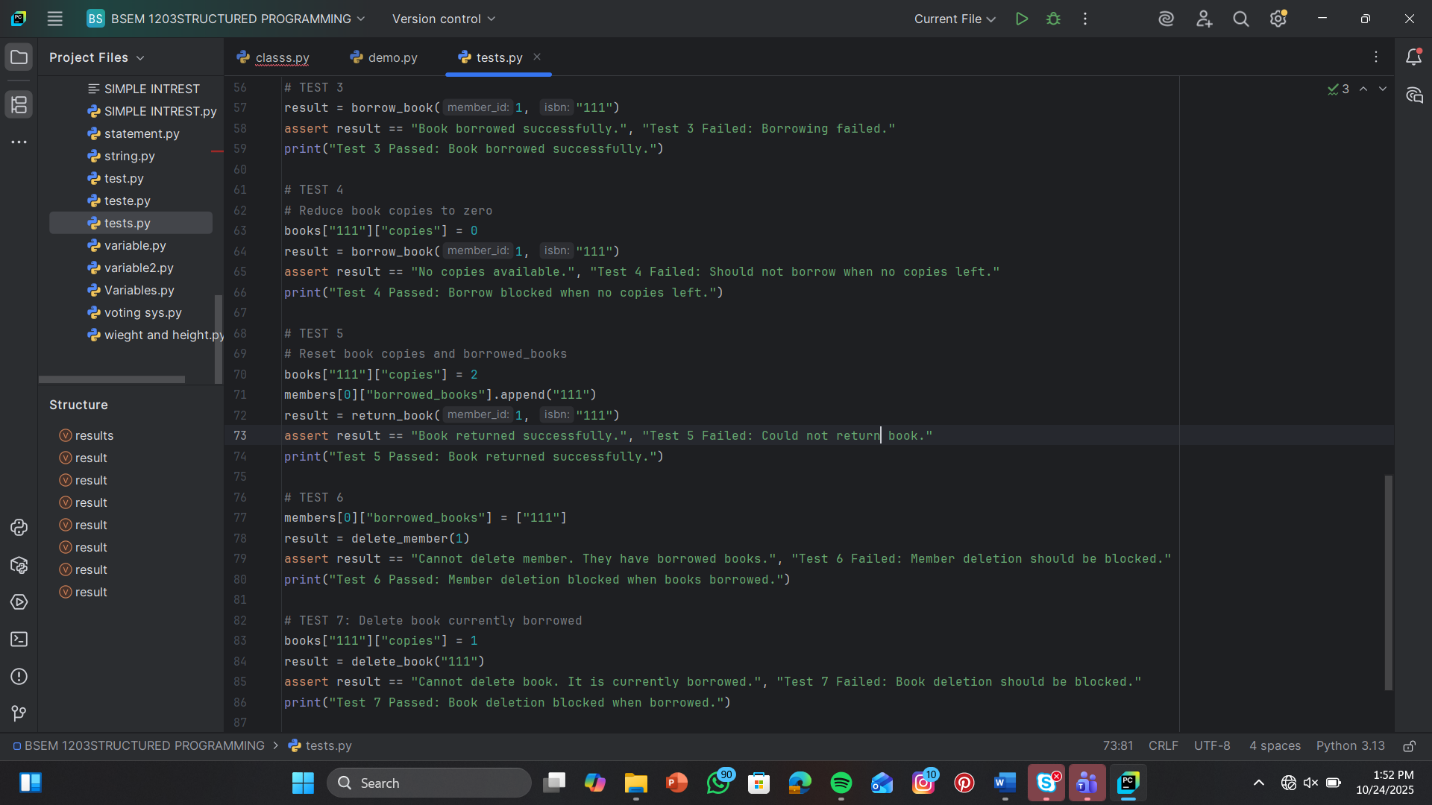
**README.dm**

Purpose: Explains what the project is, how to use it, and any setup instructions.

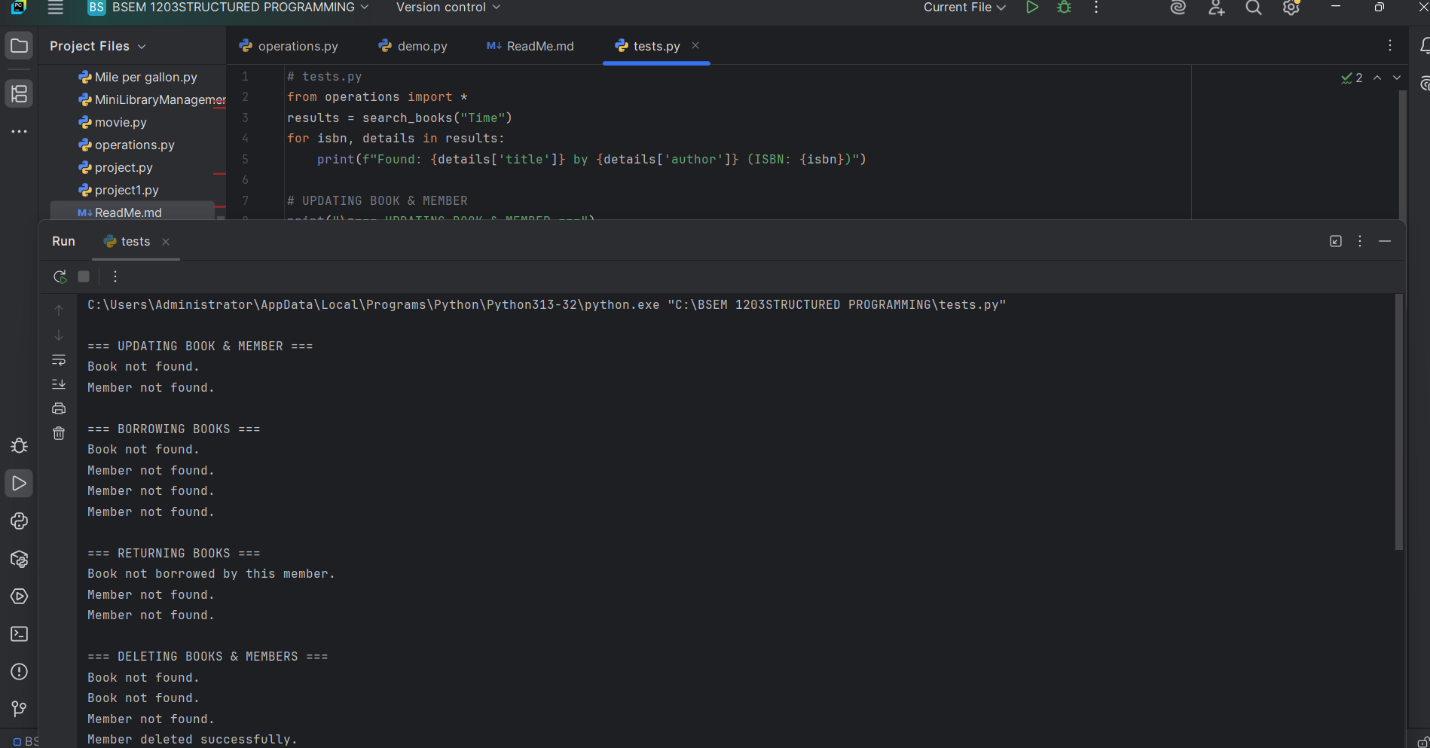
* Why it matters: Crucial for clarity, especially if others (or future you) need to understand or contribute to the project.
* Typical contents:
* Project overview
* How to run the demo
* Function descriptions
* Author info and credits

**Tests.py**

****

****

**Output**

****

Purpose: Validates that your functions behave correctly under different scenarios.

* Why it matters: Helps catch bugs early and ensures your system remains reliable as it grows.
* Typical contents: Unit tests using assert statements or a framework like unit test or py test.

**Conclusion**

The code provides a simple and effective library management system using Python. It uses dictionaries for fast book lookup, lists for storing member details, and tuples for fixed genre categories. The functions simulate real library actions like adding books, borrowing, and returning, while ensuring data is handled properly. It's a good example of apply basic Python structures to solve real-world problems.