



19th of April 2026

SPC Assignment on

Student Equipment Booking System

Module Name: Scripting and Programming Concepts

Module Code: ACCA 5036

Student Name: Andrei-Stefanita Stoica

Student ID: 2413927

Lecturer's Name: Koushik Modak

Code Link: <https://github.com/St-Stefan676/booking-python-app>

Table of Contents

Introduction	2
Main Body	2
1. Analysis	2
2. Software Design and Development	4
Design:.....	4
Implementation Decisions	10
Source Code	12
3. Testing and Debugging Report	15
3.1 Importance of Program Testing in Software Development	15
3.2 Types of Errors in Programming and Methods of Resolution	15
3.3 Test Cases and Validation Testing.....	16
4. Evaluation.....	20
4.1 Challenges Faced.....	20
4.2 Future Improvements	20
4.3 Self-Reflection	20
Conclusion	21
References.....	21
Appendices.....	22

Introduction

Bright Minds Academy is an educational institution which manages a variety of equipment resources including laptops, projectors, tablets and other learning tools. Currently, the academy relies on manual paper-based systems or spreadsheets to track equipment bookings. This approach presents several problems therefore, the computer-based solution proposed is designed to tackle them. Additionally, the application operates as a single-user console solution using data structures from the application itself (dictionaries and lists), meaning that data persists only as the program is in execution (no database required) and most importantly, no network connectivity is required to run the application making it efficient in different environments.

MainBody

1. Analysis

- Requirements Analysis

The core problems found in a paper-based system are:

- Inefficiency: Staff must manually search through paper records or scroll through large spreadsheets to find booking information
- Data integrity issues: Paper records can be lost, damaged or contain illegible handwriting. Spreadsheets do not offer input validation which leads to inconsistent data entries.
- No real-time status visibility: It can be difficult to quickly determine whether a piece of equipment is available or has already been booked for a specific date.
- Search limitations: Finding bookings by student name or equipment name requires time-consuming manual scanning.
- Risk of double-booking: Human errors may occur, meaning multiple staff members might inadvertently book the same equipment for overlapping dates, creating unnecessary circumstances.

Functional Requirements

- **Add new equipment:** The proposed computer-based solution must allow authorized staff to add new pieces of equipment to the system's inventory. Each equipment item requires a unique identifier/name (e.g., "Laptop 1", "Projector A").
- **Record bookings:** Staff must be able to record when a student borrows equipment. Each booking record must capture which piece of equipment is being borrowed, the full name of the borrowing student and the date of the booking (format: YYYY-MM-DD).
- **View booking records:** The system must display all current booking information on a clear and formatted table showing all equipment items, current booking status (available/booked) and for booked items, which student has it and on what date.
- **Simple Search:** Users must be able to search through existing booking by student name or equipment name.
- **Low-Booking Alerts:** Automatic notifications when booking fall below a set threshold.
- **User-Friendly Interface:** Simple, intuitive design suitable for staff without advanced technical skills.
- **Scalability:** Flexibility to expand the system later.

Non-Functional Requirements

In addition to functionality, the system must meet several quality criteria:

- **Reliability:** Accurate and consistent booking data at all times.
- **Performance:** Quick response when recording or retrieving booking.
- **Security:** Safe storage of data to prevent accidental loss.
- **Accessibility:** Facilitate the experience for users with visual impairments, color blindness and users who rely on assistive technologies like screen readers.

2. Software Design and Development

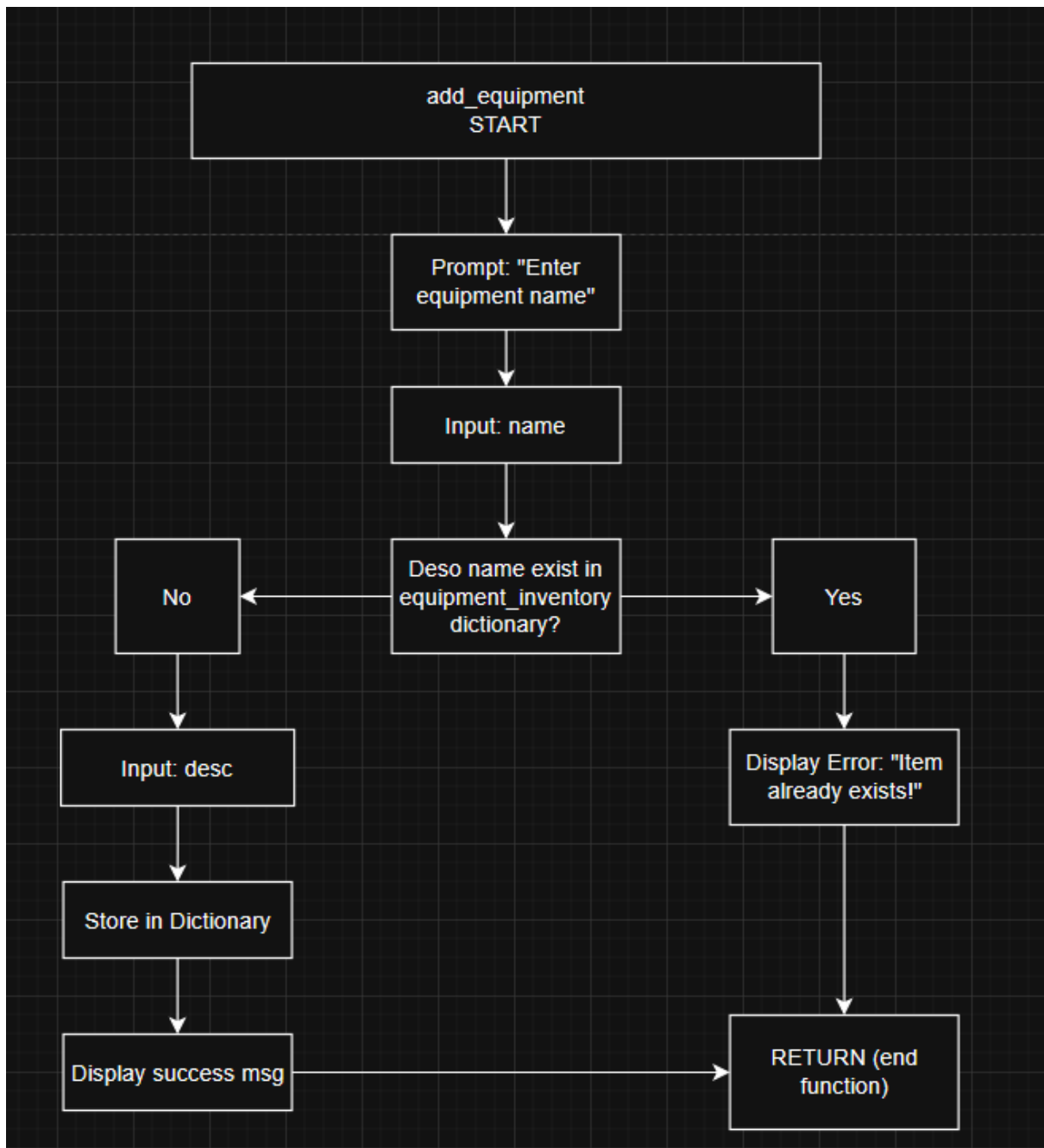
In this section of the report, I will document the design and development of the software.

Design: Representation of the algorithms of the proposed solution:

- *Flowcharts:* Main program flowchart



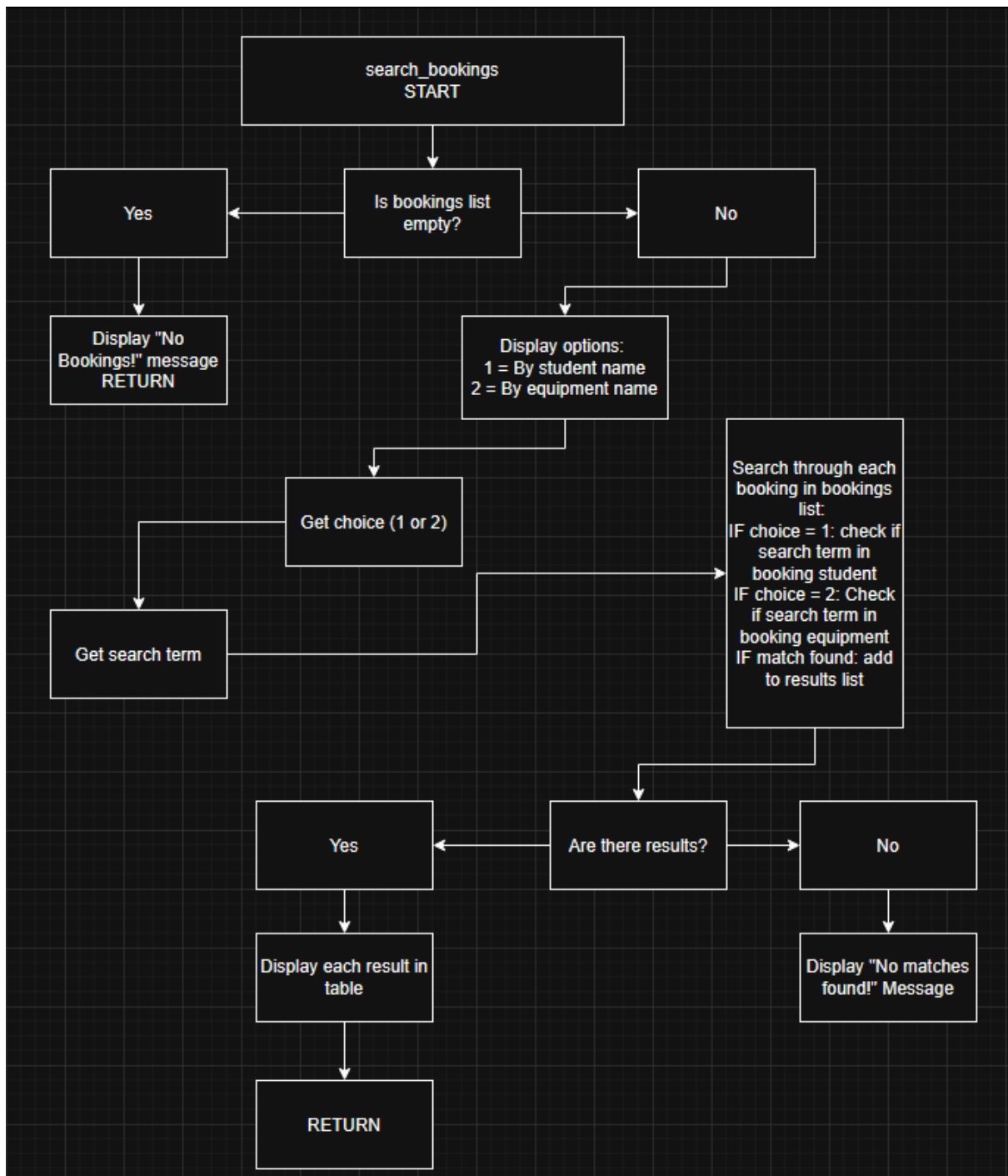
- Add equipment function flowchart



- Add booking function flowchart



- Search bookings function flowchart



- *Pseudo-code:* 1. Main program control loop pseudo-code

```

// MAIN PROGRAM LOOP
FUNCTION main():
    WHILE True:
        CALL display_menu()
        user_choice = INPUT("Enter your choice: ")
  
```

```

IF user_choice == "1":
    CALL add_equipment

ELIF user_choice == "2":
    CALL record_booking

ELIF user_choice == "3":
    CALL view_bookings

ELIF user_choice == "4":
    CALL search_bookings

ELIF user_choice == "5":
    OUTPUT("Exiting system.")
    BREAK // Stop the loop

ELSE:
    OUTPUT("Invalid option. Try again.")

```

- **2. Add equipment function pseudo-code**

```

FUNCTION add_equipment():
    name = INPUT("Enter equipment name: ")

    // Validation: Check if empty
    IF name IS EMPTY:
        OUTPUT("Error: Name cannot be empty.")
        RETURN

    // Validation: Check for duplicates
    IF name EXISTS IN equipment_inventory:
        OUTPUT("Error: Equipment already exists.")
        RETURN

    description = INPUT("Enter description: ")

    // Save to dictionary
    equipment_inventory[name] = {
        "description": description,
        "booked_by": NONE,
        "booking_date": NONE
    }
    OUTPUT("Equipment added successfully.")

```

- **3. Record booking function pseudo-code**

```

FUNCTION record_booking():

```

```

item_name = INPUT("Enter equipment name to book: ")

// Validation: Existence check
IF item_name NOT IN equipment_inventory:
    OUTPUT("Error: Equipment not found.")
    RETURN

// Availability check
IF equipment_inventory[item_name]["booked_by"] IS NOT NONE:
    OUTPUT("Error: Item is already booked.")
    RETURN

student = INPUT("Enter student name: ")
date = INPUT("Enter booking date: ")

// Update Inventory Status
equipment_inventory[item_name]["booked_by"] = student
equipment_inventory[item_name]["booking_date"] = date

// Log to History
bookings.ADD({
    "equipment": item_name,
    "student": student,
    "date": date,
    "time": CURRENT_TIMESTAMP
})
OUTPUT("Booking confirmed.")

```

- **4. View all bookings function pseudo-code**

```

FUNCTION view_bookings:
    available_count = 0
    booked_count = 0

    OUTPUT("--- EQUIPMENT STATUS ---")

    FOR EACH item IN equipment_inventory:
        status = "Available"
        IF item["booked_by"] IS NOT NONE:
            status = "Booked by " + item["booked_by"]
            booked_count = booked_count + 1
        ELSE:
            available_count = available_count + 1

        OUTPUT(item.name + ": " + status)

    OUTPUT("Total Available: " + available_count)

```

```
OUTPUT("Total Booked: " + booked_count)
```

Implementation Decisions

- ❖ *Data Structures:* The primary data structure used for this system is `equipment_inventory` which is a dictionary that maps equipment names to their details. Each key is a string representing the equipment name and each value is another dictionary containing three fields: a description of the item, the student who booked it and the booking date. The reason I chose a dictionary is because looking up equipment by name needs to be fast and dictionaries provide it. The other data structure used are `bookings` which is a list of dictionaries where each element represents one complete booking record containing the equipment name, student name and date. I chose a list because bookings are added chronologically, appending them to a list is efficient and iterating through them in order makes sense when displaying or searching history.

For simple user inputs like menu choices, names and dates I used standard strings. For counting available versus booked items during display, I used integer variables which increment within loops [1].

- ❖ *Programming Constructs:* The proposed computer-based solution is primarily formed of functions with single responsibility. The code is separated into six distinct functions: one for displaying the menu, one for adding equipment, one for recording bookings, one for viewing all bookings, one for searching bookings and one main function to operate everything. The reason each function does exactly one thing is to make the code easier to read, debug and modify. If part of the program breaks, it is easier to identify and manage.

Conditional statements like `if`, `elif` and `else` appear throughout the program for two main purposes: menu navigation and input

validation. When the user selects a menu option, an if-elif chain directs execution to the correct function. Before any data is modified, conditional checks verify that inputs are not empty, equipment names do not already exist, and equipment actually exists before booking it.

Loops serve different purposes. A while true loop continuously shows the menu until the user explicitly chooses to exit. This creates a more interactive experience for the user. For-loops iterate over the inventory dictionary items when displaying equipment lists and over the bookings list when showing or searching records.

Error handling: at the start of functions which accept user input, I placed guard clauses which check for problems such as empty strings, duplicate entries and missing equipment. What this does is it keeps the functions focused on a normal successful path and it simplifies the process without adding extra lines of if-statements to the code [2].

- ❖ *Python Modules and Libraries:* No external python modules were used in the making of this project.

No third-party libraries were used meaning this project relies solely on Python's built-in capabilities: input() and print() for console interaction, standard dictionary and list operations. String methods like .strip() and .lower() for cleaning user input [3] [4].

Source Code

```
1 equipments = [] # List of all equipment items
2 bookings = [] # List of all booking records
3
4 def add_equipment():
5     """Add a new equipment item if it doesn't already exist."""
6     name = input("Enter new equipment name (e.g., 'Laptop 1'): ").strip()
7     if not name:
8         print("Invalid name.")
9         return
10    if name in equipments:
11        print(f"{name} already exists.")
12    else:
13        equipments.append(name)
14        print(f"Added '{name}' to the system.")
15
```

This screenshot demonstrates the first function used in the program; it is the `add_equipment` function. It is simple and to the point, it prompts the user to enter an equipment name and uses `.strip()` to remove any leading/trailing whitespace. It first checks if the name is empty. If so, it prints an error and exits. It also checks whether the piece of equipment already exists in the `equipments` list. If it exists, it informs the user, otherwise it appends the name to the `equipments` list and confirms the addition.

```
16 def record_booking():
17     """Record a booking for an existing equipment item."""
18     if not equipments:
19         print("No equipment available. Add some first.")
20         return
21     print("Available equipment:", ", ".join(equipments))
22     eq_name = input("Enter equipment name: ").strip()
23     if eq_name not in equipments:
24         print(f"'{eq_name}' not found. Add it first.")
25         return
26     student = input("Enter student name: ").strip()
27     if not student:
28         print("Invalid student name.")
29         return
30     date = input("Enter date (YYYY-MM-DD): ").strip()
31     if not date:
32         print("Invalid date.")
33         return
34     # Add the booking as a dictionary
35     bookings.append({
36         'equipment': eq_name,
37         'student': student,
38         'date': date
39     })
40     print(f"Booking recorded: {eq_name} for {student} on {date}.")
```

The function demonstrated in this screenshot is the `record_booking` function. First, it checks if any equipment has been added. If the `equipments` list is empty, it asks the user to add equipment first. It displays all available equipment for easy selection. The

user must enter a valid equipment name (it must exist in the list), then it collects the student name and date (in YYYY-MM-DD format). At the same time, each input is validated to ensure it's not empty and then, if all inputs are valid, it creates a dictionary containing the equipment, student and date and appends it to the bookings list. Finally, it prints a confirmation message.

```
42 def view_bookings():
43     """View all booking records, sorted by date, showing equipment booking status across days."""
44     if not bookings:
45         print("No bookings recorded yet.")
46         return
47     # Sort bookings by date for chronological view
48     sorted_bookings = sorted(bookings, key=lambda b: b['date'])
49     print("\n=== All Booking Records (sorted by date) ===")
50     for booking in sorted_bookings:
51         print(f"{booking['date']}: {booking['equipment']} -> {booking['student']}")
52     print("=== End of records ===\n")
```

This is the `view_bookings` function. It displays all recorded bookings in chronological order. If there are no bookings, it informs the user and exists. The function uses `sorted()` with a lambda function to sort the bookings based on the 'date' key. Additionally, it prints a formatted header, then loops through each sorted booking and displays it in a readable format: YYYY-MM-DD: Equipment → Student. A footer is then printed at the end for clean output.

```
54 def view_equipment_status():
55     """View booking status per equipment, listing all booked dates."""
56     if not bookings:
57         print("No bookings recorded yet.")
58         return
59     from collections import defaultdict
60     eq_status = defaultdict(list)
61     # Group bookings by equipment
62     for booking in bookings:
63         eq_status[booking['equipment']].append(booking['date'])
64     print("\n=== Equipment Booking Status ===")
65     for eq in sorted(eq_status):
66         dates = sorted(eq_status[eq])
67         print(f"{eq}: Booked on {' '.join(dates)}")
68     print("=== End of status ===\n")
```

This function shows a summary of bookings grouped by equipment, rather than by date. It uses `defaultdict(list)` from the `collections` module to automatically create a list for each equipment. It loops through all bookings and groups the dates under their respective equipment names. It then prints each equipment (in alphabetical order) along with all the dates it was booked, sorted chronologically.

```

70 def search_bookings():
71     """Search bookings by student name or equipment name (case-insensitive partial match)."""
72     if not bookings:
73         print("No bookings recorded yet.")
74         return
75     search_term = input("Enter student name or equipment name to search: ").strip().lower()
76     if not search_term:
77         print("Invalid search term.")
78         return
79     # Find matching bookings
80     matches = [
81         b for b in bookings
82         if search_term in b['student'].lower() or search_term in b['equipment'].lower()
83     ]
84     if not matches:
85         print("No matching bookings found.")
86         return
87     print(f"\n== Search Results for '{search_term}' ==")
88     # Sort matches by date
89     sorted_matches = sorted(matches, key=lambda b: b['date'])
90     for booking in sorted_matches:
91         print(f"{booking['date']}: {booking['equipment']} -> {booking['student']}")
92     print("\n== End of results ==\n")

```

This function allows users to search for bookings using either a student name or equipment name. It accepts a search term and converts it to lowercase for case-insensitive matching. It also uses a list comprehension to filter bookings where the search term appears in either the student's name or the equipment's name, if no matches are found, it then notifies the user. Matching results are sorted by date and displayed in a formatted report. Additionally, the search is partial, meaning that if for example the user searches for "lap" it will match with "Laptop 1".

```

94 # Main menu loop
95 print("Welcome to Bright Minds Academy Equipment Booking System!")
96 while True:
97     print("\n== Main Menu ==")
98     print("1. Add new equipment")
99     print("2. Record a booking")
100    print("3. View all booking records (by date)")
101    print("4. View equipment status (by equipment)")
102    print("5. Search bookings (by student or equipment)")
103    print("6. Exit")
104
105    choice = input("Choose an option (1-6): ").strip()
106
107    if choice == '1':
108        add_equipment()
109    elif choice == '2':
110        record_booking()
111    elif choice == '3':
112        view_bookings()
113    elif choice == '4':
114        view_equipment_status()
115    elif choice == '5':
116        search_bookings()
117    elif choice == '6':
118        print("Goodbye!")
119        break
120    else:
121        print("Invalid choice. Please try again.")

```

This is the main program loop which runs the entire application. It starts by printing a welcome message. The main idea is to use an infinite while true loop to continuously display the menu until the user chooses to exit or enter the appropriate data. Any invalid input results in an error message and the menu is shown again.

3. Testing and Debugging Report

3.1 Importance of Program Testing in Software Development

Program testing is an essential component of the software development lifecycle. It serves as a systematic process to evaluate the reliability and robustness of a software application. Through rigorous testing, the developers identify defects, ensure the program behaves according to specified requirements and verify all functionalities operate as intended. In the context of the Bright Minds Academy Equipment Booking System, testing plays a vital role in confirming the input validation mechanisms function correctly, data is processed accurately and that the system successfully handles both valid and erroneous user inputs. Ultimately, thorough testing enhances software quality, reduces the risk of failures in a live environment, boosts user confidence and minimizes maintenance costs [5].

3.2 Types of Errors in Programming and Methods of Resolution

In programming, errors can generally be classified into three main categories: syntax errors, runtime errors and logical errors.

Syntax errors occur when the source code violates the grammatical rules of the programming language. Common examples include missing colons or unmatched parentheses. These errors are typically detected by the Python interpreter during execution and must be corrected before the program can run [6].

Runtime errors, also known as exceptions, happen during program execution when an operation cannot be completed. Examples include attempting to access a non-existent list index or performing operations on invalid data types. These can be mitigated through defensive programming techniques such as input validation, conditional checks and the use of exception handling constructs [7].

Logical errors are the most challenging to identify, as the program executes without crashing but produces incorrect or unintended results. These errors stem from flaws

in the underlying algorithm or logic. In this system, for example, incorrect sorting of dates or flawed search conditions would lead to logical errors. A way to fix these errors typically involves careful code review, step-by-step debugging and the development of test cases to validate program behavior against expected outcomes [8].

3.3 Test Cases and Validation Testing

Test Case 1: Add Equipment – Empty Name Validation

```
Shell >
>>> %Run 'python booking app.py'
Welcome to Bright Minds Academy Equipment Booking System!

=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 1
Enter new equipment name (e.g., 'Laptop 1'):
Invalid name.

=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6):
```

As seen in this screenshot, if by mistake, the user presses enter when prompted to enter new equipment name the system will flag it as 'Invalid name'.

Result:  Passed

Test Case 2: Add Equipment – Duplicate Equipment

```
=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 1
Enter new equipment name (e.g., 'Laptop 1'): Laptop 2
Added 'Laptop 2' to the system.

=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 1
Enter new equipment name (e.g., 'Laptop 1'): Laptop 2
Laptop 2 already exists.
```

A screenshot of the system ensuring duplicate equipment cannot be added.

Result:  Passed

Test Case 3: Record Booking – No Equipment Available

```
Shell <
>>> %Run 'python booking app.py'
Welcome to Bright Minds Academy Equipment Booking System!

=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 2
No equipment available. Add some first.

=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6):
```

Test case demonstrating how the system behaves when a user decides to record a booking with zero equipment available.

Result:  Passed

Test Case 4: Record Booking – Invalid Equipment Name

```
Shell <
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 1
Enter new equipment name (e.g., 'Laptop 1'): Laptop 3
Added 'Laptop 3' to the system.

=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 1
Enter new equipment name (e.g., 'Laptop 1'): Laptop 4
Added 'Laptop 4' to the system.

=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 2
Available equipment: Laptop 3, Laptop 4
Enter equipment name: Projector 1
'Projector 1' not found. Add it first.
```

A screenshot showcasing what happens when the user searches for a piece of equipment which is not available in the built-in database.

Result:  Passed

Test Case 5: Record Booking – Empty Student Name & Date Validation

```
Shell <
Added 'Laptop' to the system.

=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 2
Available equipment: Laptop
Enter equipment name: Laptop
Enter student name:
Invalid student name.

=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 2
Available equipment: Laptop
Enter equipment name: Laptop
Enter student name: Alice
Enter date (YYYY-MM-DD):
Invalid date.
```

This is a screenshot documenting my findings when attempting to leave the student name and date fields empty (pressing enter instead of introducing valid inputs).

Result:  Passed

Test Case 6: Search Bookings – Empty Search Term

```
=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 2
Available equipment: Projector
Enter equipment name: Projector
Enter student name: Alice Thorne
Enter date (YYYY-MM-DD): 2026-24-11
Booking recorded: Projector for Alice Thorne on 2026-24-11.

=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 5
Enter student name or equipment name to search:
Invalid search term.
```

A screenshot showcasing the system working as intended when leaving the search field empty.

Result:  Passed

Test Case 7: Search Bookings – Partial Case-Insensitive Match

```
Shell x
=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 5
Enter student name or equipment name to search: proj

=== Search Results for 'proj' ===
2026-24-11: Projector -> Alice Thorne
=== End of results ===

=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 5
Enter student name or equipment name to search: ALICE

=== Search Results for 'alice' ===
2026-24-11: Projector -> Alice Thorne
=== End of results ===
```

Testing partial and case-insensitive search. The expected output is for the system to return the appropriate booking record even when only part of the equipment name is introduced or, if for instance, the student's name is introduced in uppercase.

Result:  Passed

Test Case 8: View Functions – Empty Bookings Handling

```
4
Shell -
>>> %Run 'python booking app.py'
Welcome to Bright Minds Academy Equipment Booking System!

=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 3
No bookings recorded yet.

=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 4
No bookings recorded yet.

=== Main Menu ===
1. Add new equipment
2. Record a booking
3. View all booking records (by date)
4. View equipment status (by equipment)
5. Search bookings (by student or equipment)
6. Exit
Choose an option (1-6): 5
No bookings recorded yet.
```

Attempting to view all booking records, view equipment status and search bookings when no previous data was introduced.

Result:  Passed

4. Evaluation

4.1 Challenges Faced

I encountered several technical and design challenges throughout the development of the Bright Minds Academy Booking System. The first challenge was to understand and learn how to develop Python code via the coursework available, it required various trial and error attempts, however, with the help of extensive research and making use of the material available I was able to create a simple working application. The second challenge I had to face was time constraint. Personally, there was not enough time between lectures given and assignment deadlines to aid in successfully studying, learning and applying Python constructs. Therefore, I found myself having to do rigorous research to understand basic concepts, which in turn made it difficult to meet the deadline proposed.

4.2 Future Improvements

To enhance the system's functionality, various key improvements are proposed. First, moving from in-memory lists to a persistent database like SQLite would ensure easier data handling capabilities. Second, implementing a graphical user interface or a web-based interface would significantly improve usability and accessibility. Third, adding user authentication and role-based access control for students, technicians and administrators would make the system suitable for real-world usage. Finally, introducing features such as automated email notifications would increase the system's utility.

4.3 Self-Reflection

This project was a great hands-on experience in building software. I learned how to organize a Python program so it stays clear and easy to manage. Creating and running test cases helped me understand how important it is to check that everything works properly, including unusual situations that might cause errors. One of the biggest lessons was learning to tackle problems step by step, breaking them into smaller parts, using simple methods like print statements to find issues and improving the program through testing. Overall, this project has made me more confident in turning ideas into

working programs and showed me how important planning, testing, and clear documentation are.

Conclusion

In conclusion, building the Equipment Booking System was a useful way to apply basic programming skills in a real project. The final result is a working console-based program that meets the main requirements for managing and booking equipment. Although there were some challenges, especially with handling data and checking for errors, solving them helped improve my problem-solving and design skills. There are also clear ideas for future improvements that could turn this into a more complete, real-world system. Overall, the project shows how a structured approach can be used to create reliable and user-friendly software.

References

- [1] W3Schools, "DSA Introduction," *www.w3schools.com*.
https://www.w3schools.com/dsa/dsa_intro.php
- [2] "Python simple hack: Please considering 'Guard Clause' in your code | Kaggle," *Kaggle.com*, 2025. <https://www.kaggle.com/discussions/general/485872>
- [3] W3Schools, "Python String strip() Method," *www.w3schools.com*.
https://www.w3schools.com/python/ref_string_strip.asp
- [4] W3Schools, "Python String lower() Method," *www.w3schools.com*.
https://www.w3schools.com/python/ref_string_lower.asp
- [5] "The Importance of Software Testing," *IEEE Computer Society*, Apr. 17, 2024.
<https://www.computer.org/resources/importance-of-software-testing#why-is-software-testing-important>
- [6] GeeksforGeeks, "What is a Syntax Error and How to Solve it?," *GeeksforGeeks*, Apr. 12, 2024. <https://www.geeksforgeeks.org/c/what-is-a-syntax-error-and-how-to-solve-it/>
- [7] GeeksforGeeks, "Runtime Errors," *GeeksforGeeks*, Sep. 30, 2020.
<https://www.geeksforgeeks.org/dsa/runtime-errors/>

[8] GeeksforGeeks, "Errors and Exceptions in Python," *GeeksforGeeks*, Jun. 14, 2020. <https://www.geeksforgeeks.org/python/errors-and-exceptions-in-python/>

Appendices

Other sources used in the making of this project were:

UWTSD Official Coursework

<https://www.youtube.com/watch?v=n9eMYgk0hal> - Video used for inspiration purposes

<https://app.diagrams.net/> - Website used for the making of flowcharts