# ****Design Rationale Mini Library Management System (Python)****

## ****1. Introduction****

The Mini Library Management System was developed as part of the Object-Oriented Programming 1 course to demonstrate understanding of Python data structures, functions, and basic CRUD (Create, Read, Update, Delete) operations. The goal of the system is to simulate the management of books and members in a small library, allowing the user to add, search, update, delete, borrow, and return books.

This system was implemented using **lists**, **dictionaries**, **tuples**, and **functions**. Each of these structures was chosen carefully based on their efficiency, readability, and suitability for representing real-world data such as books, library members, and genres.

## ****2. Choice of Data Structures****

### ****a. Dictionary****

Dictionaries were used to store books and members because they allow for fast lookups and easy mapping between keys and values.

**Books** were stored as a dictionary where the key is the book’s ISBN (unique identifier) and the value is another dictionary containing the title, author, genre, and number of copies.  
Example:

books = {

"001": {"title": "Python Basics", "author": "John Doe", "genre": "Fiction", "total\_copies": 5}

}

This structure allows instant access to a book’s details using its ISBN without looping through a list.

**Members** were also stored as a dictionary where the key is the member’s ID and the value holds name, email, and borrowed books.  
Example:

members = {

"M001": {"name": "Alice", "email": "alice@email.com", "borrowed\_books": []}

}

Using dictionaries here makes it easy to find and update a member’s information quickly.

### ****b. Tuple****

Tuples were used to store the valid genres.  
Example:

genres = ("Fiction", "Non-Fiction", "Sci-Fi")

A tuple is an **immutable** data structure, meaning it cannot be changed during program execution. This ensures that the list of genres remains fixed, providing consistency and preventing accidental modification.

### ****c. List****

Lists were used for two purposes:

1 .To store multiple borrowed book ISBNs for each member.

1. To store search results when users search for books by title or author.

Lists are ideal here because they allow ordered storage and easy iteration.

## ****3. System Functions****

The system includes several functions, each performing a specific operation:

* **add\_book() / add\_member()** → Add new books or members while checking for uniqueness.
* **search\_books()** → Allows users to find books by title or author.
* **update\_book() / update\_member()** → Modify existing records.
* **delete\_book() / delete\_member()** → Remove records if no borrowed items exist.
* **borrow\_book() / return\_book()** → Handle borrowing and returning transactions, ensuring rules are respected (e.g., a member can borrow up to three books).

Using functions helps organize the code logically, reduces repetition, and makes it easier to maintain or extend in the future.

## ****4. Testing Approach****

To ensure the system works correctly, simple test cases were created using the assert statement. Each test checks a function’s output against the expected result.  
For example:

assert add\_book("001", "Python Basics", "John Doe", "Fiction", 5) == "Book added successfully."

This form of testing guarantees that key functionalities (adding, borrowing, returning, and deleting) work properly before final submission.

## ****5. Design Decisions****

1. **Simplicity:** The system avoids unnecessary complexity. It focuses only on the required features (CRUD + borrow/return).
2. **Reusability:** Functions were designed to be reusable in future extensions, such as adding a login system or user interface.
3. **Efficiency:** Dictionaries and lists provide fast access and minimal storage overhead for small library systems.
4. **Readability:** Clear naming conventions and modular functions make the code easy to understand and debug.
5. **Scalability:** The structure can easily be expanded to include more features, such as overdue tracking or genre-based filtering.

## ****6. Conclusion****

The Mini Library Management System demonstrates how core Python data structures can be combined to create a functional and organized application. Dictionaries were chosen for their key-value mapping efficiency, tuples for their immutability in storing fixed genres, and lists for managing multiple borrowed items.