

**Design Rationale for Data Structures**

In designing the library management system, careful consideration was given to the selection of data structures. The program uses **dictionaries**, **lists**, and **tuples** to manage books, members, and genres efficiently. These data structures were chosen based on their flexibility, data organization capabilities, and suitability for specific tasks in the program. Below is a detailed explanation of why each was selected.

**1. Dictionary – Efficient Data Mapping and Retrieval**

The **dictionary** was used to store book records, where each book is identified by a unique **ISBN number** (International Standard Book Number). Dictionaries are ideal for this because they use **key-value pairs**, which make searching, updating, and deleting records very fast.

For example, in the books dictionary:

books = {

"001": {"title": "1984", "author": "George Orwell", "genre": "Fiction", "total\_copies": 3, "available\_copies": 3}

}

Here, "001" is the key, and the book details are the values. This structure allows quick access to any book’s information by simply referencing its ISBN, e.g. books["001"]["title"].

**Reasons for using a dictionary:**

* Fast look-up and modification (O(1) average time complexity).
* Logical mapping between a unique identifier (ISBN) and related data.
* Easy to update book information such as title, author, or available copies.
* Ensures data organization and prevents duplication of records.

Overall, the dictionary provides a clear and scalable way to store large collections of books with unique identifiers, making it perfect for a library system.

**2. List – Managing Multiple Records Dynamically**

The **list** data structure was used to store the collection of members. Each member is represented as a dictionary, and all members are stored together in a single list:

members = [

{"member\_id": 1, "name": "Isha", "email": "isha@email.com", "borrowed\_books": []}

]

**Reasons for using a list:**

* Lists allow **dynamic addition and removal** of elements, which is suitable for an expanding library membership.
* They preserve **order**, making it easy to loop through all members for searching or reporting.
* Lists are flexible — they can store dictionaries or other complex data types inside them.
* Each member can be accessed using an index, or searched using loops when needed.

This design enables smooth implementation of features like adding new members (add\_member), searching members by ID, and updating borrowed book records. It also supports scalability as new members can easily be appended to the list.

**3. Tuple – Representing Fixed Categories**

The **tuple** was used to store the different **book genres** in the library:

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

Tuples were chosen because the set of genres is **constant** — these values are not expected to change during program execution. Unlike lists, tuples are **immutable**, meaning their contents cannot be modified once created. This ensures data integrity and prevents accidental changes to the list of allowed genres.

**Reasons for using a tuple:**

* Provides immutability, preventing accidental modification of categories.
* Saves memory and processing time compared to a list.
* Enhances data consistency by keeping a fixed set of allowed genres for validation when adding books.

Using tuples guarantees that all books belong to valid genres and maintains consistency throughout the program.

**Conclusion**

In conclusion, the choice of **dictionaries**, **lists**, and **tuples** in this design was intentional and based on their unique strengths:

* **Dictionaries** ensure quick access and modification of book data through unique identifiers.
* **Lists** provide flexible and organized storage for multiple member records.
* **Tuples** maintain stable, unchangeable genre categories for data validation and consistency.