

**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY (BSEM)**

page1image9370432

**INTRODUCTION TO DATA BASE (COMP 102)**

page1image9370432

Title : Individual Assignment

Issue Date : WEEK 2

Due Date : WEEK 4

Lecturer/Examiner : Mr. ALUSINE LAVALEY

Name of Student/s : ALPHEOUS M CONTEH

Student ID No : 905005565

Class : BSEM 1203F

Semester/Year : 1/2

Academic Honesty Policy Statement

page1image9370432

I, hereby attest those contents of this attachment are my own work. Referenced works, articles, art, programs, papers or parts thereof are acknowledged at the end of this paper. This includes data excerpted from CD-ROMs, the Internet, other private networks, and other people’s disk of the computer system.

Student’s Signature: Date: 05 May 2025

page1image9370432

LECTURER’S COMMMENTS/GRADE:

for office use only upon receive DATE :

Receiver’s Name:

Design Rationale – Mini Library Management System

1. Introduction

The Mini Library Management System was developed to demonstrate the use of Python’s core data structures lists, dictionaries, and tuples to perform basic CRUD (Create, Read, Update, Delete) operations without using Object-Oriented Programming. The system manages books, members, and borrowing operations through structured programming and well organized functions. It mimics the workflow of a small library, allowing users to add, search, update, delete, borrow, and return books in a simple console-based environment.

2. Choice of Data Structures

a. Dictionary for Books

The books data is stored in a dictionary with the ISBN as the key. This approach allows fast lookup, easy modification, and direct access to specific book records. Since every book in a library has a unique ISBN, the dictionary structure ensures uniqueness and efficient management.

b. List for Members

The members are stored in a list of dictionaries, where each dictionary represents one member’s data such as ID, name, email, and borrowed books. A list is suitable because member records are usually fewer in number and can be iterated through easily for searching and updates.

**c. Tuple for Genres**

The GENRES variable is a tuple containing valid categories (e.g., Fiction, Non-Fiction, Sci-Fi, Biography, Mystery). A tuple was chosen because it is immutable, meaning it cannot be changed during program execution. This ensures the genre categories remain fixed, maintaining consistency in book classification.

**3. Function Design**

The system uses modular functions for clarity, reuse, and separation of logic:

• Create functions: add\_book() and add\_member() handle adding new data with validation.

• Read function: search\_books() allows partial and case-insensitive search by title or author.

• Update functions: update\_book() and update\_member() allow editing selected fields.

• Delete functions: delete\_book() and delete\_member() safely remove data only when conditions are met (no borrowed copies or books).

• Borrow/Return functions: borrow\_book() and return\_book() synchronize changes between books and members, enforcing the rule that each member can borrow a maximum of three books.

Each function returns True or False to indicate success or failure, making the system easy to test and debug.

4. Data Integrity and Validation

Several validation mechanisms were implemented:

• Unique ISBN and Member IDs: Ensures no duplicate entries.

• Valid Genres: Checked against the GENRES tuple to prevent invalid categories.

• Borrowing Limits: Members can borrow a maximum of three books.

• Book Availability: Prevents borrowing when no copies remain.

• Safe Deletion: Books and members can only be deleted if they are not currently in use.

These measures ensure consistent and reliable operation of the system.

**5. Real-World Modeling**

The program models the basic operations of a real library:

• Adding a book mimics cataloging a new library item.

• Borrowing reduces available copies, just like a real checkout.

• Returning increases available copies again.

• Members and their borrowed books are tracked in real time.

The structure supports future expansion — for example, integrating file storage, GUI interfaces, or databases later using the same logic.

6. **Conclusion**

This design demonstrates how Python’s built-in data structures can effectively simulate real-world data management tasks without using classes. The combination of dictionaries, lists, and tuples ensures both efficiency and simplicity. Each function interacts cleanly with these structures, maintaining integrity and readability. The design reflects thoughtful planning, modularity, and adherence to academic coding standards while modeling realistic library operations.









