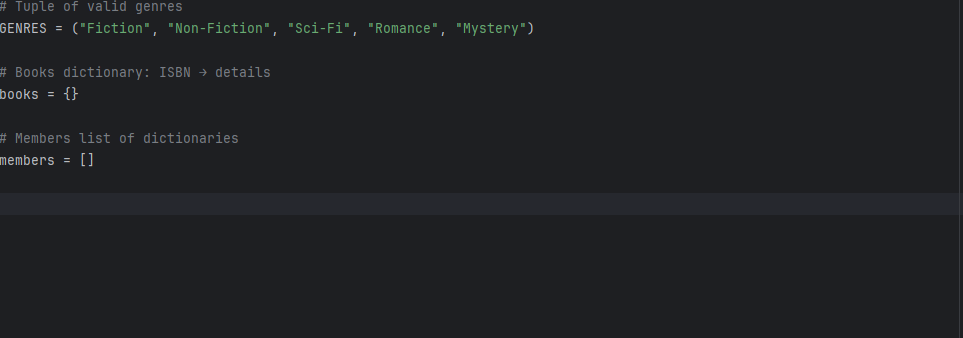
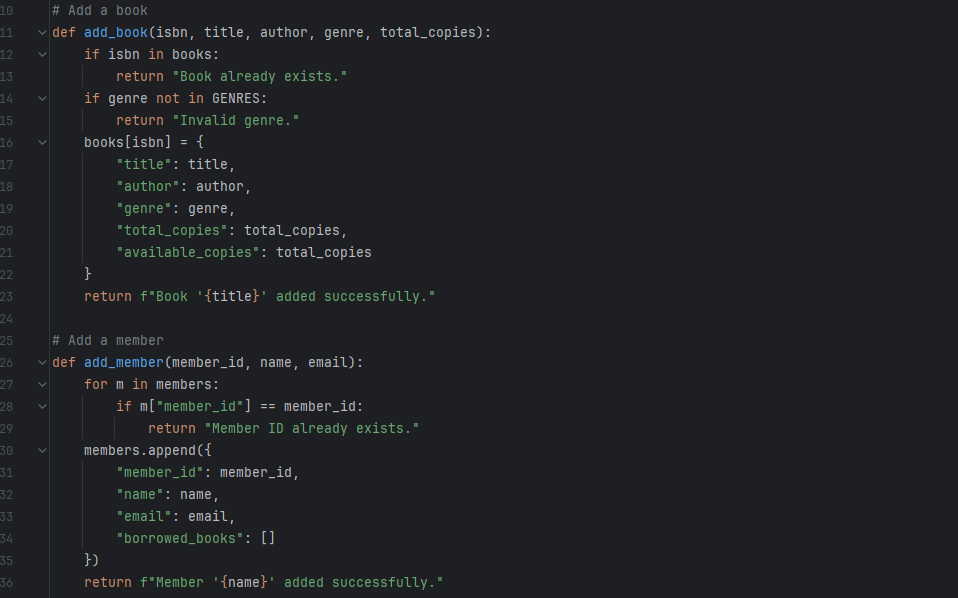
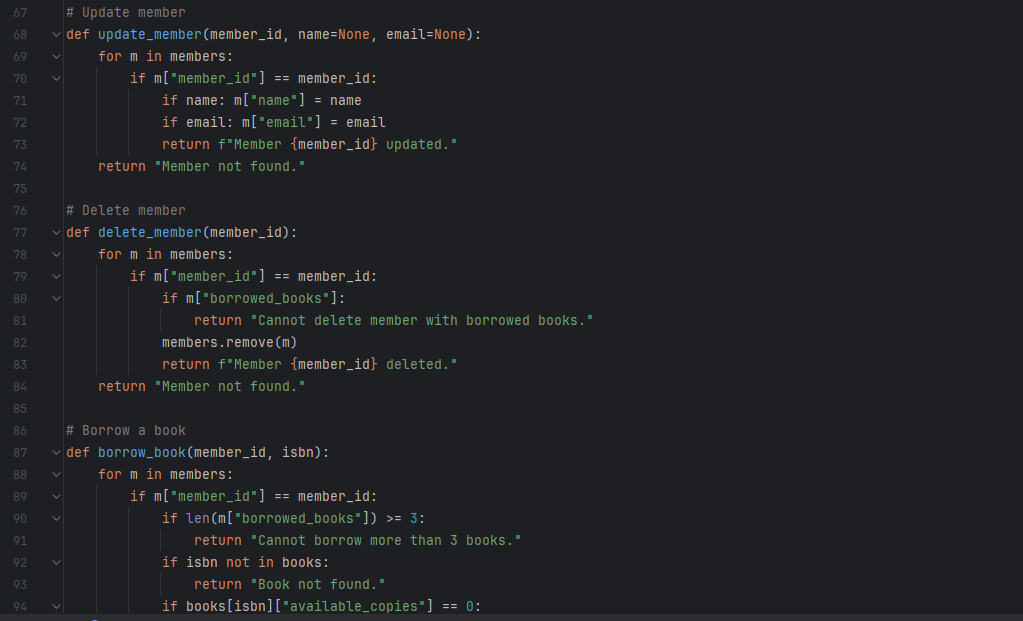
1. Data Structure

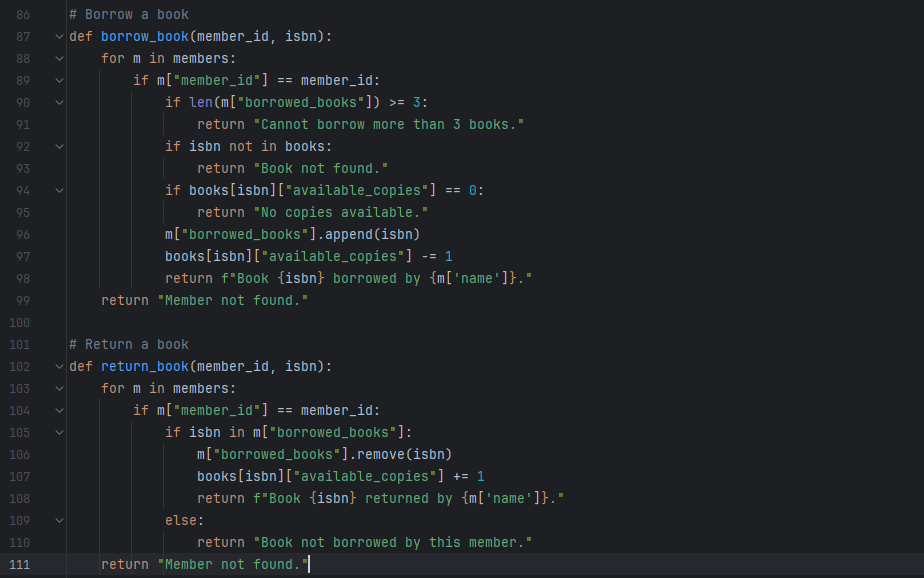


1. Core Functions (CRUD + Borrow/Return)

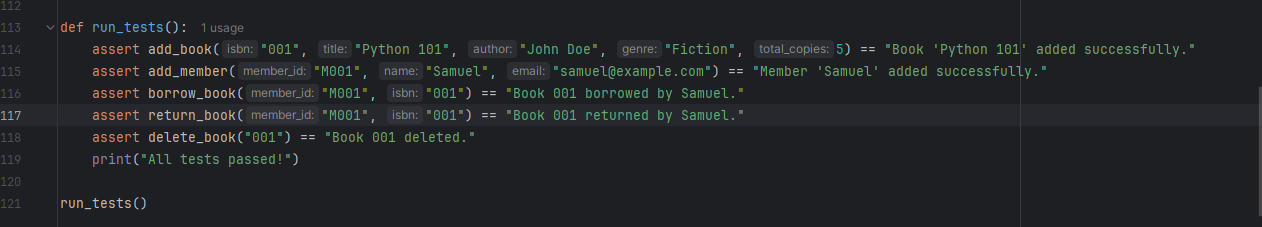








1. Testing With Assert



1. Demo Script



1. UML Diagram

Rationale for the Mini Library Management System

In designing the Mini Library Management System, careful consideration was given to how data would be stored, accessed, and managed efficiently. Since the system is built around the principles of object-oriented and structured programming, selecting the right data structures was essential to achieve efficiency, readability, and scalability. The three main Python data structures used are dictionaries, lists, and tuples. Each was chosen based on its unique characteristics and how well it fits the needs of the system’s operations and CRUD (Create, Read, Update, Delete) functionality.

1. Dictionary – For Fast Lookup of Books by ISBN

The dictionary data structure was used to store and manage information about books in the library. Each book entry is stored using its ISBN (International Standard Book Number) as the key, with its associated details (such as title, author, genre, and total copies) stored as the value.

This approach offers several advantages:

Fast data access: Dictionaries in Python use hash mapping, which allows for constant-time complexity (O(1)) when searching, updating, or deleting entries. This makes it ideal for quickly locating a book by its ISBN number.

Unique identification: Every book has a unique ISBN, which perfectly aligns with the dictionary’s key-value system. This ensures there are no duplicate entries and makes it easy to validate book uniqueness during the “add\_book” operation.

Structured storage: The value portion of each dictionary key can itself be a nested dictionary, allowing for clear and organized storage of related book details in one place.

By using dictionaries, the system can efficiently handle CRUD operations such as adding, searching, updating, and deleting books with minimal computational overhead.

2. List – For Storing Multiple Members Dynamically

The list structure was chosen to manage the library’s members. Each member is represented as a dictionary (containing member ID, name, email, and borrowed books), and all member records are stored collectively inside a single list.

The main reasons for this choice are:

Dynamic resizing: Lists can grow or shrink dynamically at runtime, allowing new members to be added or removed easily without the need to predefine a fixed size. This flexibility is crucial in real-world library systems, where the number of registered members can change frequently.

Ease of iteration: The list makes it simple to loop through members for searching or validating information (for example, checking if a member ID already exists before adding a new one).

Support for CRUD operations: Lists naturally support appending, removing, and updating data, which aligns perfectly with the system’s required core functionalities such as “add\_member,” “update\_member,” and “delete\_member.”

This combination of flexibility and simplicity makes the list structure ideal for handling the member-related operations within the system.

3. Tuple – For Fixed Genres That Don’t Change

The tuple structure is used to store the valid list of book genres (for example, Fiction, Non-Fiction, Sci-Fi, Mystery, etc.). Unlike lists and dictionaries, tuples are immutable, meaning their contents cannot be changed once created.

The reasons for using a tuple include:

Data consistency: Since the list of genres is predefined and should not change during program execution, a tuple ensures these values remain constant and protected from accidental modification.

Lightweight and memory-efficient: Tuples consume less memory than lists, which is beneficial for small sets of static data such as categories or types.

Improved readability and safety: Using a tuple communicates to other programmers that this data is meant to be constant, enhancing the clarity and maintainability of the code.

This design choice contributes to the overall stability of the system by preventing unintended alterations to core reference data.

Conclusion

In summary, the choice of dictionary, list, and tuple in the Mini Library Management System was deliberate and rooted in both efficiency and clarity. The dictionary enables quick and unique identification of books, the list provides flexible and dynamic handling of members, and the tuple ensures that fixed categories like genres remain consistent throughout the system. Together, these structures support the CRUD operations seamlessly and make the system both efficient and easy to maintain.

By aligning data structure selection with functional requirements, the design achieves an effective balance between performance, scalability, and simplicity — which are key principles in software engineering and object-oriented programming.