**Table of Contents**

Introduction…………………………………………………………………………. 2

Design Rationale…………………………………………………………………….2

Operation.py Explanation…………………………………………………….....3

Operation.py Code………………………………………………………………….4-5

Test.py Code…………………………………………………………………………6

Demo.py Code………………………………………………………………………..7

Uml Diagram…………………………………………………………………………..8

Conclusion……………………………………………………………………………….9

**Introduction**

This Library Management System was built using simple Python data structures selected for their clarity, efficiency, and suitability to the system’s needs.

Design Rationale

**Dictionary for Books**  
Use of the dictionary was because of its unique property to store a collection of key value pairs and there mutability allowing for unordered collection of information where each unique key maps to a specific value. This structure allows for efficient retrieval of values by referencing their corresponding keys. And there for it was a great fit for the library management app

**List for Members**  
Members are managed using a list of dictionaries. Each dictionary holds details like member\_id, name, email, and a list of borrowed books. Lists are simple, flexible, and efficient for iterating through a moderate number of records, which suits the system well. This makes it easy to add, search for, or update members as needed.

**Tuple for Genres**  
For the use of the tuple we took advantage of its immutability property which is very suitable for things like genre that we don’t want people to add too or change in anyway shape or form.

**Functions for Operations**  
All system operations—adding, searching, updating, deleting, borrowing, and returning books—are implemented as separate functions. Each function performs proper input validation and returns a Boolean value (True or False) to indicate success or failure. This design supports modular programming and makes automated testing easy with assert statements.

**Data Integrity**  
The system maintains accuracy and consistency of data through several rules:

* Every book must have a unique ISBN.
* Book genres must match the predefined tuple of valid genres.
* Books cannot be deleted if they are currently borrowed.
* Members cannot be deleted while they still have borrowed books.
* Each member can borrow a maximum of three books at once.

Overall, these design decisions create a system that is efficient, simple to understand, and easy to maintain. The use of dictionaries, lists, and tuples ensures fast data handling and reliability, while the function-based structure promotes reusability, clarity, and effective testing.

**Operation.py Explanation**

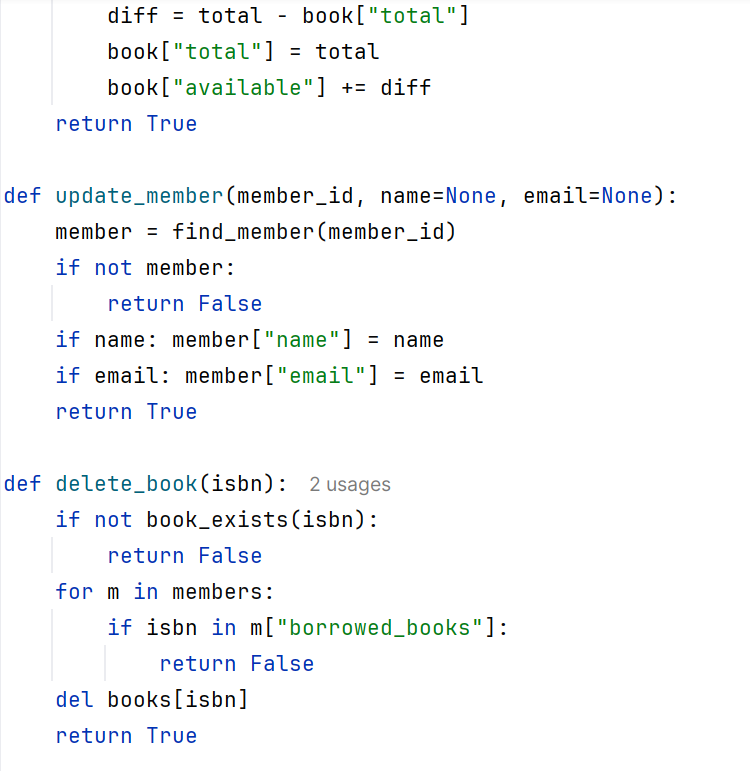
In Summary every operation needed a function for it to be possible the file **operation.py** was created to build a simple Library Management System using Python’s basic data structures and functions. The goal was to manage books and members by allowing adding, searching, updating, deleting, borrowing, and returning of books. A dictionary was used for books because each book has a unique ISBN, making it easy to look up and update details quickly. Each book record stores information such as title, author, genre, total copies, and available copies. Members were stored in a list of dictionaries since there can be many members, and a list makes it easy to loop through and find a specific member. Each member has an ID, name, email, and a list of borrowed books. The genres were stored in a tuple to keep them fixed and prevent any changes during the program’s execution.

I created Helper functions such as **find\_member** and **book\_exists** to reduce repetition and make the code easier to read. The main functions handle all operations like adding, updating, deleting, borrowing, and returning books, with checks to make sure data stays valid. For example, the system prevents adding duplicate ISBNs, borrowing more than three books, or deleting a book that is still borrowed. Instead of printing messages, each function returns **True** or **False** to make testing easier. This makes it easier to use test scripts to **assert** statements to check if the system works correctly.

Overall, I designed the **operation.py** to be clear, reliable, and easy to test. Dictionaries make managing books efficient, lists make handling members flexible, and tuples keep the genres fixed. The code structure ensures that the system runs smoothly, is easy to understand, and can be expanded or improved later if needed.

**Operations.py Code**

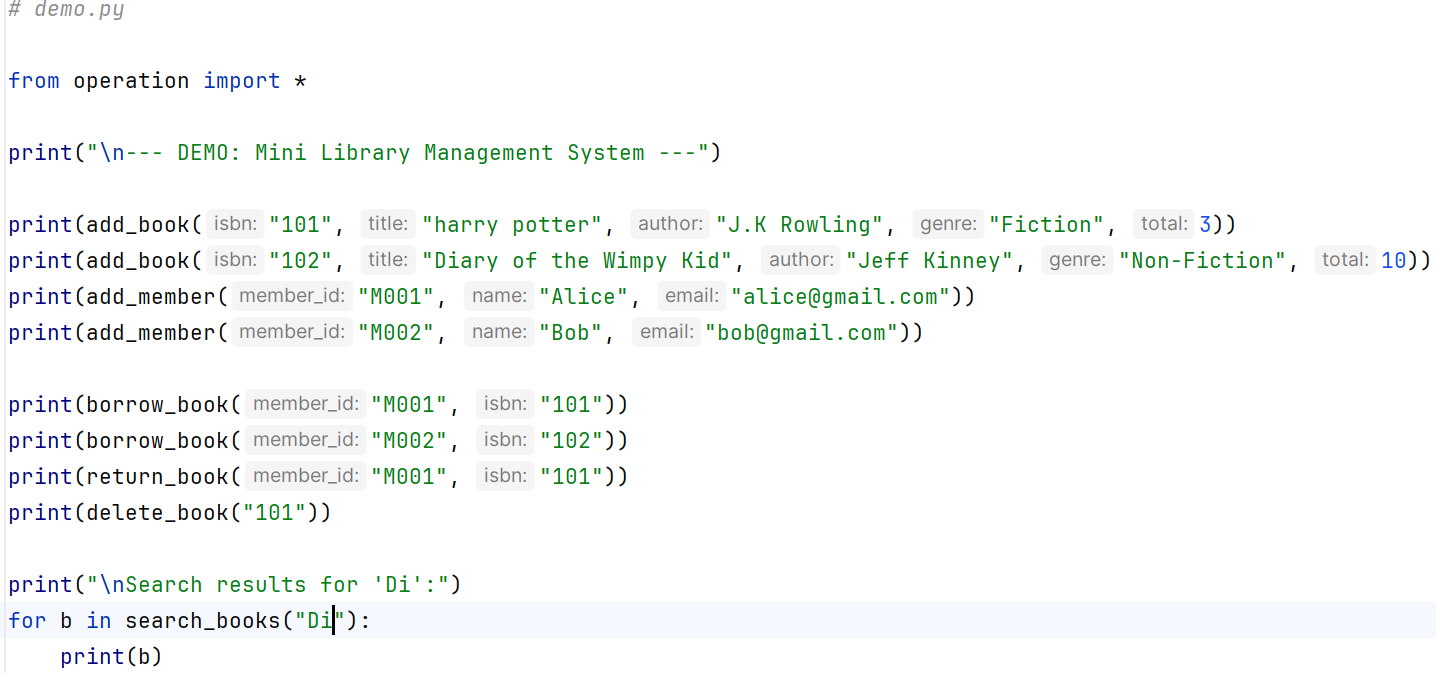
****



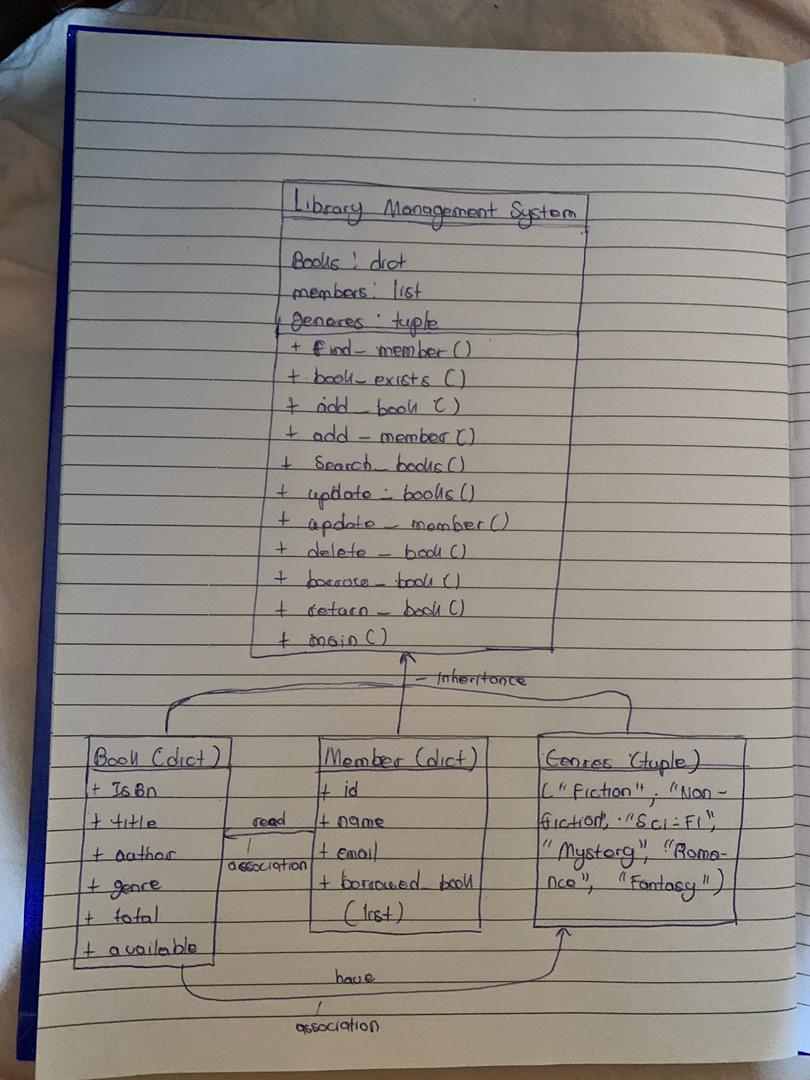
**Test.py**

****

**Demo.py**

****

**UML Diagram**

****

**Conclusion**

In summary, the Library Management System was created using simple Python data structures to make it clear, efficient, and easy to use. \*\*Dictionaries\*\* were used for books because each ISBN is unique and allows quick access to book details. \*\*Lists\*\* were used for members since they can hold many records and make it easy to search or update member information. \*\*Tuples\*\* were used for genres because they cannot be changed, keeping the list of genres fixed.

The \*\*operation.py\*\* file brings everything together with functions that handle adding, searching, updating, deleting, borrowing, and returning books. Helper functions make the code shorter and easier to understand. Each main function returns \*\*True\*\* or \*\*False\*\* so testing is simple and clear.

Overall, the system is well-structured, reliable, and easy to expand. The use of these data structures makes the program fast, organized, and suitable for building larger and more advanced systems in the future.