1. Introduction

The Mini Library Management System was developed using only functions, lists, dictionaries, and tuples as required by the assignment. The system performs all basic library operations, adding, updating, deleting, searching, borrowing, and returning books, without using object-oriented programming.

The goal was to design a simple and efficient data structure that reflects real-world library activities in a functional programming style.

2. Choice of Data Structures

a) Dictionaries for Books

Books are stored in a dictionary where the ISBN acts as the unique key and the value is another dictionary containing attributes like title, author, genre, and total copies.

This structure was chosen because:

•It allows fast lookup of books using ISBN as a key.

•It is easy to update or delete specific book details.

•The dictionary format makes the data readable and structured, similar to a real database record.

Example

books = {"123456": {"title": "Python Basics", "author": "'Tanu Jalloh", "genre": "Non-Fiction", "total copies": 5}

}

b) Lists for Members

Members are stored in a list of dictionaries, where each dictionary contains the member's ID, name, email, and a list of borrowed books.

The list was chosen because:

•It allows simple iteration to find or update a member.

•It is suitable for small datasets, such as a classroom or college library.

•Lists are flexible and can grow dynamically as new members are added.

Example

members = [{"member id": "M001", "name": "Ayisha", "email": "alice@example.com", "borrowed books": ["123456"]}

]

c) Tuples for Genres

Genres are stored as a tuple because:

•Tuples are immutable, ensuring the list of valid genres cannot be accidentally changed during program execution.

•They serve as a fixed validation list when adding or updating books.

Example

GENRES = ("Fiction", "Non-Fiction", "Sci-Fi", "Mystery", "Biography")

3. Functional Design

The system uses modular functions to perform all operations. Each function handles one responsibility:

•add book() and add member() →Create new records.

•search books() → Retrieve information based on title or author.

•update book() and update member() → Modify details.

•delete book() and delete member() → Safely remove records.

•borrow book() and return book() → Handle book lending and returning.

Functions interact directly with the global data structures (books, members, and GENRES), making it easy to trace changes and test each feature independently.

4. Data Integrity & Validation

To ensure data accuracy, several validation checks were added:

•Unique ISBNs and member IDs: Prevent duplicate entries.

•Genre validation: A book can only belong to an existing genre in GENRES.

•Borrow limits: A member cannot borrow more than three books at once.

•Deletion rules: Books cannot be deleted if borrowed; members cannot be deleted if they have borrowed books.

These checks mimic real-world library policies and maintain consistent data integrity.

5. Real-World Modeling

The system's design mirrors how a real library operates:

•Adding a book: like registering a new title in a catalog.

•Borrowing a book: decreases available copies and adds to a member’s record.

•Returning: increases the count and clears the member’s borrowed list.

•Deleting members or books: only allowed under valid conditions.

6. Conclusion

This design achieves the objectives of the assignment by using appropriate Python data structures and functions to simulate a real-world library.

It is simple, efficient, and easy to test, maintaining a balance between readability and logical structure. The chosen approach ensures reliable data handling, modularity, and clarity, which are key qualities of good software design.