TOPIC:

Library Management System

Name: Riyakrishna

Branch: Health Informatics

Registration Number:
25BHI10012

Subject: PYTHON ESSENTIALS

Introduction

A library plays a vital role in providing knowledge resources to students, teachers, and researchers. However, managing a large number of books, tracking issued books, and maintaining student records manually can be time-consuming and error-prone. To address these challenges, this project presents a Library Management System, a simple yet effective console-based application designed to streamline and automate basic library operations.

The system enables users to add student records, view available books, and issue books to registered students. It stores book information, student details, and issued book records in an organized manner using separate modules, making the project easy to understand, maintain, and expand. This modular structure ensures that each task—such as managing books, students, or issue histories—is handled by a specific component, improving the overall clarity and usability of the program.

This project demonstrates the practical application of Python programming concepts, including classes, file organization, modularization, data management, and user interaction. It serves as a simple yet effective example of how software can improve everyday tasks within an educational environment.

Problem Statement

Managing a library manually is often inefficient and prone to errors. Traditional methods such as paper records or simple logbooks make it difficult to track book availability, maintain accurate student records, and monitor issued or returned books. This can lead to problems such as misplaced books, duplicate entries, incorrect issue tracking, and delays in verifying student information.

There is a need for a simple, automated system that can store book details, manage student registrations, and record the issuing of books accurately and quickly. A digital solution will help reduce manual effort, minimize errors, and provide a clear overview of library operations at any time.

Functional requirements

◆ 1. **Student Management**

a) Add New Student

The system must allow the librarian to register a new student by entering:

- Full Name
- Class
- Student ID
- Phone Number

b) Validate Student Information

The system must validate that the entered student ID and phone number are numeric.

c) Store Student Records

The system must store student details in a structured list for later use.

◆ **2. Book Management**

a) Display Available Books

The system must show all books with details such as:

- Title
- Author
- Book ID
- Genre
- Status (Available / Issued)

b) Identify Book by Book ID

The system must allow users to search/select a book using its unique Book ID.

c) Check Book Availability

The system must verify whether a book is available before issuing it.

◆ 3. Issue Management

a) Issue a Book to a Student

The system must allow issuing a book only if:

- The book exists
- The book is available
- The student exists in the system

b) Validate Student ID

The system must ensure the entered student ID belongs to a registered student.

c) Issue Details

For every issued book, the system must store:

- Student ID
- Book ID
- Issue Date
- Return Date
- Status (Issued)

d) Prevent Duplicate Issuing

The system must prevent issuing a book that is already marked as "Issued".

◆ 4. Status & Reporting

a) View All Books Status

The system must display the current status (Available/Issued) of each book.

b) View Student Records

The system must display all registered students.

c) View Issued Book Records

The system must show a list of all books currently issued with issue details.

◆ 5. User Interaction

- Menu Navigation

The system must provide a menu with the following options:

- Add Student
- Issue Book
- Status Details
- Exit

c) Input Validation

The system must handle invalid inputs (e.g., wrong IDs, non-numeric choices) and prompt the user accordingly.

Exit Application

The system must allow the user to safely exit the program.

Non-functional requirements

1. Usability Requirements

- Easy to Use

The system should be simple and intuitive for librarians or students with basic computer knowledge.

- Clear Menu Navigation

Options must be clearly labeled, making it easy for users to navigate between operations.

2. Performance Requirements

- Fast Execution

The system should respond to user actions (adding students, issuing books, etc.) within 1–2 seconds.

- Efficient Data Processing

The system should handle student and book records efficiently without noticeable delay.

3. Reliability Requirements

- Consistent Output

The system must produce accurate results for all operations (e.g., issuing, listing books, validating inputs).

- Error Handling

The system must handle invalid inputs (wrong IDs, non-numeric entries, unavailable books) gracefully without crashing.

4. Maintainability Requirements

- Modular Structure

The system should be divided into multiple modules (book management, student management, issue management) to simplify updates.

- Code Readability

Code should be clear, properly indented, and well-commented for easy future modifications.

5. Portability Requirements

- Platform Independent

The system should run on any operating system that supports Python (Windows, Linux, MacOS).

- No External Dependencies

The system must not require heavy external libraries and should run using only the Python standard library.

6. Security Requirements

- User Input Validation

All user inputs must be validated to prevent errors or invalid data from being stored.

- Data Integrity

The system must ensure that no book is issued to an unregistered student and no unavailable book can be issued.

7. Scalability Requirements

- Data Expansion

The system should support adding more students, books, and issue records without degrading performance.

- Future Extensions

The design should allow easy addition of new features such as:

- Book return module
- Fine calculation
- File/database storage

System architecture

A. Presentation Layer (User Interface Layer)

This layer interacts directly with the user through the console.

Responsibilities:

- Display menu options
- Accept user input
- Show results, status details, and error messages
- Communicate with the logic layer

Includes:

- main.py
- menu.py (optional module for clean design)

B. Application Logic Layer (Business Logic Layer)

This layer contains the core logic of the system.
All operations are processed here.

Responsibilities:

- Add new students
- Issue books
- Validate inputs
- Update book status
- Record issue details
- Prevent invalid actions (e.g., issuing an issued book)

Includes (example modules):

- student_manager.py
- book_manager.py
- issue_manager.py
- utils.py

C. Data Layer

This layer manages the in-memory data structures.
It can later be replaced with files or a database.

Responsibilities:

- Store book records
- Store student records
- Store issue transactions
- Provide data access to managers

Includes:

- data_store.py
- books.json / students.json (optional future extension)

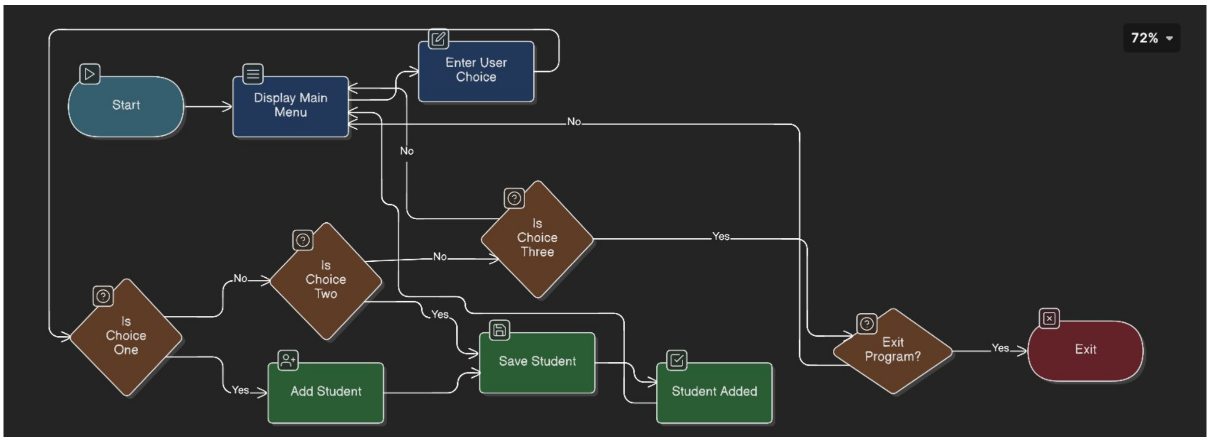
🔁 2. System Workflow

1. User selects an option from the menu
2. The request goes to the Logic Layer
3. Logic Layer validates the request
4. Data Layer provides or updates data
5. Updated information is sent back to the UI layer
6. UI displays the result to the user

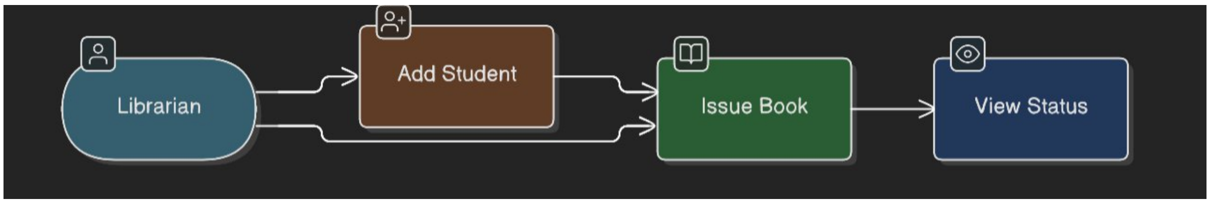
System architecture

Design Diagrams

Workflow diagram



Case diagram



Design Decisions & Rationale1.

Modular Architecture

Design Decision:

The system was split into multiple modules such as:

- Student Manager
- Book Manager
- Issue Manager
- Data Storage Module
- Main Menu Controller

Rationale:

- Improves readability and structure
- Makes debugging simpler
- Allows future expansion (returns, fines, database storage)
- Ensures separation of concerns (each module handles one task)

2. In-Memory Data Storage

Design Decision:

Books, students, and issue records are stored in Python lists and dictionaries instead of a file or database.

Rationale:

- Suitable for a small-scale academic project
- Faster development and simpler logic
- Avoids dependency on external files or database servers
- Keeps the focus on core functionality

(However, the design allows future conversion to JSON or database storage.)

3. Use of Functions and Classes

Design Decision:

The system uses modular functions (or classes if expanded) for operations such as adding students, issuing books, searching records, etc.

Rationale:

- Prevents repetition of code
- Makes the program reusable and scalable
- Enhances structure for later upgrades
- Clear separation between data models (Book, Student) and managers

◆ 4. Input Validation

Design Decision:

All user inputs (Student ID, Book ID, choices) are checked for correctness and proper format.

Rationale:

- Prevents system crashes due to invalid data
- Ensures only valid students/books are processed
- Improves reliability and user experience

◆ 5. Console-Based Interface

Design Decision:

The system uses a simple text-based interface rather than a graphical UI.

Rationale:

- Easier for beginners and academic submissions
- Avoids complexity of GUI frameworks
- Works on any system that supports Python
- Focuses on logical and functional implementation

◆ 6. Status Tracking with Flags (Available/Issued)

Design Decision:

Each book includes a Status field ("Available" or "Issued").

Rationale:

- Enables quick availability checking
- Prevents issuing already issued books
- Simplifies status reports and listing features

◆ 7. Extensible Design for Future Enhancements

Design Decision:

Modules are designed so features like **Returning books**, **Fine calculation**, **Saving records to files**, or **GUI upgrade** can be added easily.

Rationale:

- Future-proofing the project
- Allows more complex version later
- Encourages modular learning and implementation

Implementation details

1. Programming Language

The project is developed entirely in **Python**, chosen because of its:

- Simple syntax
- Rich standard library
- Ease of input/output handling
- Beginner-friendly nature

This makes Python ideal for a console-based management system.

2. Project Structure

The system is organized into multiple modules (files) to ensure separation of responsibilities:

Example structure:

main.py	— Entry point, controls menu
student_manager.py	— Add/search student functionality
book_manager.py	— Manage book list and availability
issue_manager.py	— Issue book logic and record handling
data_store.py	— Stores book, student, and issue lists
utils.py	— Input validation & helper functions

3. Data Management

The system uses **in-memory data structures**:

- **Books** → List of dictionaries
- **Students** → List of dictionaries
- **Issues** → List of dictionaries

4. Core Functional Modules

A. Student Management

Handles:

- Adding new students
- Validating student ID
- Ensuring no duplicate entries

B. Book Management

Handles:

- Displaying available books
- Searching for books via Book_ID
- Updating book status (Available → Issued)

C. Issue Management

Handles:

- Issuing books to valid students
- Checking if a book is available
- Recording Issue Date and Return Date
- Storing issuance in the issues list

User Interface (Console-Based)

The interface is purely text-based and operates through a repetitive menu loop:

1. Add Student
2. Issue Book
3. Status Details
4. Exit

6. Input Validation

To prevent crashes and ensure reliable operation, all inputs are validated.

Examples:

- Student ID → must be numeric
- Book ID → must exist in list
- Choices → must be valid menu numbers
- Book must be “Available” before issuing

Validation improves reliability and user experience.

7. Error Handling

The system includes:

- Try/except blocks for invalid number inputs
- Checks for missing book/student IDs
- Prevents issuing an already issued book

This avoids program termination due to user mistakes.

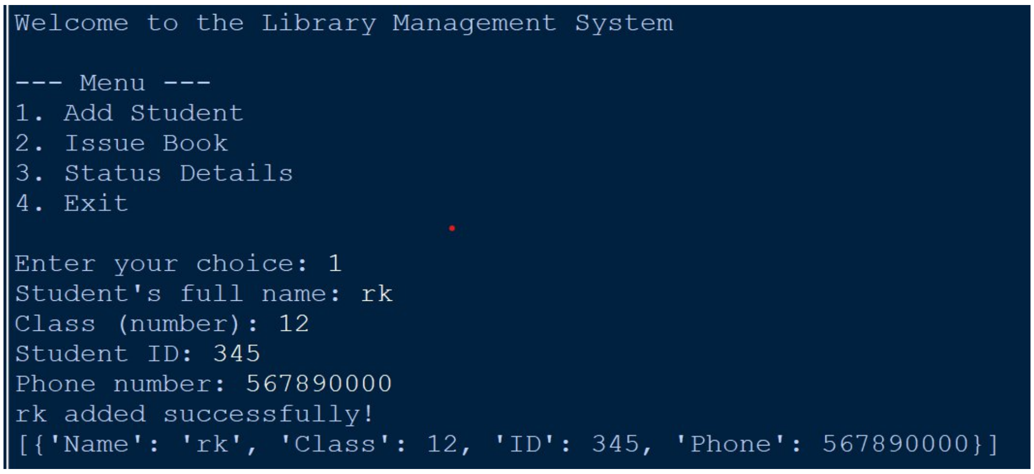
8. Status Reporting

The system displays:

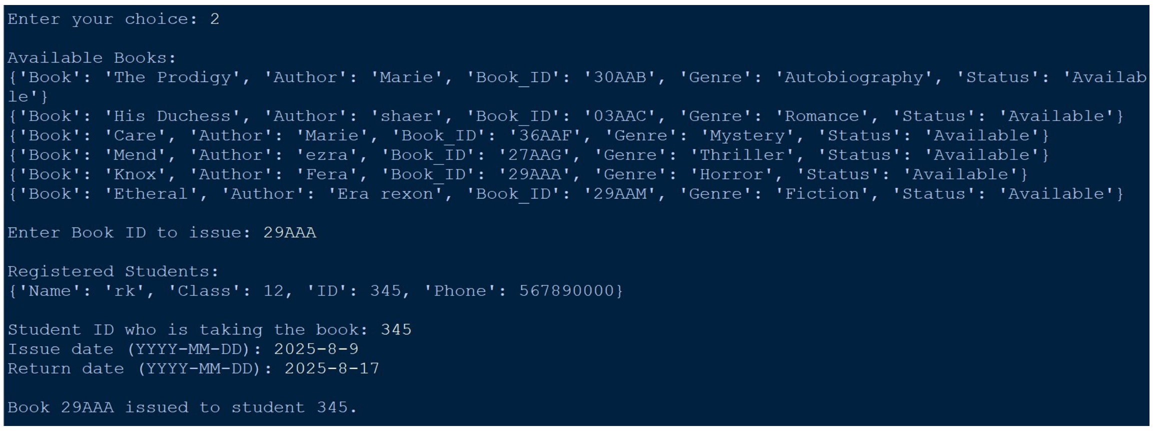
- All books and their current status
- All registered students
- All issued book records

This helps the librarian keep track of resources at all times.

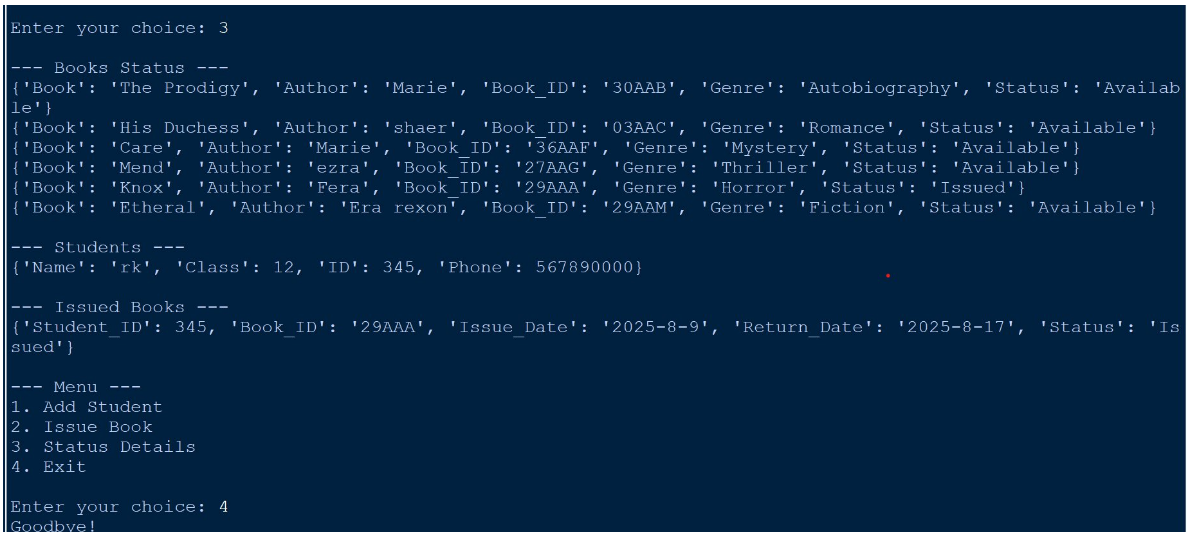
Screenshots/ outputs



```
Welcome to the Library Management System

--- Menu ---
1. Add Student
2. Issue Book
3. Status Details
4. Exit

Enter your choice: 1
Student's full name: rk
Class (number): 12
Student ID: 345
Phone number: 567890000
rk added successfully!
[{'Name': 'rk', 'Class': 12, 'ID': 345, 'Phone': 567890000}]
```



```
Enter your choice: 2

Available Books:
{'Book': 'The Prodigy', 'Author': 'Marie', 'Book_ID': '30AAB', 'Genre': 'Autobiography', 'Status': 'Availab
le'}
{'Book': 'His Duchess', 'Author': 'shaer', 'Book_ID': '03AAC', 'Genre': 'Romance', 'Status': 'Available'}
{'Book': 'Care', 'Author': 'Marie', 'Book_ID': '36AAF', 'Genre': 'Mystery', 'Status': 'Available'}
{'Book': 'Mend', 'Author': 'ezra', 'Book_ID': '27AAG', 'Genre': 'Thriller', 'Status': 'Available'}
{'Book': 'Knox', 'Author': 'Fera', 'Book_ID': '29AAA', 'Genre': 'Horror', 'Status': 'Available'}
{'Book': 'Etheral', 'Author': 'Era rexon', 'Book_ID': '29AAM', 'Genre': 'Fiction', 'Status': 'Available'}

Enter Book ID to issue: 29AAA

Registered Students:
{'Name': 'rk', 'Class': 12, 'ID': 345, 'Phone': 567890000}

Student ID who is taking the book: 345
Issue date (YYYY-MM-DD): 2025-8-9
Return date (YYYY-MM-DD): 2025-8-17

Book 29AAA issued to student 345.
```



```
Enter your choice: 3

--- Books Status ---
{'Book': 'The Prodigy', 'Author': 'Marie', 'Book_ID': '30AAB', 'Genre': 'Autobiography', 'Status': 'Availab
le'}
{'Book': 'His Duchess', 'Author': 'shaer', 'Book_ID': '03AAC', 'Genre': 'Romance', 'Status': 'Available'}
{'Book': 'Care', 'Author': 'Marie', 'Book_ID': '36AAF', 'Genre': 'Mystery', 'Status': 'Available'}
{'Book': 'Mend', 'Author': 'ezra', 'Book_ID': '27AAG', 'Genre': 'Thriller', 'Status': 'Available'}
{'Book': 'Knox', 'Author': 'Fera', 'Book_ID': '29AAA', 'Genre': 'Horror', 'Status': 'Issued'}
{'Book': 'Etheral', 'Author': 'Era rexon', 'Book_ID': '29AAM', 'Genre': 'Fiction', 'Status': 'Available'}

--- Students ---
{'Name': 'rk', 'Class': 12, 'ID': 345, 'Phone': 567890000}

--- Issued Books ---
{'Student_ID': 345, 'Book_ID': '29AAA', 'Issue_Date': '2025-8-9', 'Return_Date': '2025-8-17', 'Status': 'Is
sued'}

--- Menu ---
1. Add Student
2. Issue Book
3. Status Details
4. Exit

Enter your choice: 4
Goodbye!
```

Testing Approach

1. Testing Methodology

A. Manual Testing

Manual testing is used to verify:

- Menu navigation
- User input behavior
- Correct function execution
- Data correctness (students, books, issued records)

Each feature is tested by entering different combinations of valid and invalid inputs.

B. Functional Testing

This ensures that each function performs its intended task correctly.

Examples of functional tests:

- Adding a student with valid ID
- Trying to add the same student twice
- Issuing an available book
- Attempting to issue a book that is already issued
- Searching for a book that does not exist

Each function is tested individually ("unit-level"), then tested together ("integration-level").

C. Integration Testing

After individual modules work correctly, they are tested together to confirm smooth interaction.

Examples:

- Adding a student → issuing a book to that student
- Changing book status after issuance
- Viewing updated status in Status Details

This confirms that student, book, and issue modules work cohesively.

◆ 2. Test Data

Several test cases are created, including:

- Valid student IDs (e.g., 101, 102)
- Invalid IDs (letters, empty input)

- Valid book IDs (matching system records)
- Non-existing book IDs
- Books in both “Available” and “Issued” states

This helps evaluate normal usage, edge cases, and error handling.

◆ 3. Input Validation Testing

Tested for:

- Non-numeric student IDs
- Empty fields
- Book IDs in wrong format
- Invalid menu options (letters, numbers outside range 1–4)

Expected Result → System displays an error message and prompts again without crashing.

4. User Interface Testing

Ensures that:

- Menu displays correctly
- Returned messages are clear
- Navigation loops properly
- No unexpected program terminations occur

Because the system is console-based, UI testing focuses on clarity and flow rather than design.

5. Performance Testing (Basic)

Even though the project is small, simple checks ensure:

- Fast response time for listing books/students
- No slowdowns when adding multiple entries
- System stability during repeated operations

6. Error Handling Testing

Scenarios tested include:

- Issuing a book that is already issued
- Issuing to a non-existent student
- Selecting an invalid menu option

- Closing the system gracefully

The system should always:

- Show an informative message
- Not terminate unexpectedly

7. Regression Testing

After fixing any bug or adding a feature, previously working features are re-tested to confirm nothing broke.

Challenges

1. Handling User Input Validations

One major challenge was ensuring that all user inputs—such as student IDs, phone numbers, book IDs, and menu choices—were correctly validated.
This required:

- Checking for numeric values
- Handling empty or invalid inputs
- Preventing the system from crashing due to unexpected data

Implementing proper validation and error messages took careful design and testing.

2. Structuring the Project into Modules

Splitting the initial single file into separate modules (books, students, issues, main menu) was challenging because:

- Functions needed to communicate across files
- Data such as book lists and student records had to be shared
- Imports had to be managed properly

However, this modular approach improved clarity and maintainability.

3. Managing Book Status and Issue Records

Tracking book availability required updates to the book list and maintaining a separate issues record.
Challenges included:

- Ensuring book status changed to "Issued" at the correct time
- Preventing multiple issuance of the same book
- Ensuring issued records were stored correctly

This required careful testing and logical conditions.

4. Ensuring Smooth Menu Navigation

The menu-driven interface needed to:

- Loop properly
- Handle invalid menu choices
- Exit gracefully

Ensuring that each option returned control back to the menu without breaking the flow was initially tricky.

5. Error Handling and Program Stability

The system needed to handle errors gracefully without crashing.
Challenges included:

- Invalid menu entries
- Nonexistent book IDs
- Issuing books without registered students

Adding exception handling improved stability.

◆ 6. Keeping the Code Readable and Maintainable

As the project grew, keeping the code clean and understandable became difficult.
Steps taken included:

- Adding comments
- Following consistent naming conventions
- Separating logic into smaller functions

Learnings & Key Takeaways

1. Understanding Modular Programming

One of the major learnings was how to break a large program into smaller, manageable modules.
This included:

- Separating logic into different files
- Organizing code for books, students, and issues
- Creating functions that interact smoothly
  This improved clarity and made the project easier to maintain.

◆ **2. Improved Python Skills**

The project strengthened core Python concepts such as:

- Lists and dictionaries
- Looping and conditional statements
- Input handling
- Error control using try–except
- Working with functions and return values
  This enhanced overall confidence in coding and debugging Python programs.

◆ **3. Importance of Data Validation**

The system required strict handling of user inputs.
Key takeaways:

- Always check user-entered values
- Prevent invalid inputs from crashing the system
- Ensure data consistency (e.g., book IDs, student IDs)
  This reinforced the importance of robust validation in real-world applications.

◆ **4. Designing User-Friendly Console Interfaces**

Creating a smooth menu-driven interface taught:

- How to structure clear instructions for users
- How to manage program flow loops
- How to display data meaningfully
  This improved understanding of how users interact with CLI applications.

◆ 5. Logical Thinking & Problem Solving

Many operations, such as issuing books or updating statuses, required logical reasoning.
This helped improve:

- Flowchart thinking
- Step-by-step breakdown of tasks
- Decision-making using conditions
  This strengthened algorithmic thinking skills.

◆ 6. Handling Real-World Constraints

The project simulated real library constraints, such as:

- Preventing multiple students from issuing the same book
- Ensuring only registered students can borrow books
- Keeping accurate status tracking
  This provided insight into how software solves everyday problems.

◆ 7. Testing and Debugging Skills

While testing, many small issues were found and corrected.
Key lessons:

- Testing is essential, not optional
- Edge cases reveal hidden bugs
- Debugging builds patience and improves logi

◆ 8. Project Documentation Skills

Writing the report improved:

- Technical writing
- Organizing content
- Presenting software concepts clearly

Future enhancements

1. Add a Return Book Feature

Currently, the system only supports issuing books.
A return feature would allow:

- Updating book status back to *Available*
- Removing or updating issued book history
- Calculating late fines (optional)

◆ 2. Integrate File Storage or Database

At present, all data is stored in memory and is lost when the program closes.
Adding storage options would make the system more realistic:

Possible approaches:

- Text files
- CSV/JSON files
- SQLite database
- MySQL/PostgreSQL (for advanced version)

This will allow persistent tracking of books and students.

◆ 3. Search and Filter Options

Enhance usability by allowing:

- Search by book title
- Search by author
- Filter by genre
- Search for students by name or ID

This makes the system more dynamic and practical.

◆ 4. Improve User Interface

Upgrade from console to a graphical interface using:

- Tkinter (Python GUI)
- PyQt
- Web interface (HTML/CSS/Flask)

This will make the system easier and more appealing to use.

◆ **5. Add Login System & Roles**

Introduce login access for:

- Admin (can add books, students, issue/return)
- Librarian (limited functions)
- Students (view issued books, due dates)

This ensures security and controlled access.

◆ **6. Track Fines and Due Dates**

Enhance the issuing system by:

- Automatically calculating fines for late returns
- Generating alerts for overdue books
- Sending reminders (email/SMS in advanced versions)

◆ **7. Add Book Management Features**

Additional features such as:

- Add new books to the catalog
- Delete books
- Update book details
- Track damaged or missing books

This makes the system more aligned with real-world libraries.

◆ **8. Generate Reports**

Provide useful reports such as:

- List of all issued books
- Students with pending returns
- Daily/weekly transaction summary
- Genre-wise book count

These reports can be shown in the console or exported to a file.

◆ **9. Barcode or QR Code Integration (Advanced)**

Books and students can have barcodes that can be scanned for faster processing.
This requires:

- Barcode generation
- Barcode scanner interface

Useful for modern libraries.

◆ 10. Cloud Integration

For advanced deployment:

- Store data online
- Allow remote access
- Synchronize multiple library branches

Though optional, this can make the system enterprise-level

References

- GeeksforGeeks. *Python Programming Tutorials*.
  https://www.geeksforgeeks.org/python-programming-language/
- Notes
- Real Python. *Python Best Practices*.
  https://realpython.com/