**Rationale for Mini Library Management System**

**1. Introduction**

This Mini Library Management System was designed using Python’s fundamental data structures **dictionary**, **list**, and **tuple** to manage books, members, and genres efficiently.  
The system focuses on simplicity, clarity, and compliance with the assignment requirements by avoiding complex data structures or external libraries.

**2. Data Structure Choices**

**a. Dictionary for Books**

The **dictionary** was chosen to store and manage books because each book can be uniquely identified by its **ISBN** (International Standard Book Number), which naturally fits as a dictionary key. This provides **fast lookup**, **easy updating**, and **clear organization**.

**Structure:**

books = {

"111": {"title": "Database System", "author": "Mr Sheikh Umar", "genre": "Fiction", "copies": 3}

}

**Reasons for using a dictionary:**

* ISBN provides a unique key for each book.
* Quick access and update operations using keys (O(1) time complexity).
* Intuitive structure for representing book details.

**b. List for Members**

A **list** was used to store members because membership data grows dynamically and does not have a natural unique key like a dictionary. Each member is represented as a **dictionary** inside the list, holding individual attributes such as ID, name, and borrowed books.

**Structure:**

members = [

{"member\_id": 1, "name": "Shadrach", "email": "shadrach@gmail.com", "borrowed\_books": []}

]

**Reasons for using a list:**

* Supports easy addition and removal of members.
* Simple iteration when searching or updating a member by ID.
* Natural for storing multiple similar objects (members).

**c. Tuple for Genres**

A **tuple** was chosen for the list of valid genres because tuples are **immutable** they cannot be changed after creation. This ensures that the set of valid genres remains constant, preventing accidental modifications.

**Structure:**

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

**Reasons for using a tuple:**

* Immutability guarantees data consistency.
* Tuples are faster and use less memory than lists.
* Ideal for fixed reference data that should not change.

**3. Function Design**

Each operation (add, update, delete, borrow, return) is implemented as a **function**.  
This modular approach improves **readability**, **reusability**, and **testing**.

For example:

* add\_book() → Adds a new book record.
* borrow\_book() → Updates both the book’s available copies and the member’s borrowed list.
* return\_book() → Restores book availability and updates member records.

Each function returns a clear message to indicate success or failure, which is used in both the demo and tests for validation.

**4. Advantages of the Design**

* **Simple and Readable:** Easy to understand for beginner programmers.
* **Efficient:** Uses lightweight built-in data structures with fast operations.
* **Reusable:** Functions are modular and can be tested independently.
* **Scalable:** The design can later be extended to use databases or GUI interfaces.

**5. Conclusion**

The chosen data structures dictionary, list, and tuple provide a perfect balance between simplicity and functionality for this Mini Library Management System. They allow efficient data storage, manipulation, and retrieval while meeting the assignment’s requirements using only Python’s core features.