# Design Rationale

# 1. Introduction

The Mini Library Management System was developed using the Python programming language to manage books and members of a small library. The system supports adding, updating, deleting, borrowing, and returning of books. The goal of this design is to demonstrate the efficient use of fundamental Python data structures — lists, tuples, and dictionaries — to create a structured and maintainable application without relying on external databases.

## 2. Choice of Data Structures

### 2.1 Dictionary

The dictionary data structure was chosen to represent both books and members because it allows data to be stored in key–value pairs. Each book and member record contains multiple fields such as ISBN, title, author, genre, available, and member ID, name, email respectively. Dictionaries provide fast lookups and flexibility, making it easy to retrieve and update records based on unique identifiers like ISBN or member ID.

Example:  
book = {"isbn": "ISBN001", "title": "The Great Gatsby", "author": "F. Scott Fitzgerald", "genre": "Fiction"}

### 2.2 List

Lists are used to store collections of dictionaries, such as the entire list of books, members, and borrowed records. Lists are suitable because the number of items in the library can grow dynamically, and elements can be easily appended, removed, or iterated over.

Example:  
books = []  
members = []

### 2.3 Tuple

Tuples are used to store immutable and constant values, specifically for the list of genres ("Fiction", "Non-Fiction", "Sci-Fi", "Fantasy", "Biography", "Mystery") and borrow records. Tuples were chosen because they are faster and immutable, ensuring that predefined values like genres cannot be accidentally modified during program execution. This enhances data integrity within the system.

## 3. System Structure

The system is modularized into three main files:  
1. library.py – Core functions and data structures (CRUD operations and borrow/return logic).  
2. main.py – Command-line interface (user interaction).  
3. tests.py – Contains automated test cases to verify functionality.  
This modular design promotes code reusability, readability, and ease of maintenance, following structured programming principles.

## 4. Justification

The combination of lists, tuples, and dictionaries provides an efficient, Pythonic approach to managing structured data without a database. Dictionaries represent structured data entities. Lists manage collections of those entities. Tuples store constants and fixed sets of related values. This choice results in a robust and easy-to-understand design, suitable for both educational and small-scale practical applications.

## 5. Conclusion

The data structures chosen (list, tuple, dictionary) were deliberately selected to reflect real-world use cases in library systems and to fulfill the assignment requirement of demonstrating mastery of Python’s core structures. The design emphasizes simplicity, maintainability, and clarity, ensuring that the system runs smoothly and is easily extensible for future improvements, such as database integration or graphical user interfaces.