**Introduction**

This report presents the design and implementation of a Mini Library Management System using Python. The system models basic library operations such as adding, searching, updating, deleting, borrowing, and returning books. It uses core Python data structures and procedural programming techniques to simulate a functional library for Limkokwing University.

**System Architecture**

The system is composed of three main entities:

* **Book**: Identified by ISBN includes title, author, genre, total and available copies.
* **Member**: Identified by member ID, including name, email, and borrowed books.
* **Library**: Aggregates books and members and manages operations.

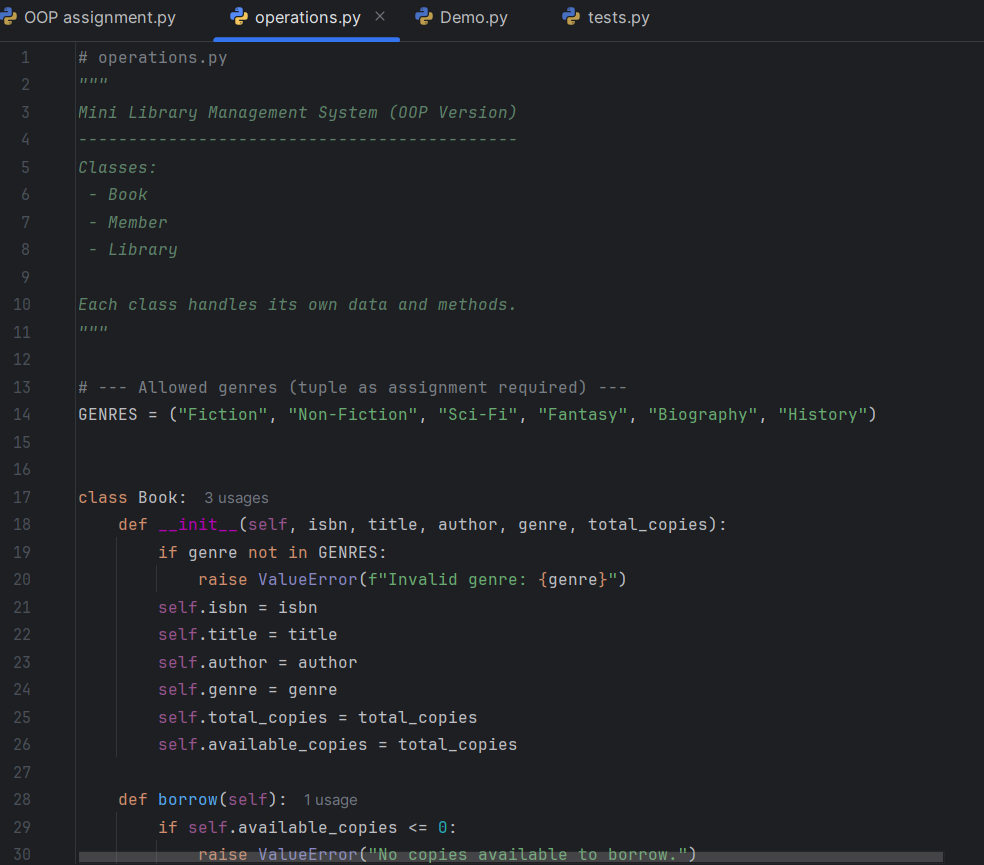
All operations are implemented as standalone functions in a single Python file.

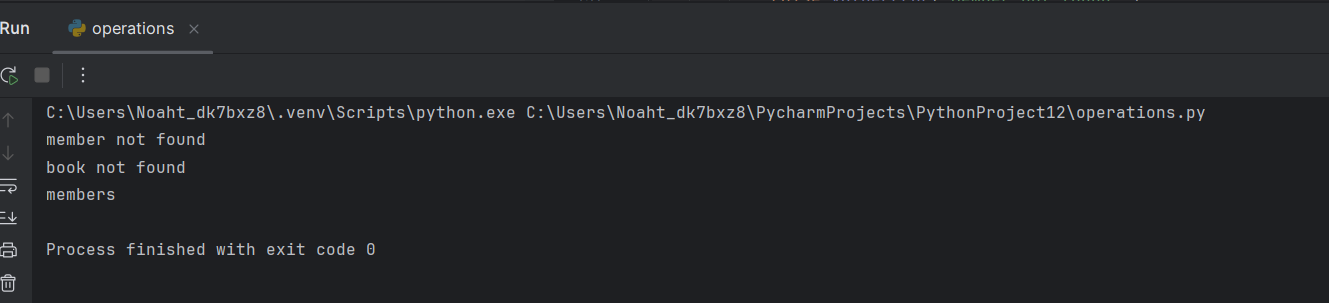
**UML Diagram**

Below is the UML diagram representing system architecture. It shows the relationships between the Library, Book, and Member entities, with aggregation arrows from Library to Book and Member.

**Python Code Implementation**

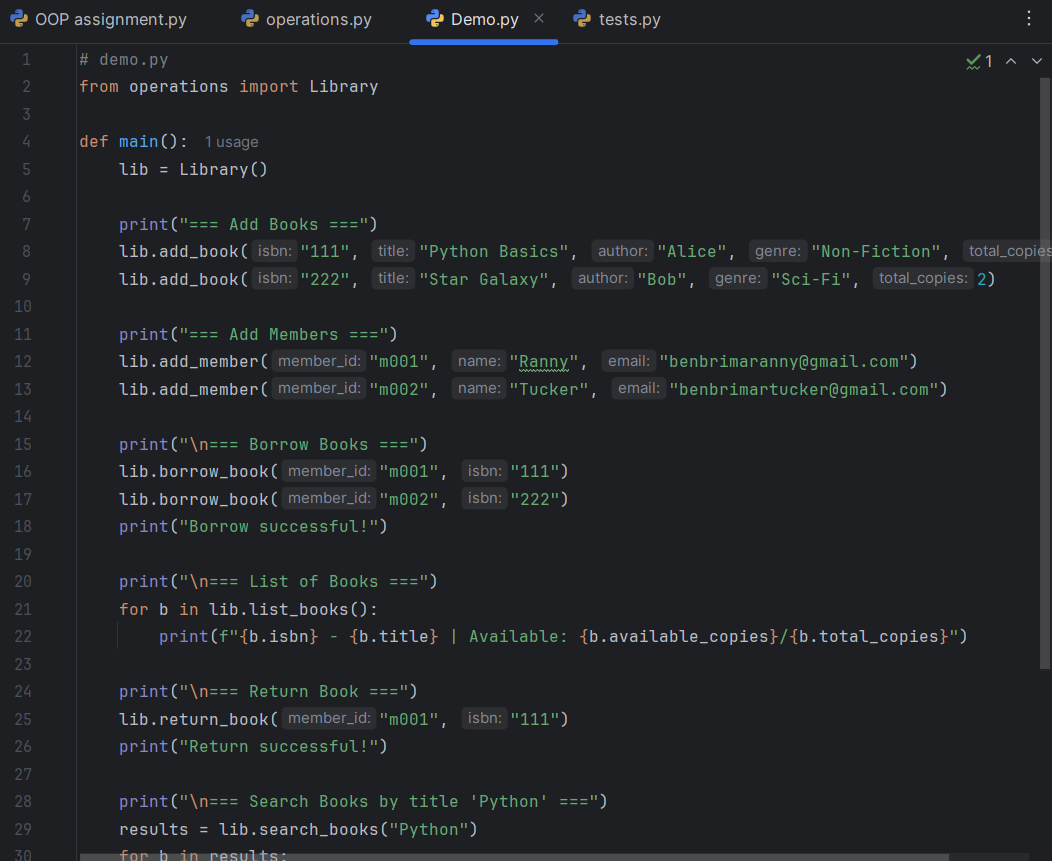
The following code implements the full functionality of the Mini Library Management System:

**operations.py**



**Demo**

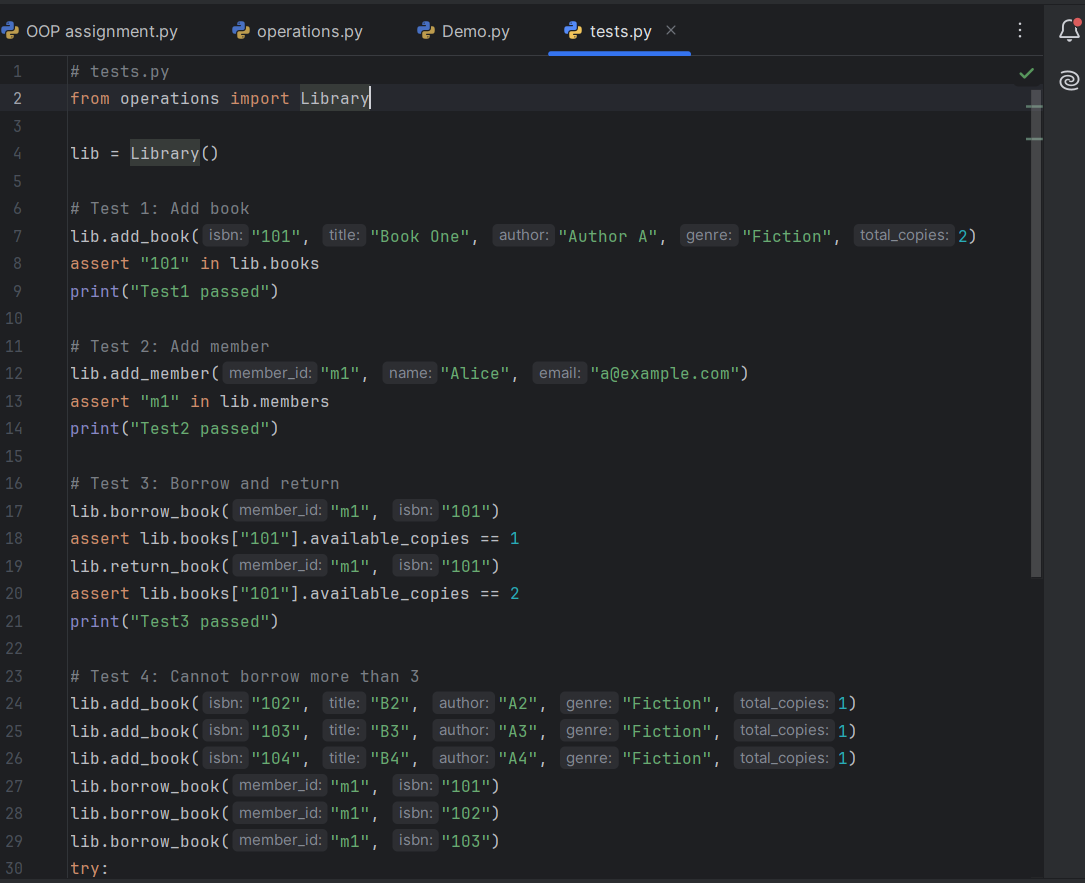
Unit tests were written using assert statements to validate core functionality. A demo script (demo.py) showcases the system in action with sample data.

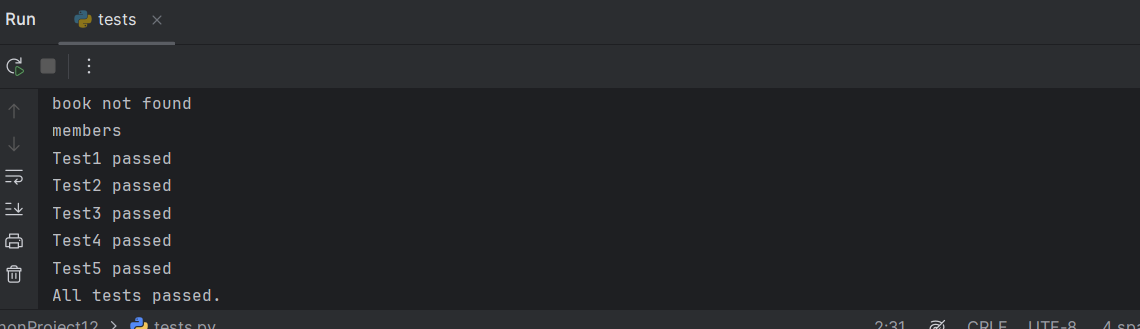
****

**A screenshot of a computer

AI-generated content may be incorrect.**

**tests.py**

****

****

**Design Rationale for Mini Library Management System**

This rationale outlines the reasoning behind the selection of data structures and programming techniques used in the development of the Mini Library Management System. The system was built using Python and adheres to the assignment requirements provided by Limkokwing University.

**1. Dictionary for Book Records**

Books are stored in a dictionary where the key is the ISBN and the value is another dictionary containing the book’s details (title, author, genre, total copies, available copies).

**Justification:**

* **Fast lookup**: ISBNs serve as unique identifiers, allowing constant-time access to book records.
* **Structured storage**: Each book’s attributes are grouped logically, making updates and retrievals efficient.
* **Scalability**: New fields (e.g., publisher, year) can be added easily without changing the structure.

This choice ensures that book data is organized, accessible, and easy to manipulate.

**2. List of Dictionaries for Member Records**

Members are stored as a list of dictionaries. Each dictionary contains the member’s ID, name, email, and a list of borrowed books.

**Justification:**

* **Flexible structure**: Lists allow dynamic addition and removal of members.
* **Ease of iteration**: Searching for a member by ID or name is straightforward using loops or list comprehensions.
* **Expandable**: Additional fields (e.g., phone number, address) can be added without restructuring.

This approach balances simplicity with flexibility, making it suitable for small-scale systems.

**3. Tuple for Genre Validation**

Genres are stored in a tuple containing predefined categories such as Fiction, Non-Fiction, Sci-Fi, etc.

**Justification:**

* **Immutability**: Tuples prevent accidental modification of genre definitions.
* **Memory efficiency**: Tuples consume less memory than lists and are faster for read-only operations.
* **Validation**: Ensures that only approved genres are used when adding books.

Using a tuple reinforces data integrity and supports consistent validation.

**4. Function-Based Design**

All operations—adding, updating, deleting, borrowing, and returning—are implemented as standalone functions.

**Justification:**

* **Modularity**: Each function performs a single task, improving readability and maintainability.
* **Reusability**: Functions can be reused across scripts, tests, and future enhancements.
* **Testability**: Functions are easy to test using assert statements, supporting robust development.

**Conclusion**

The chosen design reflects a balance between simplicity, performance, and academic rigor. By leveraging Python’s built-in data structures and modular functions, the system meets all assignment requirements while remaining extensible for future development.