**Design Rationale**

**Design Rationale for Mini Library Management System**

In this assignment, I chose specific data structures to implement the library management system efficiently using Python's built-in types: dictionaries, lists, and tuples.

**Why Dictionary for Books?**

Dictionaries are ideal for storing books with ISBN as the key because they provide O(1) average-time complexity for lookups, insertions, and deletions. Since ISBN is unique, it serves as a perfect hashable key. Each value is another dictionary holding details like title, author, genre, and total\_copies. This structure allows quick access to book information without iterating through a list, which would be slower for large numbers of books. For example, checking if a book exists or updating its details is straightforward with books[isbn]

**Why List for Members?**

Members are stored in a list of dictionaries because the order might not matter, but we need to easily append new members and remove them. Lists are mutable and support dynamic sizing. Each member's dictionary includes id (unique, but we check manually), name, email, and a list of borrowed\_books. Unlike books, members don't have a natural unique key for fast lookup (though we could use a dict with id as key, but the spec suggests list). Iteration over the list is acceptable since the number of members is likely small. For operations like finding a member, we loop through the list, which is fine for this mini system.

**Why Tuple for Genres?**

Genres are stored in a tuple because they are a fixed set of valid options that shouldn't change during runtime. Tuples are immutable, which prevents accidental modifications, ensuring data integrity. They are also more memory-efficient than lists for constants. Validation is simple: if genre in genres.

This design balances simplicity, efficiency, and adherence to Python best practices. No external libraries are needed, keeping it lightweight. For borrow/return, we calculate available copies on-the-fly by counting borrowings across members, avoiding redundant storage.