PROJECT REPORT

Library Management System

1. Cover Page

Project Title: Student Library Management System

Submitted By: Hemant Raj

Registration No: 25BEC10093

2. Introduction

Libraries are an essential part of any educational institution, providing a wealth of knowledge to students. However, managing a library manually is a tedious task. This project, the **Student Library Management System**, is a Python-based application designed to automate the core functions of a library. It allows for the efficient management of book records, issuance, returns, and fine calculation, replacing traditional paper-based registers with a digital solution.

3. Problem Statement

In the traditional manual system, librarians face several challenges:

- **Time-Consuming:** searching for a book's availability requires flipping through pages of a register.
- **Human Error:** details like return dates or student names can be recorded incorrectly.
- **Calculation Issues:** manually calculating fines for late returns is prone to mistakes and can lead to disputes.
- **Lack of Updates:** it is difficult to know the exact number of copies available on the shelf at any given moment.

This project aims to solve these problems by creating a system that tracks inventory in realtime and automates date-based calculations.

4. Functional Requirements

The system fulfills the following functional requirements:

1. **View Stock:** Users must be able to see a list of all books and the exact number of sets available.
2. **Issue Book:** The system should allow issuing a book only if stock is available (>0) and record the current date and time.

3. **Return Book:** The system must accept returns and automatically calculate the number of days the book was kept.
4. **Fine Calculation:** If the book is returned after 7 days, a fine of Rs 50 must be displayed.
5. **Add Stock:** The librarian should be able to add new books or increase the quantity of existing ones.

5. Non-functional Requirements

1. **Performance:** The system should respond instantly to user inputs without lag.
2. **Usability:** The menu-driven interface should be simple enough for anyone to understand without technical training.
3. **Reliability:** The system should not crash if a user enters a wrong input (like text instead of a number).
4. **Accuracy:** Date and fine calculations must be precise.

6. System Architecture

The system follows a simple **Monolithic Architecture** suitable for a standalone console application.

- **Interface Layer:** The command-line menu where the user interacts.
- **Application Layer:** The Python script containing the logic (Functions and Classes).
- **Data Layer:** Python Dictionaries (self.books and self.issued_books) acting as temporary in-memory storage.

7. Design Diagrams

7.1 Use Case Diagram

This diagram shows who interacts with the system and what they can do.

Actor: Librarian (User) System: Library Management System Use

Cases (The bubbles connected to the Actor):

1. **View Stock: The librarian checks which books are available.**
2. **Issue Book: The librarian gives a book to a student.** ○ ***Includes:*** **Check Availability (System checks if stock > 0).**
3. **Return Book: The librarian accepts a book back.**

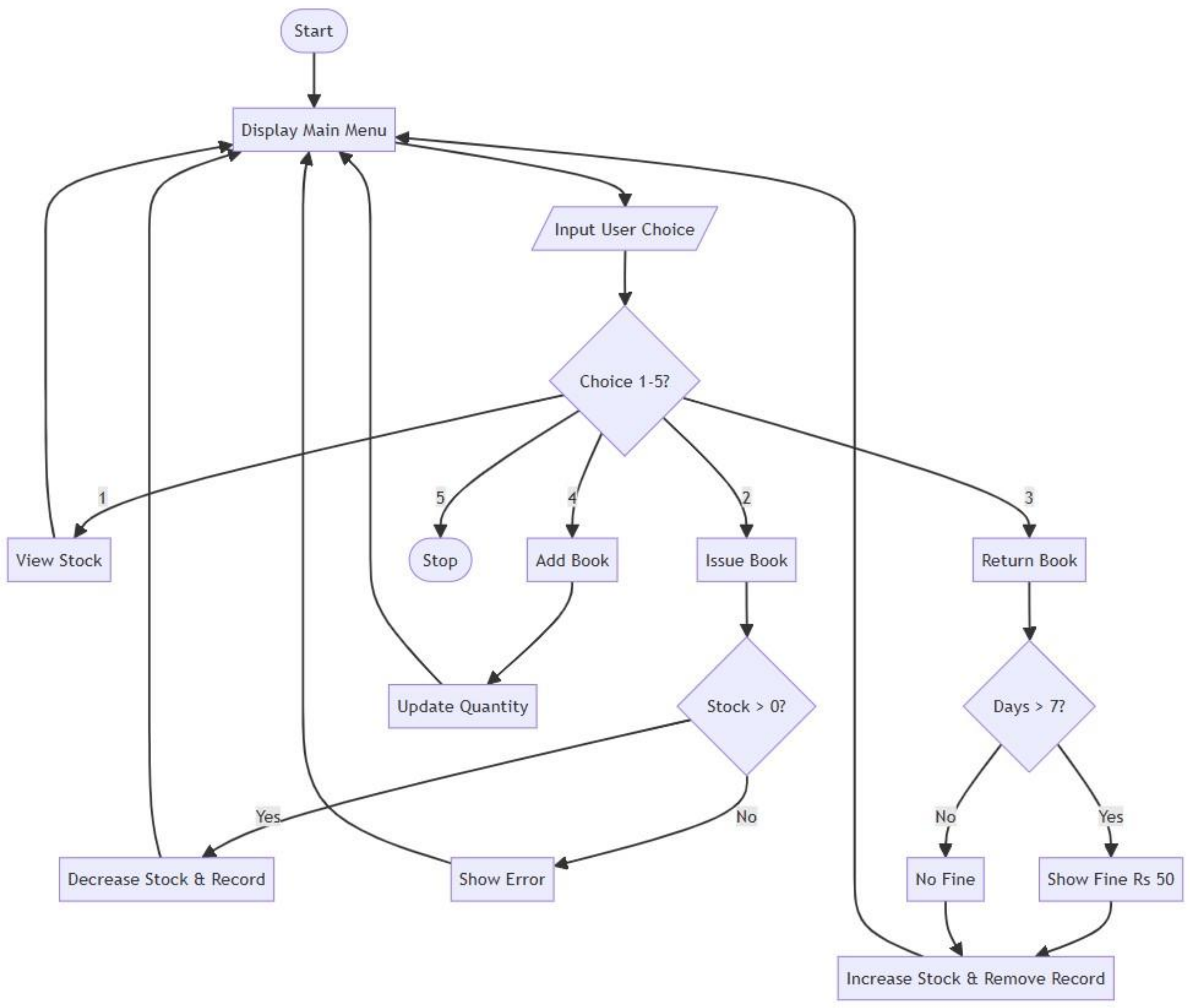
- ***Includes:*** **Calculate Fine (System checks if days > 7).**

4. Add Book: The librarian adds new books to the inventory.

7.2 Workflow Diagram (Flowchart)

This shows the step-by-step flow of the program logic.

1. **Start (Oval)**
2. **Display Main Menu (Rectangle)**
3. **Input User Choice (Parallelogram)**
4. **Decision Diamond: Is Choice 1, 2, 3, 4, or 5?**
   - **If 1 (View): Print Dictionary books -> Go back to Menu.**
   - **If 2 (Issue): Input Name & Book -> Check Stock > 0? -> Yes: Decrease Stock, Add to issued_books -> No: Show Error -> Go back to Menu.**
   - **If 3 (Return): Input Name & Book -> Calculate Days -> If > 7: Show Fine -> Increase Stock, Remove from issued_books -> Go back to Menu.**
   - **If 4 (Add): Input Book Name & Quantity -> Update books Dictionary -> Go back to Menu.**
   - **If 5 (Exit): Stop (Oval).**



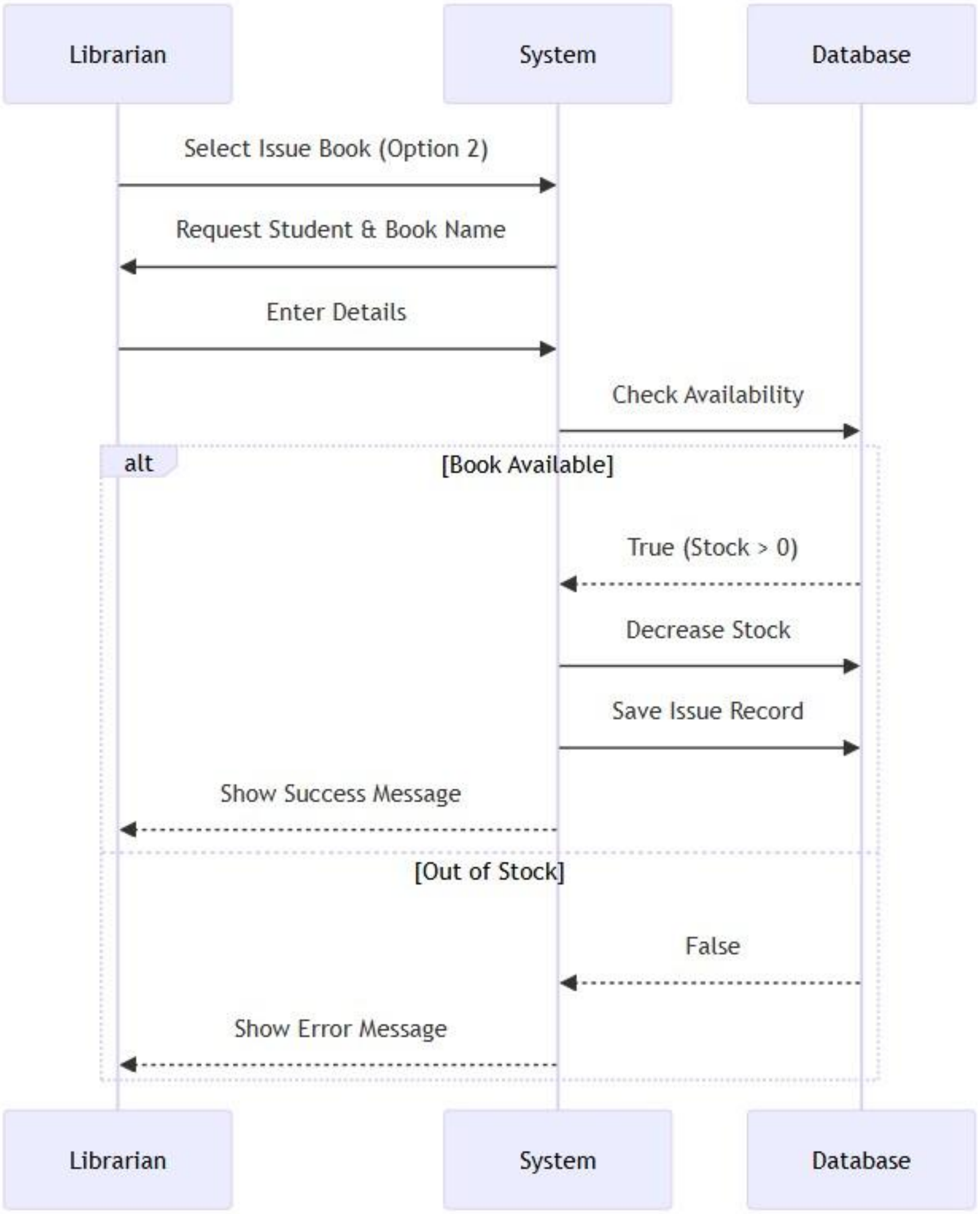
7.3 Sequence Diagram

This shows the timeline of a specific action, like "Issuing a Book".

Participants:

1. **User (Librarian)**
2. **System (The Python Script)**
3. **Database (The Dictionaries) Steps:**
   1. **User -> enters "2" (Issue Book) -> System**
   2. **System -> requests "Enter Student Name & Book Name" -> User**
   3. **User -> enters details -> System**
   4. **System -> check_availability(book) -> Database**

5. **Database -> returns "Available" (True/False) -> System**
6. **System -> (If True) update_stock(-1) and add_issue_record() -> Database**
7. **System -> prints "Book Issued Successfully" -> User**



7.4 Class Diagram

This shows the structure of the code.

Class: Library

Attributes (Variables)

- self.books: Dictionary (Stores Name : Quantity)

- self.issued_books: Dictionary (Stores Name : {Student : Date})

Methods (Functions)

+ __init__(books_dict)

+ display_available_books()

+ lend_book(user, book)

+ return_book(user, book)

+ add_book(book, quantity)

7.5 ER Diagram (Data Structure)

Since we are using Python Dictionaries instead of SQL tables, this diagram represents how our data is linked in memory.

Entity 1: Book Inventory

- **Key: Book Name (String)**
- **Value: Quantity (Integer)**

Entity 2: Issued Records

- **Key: Book Name (String)**
- **Value: Nested Dictionary containing: ○ Key: Student Name (String) ○ Value: Issue Date (Datetime Object)**

8. Design Decisions & Rationale

- **Language:** I chose **Python** because of its simplicity and strong support for data structures like lists and dictionaries, which are perfect for this project.
- **Data Structure:** I used **Dictionaries** (Hash Maps) instead of Lists for storing books.

  - *Reason:* Searching for a book in a list takes time (O(n)), but looking it up in a dictionary is instant (O(1)). This makes the program much faster.

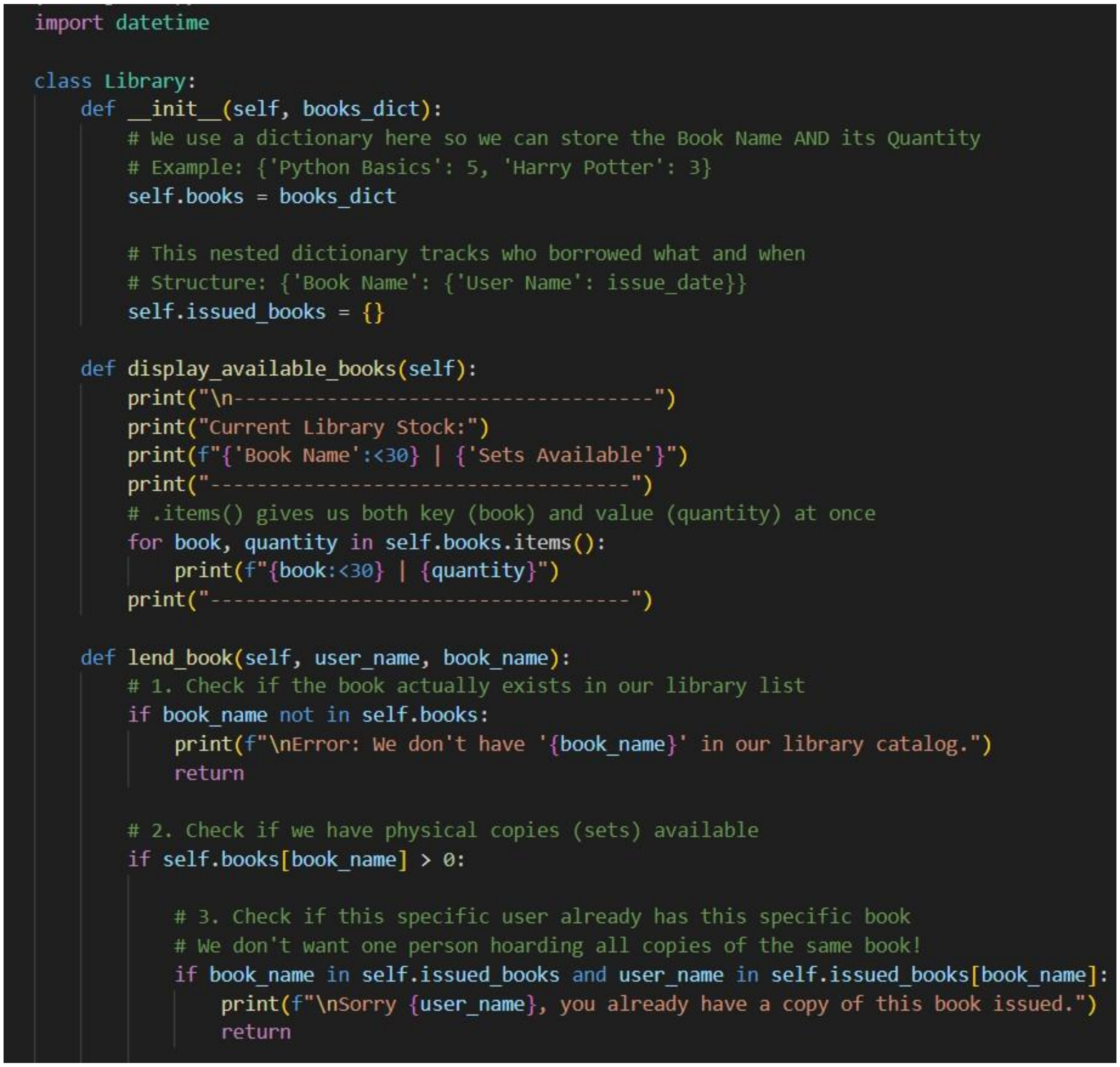
- **Class-Based Approach:** Using a Class structure keeps the code organized and modular, making it easy to upgrade in the future.

9. Implementation Details

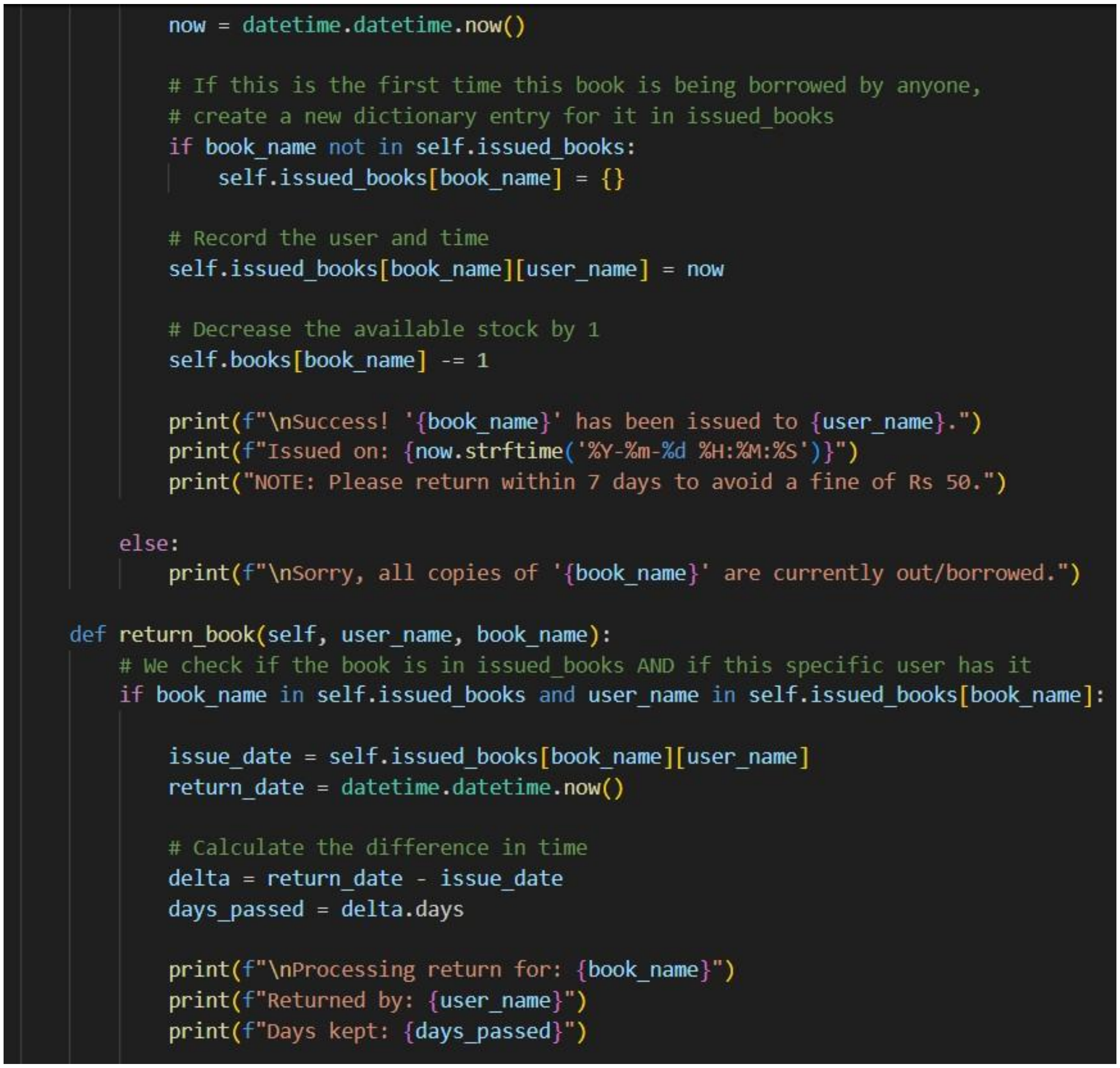
The core logic relies on the datetime module.

- **Issuing:** datetime.datetime.now() captures the exact timestamp.
- **Returning:** return_date - issue_date gives a timedelta object, from which we extract .days.
- **Logic Snippet:**
- if days_passed > 7:
- fine = 50
- else:
- fine = 0

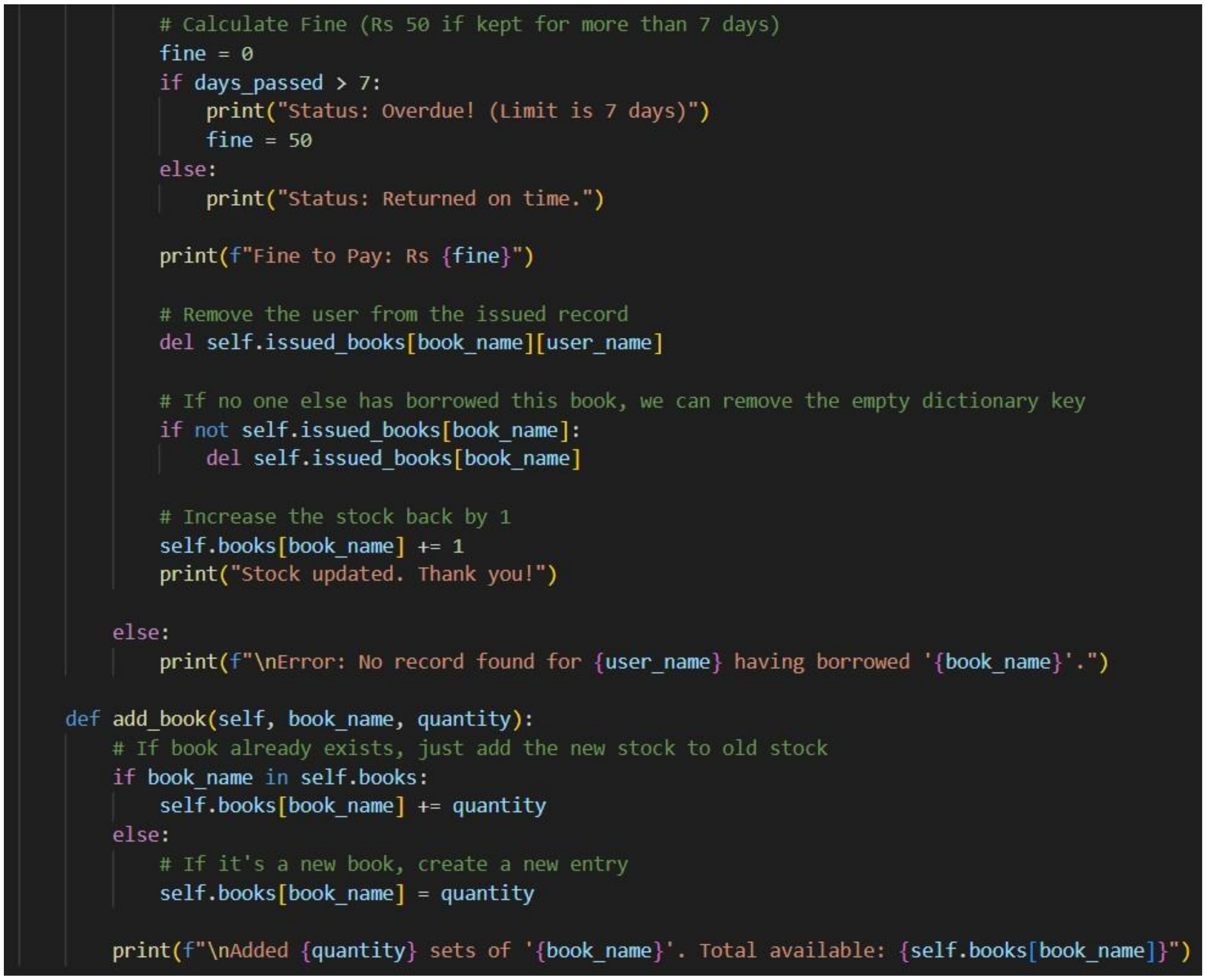
10. Screenshots / Results



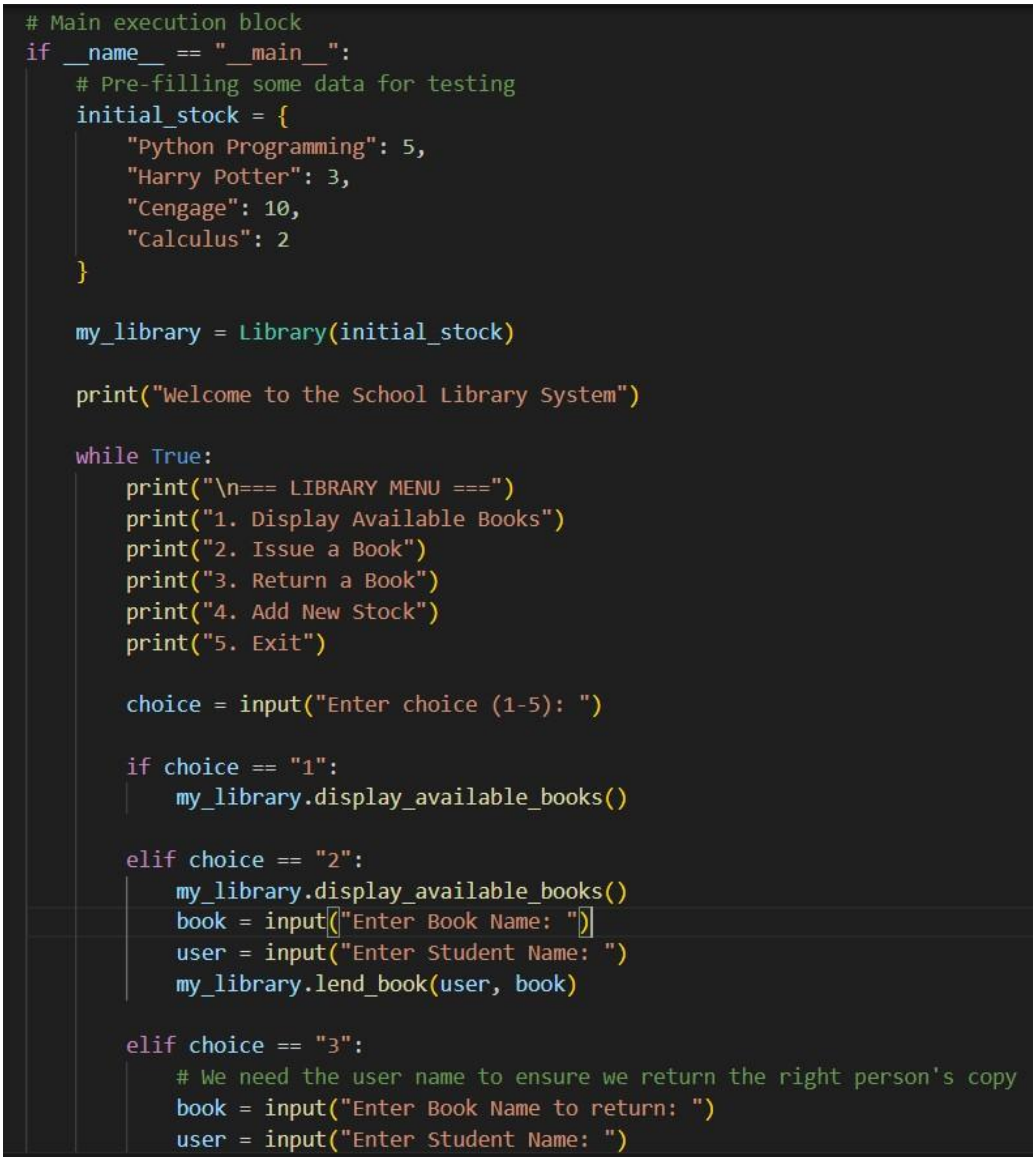
```python
import datetime

class Library:
    def __init__(self, books_dict):
        # We use a dictionary here so we can store the Book Name AND its Quantity
        # Example: {'Python Basics': 5, 'Harry Potter': 3}
        self.books = books_dict

        # This nested dictionary tracks who borrowed what and when
        # Structure: {'Book Name': {'User Name': issue_date}}
        self.issued_books = {}

    def display_available_books(self):
        print("\n------------------------------------")
        print("Current Library Stock:")
        print(f"{'Book Name':<30} | {'Sets Available'}")
        print("------------------------------------")
        # .items() gives us both key (book) and value (quantity) at once
        for book, quantity in self.books.items():
            print(f"{book:<30} | {quantity}")
        print("------------------------------------")

    def lend_book(self, user_name, book_name):
        # 1. Check if the book actually exists in our library list
        if book_name not in self.books:
            print(f"\nError: We don't have '{book_name}' in our library catalog.")
            return

        # 2. Check if we have physical copies (sets) available
        if self.books[book_name] > 0:

            # 3. Check if this specific user already has this specific book
            # We don't want one person hoarding all copies of the same book!
            if book_name in self.issued_books and user_name in self.issued_books[book_name]:
                print(f"\nSorry {user_name}, you already have a copy of this book issued.")
                return
```



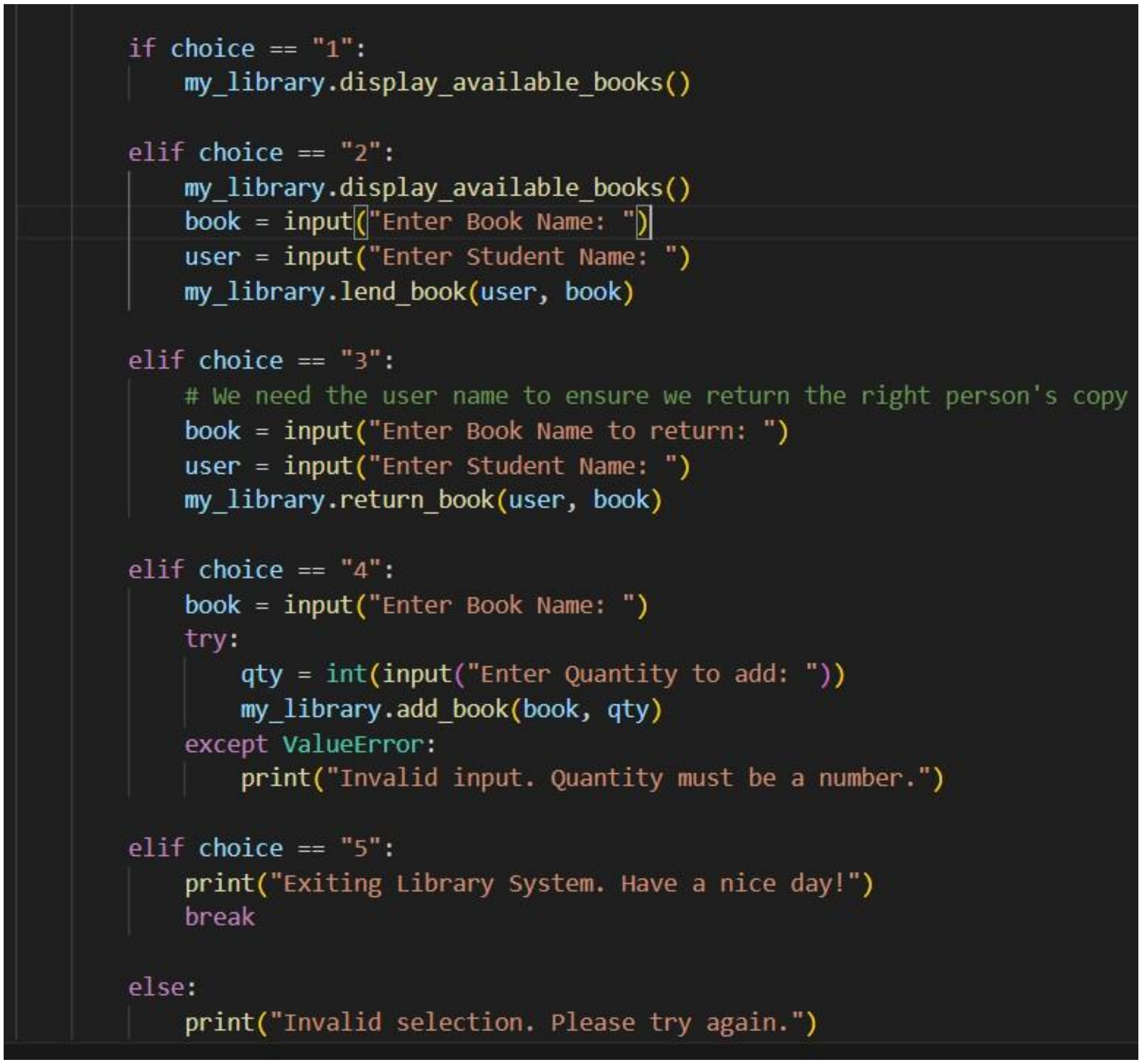
```python
            now = datetime.datetime.now()

            # If this is the first time this book is being borrowed by anyone,
            # create a new dictionary entry for it in issued_books
            if book_name not in self.issued_books:
                self.issued_books[book_name] = {}

            # Record the user and time
            self.issued_books[book_name][user_name] = now

            # Decrease the available stock by 1
            self.books[book_name] -= 1

            print(f"\nSuccess! '{book_name}' has been issued to {user_name}.")
            print(f"Issued on: {now.strftime('%Y-%m-%d %H:%M:%S')}")
            print("NOTE: Please return within 7 days to avoid a fine of Rs 50.")

        else:
            print(f"\nSorry, all copies of '{book_name}' are currently out/borrowed.")

    def return_book(self, user_name, book_name):
        # We check if the book is in issued_books AND if this specific user has it
        if book_name in self.issued_books and user_name in self.issued_books[book_name]:

            issue_date = self.issued_books[book_name][user_name]
            return_date = datetime.datetime.now()

            # Calculate the difference in time
            delta = return_date - issue_date
            days_passed = delta.days

            print(f"\nProcessing return for: {book_name}")
            print(f"Returned by: {user_name}")
            print(f"Days kept: {days_passed}")
```



```python
            # Calculate Fine (Rs 50 if kept for more than 7 days)
            fine = 0
            if days_passed > 7:
                print("Status: Overdue! (Limit is 7 days)")
                fine = 50
            else:
                print("Status: Returned on time.")

            print(f"Fine to Pay: Rs {fine}")

            # Remove the user from the issued record
            del self.issued_books[book_name][user_name]

            # If no one else has borrowed this book, we can remove the empty dictionary key
            if not self.issued_books[book_name]:
                del self.issued_books[book_name]

            # Increase the stock back by 1
            self.books[book_name] += 1
            print("Stock updated. Thank you!")

        else:
            print(f"\nError: No record found for {user_name} having borrowed '{book_name}'.")

    def add_book(self, book_name, quantity):
        # If book already exists, just add the new stock to old stock
        if book_name in self.books:
            self.books[book_name] += quantity
        else:
            # If it's a new book, create a new entry
            self.books[book_name] = quantity

        print(f"\nAdded {quantity} sets of '{book_name}'. Total available: {self.books[book_name]}")
```



```
# Main execution block
if __name__ == "__main__":
    # Pre-filling some data for testing
    initial_stock = {
        "Python Programming": 5,
        "Harry Potter": 3,
        "Cengage": 10,
        "Calculus": 2
    }

    my_library = Library(initial_stock)

    print("Welcome to the School Library System")

    while True:
        print("\n=== LIBRARY MENU ===")
        print("1. Display Available Books")
        print("2. Issue a Book")
        print("3. Return a Book")
        print("4. Add New Stock")
        print("5. Exit")

        choice = input("Enter choice (1-5): ")

        if choice == "1":
            my_library.display_available_books()

        elif choice == "2":
            my_library.display_available_books()
            book = input("Enter Book Name: ")
            user = input("Enter Student Name: ")
            my_library.lend_book(user, book)

        elif choice == "3":
            # We need the user name to ensure we return the right person's copy
            book = input("Enter Book Name to return: ")
            user = input("Enter Student Name: ")
```



```python
        if choice == "1":
            my_library.display_available_books()

        elif choice == "2":
            my_library.display_available_books()
            book = input("Enter Book Name: ")
            user = input("Enter Student Name: ")
            my_library.lend_book(user, book)

        elif choice == "3":
            # We need the user name to ensure we return the right person's copy
            book = input("Enter Book Name to return: ")
            user = input("Enter Student Name: ")
            my_library.return_book(user, book)

        elif choice == "4":
            book = input("Enter Book Name: ")
            try:
                qty = int(input("Enter Quantity to add: "))
                my_library.add_book(book, qty)
            except ValueError:
                print("Invalid input. Quantity must be a number.")

        elif choice == "5":
            print("Exiting Library System. Have a nice day!")
            break

        else:
            print("Invalid selection. Please try again.")
```

11. Testing Approach

I tested the project using the "Black Box" testing method (testing inputs and outputs without changing code manually every time).

- **Test Case 1 (Valid Issue):** Tried to issue a book that was in stock. -> *Result: Success.*
- **Test Case 2 (Out of Stock):** Tried to issue a book with 0 quantity. -> *Result: System showed "Sorry, out of stock".*
- **Test Case 3 (Duplicate Issue):** Tried to issue the same book to the same student twice. -> *Result: System blocked the transaction.*

- **Test Case 4 (Late Return):** Temporarily changed system time logic to simulate a late return. -> *Result: Fine was calculated correctly.*

12. Challenges Faced

- **Date Logic:** Initially, I struggled with subtracting dates. I learned that datetime objects can be subtracted directly to get a 'difference' object.
- **Tracking Multiple Copies:** Earlier, my code only tracked if a book existed or not. I had to rewrite the logic to use a dictionary to track *how many* copies were left.
- **User Input Errors:** The program would crash if I entered a letter when it asked for a quantity number. I fixed this by using try-except blocks.

13. Learnings & Key Takeaways

Working on this project helped me understand:

1. **OOP Concepts:** How to actually use Classes and Objects in a real scenario, not just theory.
2. **Data Management:** How important efficient data structures (like dictionaries) are for performance.
3. **Logic Building:** How to break down a complex problem (like library management) into small, solvable functions.

14. Future Enhancements

If given more time, I would like to add:

1. **SQL Connectivity:** To store records permanently in a MySQL database so data isn't lost when the program closes.
2. **Student Login:** A separate menu for students to reserve books from home.
3. **Graphical Interface (GUI):** Using Tkinter to make it look like a real Windows app with buttons.

15. References

1. Python Documentation (docs.python.org)
2. Computer Science with Python - Textbook by Sumita Arora
3. Class Notes and Teacher's Guidance