



**FALCULTY OF INFORMATION AND COMMUNICATION
TECNOLOGY**

**PROG 211: OBJECT ORIENTED PROGRAMMING
METHOD 1**

Title : Assignment 1
Issue Date : WEEK 2
Due Date : WEEK 4
Lecturer/Examiner : Mr. Alusine Laval
Name of Student : Ransolina I. Williams
Student ID No. : 905005671
Class : BSEM
Semester/Year : 3 of 2

I hereby attest that 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:

LECTURER'S COMMENTS/GRADE:

for office use only upon receive

Remark

DATE :

TIME :

RECEIVER'S NAME :

Introduction

This document explains the design decisions and data structure choices for the Library Management System implemented in Python.

Data Structure Choices

1. Books: Dictionary (ISBN → Book Details)

Choice: dict with ISBN as key and book details as value.

Rationale:

- **Fast Lookup:** Dictionaries provide $O(1)$ average time complexity for lookups, which is crucial for frequent book searches by ISBN.
- **Unique Identification:** ISBN naturally serves as a unique primary key for books.
- **Efficient Updates:** Easy to update book details using ISBN as reference.
- **Memory Efficiency:** Stores data in key-value pairs without unnecessary indexing.

Example Structure:

```
{
  "978-0451524935": {
    "title": "1984",
    "author": "George Orwell",
    "genre": "Fiction",
    "total_copies": 5,
    "available_copies": 3
  }
}
```

2. Members: List of Dictionaries

Choice: list containing dict objects for each member.

Rationale:

- **Flexible Structure:** List allows easy iteration over all members for operations like displaying all members or finding members by criteria.
- **Dynamic Growth:** Lists naturally accommodate adding/removing members without size constraints.
- **Simple Serialization:** Easy to convert to JSON or other formats if needed for persistence.
- **Order Preservation:** Maintains insertion order which can be useful for chronological analysis.

Example Structure:

```
[
```

```
{
  "member_id": "M001",
  "name": "Alice Johnson",
  "email": "alice@email.com",
  "borrowed_books": ["978-0451524935", "978-0061120084"]
}
```

3. Genres: Tuple of Valid Genres

Choice: tuple containing valid genre strings.

Rationale:

- **Immutability:** Tuples are immutable, preventing accidental modification of valid genres.
- **Performance:** Slightly faster than lists for iteration.
- **Data Integrity:** Ensures genre validation remains consistent throughout the application.
- **Memory Efficiency:** More memory efficient than lists for fixed data.

Example Structure:

("Fiction", "Non-Fiction", "Sci-Fi", "Mystery", "Romance", "Biography", "History")

Alternative Considerations

Why not use Lists for Books?

- **Slower Search:** Linear search $O(n)$ vs dictionary's $O(1)$.
- **No Natural Key:** Would require maintaining separate ISBN indexing.
- **Complex Updates:** More complicated to update specific books.

Why not use Dictionaries for Members?

- **No Natural Primary Key:** Member IDs are generated, not natural keys like ISBN.
- **Iteration Overhead:** Would need to iterate through values anyway for many operations.
- **Ordering:** Dictionaries maintain insertion order only in Python 3.7+, but lists are more explicit.

Why not use Lists/Enums for Genres?

- **Lists:** Mutable, could be accidentally modified.
- **Enums:** Overkill for simple string validation, less intuitive.

System Architecture

Function Design Principles

- **Single Responsibility:** Each function handles one specific task.
- **Error Handling:** Comprehensive validation and meaningful error messages.
- **Return Values:** Consistent (success, message) tuple pattern.

- **Data Integrity:** Constraints enforced at function level.

Key Design Patterns

- **CRUD Operations:** Consistent Create, Read, Update, Delete patterns.
- **Constraint Enforcement:** Business rules (max 3 books, no deletion with active borrows).
- **Separation of Concerns:** Core logic separated from UI/interactive components.

Performance Considerations

- **Books Dictionary:** Optimized for the most frequent operation (book lookup by ISBN).
- **Member List:** Acceptable since member operations are less frequent than book operations.
- **Genre Tuple:** Minimal memory footprint for validation data.

Scalability Considerations

The current design works well for small to medium-sized libraries. For larger scales:

- Could add database persistence.
- Might implement indexing for member searches.
- Could add caching for frequent operations

Conclusion

The chosen data structures provide an optimal balance of performance, simplicity, and maintainability for the Library Management System's requirements. The dictionary for books enables fast lookups, the list for members supports flexible member management, and the tuple for genres ensures data integrity for validation.

```
def run_demo():  
    """Try to delete book with borrowed copies (cannot delete)"""  
    print("\nX Trying to delete '1984' (borrowed copies)...")  
    library.delete_book("1984-0451264350")  
  
    print("\nY" + " " + terminal_width)  
    print("Y LIBRARY MANAGEMENT SYSTEM DEMO COMPLETED SUCCESSFULLY!")  
    print("\nY" + " " + terminal_width)  
  
    if __name__ == "__main__":  
        run_demo()
```

Run demo

C:\Users\LENOVO\PycharmProjects\PythonProject1\venv\Scripts\python.exe C:\Users\LENOVO\PycharmProjects\PythonProject1\Friend\demo.py

LIBRARY MANAGEMENT SYSTEM - DEMO

1. ADDING BOOKS TO COLLECTION

Adding 'Dune'...

ADDING NEW BOOK TO LIBRARY COLLECTION

Book 'Dune' added successfully to Library!

Adding '1984'...

ADDING NEW BOOK TO LIBRARY COLLECTION

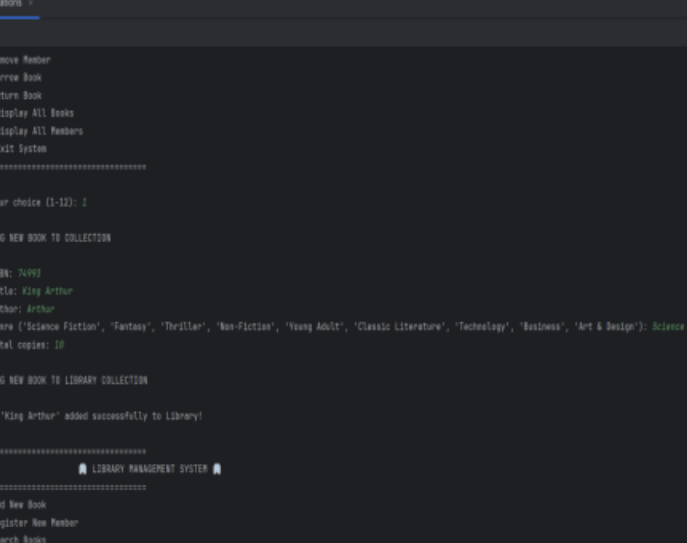
Book '1984' added successfully to Library!

```
WELCOME TO LIBRARY MANAGEMENT SYSTEM!  
Your gateway to knowledge and discovery!
```

LIBRARY MANAGEMENT SYSTEM

1. Add New Book
2. Register New Member
3. Search Books
4. Update Book Details
5. Update Member Information
6. Remove Book
7. Remove Member
8. Borrow Book
9. Return Book
10. Display All Books
11. Display All Members
12. Exit System

Enter your choice (1-12):



```

PythonProject - Version control - Current file - PythonProject
demo.py operations.py
Run operations.py
1. Remove Member
2. Borrow Book
3. Return Book
4. Display All Books
5. Display All Members
6. Exit System
Enter your choice (1-12): 1
1. ADDING NEW BOOK TO COLLECTION
Enter ISBN: 2490
Enter title: King Arthur
Enter author: Arthur
Enter genre ('Science Fiction', 'Fantasy', 'Thriller', 'Non-Fiction', 'Young Adult', 'Classic Literature', 'Technology', 'Business', 'Art & Design'): Science Fiction
Enter total copies: 10
1. ADDING NEW BOOK TO LIBRARY COLLECTION
Book 'King Arthur' added successfully to Library!
LIBRARY MANAGEMENT SYSTEM
1. Add New Book
2. Register New Member
3. Search Books
4. Update Book Details
5. Update Member Information
6. Remove Book
PythonProject - Friend - operations.py
624 UTF-8 4 spaces Python 3.13 (PythonProject)

```