2026

Bright Minds Academy Student Equipment Booking System

https://github.com/Oyebamiji2026/Scriptingandprogramming/blob/main/Assessment Python Code

OYEBAMIJI OLUWAFEMI OYEDOKUN (2409692)

| University of Wales Trinity Saint David, Birmingham Campus.

TABLE OF CONTENTS

1.0 INTRODUCTION 2

2.0 REQUIREMENT ANALYSIS 3

3.0 SOFTWARE DESIGN AND DEVELOPMENT 8

3.1 Design 8

3.1.1 Flowchart 8

3.1.2 Pseudocode 13

3.2 Implementation Decision 26

3.2.1 Data Structures 26

3.2.3 Python Modules and Libraries..... 30

4.0 TESTING AND DEBUGGING 31

4.1 Role of Testing in Software Environment 31

4.2 Type of Errors in Programming 31

4.3 Example of Test Cases 33

5.0 EVALUATION 37

Challenges 37

Future Improvements 39

Self-Reflections 40

REFERENCES 42

APPENDIX..... 45

1.0 INTRODUCTION

The Bright Minds Academy booking system was made to make booking tools for schools in a way that is organised, quick, and easy for everyone to use. It's important for schools to have a structured system because there must be needs of digital tools like computers, desktops, cables, tablets, and memory drives. If there isn't a good booking system, equipment could be double booked, lost, or not available, which would cause training delays and unnecessary stress for both teachers and students. This project aims to solve these problems by creating a reliable system that correctly records, monitors, and maintains how equipment is used clearly and easy to understand.

Organised methods were used to build the system. The first step was to analyse the users' needs. This was followed by flowchart modelling, pseudocode writing, and finally Python coding. Each step made a big difference in how well the system works, making sure that every part, from adding equipment to making booking reports, works in a logical and uniform way.

The program lets users add new equipment, see what products are accessible, make reservations, look for records, cancel reservations, and print reports. It's a complete resource management tool.

The booking system is a structured, reliable, and useful tool that helps the Bright Minds Academy handle its equipment in the best way possible.

The bookings system focusses on the purpose of the system, how the system works, the user experiences, benefit to the Academy and how the problem being solved.

2.0 REQUIREMENT ANALYSIS

2.1 Definition and the purpose of using Requirement Analysis

Requirement analysis is the methodical process of determining, understanding, and recording what a software system must accomplished. It involves obtaining user data, evaluating their needs, and converting those needs into precise, structured requirements that direct the system's design and development [1].

For the equipment booking system, requirement analysis helps to: Determine which features, such as adding equipment, booking, searching, view record, booking cancellation, print booking reports, booking duration, comment on the booking and exit, it helps to acknowledge the problems with the existing paper-based method and it help to explain the necessity solution and make sure the program fits the requirements of the Bright Minds Academy [2].

The following features outline the system's capabilities and how each one contributes to the objectives of lowering booking errors, enhancing organisation and explanation of the features.

Add New equipment

Bright Minds Academy employees must be able to register new equipment, like laptops, tablets, or projectors, through the system. This feature guarantees that every resource is digitally recorded and can be chosen at the time of booking. Additionally, it avoids duplicate or missing equipment data, which enhances inventory control and accuracy.

View all available Equipment

Viewing all available equipment is a core feature in the Bright Minds Academy booking system because it gives staff a clear, real-time picture of what resources can be used at any given moment. It removes guesswork and prevents tutors from attempting to book items that are already taken or unavailable.

Record bookings

For a given date, staff members of the Bright Minds Academy must be able to assign a piece of equipment to a student. This feature guarantees that reservations are kept regularly and replaces the manual, paper-based procedure. By determining whether the equipment is already reserved on the chosen date, it also helps to avoid double bookings.

View all booking record

Staff of Bright Minds Academy must be able to view a list of all equipment and their associated bookings. This functionality provides transparency and allows staff to monitor equipment usage, check availability, and review past or upcoming reservations. It supports planning and reduces confusion caused by scattered or incomplete records.

Simple search

A basic search function that enables staff of Bright Minds Academy to look up reservations by student or equipment name must be included in the system. By assisting workers in finding relevant booking information quickly without having to

manually search through data, this feature increases efficiency. These will support organisation and reduces administrative workload.

Booking cancellations

When plans change or mistakes are made, the system must enable employees to cancel an existing reservation. This feature maintains the accuracy of the booking database and guarantees flexibility. Additionally, it keeps conflicts and uncertainty from arising from out-of-date or inaccurate reservations.

Print booking report

Booking summaries or reports that may be printed or saved must be generated by the system. Administrative responsibilities including lesson planning, prepping equipment, and communicating schedules with other staff members in Bright Minds Academy are supported by this feature. it enhances accountability and record keeping.

Exit Program

The exit features allow the user to safely close the program from the other features. It ensures that: the program stops operating, no unintended inputs occur and user returns to the command line. Bright Minds Academy staff may not be highly technical. They can utilise the system with confidence and not worry about breaking anything if there is an easy-to-use obvious exit option

2.2 Requirements Analysis Using the MoSCoW Method

	Must have	Should have	Could have	Won't have
1.	Add equipment			
2.	View all available Equipment			
3.	Record booking			
4.	View bookings			
5.	Search bookings			
6.		Cancel bookings		
7.		Print report		
8.	Exit			

Table 1 show the requirements Analysis MoSCoW method

2.3 Requirements Analysis Using Structure diagram

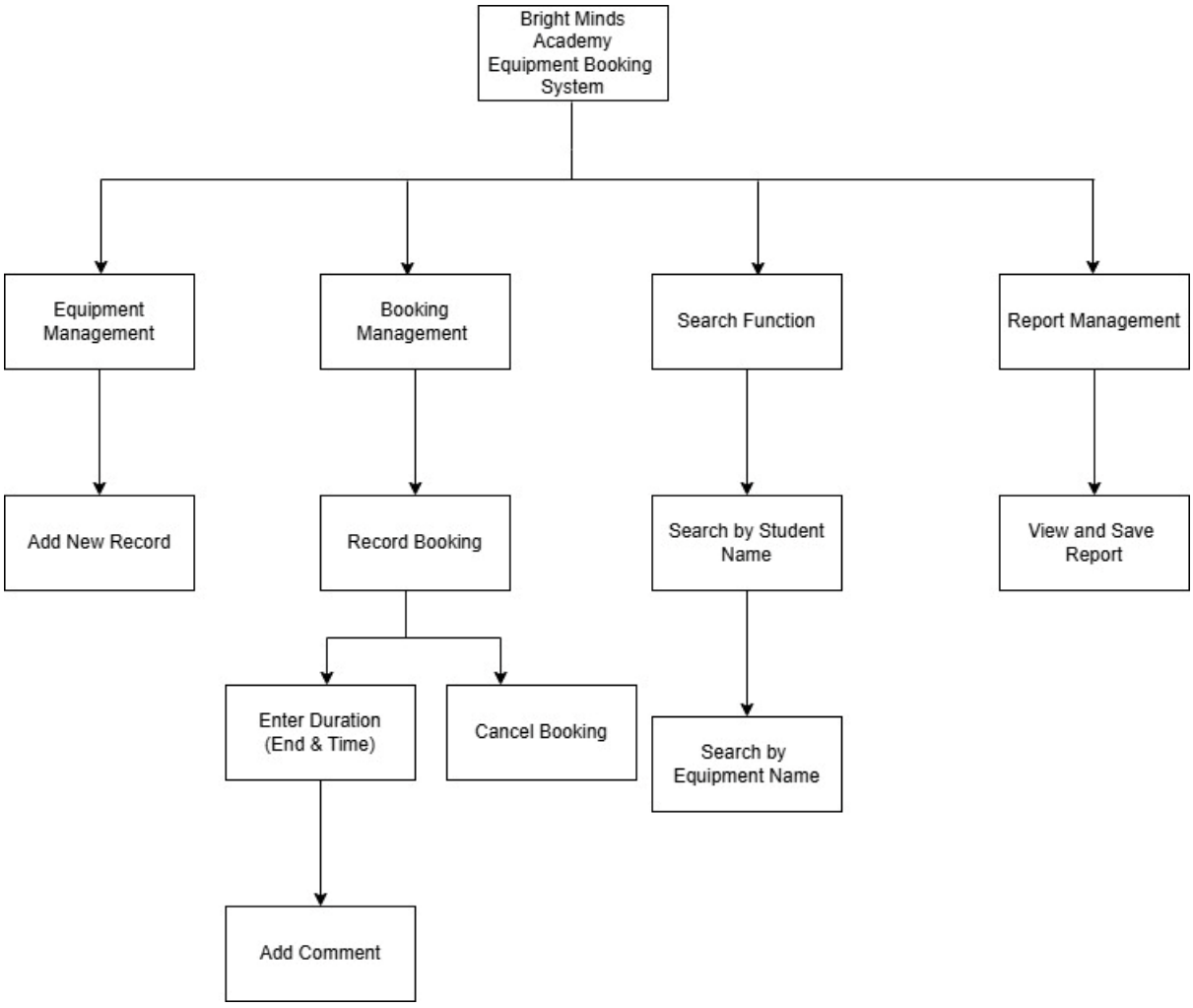


Figure 1 diagram show structure diagram

3.0 SOFTWARE DESIGN AND DEVELOPMENT

Design and development work together to make sure that the booking system is well organised, easy to keep up with, and able to solve the real-world challenge of managing equipment at Bright Minds Academy in an efficient way [8].

3.1

3.1.1 Flowchart

A flowchart is a visual diagram that show a program's logical flow [10].

Flowchart used standard symbols, like rectangles for processes, diamonds for decision, and arrows for direction, are used to show how data moves through a system. Flowcharts are very useful in programming and system design because they make it easier to understand, examine, and talk about complicated processes [10].

Regarding the In Bright Minds Academy of the Student Equipment Booking System, the goal of the flowchart is to provide a clear and visual summary of the system operations, Demonstrate the steps involved adding equipment, view all available Equipment, record booking, searching, view record, booking cancellation, print booking reports and exit. This help developer, student and staff to understand the system logic without reading the code, assist with planning and design, making sure the program structure is accurate before scripting and programming start and reduce errors by recognising logical problems early [10].

The IPO (Input, Process and Output) uses a flowchart approach to take the system requirements as input, processes, and give out the Output of Equipment Booking System. The flowchart shows how the program flows. In the end, it makes sure the program is well-organised, easy to comprehend, and works as it should [11].

flowchart

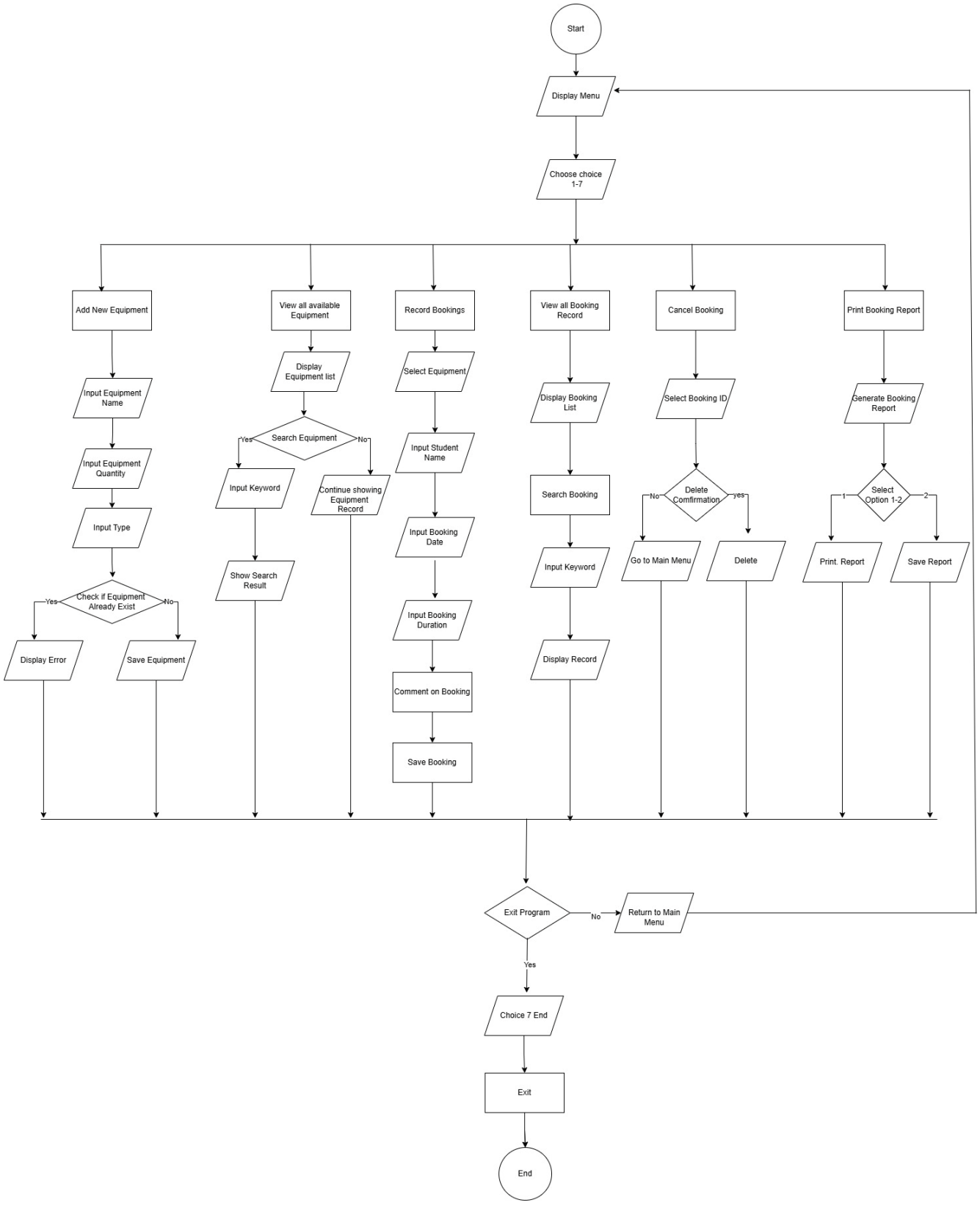


Figure 2 diagram show the flowchart of all features in the booking system from start to end

IPO Table

IPO table is simply structured method to displaying the inputs, processes, and outputs for any system features [12]. IPO Table helps programmers figure out what information a system needs, what it does with that information, and what outcomes it creates.

IPO table was made from flowchart which was included in below:

Add Equipment

INPUT	**PROCESS**	**OUTPUT**
Input Equipment details (Name, ID, type)	Save equipment record	Confirmation of saved equipment record or error message.
	Check if equipment already available and exists.	
	if exists display error	
	if not save Record.	

Table 2 show the Add Equipment IPO table

View all equipment available

INPUT	**PROCESS**	**OUTPUT**
Search Equipment	Showing available Equipment	Show search result
Input keyword	Search record	

Table 3 show the View all available Equipment IPO table

Record Booking

INPUT	PROCESS	OUTPUT
Select Equipment	Select equipment	Booking confirmation message
Booking duration	Enter booking details	
Input Student Name	Validate inputs	
Input Booking Date	Save booking record	
Commet on booking		

Table 4 show the Record booking IPO table

View all Booking Records

INPUT	PROCESS	OUTPUT
Search keyword	Display full booking list	List of all bookings
Input key word	Search Record	
	Display matching Record	

Table 5 show the View all booking Record IPO table

Cancel Booking

INPUT	PROCESS	OUTPUT
Select booking	Select booking to cancel	Cancellation success message
User confirmation **yes or no**	Ask for confirmation if yes	
	Delete booking and update record	

Table 6 show the View all cancel Booking IPO table

Print Booking Report

INPUT	PROCESS	OUTPUT
User option print or save	Generate booking report	Print report
	Ask user whether to print	Save report
	if yes print the report	
	If no, return to main menu	

Table 7 show the print booking report IPO table

Main Menu

INPUT	PROCESS	OUTPUT
User menu selection	Display Menu	System exits
Exit choice Yes or No	Execute selected operation	
	Ask if user wants to exit or return to main menu	

Table 8 show the Main Menu IPO table

3.1.2 Pseudocode

Pseudocode is a structured, human accessible description of an algorithm that uses plain language and programming language programming like; constructs to outline the logical steps of a system without following the strict syntax of any specific programming language [13].

The purpose of pseudocode in the booking is to provide a clear, structured and language independent explanation of how it should operate. By breaking down the logic of the system into easily understood phases. Pseudocode acts as a bridge between the flowchart and the actual code by expressing each features step such as installing equipment, making reservations, or producing reports [13].

Pseudocode sits between the system flowchart, and the final program. Its goal is to express the logic of the system in a form that is simple enough for people to understand but structured enough to be converted straight into code [14].

All the booking system features, including Add Equipment, Record Booking, and Cancel Booking; has a sequence of actions and decision. The logic steps are “What inputs the system needs” “What decisions must be made” “What actions follow those decisions” “What outputs are produced”. For example, in the flowchart “Check if equipment exists” “If yes show error” “If no → save record”. Pseudocode turns that into readable logic.

This below diagram expresses the logic clearly when there is a condition, there are two possible outcomes and the system must one.

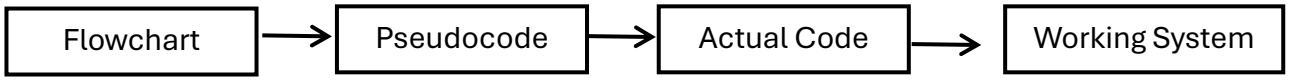


Figure 3 diagram show the clear logic.

Pseudocode

```
START
        SET EquipmentList AS ARRAY
        SET BookingList AS ARRAY
        SET choice AS INTEGER
        SET ExitProgram  FALSE
        WHILE ExitProgram AS BOOLEAN= FALSE DO

WHILE ExitProgram = FALSE DO


DISPLAY " ========BRIGHT MINDS ACADEMY BOOKING SYSTEM==========”
        DISPLAY "1. Add New Equipment"
         DISPLAY "2. View All Available Equipment"
        DISPLAY "3. Record Booking"
        DISPLAY "4. View All Booking Record"
        DISPLAY "5. Cancel Booking Report"
        DISPLAY "6. Print booking Report"
        DISPLAY "7. Exit Program"
        DISPLAY "----------------------------------"

        PROMPT user for choice (1-7)
        STORE user input in choice

        IF choice IS '1' THEN
                PROMPT user for // ADD NEW EQUIPMENT
                INPUT EquipmentName
```

```
        PROMPT “Enter Equipment Type”
        INPUT EquipmentType

        IF EquipmentName EXISTS IN EquipmentList THEN
                DISPLAY "Error: Equipment already exists."

        ELSE
                SAVE EquipmentName, EquipmentType INTO EquipmentList
                DISPLAY "Equipment successfully added."
        END IF

ELSE IF choice IS '2' THEN
        PROMPT user for // VIEW ALL AVAILABLE EQUIPMENT
        DISPLAY “AvailableEquipment”

        IF EquipmentList IS EMPTY THEN
                DISPLAY "No equipment available."
        ELSE
                DISPLAY “EquipmentList”
        END IF
        PROMPT “SearchEquipment”
        DISPLAY Yes or No
        INPUT “SearchChoice”
        IF SearchChoice = "Y" THEN
                PROMPT "Enter keyword: "
                INPUT keyword
```

```
            SET Found TO FALSE

            IF EquipmentList Keyword is matching
                DISPLAY Matching Equipment
            END IF
            IF found = FALSE THEN
                DISPLAY “No matching equipment found”
            END IF
        END IF

ELSE IF choice IS '3' THEN
        PROMPT user for // RECORD BOOKING
        IF Equipment is EMPTY THEN
            DISPLAY “No equipment is available”
        ELSE
            DISPLAY EquipmentList with index numbers

            PROMPT "Select equipment number: "
            INPUT Selection

        IF Selected NOT AVAILABLE THEN
            DISPLAY "Invalid Selection."

        ELSE
            INPUT StudentName
                INPUT BookingDate
```

```
                INPUT StartTime
                        INPUT EndTime
                INPUT Comment

                IF booking overlaps existing booking THEN
                        DISPLAY "Error: Equipment already booked during this time."

                ELSE
                                SAVE booking        (SelectedEquipment,
                                                    StudentName,BookingDate,
                                                    StartTime, EndTime, Comment)
                        INTO BookingList
                        DISPLAY "Booking saved successfully."
                END IF
            END IF
        END IF

ELSE IF choice IS '4' THEN
        PROMPT user for // VIEW ALL BOOKING RECORDS
        IF BookingList is EMPTY THEN
            DISPLAY “No booking record found”
        ELSE
            DISPLAY BookingList
            END For

            DISPLAY "Search? (Y/N)"
             INPUT SearchChoice
```

```
            IF SearchChoice = "Y" THEN
                INPUT keyword
                SEARCH BookingList FOR keyword
                DISPLAY matching records
            END IF
        END IF

ELSE IF choice IS '5' THEN
        PROMPT user for // CANCEL BOOKING
        IF BookingList is EMPTY THEN
            DISPLAY “No bookings to cancel”
        ELSE
            DISPLAY BookingList
            INPUT BookingID

            DISPLAY "Confirm Delete? (Y/N)"
              INPUT ConfirmDelete

            IF ConfirmDelete = "Y" THEN
                DELETE booking FROM BookingList
                DISPLAY "Booking cancelled successfully."
            ELSE
                DISPLAY "Cancellation aborted."
            END IF
        END IF
```

```
ELSE IF choice IS '6' THEN
        PROMPT user for // PRINT BOOKING REPORT
        IF BookingList IS EMPTY THEN
                DISPLAY “No bookings available”
        ELSE
                GENERATE BookingReport FROM BookingList

                DISPLAY "1. Print Report"
                DISPLAY "2. Save Report"
                INPUT ReportChoice

                IF ReportChoice = 1 THEN
                        PRINT BookingReport TO FILE
                END IF
        END IF

ELSE IF choice IS '7' THEN
        PROMPT user for // EXIT PROGRAM

        DISPLAY "Are you sure you want to exit? (Y/N)"
        INPUT ExitChoice

        IF ExitChoice = "Y" THEN
                SET ExitProgram TRUE
        END IF
```

```
    ELSE
        DISPLAY "Invalid choice. Please try again."
    END IF
  END WHILE
        DISPLAY "Program Closed Successfully."
END
```

How pseudocode translates into actual code

Pseudocode is written to mirror a program's thought process. It does not worry about the logic, decision and sequence of a programming language; instead, it records the reasoning, choices, and flow of actions. Each instruction in pseudocode corresponds directly to a programming construct when it is ultimately converted into actual code [14].

This below following stated how pseudocode translates into actual code in the booking system:

Start and Variable

Pseudocode starts with START and setting up variables. This turns into real code by setting up the program flow and saving data. In Python, `START` becomes the program's start point and `SET ExitProgram = FALSE` becomes `ExitProgram = False`. `equipment_list = []` turns variables like `EquipmentList` into lists, which lets the computer store, process, and control data in a logical way.

```
    ["Keyboard", 100, "Desktop"],
    ["Monitor", 100, "Desktop"],
    ["CPU", 100, "Desktop"],
    ["Tablet", 200, "Phone"],
    ["Webcam", 100, "Desktop"]
]
# Stores booking dictionaries [student name, equipment name, date,time, comment]
booking_list = [
    ["Oyebamiji", "HP", "25/01/2026", "13:00pm-16:00pm" ],
    ["Olufunke", "Tablet", "29/01/2026", "15:00pm-17:00pm"],
    ["Abolade", "Dell", "26/01/2026", "16:20pm-17:00pm"],
    ["Temilola", "Dell","15/02/2026", "13:00pm-16:00pm"],
    ["Tomiwa", "HP","16/02/2026", "17:13pm-19:00pm"]
]
ExitProgram = False
# ------------------ MAIN LOOP ----------------------------
while ExitProgram == False:
```

Figure 1 image show the list and Exit program in my python code.

Display menu

In pseudocode, DISPLAY statements are used to show a menu. In real code, these statements are output orders. In Python, DISPLAY "1. Add Equipment" would be print ("1. Add Equipment"). A structured menu interface is made up of several DISPLAY lines. This lets users see their options easily and interact with the program by using choices and input prompts.

```python
    # ---------- ADD NEW EQUIPMENT ----------
    if choice == "1":

        equipment_name = input("Enter Equipment Name: ")
        equipment_type = input("Enter Equipment Type: ")

        exists = False
        for eq in equipment_list:
            for eq in equipment_list:
                if eq[0].lower() == equipment_name.lower():  # index 0 = name
                    exists = True
                    break

        if exists:
            print("Error: Equipment already exists.")

        else:
            equipment_list.append({
                "name": equipment_name,
                "type": equipment_type
            })
            print("Equipment successfully added.")
```

Figure 2 image show how the python code print successfully.

Get user choice

INPUT is used by pseudocode to get user input, which is the same thing that happens in real code. In Python, INPUT choice is changed to choice = input("Enter choice:"). The value is saved in a variable and used in conditions like IF choice = 1. This lets the program respond to what the user chooses and manage the flow of the program.

```python

    choice = input("Enter your choice (1-7): ")

    # ----------------- ADD NEW EQUIPMENT -----------------
    if choice == "1":

```

Figure 3 image show how pseudocode translate user choice into actual code.

View all equipment

Pseudocode uses DISPLAY and loops, which is the same thing as iteration in code. DISPLAY EquipmentList, for instance, turns into a loop like for eq in equipment_list: print("……") in Python. If the list of equipment is empty, IF EquipmentList IS EMPTY changes to if len(equipment_list) == 0. This makes sure that the computer correctly handles and shows data.

```
    elif choice == "2":
        print("\n----- AVAILABLE EQUIPMENT -----")
        if len(equipment_list) == 0:
            print("No equipment available.")
        else:
            for i, eq in enumerate(equipment_list):
                print(f"{i}. Name: {eq[0]} | Quantity: {eq[1]} | Type: {eq[2]}")
        print("--------------------------------")

```

Figure 4 image show *how pseudocode translate into actual code.*

Record booking

The INPUT and SAVE steps in pseudocode are used to create records, which are equivalent to variable input and list storage in code. INPUT StudentName changes to student = input(), and SAVE booking INTO BookingList changes to booking_list.append([...]). Conditions like checking for availability are turned into if statements, which make sure that only real bookings are saved and that conflicts don't happen.

```python
if len(equipment_list) == 0:
    print("No equipment available.")
else:
    print("\nAvailable Equipment:")
    for i, eq in enumerate(equipment_list):
        print(f"{i + 1}. {eq[0]} ({eq[2]})")  # index 0 = name, index 2 = type

    try:
        selection = int(input("Select equipment number: ")) - 1
        selected_equipment = equipment_list[selection][0]
    except:
        print("Invalid selection.")
        continue

    student_name = input("Enter Student Name: ")
    booking_date = input("Enter Booking Date (DD/MM/YYYY): ")
    start_time = input("Enter Start Time (HH:MM): ")
    end_time = input("Enter End Time (HH:MM): ")
    comment = input("Enter Comment: ")

    overlap = False
```

Figure 5 image show how pseudocode translate into actual code.

Exit program

SET ExitProgram = TRUE is an example of pseudocode that ends a program. In code, this is the same thing as changing a loop control variable. In Python, IF choice = 7 changes to if choice == "7": ExitProgram = True. This ends processing and stops the loop, but it doesn't exit the program. After the user confirms, the program can close safely.

```python
    # ------------------ EXIT PROGRAM ------------------
    elif choice == "7":

        exit_choice = input("Are you sure you want to exit? (Y/N): ").upper()

        if exit_choice == "Y":
            exitProgram = True
        else:
            print("Returning to main menu.")

    # ------------------ INVALID INPUT ------------------
    else:
        print("Invalid choice. Please try again.")

print("Program closed successfully.")
```

Figure 6 image show how pseudocode translate into actual code.

3.2 Implementation Decision

3.2.1 Data Structures

A data structure is an organised way to managing, arranging, and storing data on a computer for effective access and usage. It specifies how information is related, how it is stored, and what can be done with it, including searching, inserting, deleting, and updating data [16].

The system used Lists to store collections of items such as “EquipmentList” “BookingList”. This was chosen because of dynamic growth as new equipment or booking are added and make it easy to implement in pseudocode and can be translated into python programming,

The program uses variables to store and organise data in a clear and efficient way. For for example; EquipmentList keeps track of all records for equipment, while choice records what the user types to control the menu. ExitProgram handles the end of loops. Using meaningful variable names makes the code easier to read, cuts down on

mistakes, and makes sure the program can process, update, and receive data in a way that makes sense.

3.2.2 Variables

Variables are named storage location that are used to hold data that can change while a program is running. They let programs save things like numbers, text, or groups and use them again when they need to [16].

In the Bright Mind Academy System, variable was used where only two possible states exist such as TRUE or FALSE. For example, “ExitProgram control whether is continues running” “EquipmentExists checks if equipment is already registered” .

The system used String Variables such as StudentName, EquipmentName and comment.

The system used Integer Variables where is stated choice like: 1-7 and this numeric input and menu navigation.

This system used Date and Time to manage booking duration such as BookingDate, StartTime and EndTime. This was used because to allow accurate scheduling and prevent overlapping bookings.

All above was used for solution because it simple to implement, easy to understand and maintain, managing multiple records, suitable for Bright Minds Academy Booking System and efficient for storing in the variable table below.

Variable table with their data types

Variable Name	Data type	Example in the python code in the booking system	Description
EquipmentList	List	[["HP", 50, "Laptop"], ["Drive", 70, "Memory"]]	It keeps all the tools and gives each item a name, a quantity, and a type.
BookingList	List	[["Oyebamiji", "HP", "25/01/2026", "13:00pm-16:00pm"]]	It keeps track of reservations, and each one has a student's name, an equipment's name, a date, a start time, an end time, and a note field.
ExitProgram	Bool	False	A flag that controls the main program loop
Choice	String	“!”	Input from the user menu
EquipmentName	String	“HP”	Name of equipment to add
ExitChoice	String	“Y”	User confirmation to exit program

Table 9 show variable with their data type.

3.2.2 Programming Constructs

Programming constructs are the fundamental building blocks used to design and implement any program [19]. They are selection (making choices based on conditions,

like if statements), iteration (doing things repeatedly using loops, like for or while), and sequences (running statements in order). These building blocks help programmers manage how a program works and quickly solve issues [19].

The Control structures used in the Python code

Control structures are used by the Bright Minds Academy Booking System to keep track of the flow of programs [19]. While loops constantly show the main menu, for loops go through equipment and reservations, and if elif and else statements handle user choices and checks. Conditions and nested loops check for availability, stop overlaps, and handle searches, cancels, and reports, making sure that booking management is quick and flexible [19].

The booking system follows the order when performing tasks: for example, when record the booking, the program must selected equipment, enter student name, enter booking date, add comment and save booking. Each step must be completed in the exact for the booking to be valid.

If/ELSE or Yes/No decisions are used in a system to handle options like checking if equipment is available, confirming deletions, printing reports, or exiting. These options are often controlled by loops (WHILE, FOR, or REPEAT), so when the system asks, "Exit: Yes/No?" and the user chooses "No," it goes back to the main menu and keeps looping until the user chooses "Exit," which is option 7.

Each control structure makes sure that the program is user friendly, effective, and organised in a way that makes sense. This makes booking management easy and error-free.

The program's built-in functions, such as input(), print(), len(), int(), any(), enumerate(), and open(), handle user input, check data, control lists, stop errors, and save reports. This makes sure that the booking system is reliable, easy to use, and interactive.

The booking system is strong and reliable because it handles errors with try except blocks and conditional checks. This makes sure that the program handles incorrect inputs, stops crashes, and gives users clear messages.

3.2.3 Python Modules and Libraries

Python Modules is a singular Python file (.py) that contain functions, variables, or classes that can be imported into another program. Libraries is a collection of various modules packages together. It offers an expanded array of tools for addressing particular issues.

They make it easier for developers to make complicated programs quickly and with fewer mistakes. Python also has a large standard library with built-in modules for working with files, networking, and the system. To add to these modules, external libraries can be added using pip.

Bright Minds Academy Booking System used python modules and libraries, these are input(), print(), len(), int(), any(), enumerate(), and open(). They take in user input, show data, control lists, change data types, stop duplicates, keep track of indices, and save reports. This keeps the program interactive, reliable, and effective without crashing.

4.0 TESTING AND DEBUGGING

4.1 Roles of Testing in Software Environment.

Program testing in software development is the process of executing a program to verify its correctness working and produces the expected results [18].

Program testing is important in software development since it facilitates the identification of mistakes and bugs in a program before it is used by users. Through software testing, developer can ascertain if the program operates accurately and yields the anticipated outcomes [18].

4.2 Type of Errors in Programming

Errors in Programming are mistakes in code that prevent a program from running correctly or producing the desired output [20]. They can occur due to syntax mistakes, incorrect logic, or invalid operations. The type of error is syntax error, logical error and runtime error [20].

Syntax error is when code doesn't follow the rules of the computer language, it's called a syntax error. Python can't read it because it's like making a sentence with bad grammar. Example is missing brackets or quotation marks [20].

Example 1

Missing colon in a loop or condition is when there is no colon after "if" in a loop or condition: if 5 > 3 No colon Print("5 is greater than 3") [20].

Example 2

Unclosed parentheses are when Print("Hello World" . the missing bracket is the unclosed [20].

Example 3

If or elif statements are missing a colon. For example, if choice == “1” Missing colon, print(“Add New Equipment”). Python needs a colon at the end of if statements.
The program will not run and will give a SyntaxError.

Logical Error

Logical error is when the program starts but doesn't do what programmer want it to; the programmer will have a logical error. The code works, but the way it works is wrong [20].

Example 1

The check for overlapping bookings might not work. overlap = False for booking in booking_list: if booking[1] == selected_equipment and booking[2] == booking_date: existing_start = booking[3], if start_time is same to existing_start: overlap = True; break;
This just tests if the start timings are the same, which is a logical fallacy. It won't find a conflict if two bookings overlap in part (for example, 13:00–15:00 vs. 14:00–16:00).
Effect: Even after checking, equipment may still be double-booked.

Example 2

If age > 18, it should be >= or > depending on what you need. else: print("You can't vote") # Wrong output for age = 15 [20].

Example 3

Adding the wrong variable: x = 5, y = 10, sum = x - y # Should be x + y print(sum). The output is -5, not 15 [20].

Runtime Errors

Runtime errors mistake while the program is running. The code is right in terms of syntax, but when it runs, something strange happens [20].

Example 1

Trying to access equipment or a booking that doesn't exist. selection = int(input("Select equipment number: ")) - 1; selected_equipment = equipment_list[selection][0] If the selection is out of range, there will be a runtime error.

Example 2

Can't change a string to an integer. equipment_quantity = int(input("Enter Quantity: ")). If the user types "ten" instead of 10, there will be a runtime error.

Example 3

Trying to read a file that doesn't exist (if extended code reads reports) with open("BookingReport.txt", "r") as file: Error if file doesn't exist data = file.read()

4.3 Example of Test Case

Test case is when to make sure that a program or system works right, test case should be used, which is a particular set of inputs, conditions, and expected outcomes. It's a planned way to check out one function or trait at a time [20].

Test Case Title: Add equipment

In the test case table below, the user can add a new piece of equipment to the system. The user puts in the name, quantity, and type of the equipment. The system checks to see if the item is already on the list of equipment. If it isn't, it adds it and makes sure it's there. It gives an error message if it already exists.

Test case ID	**TC001**
Test Case Tittle	Add Equipment
Objective	Verify that the system correctly adds a new equipment item when it does not exist.
Preconditions	“System is running” equipment list already contains several items (HP, Dell, Drive, Keyboard, Tablet, Webcam, CPU and Lenovo.)
Test step	Enter equipment name, enter quantity, enter equipment type and If equipment does not exist, “system adds” it and If equipment already exists, “system rejects it”
Test data	Cord (USB)
Expected Result	The system display adds successfully, or equipment Already exist.
Pass or fail	Pass
Comments	Adding a New Item

Table 10 show the test case table for Add Equipment.

```
-----------------------------------------------------
Enter your choice (1-7): 1
Enter Equipment Name: Cord
Enter Quantity: 50
Enter Equipment Type: Desktop
Equipment successfully added.

======== BRIGHT MINDS ACADEMY BOOKING SYSTEM ========
```

Figure 7 image show the test case for Add Equipment.

Test Case Title: View All Available Equipment

In the test case table below, the function shows all the equipment that is saved in the system. Each item's name, quantity, and type are shown. Users are told that there is no equipment available if the list is empty. The user using it can also look for tools by name or type.

Test Case ID	**TC002**
Test Case Tittle	View all available Equipment
Objective	Test that the system displays all equipment stored in the equipment list.
Preconditions	Equipment list contains at least Ten (10) items.
Test step	View all available equipment and search equipment: if user select “NO” System return to main menu, if user select “YES” enter keyword.
Test data	Cord (USB)
Expected Result	System displays a list such as Cord (USB)
Pass or fail	Yes
Comments	Displaying Full Equipment List

Table 11 show the test case table for View all available Equipment.

```
4. Name: Mouse | Quantity: 100 | Type: Desktop
5. Name: Keyboard | Quantity: 100 | Type: Desktop
6. Name: Monitor | Quantity: 100 | Type: Desktop
7. Name: CPU | Quantity: 100 | Type: Desktop
8. Name: Tablet | Quantity: 200 | Type: Phone
9. Name: Webcam | Quantity: 100 | Type: Desktop
10. Name: Cord | Quantity: 50 | Type: Desktop
--------------------------------
Search equipment? (Y/N): y
Enter keyword: Cord

----- SEARCH RESULTS -----
10. Name: Cord | Quantity: 50 | Type: Desktop
```

Figure 8 image show the test case out for view all available Equipment.

5.0 EVALUATION

Evaluation is the process of assessing a system, project, or solution to judge how well it meets its goals, identify strengths and weaknesses, measure performance, and determine what improvements are needed to make the system more effective, reliable, and suitable for its intended purpose [20].

5.1 Challenges and how it was addressed.

There were a lot of problems that came up when making the Bright Minds Academy booking system. These included structured design, flowchart building, pseudocode development, Python implementation, and report writing. One of the first problems was putting the system needs into a clear order. The system included many functionalities, such as adding equipment, examining records, searching, booking equipment, cancelling, and generating reports. This made it hard to divide the system down into logical parts. This was fixed when asking questions from lecturer (Dr Mobolanle) in the class, making research on internet, journals and using AI to get idea.

Making the flowchart was equally hard. It took a lot of care to highlight every decision point, loop, and process in a visual way, especially when showing activities that happen again, like going back to the main menu or confirming deletions. We made the flowchart easier to read by breaking it up into clear sections, using the same symbols that was teaches by Dr Mobolanle in class, and making sure that each path logically led to the next. This made the flowchart easier to understand and made it match how the software worked.

To write IPO table and pseudocode, I had to turn the flowchart into stages that were structured and follow the standard pseudocode template format that was given in the class. The biggest problem was making sure the pseudocode was precise enough to help with coding but also easy to understand. This was fixed by standard pseudocode template format, watching online video and asking question in the class. This makes it easier to switch from pseudocode to Python.

There were several technical problems with putting the system into Python. It took a lot of rigorous debugging to make sure that the menu loop functioned right, that user input was legal, and that the program didn't crash. Step by step testing, adding safe access checks, and rewriting bad logic fixed problems including wrong indexing, search mismatches, and cancellation mistakes. Updating variables inside loops made search functions better, while fixing indentation and input validation corrected cancellation issues.

Producing the report meant making technical choices apparent. To solve the problem, each step design, development, testing, and evaluation was written down with examples, screenshots, and clear explanations. In general, these problems made it easier to solve problems and made the final system better.

There were a lot of technical problems that came up when we put the system into Python. It took a lot of rigorous debugging to make sure that the menu loop functioned right, that user input was legal, and that the program didn't crash. Step by step testing, adding safe access checks, and rewriting bad logic fixed problems including wrong indexing, search mismatches, and cancellation mistakes. Updating variables inside

loops made search functions better, while fixing indentation and input validation corrected cancellation issues.

Lastly, producing the report meant making technical choices apparent. The problem was solved by writing out each step design, development, testing, and evaluation using examples, screenshots, and clear explanations. In general, these problems made people better at addressing problems and made the final system better.

5.2 Future Improvements

The Bright Minds Academy booking system have the functionalities features like adding equipment, keeping track of bookings, searching records, cancelling reservations, and making reports. However, there are several changes that could make it more efficient, easier to use, and more reliable in the long run. One big area for future growth is the creation of a better way to store data. Right now, the system uses in memory lists, which means that all data is destroyed when the program ends. Using a database or even a simple CSV or JSON storage system would let bookings and equipment records stay the same between sessions, making the solution more useful and professional.

Another useful change would be to offer a way to find out when a booking has expired. This feature would automatically check the current date and time to see if a booking is coming up, is active, or has already ended. It might highlight, archive, or take expired bookings from the active list. This improvement would cut down on clutter, stop confusion, and help with improved equipment management.

Adding a section for past bookings is another change that goes along with this one. The system might automatically relocate expired bookings to a separate archive instead of mixing them with new ones. This would help staff keep better records, look at how things were used in the past, and keep track of how much equipment is needed. Adding input validation, including validating date formats, stopping invalid times, and making sure equipment names are always entered the same way, could potentially make the user experience better

Being able to change existing records, like changing booking times, updating comments, or changing equipment data, is another improvement that will come. This would make the system more adaptable and cut down on the number of times you have to delete and recreate entries.

5.3 Self- Reflections

Building the booking system has taught me a lot and made me better at both technical tasks and thinking rationally and creatively. At first, it was hard for me to break the system down into smaller, more manageable parts. However, going through structured design helped me see how important it is to plan before coding. The flowchart let me see the whole process clearly and showed how each part is connected to the next. This stage taught me that a well-designed system makes things less confusing later in development.

Writing pseudocode helped me turn my thoughts into organised logic, and I learned how important it is to avoid making mistakes while putting things into action. This is my first time of learning about Python code although I had a lot of problems, like wrong indexing, wrong input, and search methods that didn't operate as I expected. I learned to look at problems more closely and modify parts of code to make them more dependable via testing and debugging. This made me more confident in dealing with mistakes and made me better at fixing problems.

This project has helped me get better at planning, coding, debugging, and evaluating, and it has also taught me how important it is to be patient, keep going, and keep learning in software development.

REFERENCES

[1] I. Sommerville, Software Engineering, 10th ed. Boston, MA: Pearson, 2022.

[2] A. Rahman and M. Abdullah, “Design and implementation of a resource booking system for educational institutions,” International Journal of Computer Applications, vol. 175, no. 3, pp. 12–18, 2021.

[3] B. Nuseibeh and S. Easterbrook, “Requirements engineering: A roadmap,” in *Proc. Conf. Future of Software Engineering*, 2000, pp. 35–46.

[4] J. Nielsen, “Usability engineering and system design,” *Communications of the ACM*, vol. 33, no. 3, pp. 32–48, 2021.

[5] S. Lauesen, “User requirements: What are they and why do they matter?” Computer Science Review, vol. 3, no. 1, pp. 1–21, 2022.

[6] T. Catarci, M. F. Costabile, and S. Levialdi, “Visual interaction with information systems,” Journal of Visual Languages & Computing, vol. 10, no. 4, pp. 341–363, 2021.

[7] P. Forbrig, M. Herczeg, and A. Dittmar, “Improving booking systems through usability-driven design,” Human-Computer Interaction, vol. 25, no. 2, pp. 145–168, 2021.

[8] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 8th ed. New York, NY, USA: McGraw-Hill, 2021.

[9] I. Sommerville, *Software Engineering*, 10th ed. Boston, MA, USA: Pearson, 2016.

[10] A. Dennis, B. Wixom, and R. Roth, *Systems Analysis and Design*, 7th ed. Hoboken, NJ, USA: Wiley, 2021.

[11] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2021.

[12] Feldman, *"A Comprehensive Guide to Input-Process-Output Models,"* iSixSigma, Jan. 23, 2025.

[13] Glenn Brookshear and D. Brylow, *Computer Science: An Overview*, 13th ed. Boston, MA, USA: Pearson, 2021.

[14] Patterson and J. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 6th ed. Burlington, MA, USA: Morgan Kaufmann, 2021.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.

[16] J. G. Brookshear and D. Brylow, *Computer Science: An Overview*, 13th ed. Boston, MA, USA: Pearson, 2021.

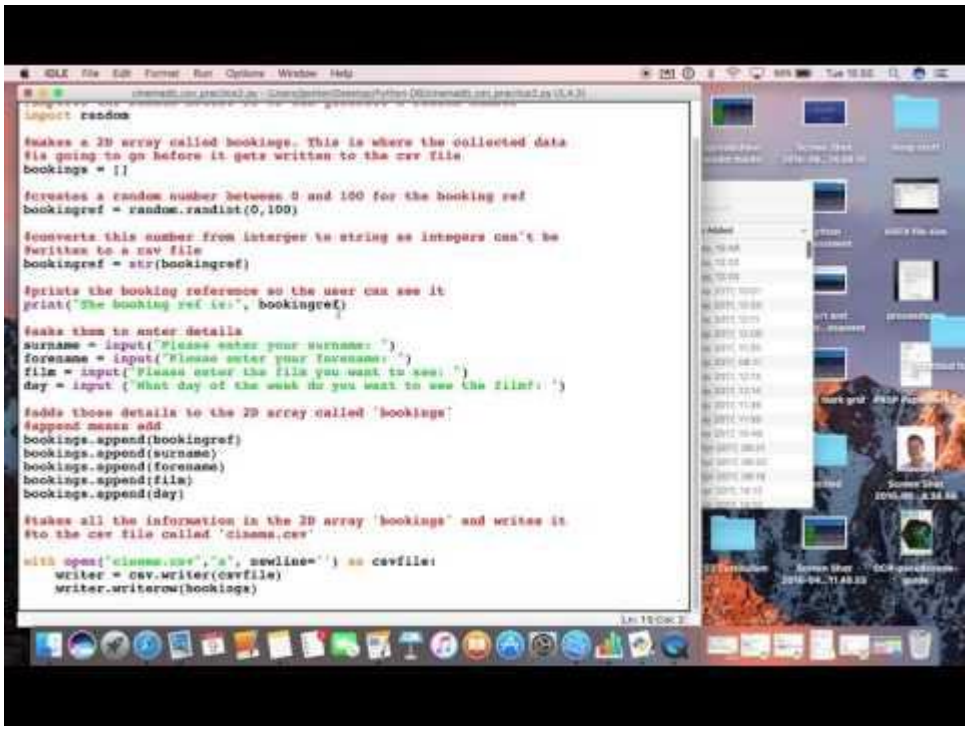
[17] M. Goodrich, R. Tamassia, and M. Goldwasser, *Data Structures and Algorithms in Python*, Hoboken, NJ, USA: Wiley, 2021.

[18] B. W. Kernighan and D. M. Ritchie, The C Programming Language, 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 2021.

[19] A. Hunt and D. Thomas, *The Pragmatic Programmer: Your Journey to Mastery*, 20th Anniversary ed. Boston, MA, USA: Addison-Wesley, 2021.

[20] Sommerville, I. *Software Engineering*, 10th ed., Pearson Education, 2022.

APPENDIX



https://codingfleet.com/transformation-details/creating-a-simple-booking-app-with-python/