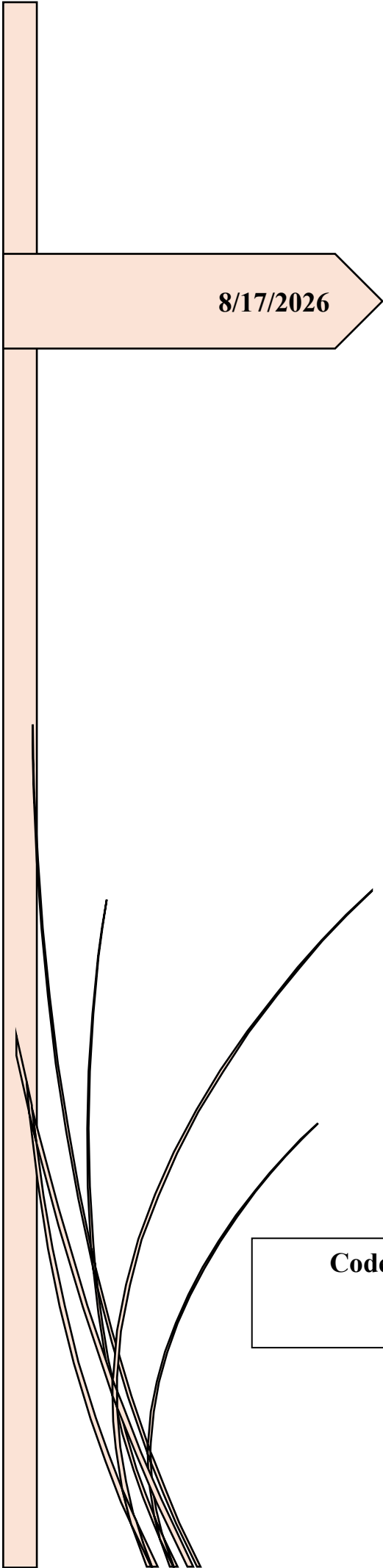


A vertical orange bar on the left side of the page. A horizontal orange arrow points to the right from the bar, containing the date 8/17/2026.

8/17/2026

**SPC Assignment on
Smart Home Device Manager System for
Bright Minds Academy**

**Module Name: Scripting and
Programming Concepts**

Module Code: ACCA 5036

Student Name: Elaine Rodovalho

Student ID:2427102

Code Link: https://github.com/Foxgirl70/Elaine-Lopes-bright_minds_devices.py-.git

**University of Wales Trinity Saint David
LONDON CAMPUS**

Table of Contents

Introduction.....	2
Main Body	2
1. Analysis.....	2
Requirements Analysis.....	3
3. Testing and Debugging Report	21
4. Evaluation	23
Conclusion	27
References.....	28
Appendices.....	Error! Bookmark not defined.

Introduction

Bright Minds Academy provides private tutoring and is introducing new technology into its classrooms. The centre is testing different smart devices, such as lighting systems, projectors and temperature controls. However, staff currently have no central system for managing this equipment, as most checks are completed manually. This can cause difficulties when identifying where a device is located, whether it is connected and when it was last used. To improve this process, the project will develop a basic software application that stores and displays important device information. The proposed system will make device management clearer, quicker and more reliable for academy staff.

Main Body

1. Analysis

Bright Minds Academy requires a basic digital system to organise information about the smart equipment used in its classrooms. Now, staff manage devices such as projectors, thermostats and intelligent lighting by hand. Because there is no central record, it may be difficult to identify which devices are available, where they are located and whether they are online or offline. Carrying out a requirements analysis before development is essential because it helps define what the program must do and what information it should manage [1].

To address this issue, a Python application will be created using data structures such as lists and dictionaries. These will store important details about each device, including its name, location, condition and connection status. The purpose of the program is to make the management process more organised, efficient and easy for staff to follow. The software will not connect to or operate the devices directly. Its role will be limited to storing, updating, searching and displaying device information.

Requirements Analysis

The functional requirements describe the main activities that the Smart Device Management System must allow staff to complete. Defining these requirements before development helps ensure that the final program meets the needs of Bright Minds Academy [1].

Add New Devices:

The system must allow staff to register new classroom devices, such as “Smart Light A1” or “Projector B2.” The user should enter the device name and the room where it is located. This function will help the academy maintain an organised inventory of its smart equipment. Maintaining an accurate system component inventory supports effective monitoring and management [2].

Update Device Status:

Staff should be able to change the status of an existing device. The available options will be online, offline, or maintenance. This will allow staff to identify whether a device is connected, unavailable, or requires technical attention.

View Device List:

The application must display all devices recorded in the system. For each device, the program should show its name, classroom location, and status. This will provide staff with a central view of the academy’s smart classroom equipment instead of requiring them to check each room manually.

Simple Search:

The system should allow staff to search for a device by entering its name. When the device is found, the application should display its location and status. A suitable message, such as “Device not found”, should appear when no matching record exists.

User-Friendly Interface:

The application should provide a clear menu with numbered options for adding, updating, viewing, and searching for devices. Instructions, results, and error messages should be simple to understand because some staff members may have limited technical experience. Ease of interaction is recognised as an important software quality characteristic [3].

Scalability:

The program should support the addition of more devices as Bright Minds Academy expands its smart classroom system. Although the first version will use a Python dictionary, its structure should make it possible to add more projectors, thermostats, smart lights, or other types of equipment in the future. Flexibility and capacity are important considerations when software is expected to support future changes and growth [3].

Non-Functional Requirements

In addition to the functional requirements, the Smart Device Management System must meet several quality standards. These requirements describe how effectively and safely the application should operate. ISO/IEC 25010 provides a recognised model for specifying and evaluating software quality [3].

Reliability:

The system should provide accurate and consistent device information. When staff add, update, search for or view a device, the correct name, location and status should be displayed. Input validation should also be used to reduce errors and prevent incomplete records.

Performance:

The application should respond quickly when recording or retrieving device information. Adding a device, changing its status, displaying the device list and completing a search should occur without unnecessary delays. As the program uses a Python dictionary and manages a small amount of data, operations should be completed efficiently.

Security:

The system should protect device records from accidental deletion, incorrect changes and loss. Validation should prevent invalid information from being entered. In a future version, device records could be saved in a protected JSON file or database, with backup controls used to support recovery. NIST SP 800-53 includes controls relating to access control, system integrity, information protection and contingency planning [4].

Accessibility:

The application should be accessible and easy to understand for staff with different levels of technical ability. Menu options, instructions and error messages should use clear language and readable formatting. Users should be able to operate the program using simple keyboard inputs. Software accessibility aims to support users with a wide range of physical, sensory and cognitive abilities [5].

2. Software Design and Development

This section explains the design and development of the Smart Device Management System. The program was designed to help Bright Minds Academy organise information about smart classroom devices, including projectors, thermostats and smart lighting. The main functions allow staff to add devices, update their status, view all registered devices, search for a device and exit the application.

2.1 Design

Before developing the Python program, the system logic was planned to use a flowchart and pseudocode. These design tools helped to organise the sequence of operations and identify the decisions that the program must make.

Flowchart

The flowchart shows the main sequence of the Smart Device Management System. The program begins by creating an empty dictionary called devices. It then displays a menu containing five options:

Add a device

Update device status

View all devices

Search for a device

Exit the program

After the user selects an option, the program performs the selected action. When the action is completed, the system returns to the main menu. The program only stops when the user chooses the exit option.

Flowchart

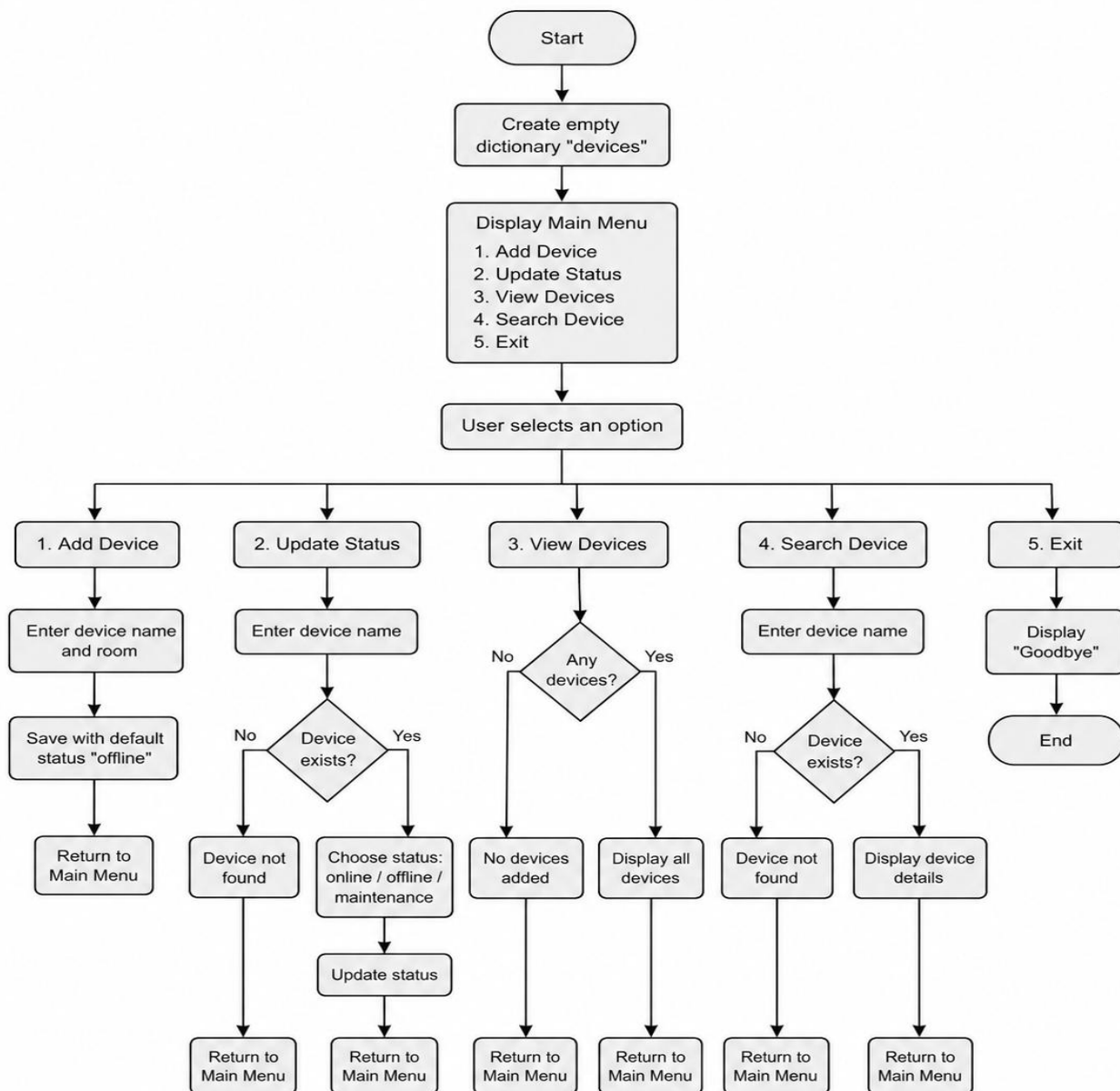


Figure 1: Flowchart of the Smart Device Management System

Pseudocode

The pseudocode describes the main operations of the application using structured English. It shows how the menu, conditions, loops and device-management functions will operate before they are converted into Python code.

Pseudocode:

Begin

CREATE empty dictionary called devices

WHILE program is running

DISPLAY main menu

DISPLAY "1. Add Device"

DISPLAY "2. Update Device Status"

DISPLAY "3. View Devices"

DISPLAY "4. Search Device"

DISPLAY "5. Exit"

INPUT userChoice

IF userChoice = 1 THEN

INPUT deviceName

INPUT deviceLocation

SET deviceStatus = "offline"

STORE device information in devices

DISPLAY "Device added successfully"

ELSE IF userChoice = 2 THEN

INPUT deviceName

IF deviceName exists in devices THEN

INPUT newStatus

IF newStatus is online, offline or maintenance THEN

UPDATE device status

DISPLAY "Status updated successfully"

ELSE

DISPLAY "Invalid status"

END IF

ELSE

```
        DISPLAY "Device not found"

    END IF

ELSE IF userChoice = 3 THEN

    IF devices are empty THEN

        DISPLAY "No devices added"

    ELSE

        DISPLAY all device names, locations and statuses

    END IF

ELSE IF userChoice = 4 THEN

    INPUT deviceName

    IF deviceName exists in devices THEN

        DISPLAY device name, location and status

    ELSE

        DISPLAY "Device not found"

    END IF

ELSE IF userChoice = 5 THEN

    DISPLAY "Goodbye"

    STOP program
```

```
ELSE  
DISPLAY "Invalid menu option"  
END IF  
END WHILE  
END
```

Figure 2: Pseudocode of the Smart Device Management System

The flowchart provides a visual representation of the program, while the pseudocode gives a written description of the algorithm. Together, they made the development process more organised and helped ensure that all functional requirements were included in the final Python solution.

Implementation Decisions

The Smart Device Management System was developed using Python because it provides clear syntax and is appropriate for a beginner-level software project. The main implementation decisions included the use of a dictionary, separate functions, selection statements and loops. These choices helped to keep the program organised, understandable and easy to test.

Data Structures:

A Python dictionary called `devices` was used as the main data structure. A dictionary stores information using key-value pairs, making it suitable for managing device records. The device name is used as the main key. Each device then contains its location and status.

```
devices["Projector B2"] = {  
    "location": "Room 4",  
    "status": "offline"  
}
```

This structure was selected because it allows the program to quickly add, search and update device information. A list could also store the devices, but searching for a particular device would require checking each item individually. Using the device name as a dictionary key makes the search process simpler

Variable	Purpose
<code>devices</code>	Stores all registered device information.
<code>device_name</code>	Stores the name entered by the user.
<code>location</code>	Stores the classroom or room of the device.
<code>user_choice</code>	Stores the option selected from the main menu.
<code>status_choice</code>	Stores the status option selected by the user.
<code>new_status</code>	Stores the new status before updating the dictionary.
<code>information</code>	Temporarily stores the location and status of a device while displaying the device list.

Programming Constructs: Several programming constructs were used to control how the application operates.

Functions: The program was divided into separate functions:

```
=====
SMART DEVICE MANAGEMENT SYSTEM
=====
1. Add New Device
2. Update Device Status
3. View All Devices
4. Search for a Device
5. Exit
=====
Choose an option from 1 to 5:
```

Each function performs one specific task. For example, **add_device()**, registers a **new device**, while **search_device()** finds and displays a device record. Using functions makes the code easier to understand, test and maintain. It also reduces repetition because each function can be called whenever it is needed.

Selection Statements

The program uses if, Elif and else statements to make decisions.

```
if user_choice == "1":
    add_device()
elif user_choice == "2":
    update_status()
else:
    print("Invalid menu option.")

elif user_choice == "3":
    view_devices()
elif user_choice == "4":
    search_device()
elif user_choice == "5":
    print("Goodbye.")
    break
```

These statements check the option selected by the user and call the correct function. They are also used to check whether a device exists and whether the user entered a valid status.

While Loop:

A while loop keeps the application running

while True:

```
    display_menu()
```

```
    user_choice = input("Choose an option: ")
```

The menu continues to appear after each task. The loop only stops when the user selects option 5, which runs the break statement.

This is appropriate because staff may need to complete several tasks during the same session.

For Loop:

A for loop is used to display every device stored in the dictionary:

```
for device_name, information in devices.items():
```

```
    print("Device name:", device_name)
```

```
    print("Location:", information["location"])
```

```
    print("Status:", information["status"])
```

The loop goes through each device record and displays its name, location and status.

Input Validation and Error Handling

The application checks user input to reduce mistakes. For example, it prevents empty device names and locations:

```
if device_name == "" or location == "":
```

```
    print("Device name and location cannot be empty.")
```

```
    return
```

The system also checks whether a device already exists:

```
if device_name in devices:
```

```
    print("This device already exists.")
```

```
    return
```

Invalid menu options and status selections produce clear messages instead of causing the program to stop. The program does not currently require try and except because menu entries are handled as strings. However, exception handling could be added in a future version if the application processes numbers, files or database connections.

Python Modules and Libraries

No external Python modules or libraries were required for this version of the project. The application only uses Python's built-in features, including:

`input()` to receive information from staff;

`print()` to display menus and results;

dictionaries to store device records;

functions to separate the program into manageable sections;

loops and conditional statements to control the program.

This decision keeps the software simple and means that users do not need to install additional libraries before running it in Thonny.

A future version could use the built-in json module to save device information permanently:

```
import json
```

However, the current version stores information temporarily in the dictionary. The records are lost when the application closes.

Dictionary and Add Device Function

```
def add_device():
    """Add a new smart device to the dictionary."""

    print("\n--- ADD NEW DEVICE ---")

    # Collect device information from the user
    device_name = input("Enter device name: ").strip()
    location = input("Enter room/location: ").strip()

    # Check whether the fields are empty
    if device_name == "" or location == "":
        print("Error: Device name and location cannot be empty.")
        return
```

Figure 3: Dictionary and add-device function

The code creates an empty dictionary called `devices`. The `add_device()` function collects the device name and location from the user. The `strip()` method removes unnecessary spaces, while the validation prevents empty information from being stored.

Saving Device Information

```
39     if device_name in devices:
40         print("Error: This device already exists.")
41         return
42
43     # Store the device with offline as its default status
44     devices[device_name] = {
45         "location": location,
46         "status": "offline"
47     }
48
49     print("Device added successfully.")
```

Figure 4: Storing a new device in the dictionary

This section checks whether the device has already been registered. If it does not exist, the program stores its location and gives it the default status of offline. This avoids duplicate device records.

Update Status Function

```
61     if device_name not in devices:
62         print("Error: Device not found.")
63         return
64
65     print("\nSelect the new status:")
66     print("1. Online")
67     print("2. Offline")
68     print("3. Maintenance")
69
70     status_choice = input("Enter option 1, 2 or 3: ").strip()
71
72     # Convert the selected number into a device status
73     if status_choice == "1":
74         new_status = "online"
75
76     elif status_choice == "2":
77         new_status = "offline"
78
79     elif status_choice == "3":
80         new_status = "maintenance"
81
82     else:
83         print("Error: Invalid status option.")
84         return
85
```

Figure 5: Device-status validation and selection

The function first checks whether the entered device exists. The user can then choose between online, offline and maintenance. Conditional statements are used to convert the selected number into the correct device status.

View and Search Functions

```
104     for device_name, information in devices.items():
105         print("\nDevice name:", device_name)
106         print("Location:", information["location"])
119     if device_name in devices:
120         print("\nDevice found.")
121         print("Device name:", device_name)
122         print("Location:", devices[device_name]["location"])
123         print("Status:", devices[device_name]["status"])
124
125     else:
126         print("Error: Device not found.")
127
```

Figure 6: Viewing and searching for device records

The for loop displays every device stored in the dictionary. The search function checks whether the entered device name exists and displays its location and status. If no matching record is found, the program displays an appropriate message.

Main Program Loop

```
129 def main():
130     """Control the main operation of the program."""
131
132     # Keep displaying the menu until the user chooses Exit
133     while True:
134
135         display_menu()
136
137         user_choice = input("Choose an option from 1 to 5: ").strip()
138
139         if user_choice == "1":
140             add_device()
141
142         elif user_choice == "2":
143             update_status()
144
145         elif user_choice == "3":
146             view_devices()
147
148         elif user_choice == "4":
149             search_device()
150
151         elif user_choice == "5":
152             print("\nGoodbye.")
153             print("The program has ended.")
154             break
155
156         else:
157             print("Invalid menu option.")
```

Figure 7: Main menu and program-control loop

The `main()` function controls the application. A while loop repeatedly displays the menu and uses conditional statements to call the selected function. The `break` statement ends the program when the user chooses option five.

Annotated Code Listing

The source code contains comments and docstrings to explain the purpose of important sections. Consistent indentation, spacing and meaningful variable names were used to improve readability.

```
6 # Dictionary used to store all registered devices
7 devices = {}
8
9
10 def display_menu():
11     """Display the main menu options."""
12
13     print("\n=====")
14     print(" SMART DEVICE MANAGEMENT SYSTEM")
15     print("=====")
16     print("1. Add New Device")
17     print("2. Update Device Status")
18     print("3. View All Devices")
19     print("4. Search for a Device")
20     print("5. Exit")
21     print("=====")
22
23
24 def add_device():
25     """Add a new smart device to the dictionary."""
26
27     print("\n--- ADD NEW DEVICE ---")
28
```

```
29 # Collect device information from the user
30 device_name = input("Enter device name: ").strip()
31 location = input("Enter room/location: ").strip()
32
33 # Check whether the fields are empty
34 if device_name == "" or location == "":
35     print("Error: Device name and location cannot be empty.")
36     return
37
38 # Prevent the same device from being added twice
39 if device_name in devices:
40     print("Error: This device already exists.")
41     return
42
43 # Store the device with offline as its default status
44 devices[device_name] = {
45     "location": location,
46     "status": "offline"
47 }
48
49 print("Device added successfully.")
```

The annotated listing demonstrates good coding practices through comments, meaningful function and variable names, consistent indentation and input validation. These practices make the program easier to understand, debug and improve.

3. Testing and Debugging Report

Program testing is an important stage of software development because it helps developers confirm that the application works as expected. Testing can identify problems before the system is used by real users. It also verifies that valid information is processed correctly and that incorrect inputs do not cause the application to stop unexpectedly [6].

For the Smart Device Management System, testing was used to check the main functions, including adding devices, updating their status, viewing the device list, searching for devices and exiting the program. Testing also helped confirm that the validation messages appeared when users entered empty information, duplicate devices or invalid menu options.

A well-tested program is generally more reliable, accurate and easier to maintain. However, testing cannot prove that software has no errors. It provides evidence that the program works correctly under the conditions that were tested [6].

3.2 Types of Programming Errors

Programming errors can prevent a program from running correctly or cause it to produce an incorrect result. The main types of errors are syntax errors, runtime errors and logical errors [7]

Syntax errors happen when the code does not follow Python's rules. For example, forgetting the colon after an if statement prevents the program from running.

```
if user_choice == "1":  
    add_device()
```

Runtime errors happen while the program is running. For example, using a variable that has not been defined can produce a `NameError`. These errors can be corrected by reading the error message and checking the affected line.

Logical errors occur when the program runs but produces the wrong result. For example, assigning "offline" when the user selects the "online" option would be a logical error. These errors are normally identified by comparing the expected result with the actual result during testing.

During this project, input validation was also used to prevent empty device names, duplicate records and invalid menu or status options [5].

3.3 Test Cases

Test ID	Function Tested	Test Input	Expected Result	Actual Result
TC01	Add a valid device	Projector B2, Room 4	Device is added with the default status offline.	Pass
TC02	Empty device name	Empty name, Room 4	Error message states that the name and location cannot be empty.	Pass
TC03	Add a duplicate device	Add Projector B2 twice	The second entry is rejected with a duplicate-device message.	Pass
TC04	Update an existing device	Projector B2, status option 1	Device status changes from offline to online.	Pass
TC05	Update a device that does not exist	Thermostat C5	Message displays Device not found.	Pass
TC06	Enter an invalid status	Projector B2, status option 8	Message displays Invalid status option.	Pass
TC07	View the device list	Select option 3 after adding a device	The device name, location and status are displayed.	Pass
TC08	View an empty device list	Select option 3 before adding a device	Message displays No devices have been added.	Pass
TC09	Search for an existing device	Search for Projector B2	The correct device details are displayed.	Pass
TC10	Search for a missing device	Search for Smart Light A9	Message displays Device not found.	Pass
TC11	Invalid main-menu choice	Enter option 9	Message asks the user to select a number from 1 to 5.	Pass
TC12	Exit the application	Select option 5	The program displays a goodbye message and ends.	Pass

Table 1: Test Cases for the Smart Device Management System

3.4 Testing Screenshots

```
Choose an option from 1 to 5: 1

--- ADD NEW DEVICE ---
Enter device name: Projector B2
Enter room/location: Room 4
Device added successfully.
Default status: offline
```

Figure 8: Successful addition of a new device

This test demonstrates that the program accepts a valid device name and location. The device was successfully stored with the default status of offline.

```
=====
Choose an option from 1 to 5: 1

--- ADD NEW DEVICE ---
Enter device name:
Enter room/location: room 4
Error: Device name and location cannot be empty.
```

Figure 9: Validation of an empty device name

This test confirms that the application does not accept incomplete device records. A clear error message is displayed, and the user is returned to the main menu.

```
=====
Choose an option from 1 to 5: 1

--- ADD NEW DEVICE ---
Enter device name: project b2
Enter room/location: room 4
Error: This device already exists.
```

Figure 10: Prevention of duplicate device records

The program checks the dictionary before saving a device. This prevents the same device name from being registered more than once.

```
=====
Choose an option from 1 to 5: 2

--- UPDATE DEVICE STATUS ---
Enter device name: project b2

Select the new status:
1. Online
2. Offline
3. Maintenance
Enter option 1, 2 or 3: |
```

```
Enter option 1, 2 or 3: 1
Device status updated successfully.
New status: online
```

Figure 11: Successful update of a device status

This test demonstrates that the status of an existing device can be changed successfully. The application converted option 1 into the online status.

```
-----
Choose an option from 1 to 5: 4

--- SEARCH DEVICE ---
Enter device name to search: Smart Light A9
Error: Device not found.
```

Figure 12: Search validation for a missing device

The application handled an unsuccessful search without stopping. It displayed a suitable error message and returned the user to the main menu.

```
=====
Choose an option from 1 to 5: 3

--- DEVICE LIST ---

Device name: project b2
Location: room 4
Status: online
-----
```

Figure 13: Display of all registered devices

This test confirms that the program retrieves and displays the correct name, location and status of the stored device.

4. Evaluation

4.1 Challenges Faced

Challenges Faced

One of the main challenges was deciding how to organise device information. Initially, a list was considered, but a dictionary was more appropriate because each device could be identified using its name. The location and status could then be stored as values connected to that name.

Another challenge was creating a menu that continued running after each action. This was solved by using a while loop. The loop repeatedly displays the main menu and only ends when the user selects the exit option.

Input validation was also challenging because the application needed to handle empty fields, duplicate devices, missing devices and invalid options. Conditional statements and the return statement were used to stop unsuitable information from being processed.

Indentation and variable-name errors were experienced during coding. These were addressed by reading the Thonny error messages, checking the affected lines and testing each function separately before running the complete application [4].

4.2 Future Improvements

Future Improvements

The current application meets the basic requirements, but several improvements could be introduced.

The most important improvement would be permanent data storage. At present, the dictionary only stores information while the program is running. When the application closes, all device records are lost. A future version could use a JSON file, CSV file or database to preserve the information [8].

The program could also include options to delete a device, update its room location and record when it was last used. These features would provide staff with more control over the device inventory.

Another improvement would be a graphical user interface. A future version could use a Python library such as Tkinter to provide buttons, text boxes and visual menus instead of a command-line interface.

Security could also be improved by adding staff login details and different access levels. For example, ordinary staff could view devices, while authorised administrators could add, update or delete records.

The system could eventually connect directly to smart classroom equipment and automatically collect connectivity information. However, this would require network access, additional security controls and more advanced programming [9].

4.3 Self-Reflection

Self-Reflection

This project helped me understand how a software solution develops from requirements into a working Python program. I learned that planning is important before coding. The flowchart and pseudocode helped me organise the program logic and identify the functions that were needed.

I improved my understanding of Python dictionaries, variables, functions, loops and conditional statements. I also learned how input validation can prevent incorrect information from being stored [8].

Testing helped me understand the difference between syntax, runtime and logical errors. Instead of only checking whether the program opened, I learned to test individual functions using both correct and incorrect inputs.

I also became more confident in using Thonny and reading Python error messages. At the beginning, indentation and variable names were difficult, but regular testing helped me recognise and correct these problems.

Overall, the project improved my problem-solving and debugging skills. I now understand that software development is an iterative process that involves planning, coding, testing, correcting and evaluating the final solution [10].

conclusion

The Smart Device Management System was developed to help Bright Minds Academy manage information about its smart classroom equipment. The Python application allows staff to add devices, update their status, view all registered devices and search for a specific device.

The solution uses a dictionary to store device names, locations and statuses. Functions were used to separate the main tasks, while loops and conditional statements controlled the program flow. Input validation was implemented to manage empty fields, duplicate devices, invalid options and unsuccessful searches.

Testing demonstrated how the program responds to both correct and incorrect user inputs. The results showed that the main functional requirements were met. However, the current version does not permanently save device information and does not directly control smart devices.

Despite these limitations, the project provides a simple, organised and user-friendly solution for Bright Minds Academy. It also creates a suitable foundation for future improvements, including permanent storage, login security, a graphical interface and real-time device monitoring.

Appendices

Link: https://github.com/Foxgirl70/Elaine-Lopes-bright_minds_devices.py-.git

References:

- [1] J. T. Force, “No title,” 2020.
- [2] D. Gobov and O. Zuieva, “Software quality attributes in requirements engineering,” vol. 17, no. 4, pp. 38–48, 2025.
- [3] W. Xu, D. Furie, M. Mahabhaleshwar, B. Suresh, and H. Chouhan, “Applications of an interaction, process, integration and intelligence (IPII) design approach for ergonomics solutions,” 2019.
- [4] J. T. Force, “No title,” 2020.
- [5] J. C. Erb, “A Decade of Programming Languages: Trends in Popularity and Influence,” 2026.
- [6] M. Senbagavalli, S. Subashini, V. Revathi, and V. A. Kumar, *Problem Solving and Python Programming*. BR Publications, 2026.
- [7] I. Glossary, “International software testing qualification board,” 2016.
- [8] C. Pacheco, I. Garcia, J. A. Calvo-Manzano, and M. Reyes, “Measuring and improving software requirements elicitation in a small-sized software organization: a lightweight implementation of ISO/IEC/IEEE 15939: 2017—systems and software engineering—measurement process,” vol. 28, no. 2, pp. 257–281, 2023.
- [9] G. Van Rossum, B. Warsaw, and N. Coghlan, “PEP 8—style guide for python code,” vol. 1565, p. 28, 2001.
- [10] M. Senbagavalli, S. Subashini, V. Revathi, and V. A. Kumar, *Problem Solving and Python Programming*. BR Publications, 2026.