LIBRARY MANAGEMENT SYSTEM APPLICATION

MAJOR PROJECT

- Institute Name: GH Patel College Of Engineering, Vallabh Vidyanagar, Anand
- Submitted to: MyJobGrow
- Submitted By: Meenakshi T.S

- Date Of Submission: 2/OCT/2025

Deployment link: https://github.com/nul-lhypothesis/Library-Management-System

ACKNOWLEDGEMENT

I would like to express my gratitude to **MyJobGrow** and my online internship tutors for their constant guidance, support, and the valuable resources they provided throughout this project. Their mentorship helped me understand programming concepts and apply them practically while developing this Library Management System. I am also thankful to my college and faculty for encouraging me to explore new technologies as part of my learning journey. As a first-year Computer Engineering student, this project was my way of learning how logic turns into a working system - step by step, one error at a time.

TABLE OF CONTENTS

INTRODUCTION TO PYTHON

Python is one of the most popular and powerful programming languages used in today's world. It is an interpreted, high-level, and general-purpose language that emphasizes readability and simplicity. Python was created by **Guido van Rossum** and first released in **1991**. The language was designed with the philosophy of making code easy to read and write, allowing programmers to express their ideas in fewer lines of code compared to other languages like C++ or Java.

Python supports multiple programming paradigms, including **procedural, object-oriented, and functional programming**, making it highly flexible for a wide range of applications. Its syntax is clean and straightforward, which helps beginners quickly understand programming concepts. This is one of the main reasons Python is widely taught in schools and colleges as a first programming language.

FEATURES OF PYTHON

Simple and easy to learn

Python’s syntax is clear and similar to english, making it very easy for beginners to learn and use.

Interpreted language

Python programs do not require compilation. The interpreter reads and executes the code line by line, making debugging simpler.

Object-oriented

Python supports object oriented programming concepts like classes, inheritance, polymorphism, which help in organizing and reusing code efficiently.

Extensive library support

Provides a large number of built-in libraries and modules that make it easy to perform various tasks such as file handling, data analysis, and web development.

Portable and open source

Python is available for free and can run on various platforms such as windows, macOS, linux without modification.

Dynamic Typing & large community support

In python we don’t need to declare the type of variable as type is automatically determined at run time and it also has a huge active global community that contributes to continuos development

APPLICATIONS OF PYTHON

Python is a versatile language used in almost every field of technology. Some of its major applications include:

- **Web Development:** Frameworks like Django and Flask are used to create powerful web applications.
- **Data Science and Machine Learning:** Libraries such as NumPy, Pandas, TensorFlow, and Scikit-learn make Python a leading language for data analysis and AI development.
- **Automation and Scripting:** Python is used for automating repetitive tasks such as file organization, email handling, and testing.
- **Software Development:** It is used to build desktop and mobile applications.
- **Game Development:** Libraries like Pygame help in creating simple 2D games.
- **Cybersecurity and Ethical Hacking:** Python scripts are used in security testing and penetration testing tools.

Python continues to grow in popularity due to its simplicity and wide range of applications. It bridges the gap between academic learning and real-world problem solving.

INTRODUCTION TO THE PROJECT

- The **Library Management System (LMS)** is a simple console-based Python project developed to efficiently manage a collection of books in a library. It allows users to display, issue, return, add, and sort books. This project also categorizes books according to their genres, helping users browse the collection more easily.
- The main purpose of this project is to make library operations easier for both users and administrators. Instead of maintaining paper records, all details such as book titles, borrower names, issue dates, and availability statuses are stored and managed digitally. This reduces errors, saves time, and ensures quick access to information.
- The project is implemented using the **Python programming language**, which provides simplicity and flexibility through features like file handling, data structures, and object-oriented programming. The system reads book data from a text file, displays it neatly, and updates it dynamically when books are issued or added. This project helped me strengthen my understanding of basic programming concepts like loops, conditionals, file I/O, and functions. It also gave me practical exposure to implementing algorithms such as **merge sort** for sorting book titles and using logic to group books under specific **genres**. Overall, this project acts as a stepping stone toward building more advanced data-driven applications in the future.

CONCEPTS USED IN THE PROJECT

This Library Management System integrates multiple Python programming concepts to create a functional and efficient application:

1. **File Handling**

The system reads book data from a text file (list_of_books.txt) and updates it when new books are added. This demonstrates Python's ability to work with external files using file operations like open(), readlines(), and write().

2. **Dictionaries and Lists**

All book details including title, lender name, issue date, and status are stored in dictionaries with book ID as the key, enabling efficient data retrieval. Lists are utilized for managing book collections during sorting operations and temporary data storage.

3. **Data Structures and Algorithms**

The implementation incorporates dictionaries for O(1) time complexity book lookup, lists for collection management, and a tree-like hierarchical structure for genre categorization. The merge sort algorithm provides efficient O(n log n) sorting of book titles alphabetically.

4. **Object-Oriented Programming**

The program follows a class-based structure using the LMS class, where each operation such as issuing or returning books is implemented as separate methods, enhancing code modularity and maintainability.

5. **Control Structures**

Conditional statements (if, elif, else) manage program flow based on user input, while loops (for, while) facilitate repetitive operations like displaying book collections and processing sorted data.

6. **String Operations and Validation**

The system validates user inputs by checking book ID formats, title lengths, and empty entries. String methods like .lower() and .strip() ensure consistent data processing and genre classification.

7. **Error Handling**

Try-except blocks are implemented to handle file operations and user input errors gracefully, preventing program crashes and ensuring robust system performance.

SOURCE CODE (screenshots)

major project(LMS).py > ...

```python
import datetime

class LMS:
    def __init__(self, list_of_books, library_name):
        self.list_of_books = list_of_books
        self.library_name = library_name
        self.books_dict = {}
        id_count = 101
        try:
            with open(self.list_of_books, "r") as bk:
                content = bk.readlines()
                for line in content:
                    line = line.strip()
                    if line:
                        self.books_dict[str(id_count)] = {
                            "books_title": line,
                            "lender_name": "",
                            "Issue_date": "",
                            "Status": "Available"
                        }
                        id_count += 1
        except FileNotFoundError:
            print("Book list file not found. Make sure it exists in the folder.")

    # Display all books
    def display_books(self):
        print("\n---------------------------------------- LIST OF BOOKS ----------------------------------------")
        print("Book ID\t\tTitle")
        print("-----------------------------------------------------------------------------------------------")
        for key, value in self.books_dict.items():
            print(f"{key}\t\t{value['books_title']} - [{value['Status']}]")

```

⊗ 0 ⚠ 1 OVR Ln 227, Col 1 Spaces: 4 UTF-8

```
32
33
34      def Issue_books(self):
35          book_id = input("Enter Book ID to issue: ").strip()
36          if book_id in self.books_dict.keys():
37              if self.books_dict[book_id]["Status"] == "Available":
38                  lender_name = input("Enter your name: ").strip()
39                  self.books_dict[book_id]["lender_name"] = lender_name
40                  self.books_dict[book_id]["Issue_date"] = datetime.date.today().strftime("%d-%m-%Y")
41                  self.books_dict[book_id]["Status"] = "Issued"
42                  print(f"\nBook '{self.books_dict[book_id]['books_title']}' has been issued to {lender_name}.")
43              else:
44                  print(f"\nThis book is already issued to {self.books_dict[book_id]['lender_name']}.")
45          else:
46              print("Book ID not found. Please check again.")
47
48      def add_books(self):
49          new_books = input("Enter book title: ").strip()
50          if new_books == "":
51              print("Book title cannot be empty!")
52              return self.add_books()
53          elif len(new_books) > 50:
54              print("BOOK TITLE TOO LONG! Keep it under 50 characters.")
55              return self.add_books()
56          else:
57              with open(self.list_of_books, "a") as bk:
58                  bk.write(f"{new_books}\n")
59              new_id = str(int(max(self.books_dict)) + 1)
60              self.books_dict[new_id] = {
```

```
            "books_title": new_books,
            "lender_name": "",
            "Issue_date": "",
            "Status": "Available"
        }
        print(f"Book '{new_books}' added successfully!")

    def return_books(self):
        book_id = input("Enter Book ID to return: ").strip()
        if book_id in self.books_dict.keys():
            if self.books_dict[book_id]["Status"] == "Issued":
                self.books_dict[book_id]["lender_name"] = ""
                self.books_dict[book_id]["Issue_date"] = ""
                self.books_dict[book_id]["Status"] = "Available"
                print(f"\nBook '{self.books_dict[book_id]['books_title']}' returned successfully!")
            else:
                print("That book wasn't issued.")
        else:
            print("Invalid Book ID!")

    def merge_sort(self, books):
        if len(books) > 1:
            mid = len(books)//2
            left = books[:mid]
            right = books[mid:]

            self.merge_sort(left)
            self.merge_sort(right)

```

```
90                  i = j = k = 0
91                  while i < len(left) and j < len(right):
92                      if left[i][1].lower() < right[j][1].lower():
93                          books[k] = left[i]
94                          i += 1
95                      else:
96                          books[k] = right[j]
97                          j += 1
98                      k += 1
99                  while i < len(left):
100                     books[k] = left[i]
101                     i += 1
102                     k += 1
103                 while j < len(right):
104                     books[k] = right[j]
105                     j += 1
106                     k += 1
107         return books
108
109     def merge_sort_books(self):
110         books = [(key, value["books_title"], value["Status"]) for key, value in self.books_dict.items()]
111         sorted_books = self.merge_sort(books)
112
113         print("\n---------------------------------- SORTED BOOK LIST ----------------------------------")
114         print("Book ID\t\tTitle")
115         print("------------------------------------------------------------------------------------------")
116         for b in sorted_books:
117             print(f"{b[0]}\t\t{b[1]} - [{b[2]}]")
118
```

```
    def get_genre(self, title):
        title = title.lower()
        if any(word in title for word in ["gone girl", "girl on the train", "da vinci", "silent patient", "sharp objects", "big little lies", "couple next door"]):
            return "Mystery / Thriller"
        elif any(word in title for word in ["hobbit", "harry potter", "game of thrones", "witcher", "eragon", "percy jackson", "narnia"]):
            return "Fantasy / Adventure"
        elif any(word in title for word in ["pride and prejudice", "gatsby", "mockingbird", "book thief", "les mis", "anna karenina", "war and peace"]):
            return "Historical / Classic"
        elif any(word in title for word in ["1984", "brave new world", "fahrenheit", "maze runner", "dune", "left hand of darkness", "neuromancer"]):
            return "Sci-Fi / Dystopian"
        elif any(word in title for word in ["meluha", "nagas", "vayuputras", "circe", "norse mythology", "achilles", "iliad"]):
            return "Mythology / Epic"
        elif any(word in title for word in ["dracula", "frankenstein", "shining", "haunting", "interview with the vampire", "it ", "woman in black"]):
            return "Horror / Supernatural"
        elif any(word in title for word in ["bossypants", "good omens", "catch-22", "hitchhiker", "rosie project", "yes please", "me talk pretty"]):
            return "General / Others"
        else:
            return "General / Others"

    def browse_by_genre(self):
        genres = [
            "General / Others",
            "Fantasy / Adventure",
            "Historical / Classic",
            "Sci-Fi / Dystopian",
            "Mythology / Epic",
            "Horror / Supernatural",
            "Mystery / Thriller"
        ]
```

```
149         print("\n-------------------- AVAILABLE GENRES --------------------")
150         for i, g in enumerate(genres, start=1):
151             print(f"{i}. {g}")
152 
153         try:
154             choice = int(input("\nEnter the number of the genre you want to browse: "))
155             if 1 <= choice <= len(genres):
156                 selected = genres[choice - 1]
157                 print(f"\n-------------------- BOOKS IN {selected.upper()} --------------------")
158                 print("Book ID\t\tTitle")
159                 print("------------------------------------------------------------")
160 
161                 found = False
162                 for key, value in self.books_dict.items():
163                     if self.get_genre(value["books_title"]) == selected:
164                         print(f"{key}\t\t{value['books_title']} - [{value['Status']}]")
165                         found = True
166 
167                 if not found:
168                     print("No books found in this genre.")
169             else:
170                 print("Invalid choice. Try again.")
171         except ValueError:
172             print("Please enter a valid number.")
173 
174 
175 
```

```python
myLMS = LMS("list_of_books.txt", "Python's")
press_key_list = {
    "D": "Display Books",
    "I": "Issue Books",
    "A": "Add Books",
    "R": "Return Books",
    "S": "Sort Books (A-Z)",
    "G": "Browse Books by Genre",
    "Q": "Quit"
}

key_press = ""
while key_press != "q":
    print(f"\n---------------------- Welcome to {myLMS.library_name} Library Management System ----------------------\n")
    for key, value in press_key_list.items():
        print("Press", key, "To", value)

    key_press = input("\nPress key: ").lower()

    if key_press == "d":
        print("\nCurrent Selection : Display Books\n")
        myLMS.display_books()
    elif key_press == "i":
        print("\nCurrent Selection : Issue Books\n")
        myLMS.Issue_books()
    elif key_press == "a":
        print("\nCurrent Selection : Add Books\n")
        myLMS.add_books()
    elif key_press == "r":
        print("\nCurrent Selection : Return Books\n")
```

```
192
193         key_press = input("\nPress key: ").lower()
194
195         if key_press == "d":
196             print("\nCurrent Selection : Display Books\n")
197             myLMS.display_books()
198         elif key_press == "i":
199             print("\nCurrent Selection : Issue Books\n")
200             myLMS.Issue_books()
201         elif key_press == "a":
202             print("\nCurrent Selection : Add Books\n")
203             myLMS.add_books()
204         elif key_press == "r":
205             print("\nCurrent Selection : Return Books\n")
206             myLMS.return_books()
207         elif key_press == "s":
208             print("\nCurrent Selection : Sort Books\n")
209             myLMS.merge_sort_books()
210         elif key_press == "g":
211             print("\nCurrent Selection : Browse Books by Genre\n")
212             myLMS.browse_by_genre()
213         elif key_press == "q":
214             print("Exiting Library System... Goodbye!")
215             break
216         else:
217             print("Invalid input! Try again.")
218
219     except Exception as e:
220         print("Something went wrong. Please check your input or file.")
221
```

WORKFLOW OF THE CODE

The workflow of the Library Management System follows a clear, step-by-step process:

1. **Program Initialization**
   When the program starts, the LMS class is created and initialized with the library name and book list file. It automatically loads all the books into a dictionary from the file.
2. **Main Menu Display**
   The program displays a main menu with options to **Display, Issue, Add, Return, Sort, Browse by Genre, or Quit**. The user can enter a letter corresponding to the desired action.
3. **Display Books**
   The display_books() method shows all available books with their IDs, titles, and current status (Available or Issued).
4. **Issue a Book**
   When a user selects “Issue Books,” they enter the Book ID and their name. The program marks the book as “Issued” and stores the lender name and issue date.

5. **Add a Book**
   Using the add_books() method, new books can be added. The system validates the input (title length, empty field) and updates both the text file and dictionary.

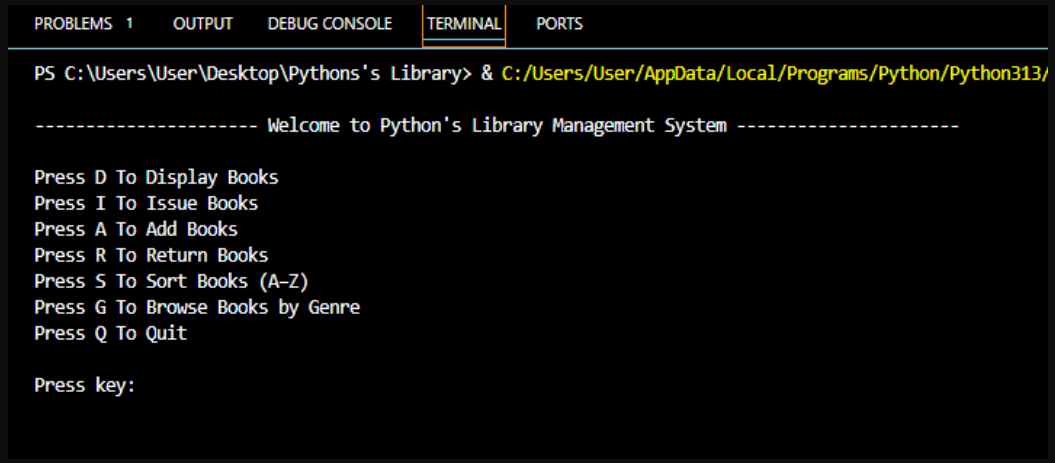
6. **Return a Book**
   The return_books() method changes the book's status back to "Available" and clears the lender's details.

7. **Sorting Books (A–Z)**
   When "Sort Books" is selected, the system applies the **merge sort algorithm** to arrange book titles alphabetically.

8. **Browse by Genre**
   The program automatically detects a book's genre (Mystery, Fantasy, Horror, etc.) based on keywords in its title. Users can view books grouped by their categories.

9. **Exit**
   When the user presses **Q**, the program ends gracefully with a closing message.

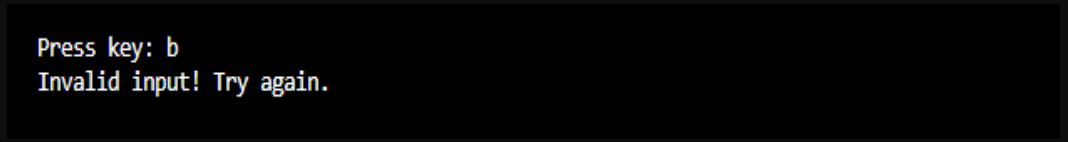
OUTPUT EXPLANATION

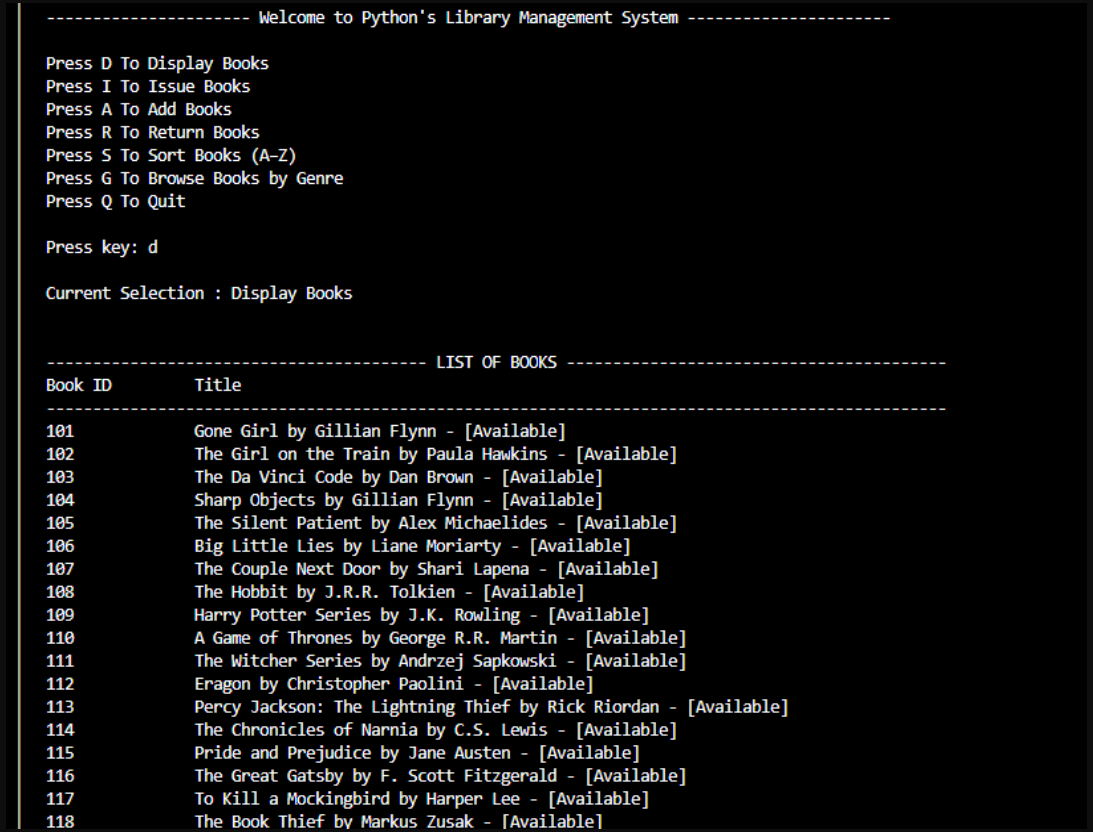


1. Main Menu:

- When the program runs, it loads the list of books from the text file and displays the main menu.
- All options are shown clearly — Display, Issue, Add, Return, Sort, Browse by Genre, and Quit.
- The user just has to type the letter of the option to perform that action.
- This shows the starting interface of the Library Management System.

When the user enters any input other than the specified keys (D, I, A, R, S, G, or Q — in either uppercase or lowercase), the system displays an “Invalid input” message.





2. Display Books :

This output shows all the books that are currently present in the library system.

Each entry displays the book ID, Title, and the Status of the book: whether it’s available or issued.

It helps users or librarians quickly check the full list of books stored in the system.

This is also the default unsorted view of the library before applying any sorting function.

```
---------- Welcome to Python's Library Management System ----------

Press D To Display Books
Press I To Issue Books
Press A To Add Books
Press R To Return Books
Press S To Sort Books (A-Z)
Press G To Browse Books by Genre
Press Q To Quit

Press key: i

Current Selection : Issue Books

Enter Book ID to issue: 101
Enter your name: Meenakshi T.S

Book 'Gone Girl by Gillian Flynn' has been issued to Meenakshi T.S.
```

3. Issue Books:

In this output, the user selects the option to issue a book and enters the Book ID.
The program asks for the lender's name and updates the book's status from *Available* to *Issued*.
It also stores the issue date automatically using the system's current date.
A confirmation message is then displayed showing that the book has been issued successfully.

```
Current Selection : Display Books


---------------------------------------- LIST OF BOOKS ----------------------------------------
Book ID         Title
-----------------------------------------------------------------------------------------------
101             Gone Girl by Gillian Flynn - [Issued]
102             The Girl on the Train by Paula Hawkins - [Available]
103             The Da Vinci Code by Dan Brown - [Available]
104             Sharp Objects by Gillian Flynn - [Available]
```

```
---------------------- Welcome to Python's Library Management System ----------------------

Press D To Display Books
Press I To Issue Books
Press A To Add Books
Press R To Return Books
Press S To Sort Books (A-Z)
Press G To Browse Books by Genre
Press Q To Quit

Press key: i

Current Selection : Issue Books

Enter Book ID to issue: 101

This book is already issued to Meenakshi T.S.
```

- The borrowed book is then updated from available to issued in the book list.
- The 2nd output appears when a user tries to issue a book that has already been issued to someone else.
- The system checks the book's current status before allowing it to be issued again.
- If the book is not available, it displays a message showing the name of the person who already has it.
- This helps prevent duplicate issuing and maintains proper tracking of borrowed books.

```
---------------------- Welcome to Python's Library Management System ----------------------

Press D To Display Books
Press I To Issue Books
Press A To Add Books
Press R To Return Books
Press S To Sort Books (A-Z)
Press G To Browse Books by Genre
Press Q To Quit

Press key: a

Current Selection : Add Books

Enter book title: The Hundred-Year-Old Man Who Climbed Out the Window and Disappeared by Jonas Jonasson
BOOK TITLE TOO LONG! Keep it under 50 characters.
Enter book title: A Confederacy of Dunces by John Kennedy Toole
Book 'A Confederacy of Dunces by John Kennedy Toole' added successfully!
```

```
146             The Hitchhiker's Guide to the Galaxy by Douglas Adams - [Available]
147             The Rosie Project by Graeme Simsion - [Available]
148             Yes Please by Amy Poehler - [Available]
149             Me Talk Pretty One Day by David Sedaris - [Available]
150             The Shining by Stephen King - [Available]
151             A Confederacy of Dunces by John Kennedy Toole - [Available]
```

4. Add Books:

This output shows how a new book can be added to the library system.

The user types the title of the book, and once entered, it is saved to the main list.

The program checks that the title is not empty or too long before adding it.

A success message confirms that the book has been added to the library records.

```
Press key: d

Current Selection : Display Books


---------------------------------------- LIST OF BOOKS ----------------------------------------
Book ID         Title
-----------------------------------------------------------------------------------------------
101             Gone Girl by Gillian Flynn - [Issued]
```

```
Press key: r

Current Selection : Return Books

Enter Book ID to return: 101

Book 'Gone Girl by Gillian Flynn' returned successfully!
```

```

---------------------------------------- LIST OF BOOKS ----------------------------------------
Book ID         Title
-----------------------------------------------------------------------------------------------
101             Gone Girl by Gillian Flynn - [Available]
```

```
Current Selection : Return Books

Enter Book ID to return: 123
That book wasn't issued.
```

5. Return Books:

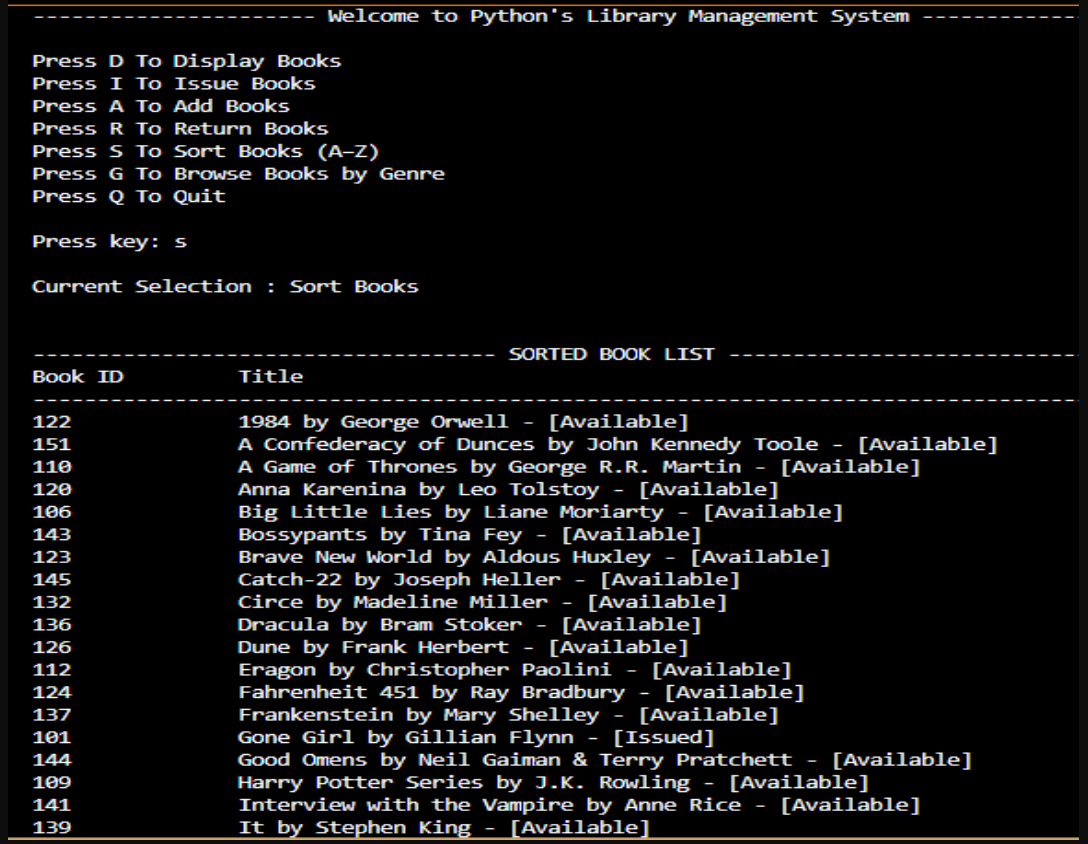
This output shows the process of returning an issued book.

The user enters the Book ID of the borrowed book, and the system checks its current status.

If the book was issued, it resets the lender's details and changes the status back to *Available*.

A confirmation message is then displayed to indicate that the book has been returned successfully.

If a book ID not issued yet is returned, then "this book wasn't issued" message is displayed



6. Sort Books (A–Z):

This output shows the list of books after applying the Merge Sort algorithm.

The titles are now arranged alphabetically from A–Z, along with their IDs and current status.

This confirms that the sorting function works correctly and organizes the data efficiently.

The Display Books slide earlier represents how the list looked before sorting.

```
Press key: g

Current Selection : Browse Books by Genre


-------------------- AVAILABLE GENRES --------------------
1. General / Others
2. Fantasy / Adventure
3. Historical / Classic
4. Sci-Fi / Dystopian
5. Mythology / Epic
6. Horror / Supernatural
7. Mystery / Thriller

Enter the number of the genre you want to browse: 1

-------------------- BOOKS IN GENERAL / OTHERS --------------------
Book ID         Title
------------------------------------------------------------
143             Bossypants by Tina Fey - [Available]
144             Good Omens by Neil Gaiman & Terry Pratchett - [Available]
145             Catch-22 by Joseph Heller - [Available]
146             The Hitchhikerâ€™s Guide to the Galaxy by Douglas Adams - [Available]
147             The Rosie Project by Graeme Simsion - [Available]
148             Yes Please by Amy Poehler - [Available]
149             Me Talk Pretty One Day by David Sedaris - [Available]
151             A Confederacy of Dunces by John Kennedy Toole - [Available]

---------------------- Welcome to Python's Library Management System ----------------------
```

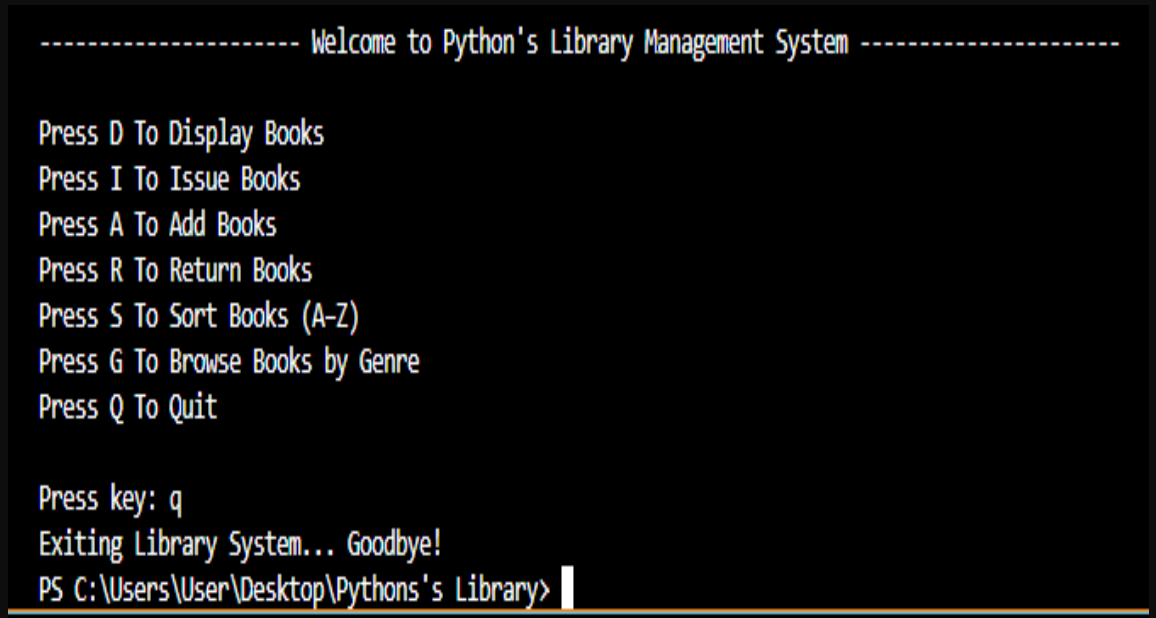
7. Browse by Genre:

This output shows how users can browse books based on different genres, similar to a real library system.

The program displays a list of available genres and asks the user to choose one by entering its number.

After selection, it shows only the books that match the chosen genre, along with their IDs and status.

This makes the system more organized and easier for users to find books of their interest.



8. Quit:

This output appears when the user chooses to exit the Library Management System by pressing **'Q'**.

The program displays a farewell message — *"Exiting Library System... Goodbye!"* — confirming that the system has closed successfully.

It signifies the end of the program's execution and returns the user to the command prompt.

CONCLUSION

Working on this Library Management System showed me how different programming concepts come together to form something that actually functions beyond the console. Through this project, I got to see how data structures, file handling, and algorithms connect in a real program instead of just being separate topics from class.

Using Python made it easier to experiment with logic and figure out how small errors can completely change the output. The process of writing, testing, and fixing code helped me understand not just how a program runs, but also how to think through it when it doesn't. Implementing features like sorting, adding, and browsing books gave me a better sense of how programs handle data efficiently while staying readable.

Overall, this project helped me see programming as a problem-solving process rather than just syntax. It's a small system, but it reflects how structure, logic, and a bit of patience can turn simple lines of code into something practical.

BIBLIOGRAPHY

- Python Official Documentation – https://docs.python.org/3/
- W3Schools Python Tutorials – https://www.w3schools.com/python/
- GeeksforGeeks Python Programming – https://www.geeksforgeeks.org/python-programming-language/
- Stack Overflow – Community discussions and code references
- MyJobGrow Online Internship Resources
- College Notes and Python Materials