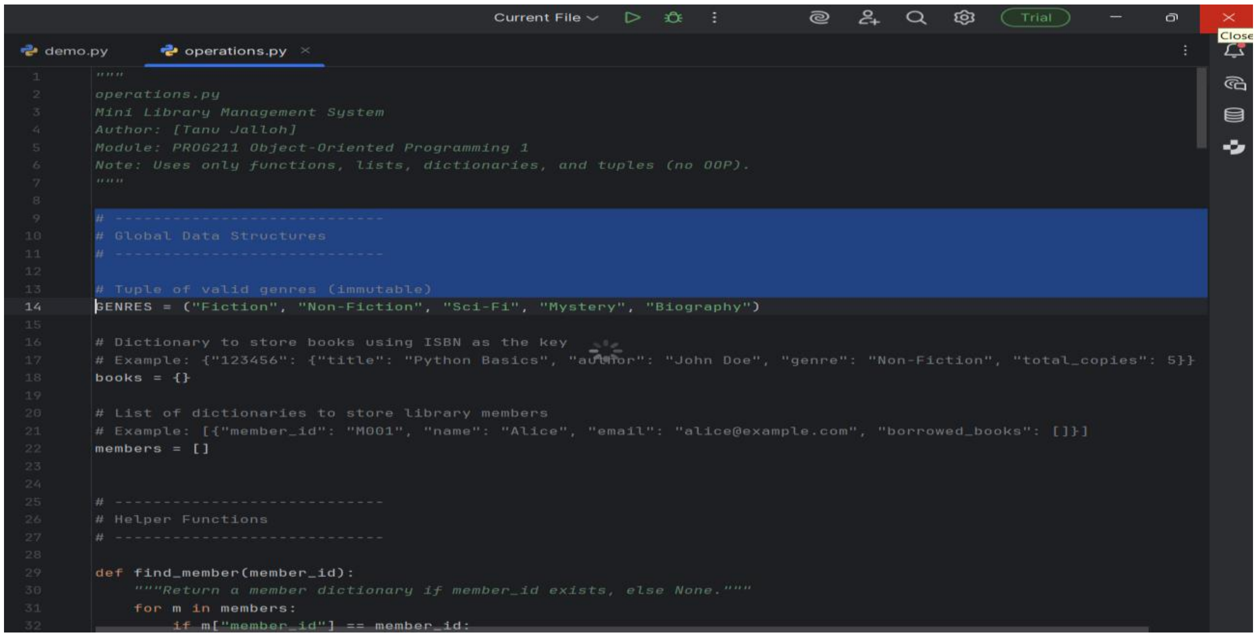
Contents

1. Introduction

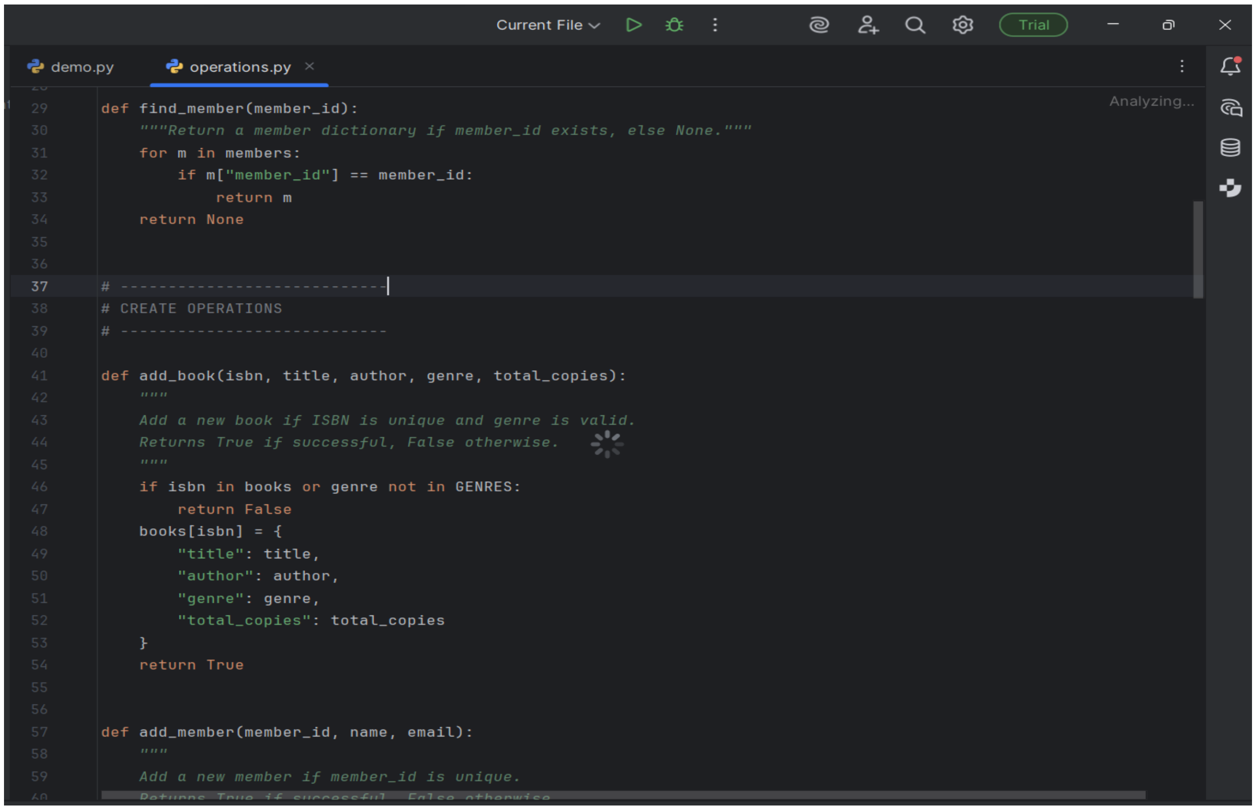
The Mini Library Management System was developed using only functions, lists, dictionaries, and tuples as required by the assignment. The system performs all basic library operations, adding, updating, deleting, searching, borrowing, and returning books, without using object-oriented programming.

The goal was to design a simple and efficient data structure that reflects real-world library activities in a functional programming style.

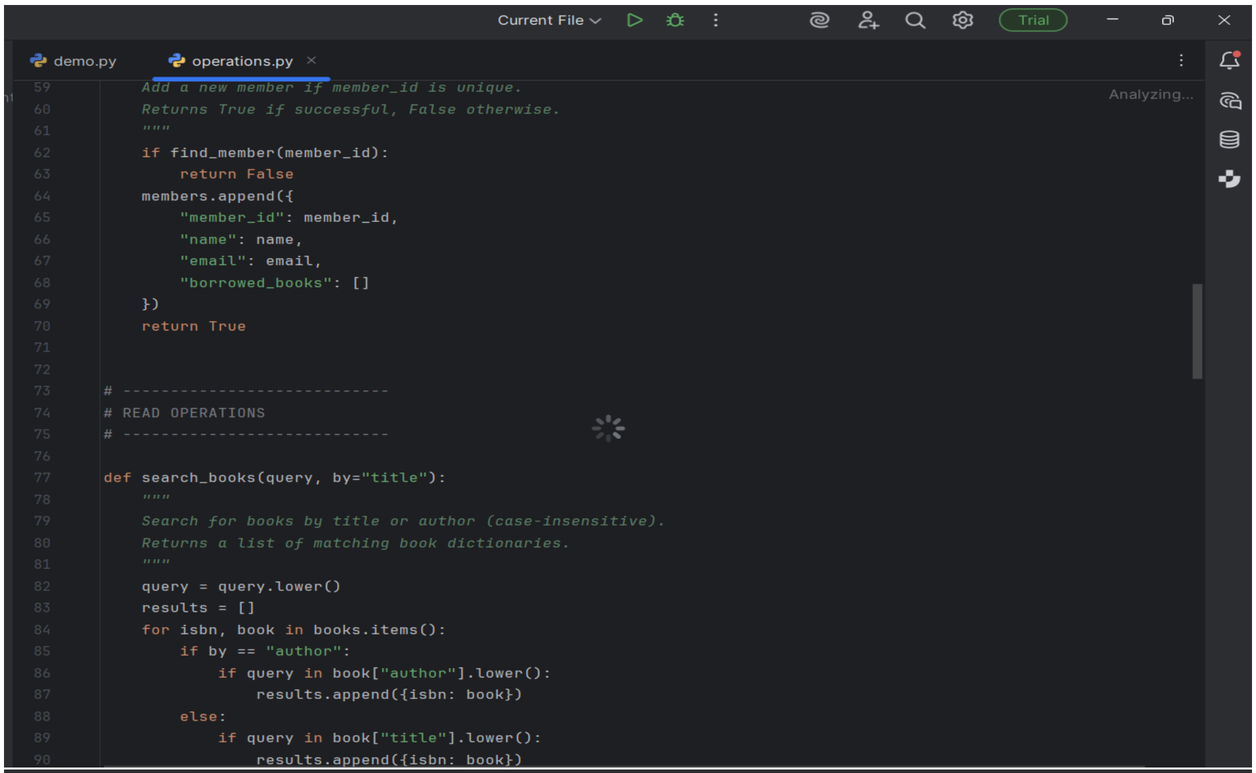
operations.py



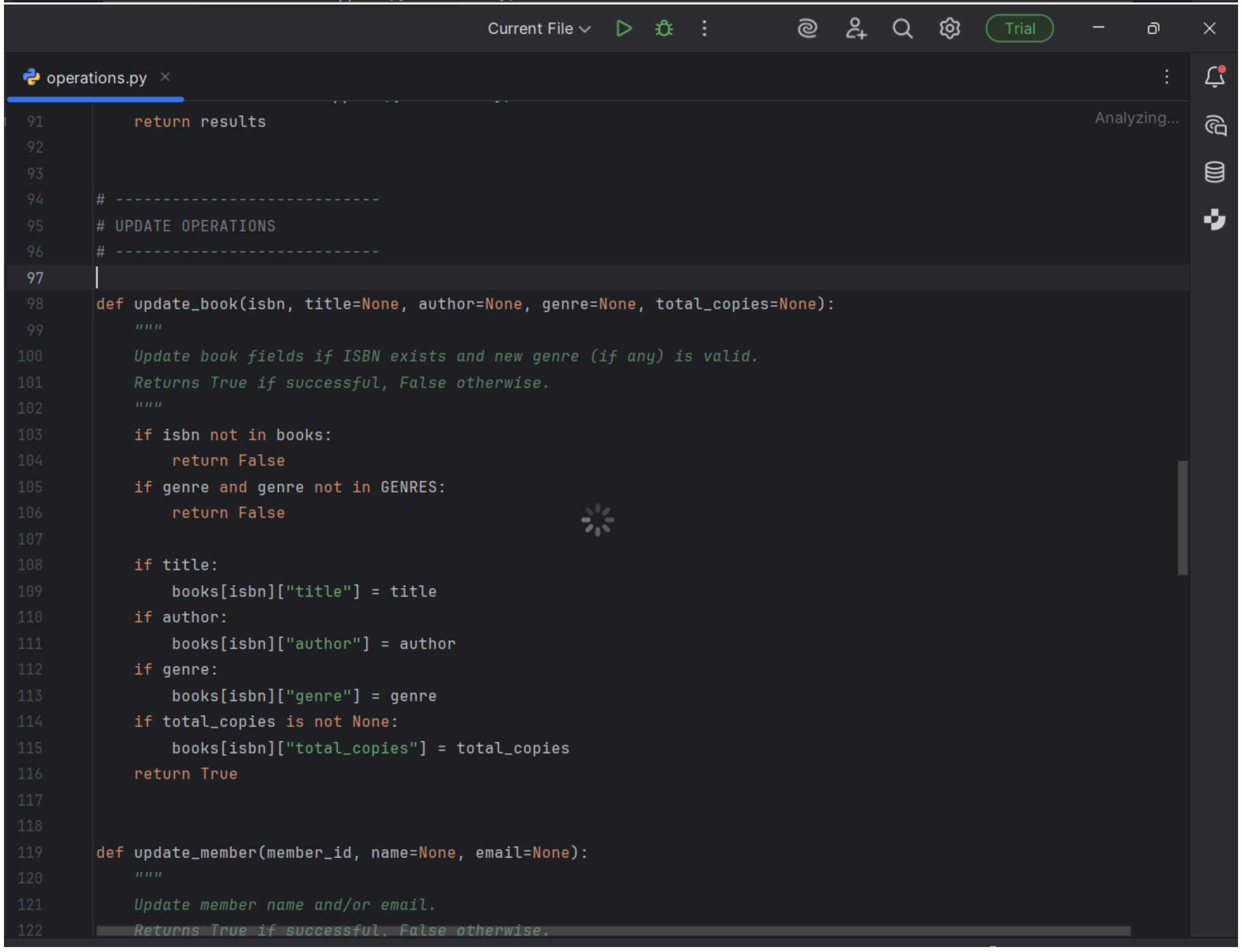
```python
"""
operations.py
Mini Library Management System
Author: [Tanu Jalloh]
Module: PROG211 Object-Oriented Programming 1
Note: Uses only functions, lists, dictionaries, and tuples (no OOP).
"""

# ----------------------------
# Global Data Structures
# ----------------------------

# Tuple of valid genres (immutable)
GENRES = ("Fiction", "Non-Fiction", "Sci-Fi", "Mystery", "Biography")

# Dictionary to store books using ISBN as the key
# Example: {"123456": {"title": "Python Basics", "author": "John Doe", "genre": "Non-Fiction", "total_copies": 5}}
books = {}

# List of dictionaries to store library members
# Example: [{"member_id": "M001", "name": "Alice", "email": "alice@example.com", "borrowed_books": []}]
members = []


# ----------------------------
# Helper Functions
# ----------------------------

def find_member(member_id):
    """Return a member dictionary if member_id exists, else None."""
    for m in members:
        if m["member_id"] == member_id:
```



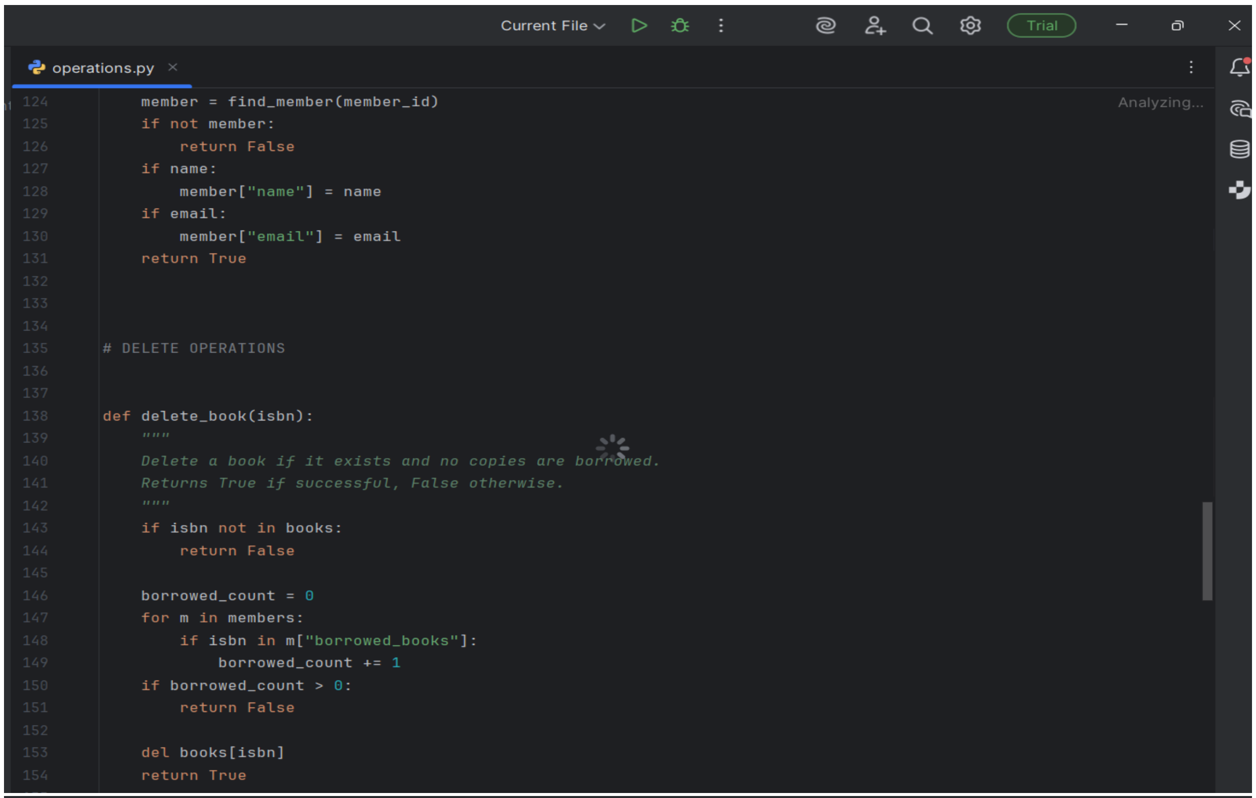
```python
def find_member(member_id):
    """Return a member dictionary if member_id exists, else None."""
    for m in members:
        if m["member_id"] == member_id:
            return m
    return None


# ----------------------------
# CREATE OPERATIONS
# ----------------------------

def add_book(isbn, title, author, genre, total_copies):
    """
    Add a new book if ISBN is unique and genre is valid.
    Returns True if successful, False otherwise.
    """
    if isbn in books or genre not in GENRES:
        return False
    books[isbn] = {
        "title": title,
        "author": author,
        "genre": genre,
        "total_copies": total_copies
    }
    return True


def add_member(member_id, name, email):
    """
    Add a new member if member_id is unique.
    Returns True if successful, False otherwise.
```



```python
    Add a new member if member_id is unique.
    Returns True if successful, False otherwise.
    """
    if find_member(member_id):
        return False
    members.append({
        "member_id": member_id,
        "name": name,
        "email": email,
        "borrowed_books": []
    })
    return True


# ---------------------------
# READ OPERATIONS
# ---------------------------

def search_books(query, by="title"):
    """
    Search for books by title or author (case-insensitive).
    Returns a list of matching book dictionaries.
    """
    query = query.lower()
    results = []
    for isbn, book in books.items():
        if by == "author":
            if query in book["author"].lower():
                results.append({isbn: book})
        else:
            if query in book["title"].lower():
                results.append({isbn: book})
```



```python
    return results


# ---------------------------
# UPDATE OPERATIONS
# ---------------------------

def update_book(isbn, title=None, author=None, genre=None, total_copies=None):
    """
    Update book fields if ISBN exists and new genre (if any) is valid.
    Returns True if successful, False otherwise.
    """
    if isbn not in books:
        return False
    if genre and genre not in GENRES:
        return False

    if title:
        books[isbn]["title"] = title
    if author:
        books[isbn]["author"] = author
    if genre:
        books[isbn]["genre"] = genre
    if total_copies is not None:
        books[isbn]["total_copies"] = total_copies
    return True


def update_member(member_id, name=None, email=None):
    """
    Update member name and/or email.
    Returns True if successful, False otherwise.
```

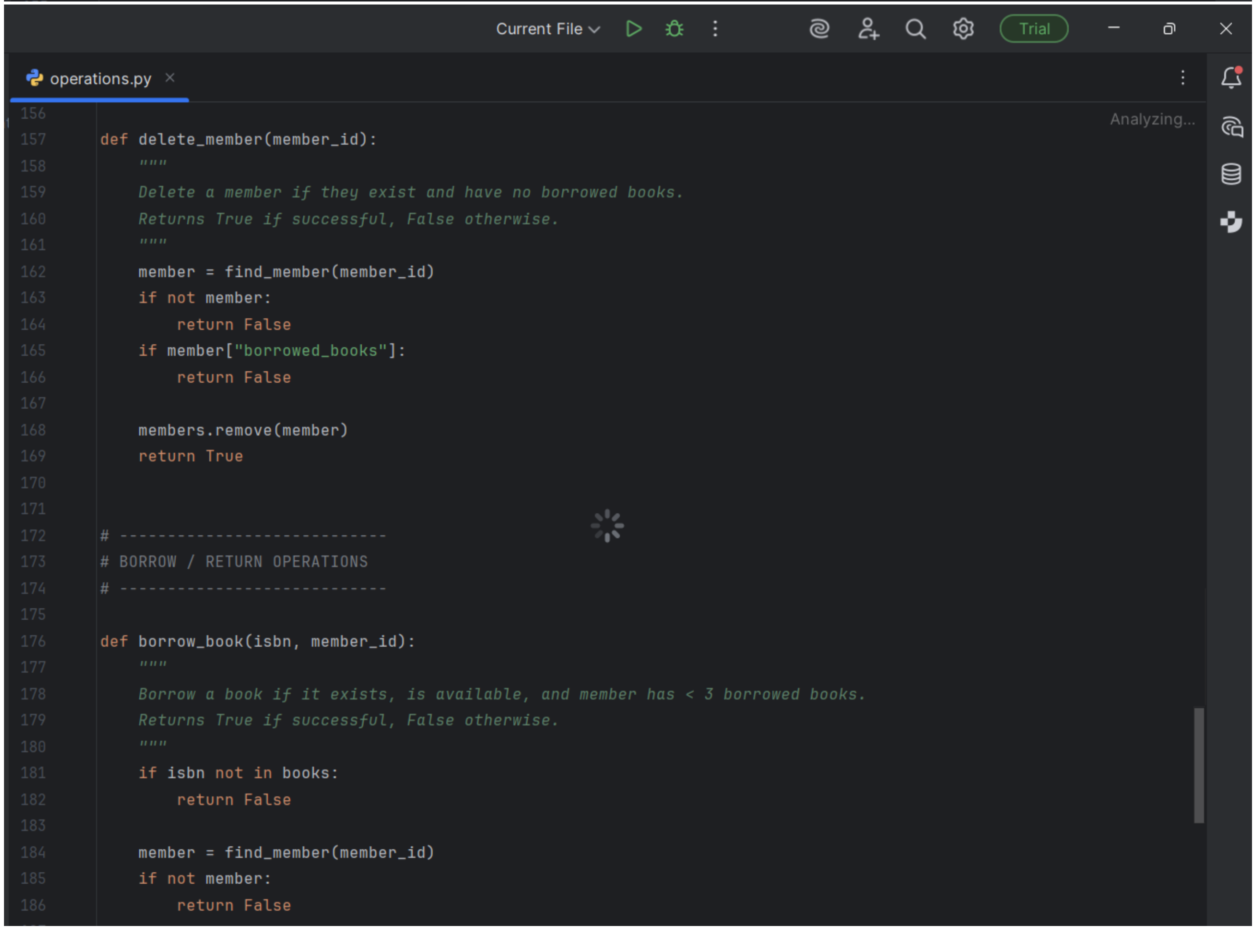


Current File

operations.py

Analyzing...

```python
    member = find_member(member_id)
    if not member:
        return False
    if name:
        member["name"] = name
    if email:
        member["email"] = email
    return True



# DELETE OPERATIONS


def delete_book(isbn):
    """
    Delete a book if it exists and no copies are borrowed.
    Returns True if successful, False otherwise.
    """
    if isbn not in books:
        return False

    borrowed_count = 0
    for m in members:
        if isbn in m["borrowed_books"]:
            borrowed_count += 1
    if borrowed_count > 0:
        return False

    del books[isbn]
    return True
```

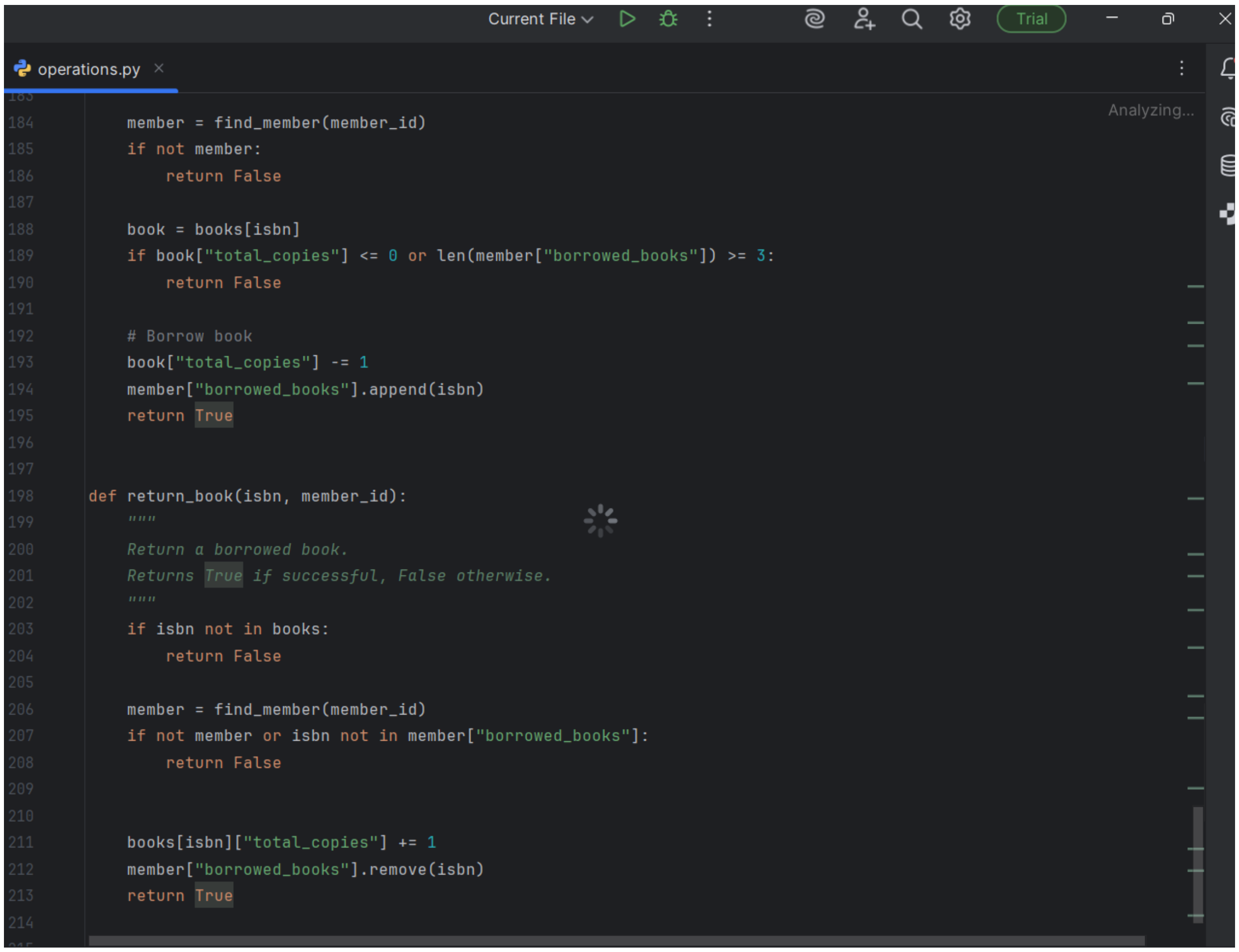


Current File

operations.py

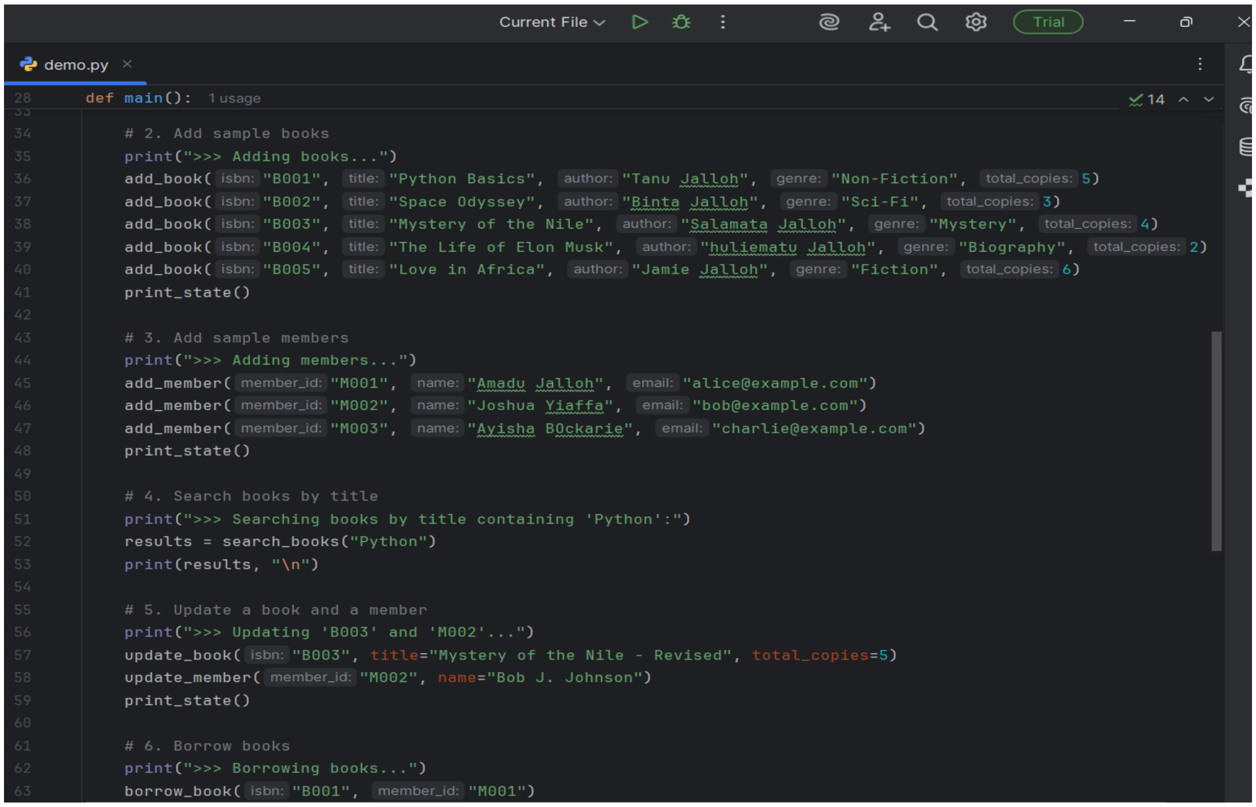
Analyzing...

```python

def delete_member(member_id):
    """
    Delete a member if they exist and have no borrowed books.
    Returns True if successful, False otherwise.
    """
    member = find_member(member_id)
    if not member:
        return False
    if member["borrowed_books"]:
        return False

    members.remove(member)
    return True


# ----------------------------
# BORROW / RETURN OPERATIONS
# ----------------------------

def borrow_book(isbn, member_id):
    """
    Borrow a book if it exists, is available, and member has < 3 borrowed books.
    Returns True if successful, False otherwise.
    """
    if isbn not in books:
        return False

    member = find_member(member_id)
    if not member:
        return False
```

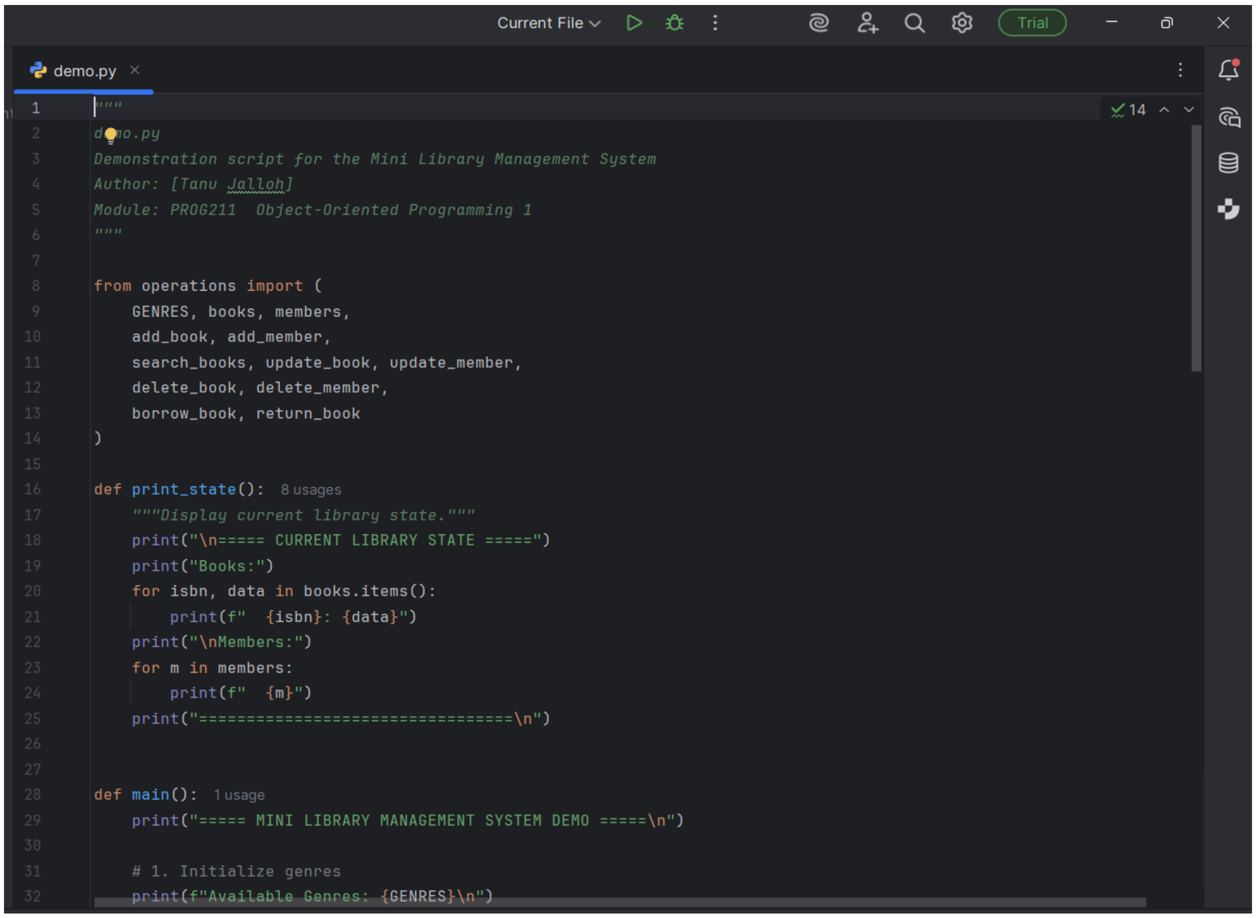


```python
    member = find_member(member_id)
    if not member:
        return False

    book = books[isbn]
    if book["total_copies"] <= 0 or len(member["borrowed_books"]) >= 3:
        return False

    # Borrow book
    book["total_copies"] -= 1
    member["borrowed_books"].append(isbn)
    return True


def return_book(isbn, member_id):
    """
    Return a borrowed book.
    Returns True if successful, False otherwise.
    """
    if isbn not in books:
        return False

    member = find_member(member_id)
    if not member or isbn not in member["borrowed_books"]:
        return False


    books[isbn]["total_copies"] += 1
    member["borrowed_books"].remove(isbn)
    return True
```

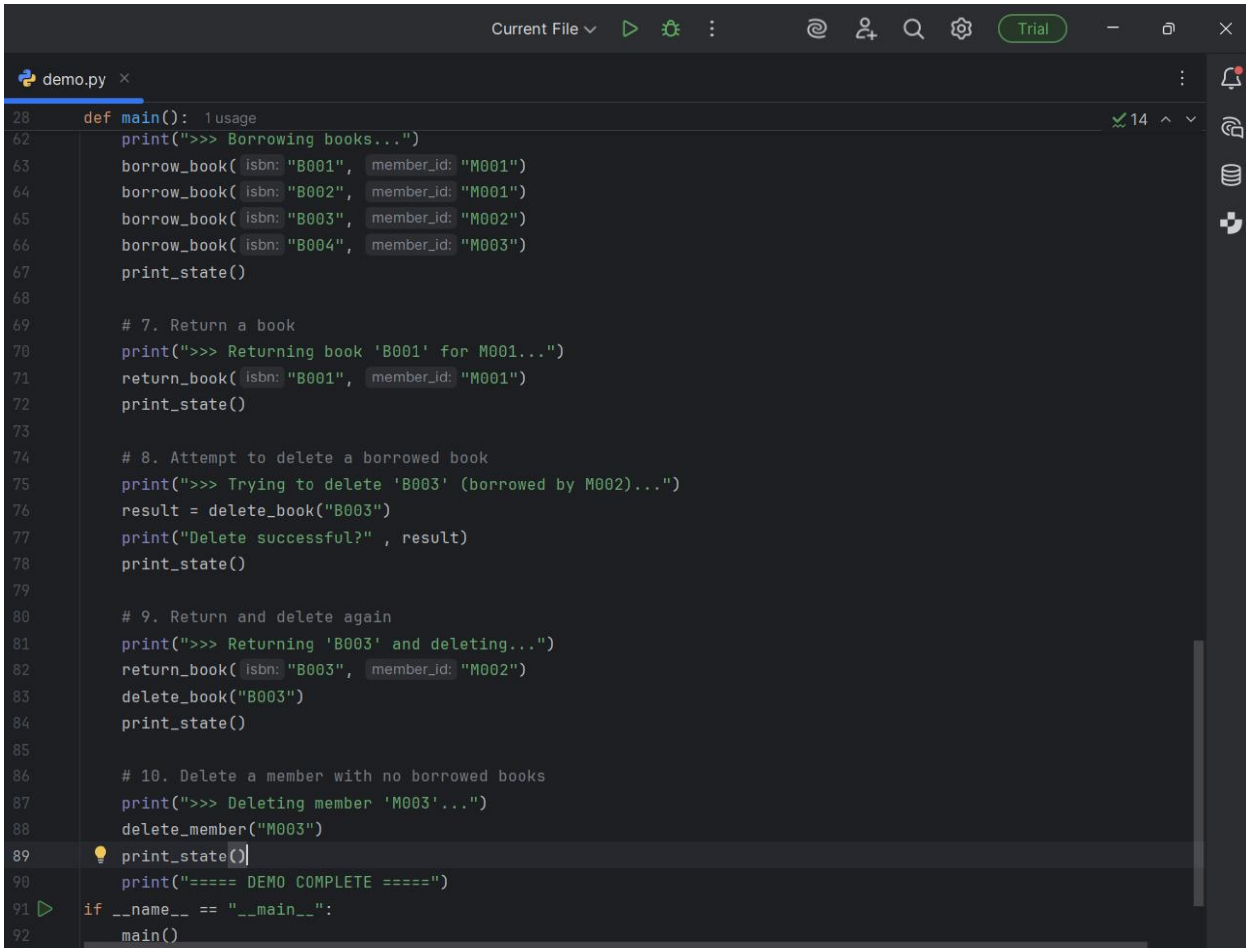
Demo.py



```python
def main():  1 usage

    # 2. Add sample books
    print(">>> Adding books...")
    add_book( isbn: "B001",  title: "Python Basics",  author: "Tanu Jalloh",  genre: "Non-Fiction",  total_copies: 5)
    add_book( isbn: "B002",  title: "Space Odyssey",  author: "Binta Jalloh",  genre: "Sci-Fi",  total_copies: 3)
    add_book( isbn: "B003",  title: "Mystery of the Nile",  author: "Salamata Jalloh",  genre: "Mystery",  total_copies: 4)
    add_book( isbn: "B004",  title: "The Life of Elon Musk",  author: "huliematu Jalloh",  genre: "Biography",  total_copies: 2)
    add_book( isbn: "B005",  title: "Love in Africa",  author: "Jamie Jalloh",  genre: "Fiction",  total_copies: 6)
    print_state()

    # 3. Add sample members
    print(">>> Adding members...")
    add_member( member_id: "M001",  name: "Amadu Jalloh",  email: "alice@example.com")
    add_member( member_id: "M002",  name: "Joshua Yiaffa",  email: "bob@example.com")
    add_member( member_id: "M003",  name: "Ayisha BOckarie",  email: "charlie@example.com")
    print_state()

    # 4. Search books by title
    print(">>> Searching books by title containing 'Python':")
    results = search_books("Python")
    print(results, "\n")

    # 5. Update a book and a member
    print(">>> Updating 'B003' and 'M002'...")
    update_book( isbn: "B003", title="Mystery of the Nile - Revised", total_copies=5)
    update_member( member_id: "M002", name="Bob J. Johnson")
    print_state()

    # 6. Borrow books
    print(">>> Borrowing books...")
    borrow_book( isbn: "B001",  member_id: "M001")
```

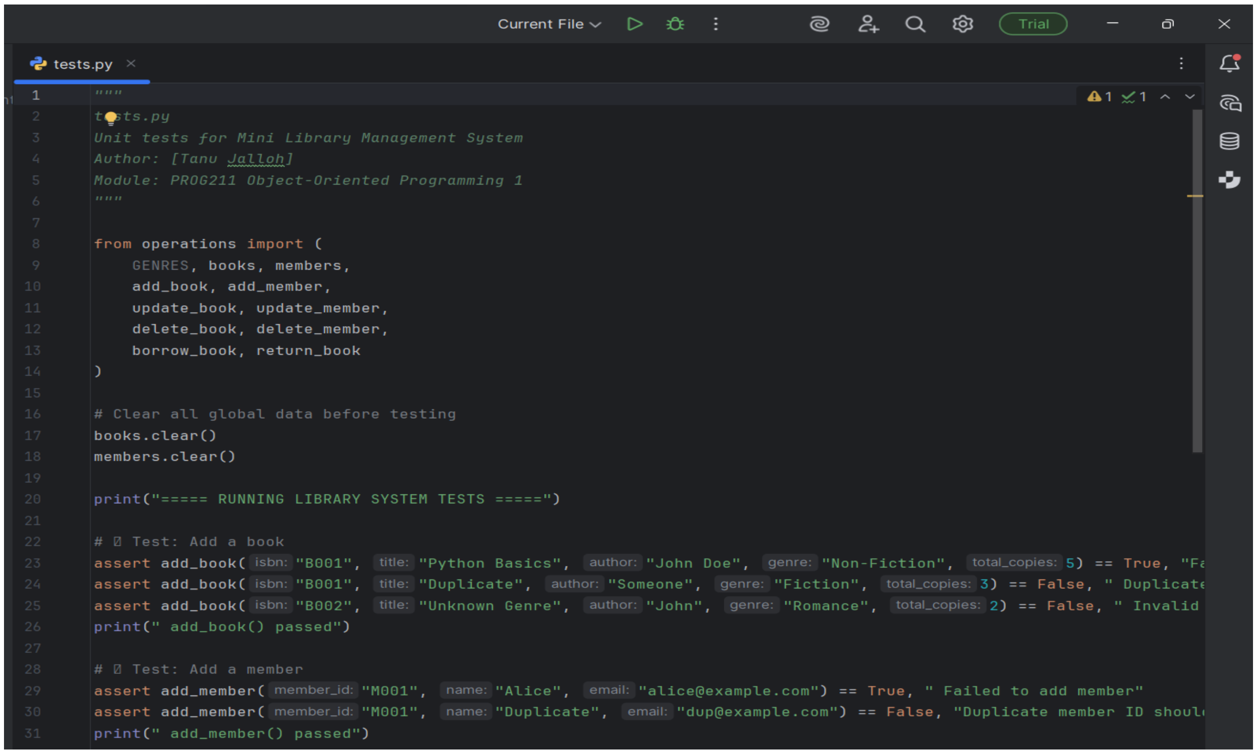


```python
"""
demo.py
Demonstration script for the Mini Library Management System
Author: [Tanu Jalloh]
Module: PROG211  Object-Oriented Programming 1
"""

from operations import (
    GENRES, books, members,
    add_book, add_member,
    search_books, update_book, update_member,
    delete_book, delete_member,
    borrow_book, return_book
)

def print_state():  8 usages
    """Display current library state."""
    print("\n===== CURRENT LIBRARY STATE =====")
    print("Books:")
    for isbn, data in books.items():
        print(f"  {isbn}: {data}")
    print("\nMembers:")
    for m in members:
        print(f"  {m}")
    print("=================================\n")


def main():  1 usage
    print("===== MINI LIBRARY MANAGEMENT SYSTEM DEMO =====\n")

    # 1. Initialize genres
    print(f"Available Genres: {GENRES}\n")
```

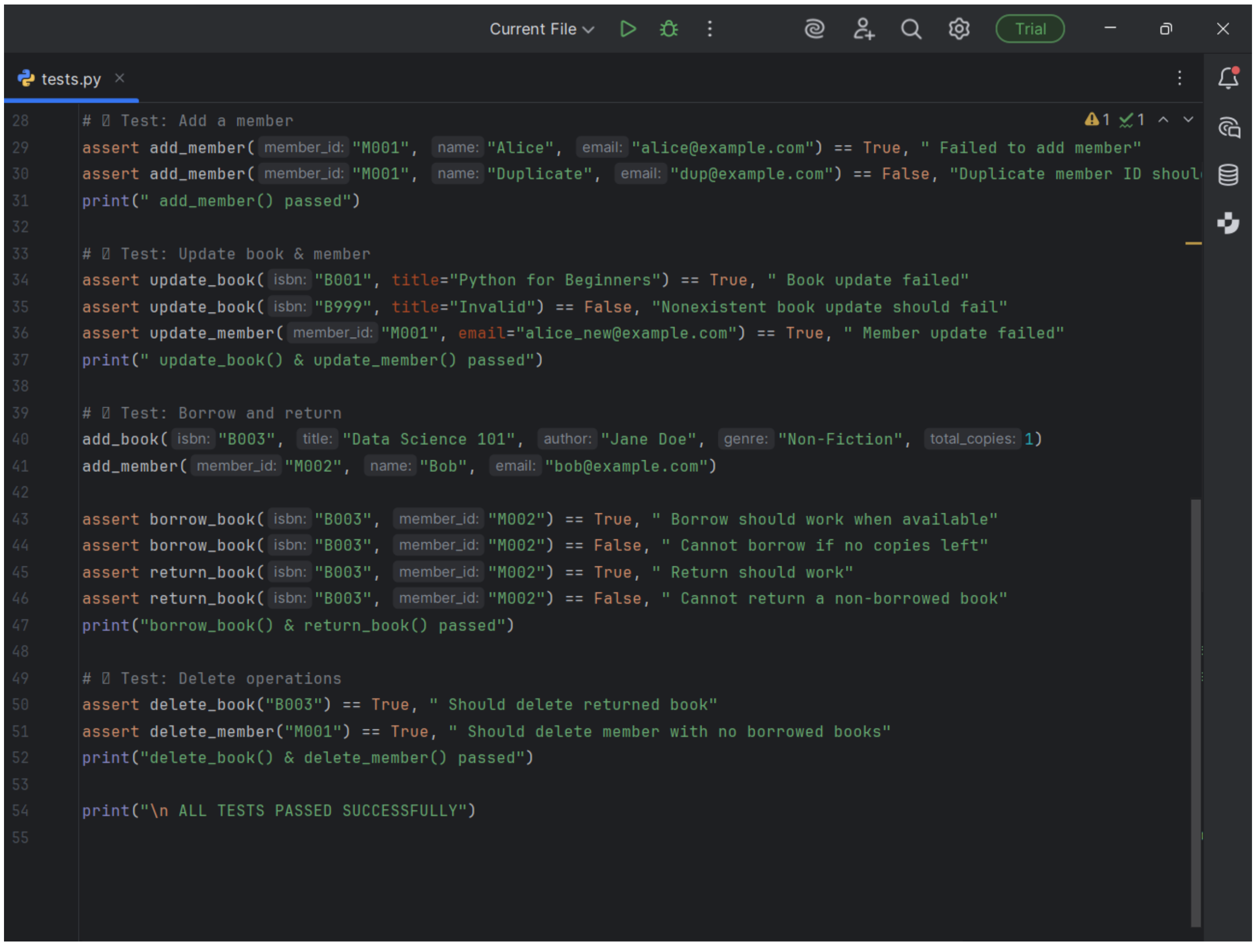


```python
def main():  1 usage
    print(">>> Borrowing books...")
    borrow_book( isbn: "B001",  member_id: "M001")
    borrow_book( isbn: "B002",  member_id: "M001")
    borrow_book( isbn: "B003",  member_id: "M002")
    borrow_book( isbn: "B004",  member_id: "M003")
    print_state()

    # 7. Return a book
    print(">>> Returning book 'B001' for M001...")
    return_book( isbn: "B001",  member_id: "M001")
    print_state()

    # 8. Attempt to delete a borrowed book
    print(">>> Trying to delete 'B003' (borrowed by M002)...")
    result = delete_book("B003")
    print("Delete successful?" , result)
    print_state()

    # 9. Return and delete again
    print(">>> Returning 'B003' and deleting...")
    return_book( isbn: "B003",  member_id: "M002")
    delete_book("B003")
    print_state()

    # 10. Delete a member with no borrowed books
    print(">>> Deleting member 'M003'...")
    delete_member("M003")
    print_state()
    print("===== DEMO COMPLETE =====")
if __name__ == "__main__":
    main()
```

Tests.py


Current File
tests.py

```python
"""
tests.py
Unit tests for Mini Library Management System
Author: [Tanu Jalloh]
Module: PROG211 Object-Oriented Programming 1
"""

from operations import (
    GENRES, books, members,
    add_book, add_member,
    update_book, update_member,
    delete_book, delete_member,
    borrow_book, return_book
)

# Clear all global data before testing
books.clear()
members.clear()

print("===== RUNNING LIBRARY SYSTEM TESTS =====")

# ☐ Test: Add a book
assert add_book( isbn: "B001",  title: "Python Basics",  author: "John Doe",  genre: "Non-Fiction",  total_copies: 5) == True, "Fa
assert add_book( isbn: "B001",  title: "Duplicate",  author: "Someone",  genre: "Fiction",  total_copies: 3) == False, " Duplicate
assert add_book( isbn: "B002",  title: "Unknown Genre",  author: "John",  genre: "Romance",  total_copies: 2) == False, " Invalid
print(" add_book() passed")

# ☐ Test: Add a member
assert add_member( member_id: "M001",  name: "Alice",  email: "alice@example.com") == True, " Failed to add member"
assert add_member( member_id: "M001",  name: "Duplicate",  email: "dup@example.com") == False, "Duplicate member ID shoul
print(" add_member() passed")
```



```python
# ▯ Test: Add a member
assert add_member( member_id: "M001",  name: "Alice",  email: "alice@example.com") == True, " Failed to add member"
assert add_member( member_id: "M001",  name: "Duplicate",  email: "dup@example.com") == False, "Duplicate member ID shoul
print(" add_member() passed")

# ▯ Test: Update book & member
assert update_book( isbn: "B001", title="Python for Beginners") == True, " Book update failed"
assert update_book( isbn: "B999", title="Invalid") == False, "Nonexistent book update should fail"
assert update_member( member_id: "M001", email="alice_new@example.com") == True, " Member update failed"
print(" update_book() & update_member() passed")

# ▯ Test: Borrow and return
add_book( isbn: "B003",  title: "Data Science 101",  author: "Jane Doe",  genre: "Non-Fiction",  total_copies: 1)
add_member( member_id: "M002",  name: "Bob",  email: "bob@example.com")

assert borrow_book( isbn: "B003",  member_id: "M002") == True, " Borrow should work when available"
assert borrow_book( isbn: "B003",  member_id: "M002") == False, " Cannot borrow if no copies left"
assert return_book( isbn: "B003",  member_id: "M002") == True, " Return should work"
assert return_book( isbn: "B003",  member_id: "M002") == False, " Cannot return a non-borrowed book"
print("borrow_book() & return_book() passed")

# ▯ Test: Delete operations
assert delete_book("B003") == True, " Should delete returned book"
assert delete_member("M001") == True, " Should delete member with no borrowed books"
print("delete_book() & delete_member() passed")

print("\n ALL TESTS PASSED SUCCESSFULLY")
```

2. Choice of Data Structures

a) Dictionaries for Books

Books are stored in a dictionary where the ISBN acts as the unique key and the value is another dictionary containing attributes like title, author, genre, and total copies.

This structure was chosen because:

•It allows fast lookup of books using ISBN as a key.

•It is easy to update or delete specific book details.

•The dictionary format makes the data readable and structured, similar to a real database record.

Example

```
books = {"123456": {"title": "Python Basics", "author": "Daniel", "genre": "Non-Fiction", "total
copies": 5}

}
```

b) Lists for Members

Members are stored in a list of dictionaries, where each dictionary contains the member's ID, name, email, and a list of borrowed books.

The list was chosen because:

•It allows simple iteration to find or update a member.

•It is suitable for small datasets, such as a classroom or college library.

•Lists are flexible and can grow dynamically as new members are added.

Example

```
members = [ {"member id": "M001", "name": "Ayisha", "email": "alice@example.com",
"borrowed books": ["123456"]}

]
```

c) Tuples for Genres

Genres are stored as a tuple because:

•Tuples are immutable, ensuring the list of valid genres cannot be accidentally changed during program execution.

•They serve as a fixed validation list when adding or updating books.

Example

GENRES = ("Fiction", "Non-Fiction", "Sci-Fi", "Mystery", "Biography")

3. Functional Design

The system uses modular functions to perform all operations. Each function handles one responsibility:

•add book() and add member() →Create new records.

•search books() → Retrieve information based on title or author.

•update book() and update member() → Modify details.

•delete book() and delete member() → Safely remove records.

•borrow book() and return book() → Handle book lending and returning.

Functions interact directly with the global data structures (books, members, and GENRES), making it easy to trace changes and test each feature independently.

4. Data Integrity & Validation

To ensure data accuracy, several validation checks were added:

•Unique ISBNs and member IDs: Prevent duplicate entries.

•Genre validation: A book can only belong to an existing genre in GENRES.

•Borrow limits: A member cannot borrow more than three books at once.

•Deletion rules: Books cannot be deleted if borrowed; members cannot be deleted if they have borrowed books.

These checks mimic real-world library policies and maintain consistent data integrity.

5. Real-World Modeling

The system's design mirrors how a real library operates:

•Adding a book: like registering a new title in a catalog.

•Borrowing a book: decreases available copies and adds to a member's record.

•Returning: increases the count and clears the member's borrowed list.

•Deleting members or books: only allowed under valid conditions.

6. Conclusion

This design achieves the objectives of the assignment by using appropriate Python data structures and functions to simulate a real-world library.

It is simple, efficient, and easy to test, maintaining a balance between readability and logical structure. The chosen approach ensures reliable data handling, modularity, and clarity, which are key qualities of good software design.