LIMKOKWING UNIVERSITY OF CREATIVE TECHNOLOGY
SIERRA LEONE

**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY**

# PROG211 – OBJECT-ORIENTED PROGRAMMING 1

Title : Mini Library Management System.
Issue Date : Week 2
Due Date : Week 4
Lecturer/Examiner : Mr. Amandus coker
Name of Students : Makiba Hassan
Student ID No. : 905005090
Class : BIT 1102F
Year/Semester : 2/1

Academic Honesty Policy Statement

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:

LECTURER'S COMMMENTS/GRADE:

for office use only upon receive

Remark

DATE:
TIME:

RECEIVER'S NAME:

# Operations.py

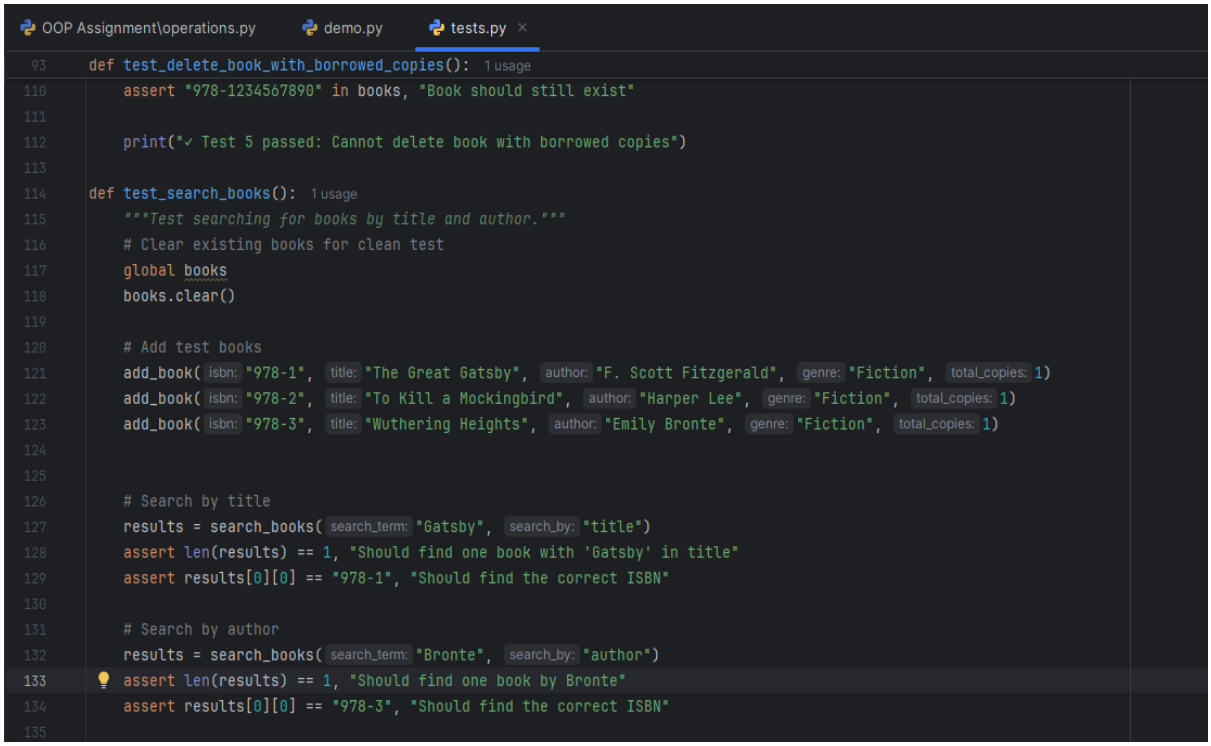
OOP Assignment\operations.py × demo.py tests.py

```python
"""
Mini Library Management System
A simple library management system using Python data structures.
"""

# Data Structures
# Books: Dictionary where key is ISBN, value is book details
books = {}

# Members: List of dictionaries containing member information
members = []

# Genres: Tuple of valid genres
GENRES = ("Fiction", "Non-Fiction", "Sci-Fi", "Mystery", "Romance", "Biography", "History", "Science")

def add_book(isbn, title, author, genre, total_copies):
    """
    Add a book to the system.

    Args:
        isbn (str): Unique ISBN identifier
        title (str): Book title
        author (str): Book author
        genre (str): Book genre (must be in GENRES tuple)
        total_copies (int): Number of copies available

    Returns:
```

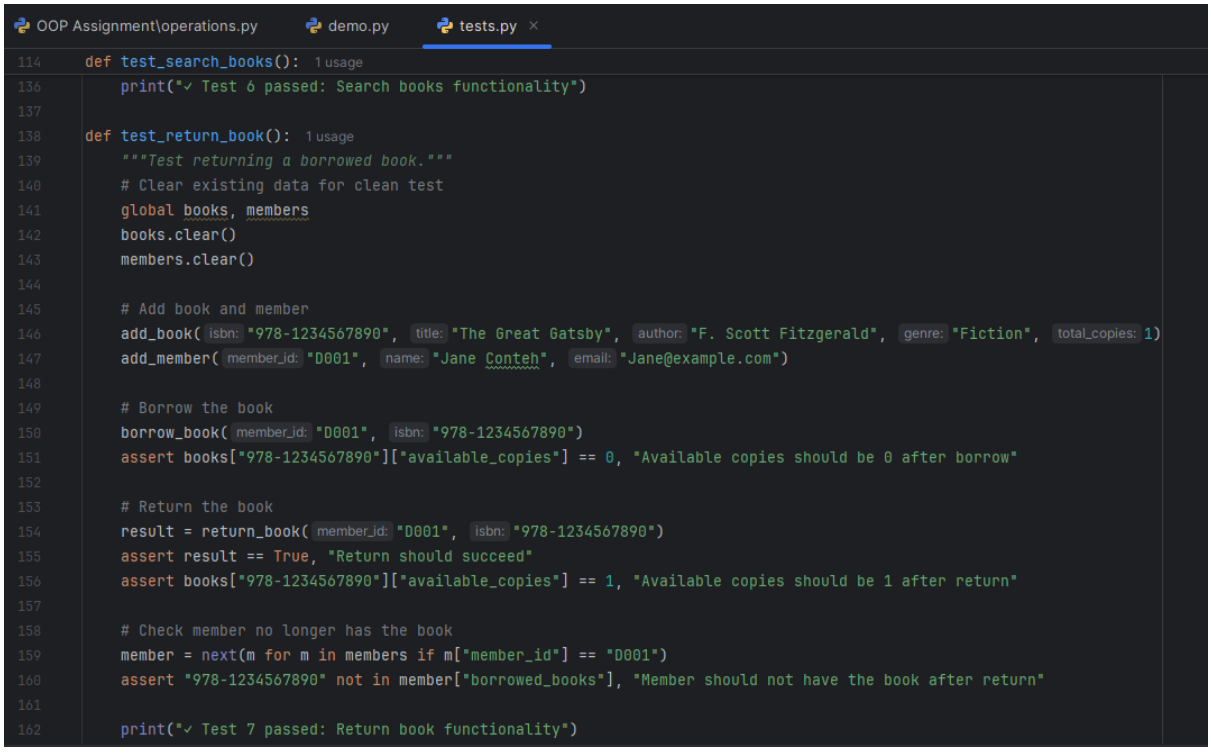
OOP Assignment\operations.py × demo.py tests.py

```python
def add_book(isbn, title, author, genre, total_copies):
    Returns:
        bool: True if book added successfully, False otherwise
    """
    # Validate ISBN uniqueness
    if isbn in books:
        print(f"Error: Book with ISBN {isbn} already exists.")
        return False

    # Validate genre
    if genre not in GENRES:
        print(f"Error: Invalid genre '{genre}'. Valid genres are: {', '.join(GENRES)}")
        return False

    # Validate total_copies
    if total_copies <= 0:
        print("Error: Total copies must be greater than 0.")
        return False

    # Add book to dictionary
    books[isbn] = {
        'title': title,
        'author': author,
        'genre': genre,
        'total_copies': total_copies,
        'available_copies': total_copies
    }
```

# Test.py



```
OOP Assignment\operations.py   demo.py   tests.py

def test_delete_book_with_borrowed_copies():  1 usage
    assert "978-1234567890" in books, "Book should still exist"

    print("✓ Test 5 passed: Cannot delete book with borrowed copies")

def test_search_books():  1 usage
    """Test searching for books by title and author."""
    # Clear existing books for clean test
    global books
    books.clear()

    # Add test books
    add_book( isbn: "978-1", title: "The Great Gatsby", author: "F. Scott Fitzgerald", genre: "Fiction", total_copies: 1)
    add_book( isbn: "978-2", title: "To Kill a Mockingbird", author: "Harper Lee", genre: "Fiction", total_copies: 1)
    add_book( isbn: "978-3", title: "Wuthering Heights", author: "Emily Bronte", genre: "Fiction", total_copies: 1)



    # Search by title
    results = search_books( search_term: "Gatsby", search_by: "title")
    assert len(results) == 1, "Should find one book with 'Gatsby' in title"
    assert results[0][0] == "978-1", "Should find the correct ISBN"

    # Search by author
    results = search_books( search_term: "Bronte", search_by: "author")
    assert len(results) == 1, "Should find one book by Bronte"
    assert results[0][0] == "978-3", "Should find the correct ISBN"
```



```
OOP Assignment\operations.py   demo.py   tests.py

def test_search_books():  1 usage
    print("✓ Test 6 passed: Search books functionality")

def test_return_book():  1 usage
    """Test returning a borrowed book."""
    # Clear existing data for clean test
    global books, members
    books.clear()
    members.clear()

    # Add book and member
    add_book( isbn: "978-1234567890", title: "The Great Gatsby", author: "F. Scott Fitzgerald", genre: "Fiction", total_copies: 1)
    add_member( member_id: "D001", name: "Jane Conteh", email: "Jane@example.com")

    # Borrow the book
    borrow_book( member_id: "D001", isbn: "978-1234567890")
    assert books["978-1234567890"]["available_copies"] == 0, "Available copies should be 0 after borrow"

    # Return the book
    result = return_book( member_id: "D001", isbn: "978-1234567890")
    assert result == True, "Return should succeed"
    assert books["978-1234567890"]["available_copies"] == 1, "Available copies should be 1 after return"

    # Check member no longer has the book
    member = next(m for m in members if m["member_id"] == "D001")
    assert "978-1234567890" not in member["borrowed_books"], "Member should not have the book after return"

    print("✓ Test 7 passed: Return book functionality")
```

# Demo.py

OOP Assignment\operations.py | demo.py | tests.py

```python
"""
Demo Script for Mini Library Management System
Demonstrates the usage of all core functionalities.
"""

from operations import *

def demo():
    """Demonstrate the library management system functionality."""
    print("=" * 60)
    print("MINI LIBRARY MANAGEMENT SYSTEM - DEMO")
    print("=" * 60)

    # Clear any existing data
    global books, members
    books.clear()
    members.clear()

    print("\n1. ADDING BOOKS")
    print("-" * 20)
    add_book( isbn: "0001",  title: "The Great Gatsby",  author: "F. Scott Fitzgerald",  genre: "Fiction",  total_copies: 3)
    add_book( isbn: "0002",  title: "Wuthering Height",  author: "Emily Bronte",  genre: "Fiction",  total_copies: 2)
    add_book( isbn: "0003",  title: "To Kill a Mockingbird",  author: "Harper Lee",  genre: "Fiction",  total_copies: 4)
    add_book( isbn: "0004",  title: "Ender's Game",  author: "Orson Scott Card",  genre: "Sci-Fi",  total_copies: 1)

    print("\n2. ADDING MEMBERS")
    print("-" * 20)
    add_member( member_id: "D001",  name: "Mary Small",  email: ".Mary@email.com")
```

OOP Assignment\operations.py | demo.py | tests.py

```python
def demo():
    add_member( member_id: "D002",  name: "Jon Smith",  email: "Jon@email.com")
    add_member( member_id: "D003",  name: "James saidu",  email: "saidu@email.com")

    print("\n3. DISPLAYING BOOKS")
    print("-" * 20)
    display_books()

    print("\n4. DISPLAYING MEMBERS")
    print("-" * 20)
    display_members()

    print("\n5. SEARCHING BOOKS")
    print("-" * 20)
    print("Searching for books with 'Dune' in title:")
    results = search_books( search_term: "Dune",  search_by: "title")
    for isbn, book in results:
        print(f"  Found: {book['title']} by {book['author']} (ISBN: {isbn})")

    print("\nSearching for books by 'Bronte':")
    results = search_books( search_term: "Bronte",  search_by: "author")
    for isbn, book in results:
        print(f"  Found: {book['title']} by {book['author']} (ISBN: {isbn})")

    print("\n6. BORROWING BOOKS")
    print("-" * 20)
    print("Jon borrows To Kill a Mockingbird:")
    borrow_book( member_id: "D002",  isbn: "0003")
```

## UML Diagram showing structures and functions

## Design Rationale

### 1. The use of dictionary

Dictionaries were used for books because they store data as key-value pairs, which makes searching and updating very fast. Each book's ISBN is unique, so it was used as the key, while the value stores the book details such as title, author, genre, and total copies.

**Why we use dictionary**

- ✓ Quick access to book details using ISBN.
- ✓ Easy to update and delete specific books.
- ✓ Prevents duplicate book entries.

**Example:**

```
books = {
   "001": {"title": "Python Basics", "author": "John Smith", "genre": "Non-Fiction", "total_copies": 3}
}
```

**2. The use of list**

A list was used for members because it allows storing multiple items in order, and each member can be represented as a dictionary. Each member dictionary contains attributes such as member ID, name, email, and borrowed books.

**Why we use list**

- ✓ Simple and flexible structure for adding or removing members.
- ✓ Easy to loop through and search by member ID.
- ✓ Works well for systems with a moderate number of users.

**Example:**

```
members = [
   {"member_id": 1, "name": "Makiba Hassan", "email": "makiba@example.com", "borrowed_books": []}
]
```

**3. The use of tuple**

A tuple was used for genres because it contains a fixed set of values that should not be changed (e.g., Fiction, Non-Fiction, Sci-fi). Tuples are immutable, which means their value cannot be modified which is perfect for storing constant data.

**Why we use tuple**

- ✓ Ensures that valid genres are consistent throughout the system.
- ✓ Prevents accidental editing or addition of invalid genres.
- ✓ Saves memory since tuples are lightweight.

Example:

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