



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Coursework – Programming Assignment with Associated Documentation

Smart Home Device Manager for Bright Minds Academy

Student Name: Sonya Vakarelova

Student Number: 2426532

Module: ACCA5036 -Scripting and Programming Concepts

Course: BSc Applied Computing

Lecturer: Dr Mobolanle Bello

Submission Date: 10.07.2026

GitHub Repository: [smart-home-device-manager/assessment at main · sonyavakarelova/smart-home-device-manager](https://github.com/sonyavakarelova/smart-home-device-manager/assessment%20at%20main)

Contents

1.0 INTRODUCTION.....	2
2.0 REQUIREMENT ANALYSIS.....	3
3.0 SOFTWARE DESIGN AND DEVELOPMENT	5
3.1 Design.....	6
3.1.1 Flowchart.....	6
3.1.2 Pseudocode	8
3.2 Implementation Decision	13
3.2.1 Data Structures	13
3.2.2 Programming Constructs.....	14
3.2.3 Python Modules and Libraries	16
4.0 TESTING AND DEBUGGING	18
4.1 Role of Testing in Software Environment.....	18
4.2 Types of Errors in Programming	19
4.3 Example of Test Cases.....	19
5.0 EVALUATION	22
References	24

1.0 INTRODUCTION

Smart technology is becoming increasingly popular in schools, businesses and homes because it helps improve efficiency, security and the management of connected devices. Many educational organisations are now introducing smart devices such as projectors, smart lights, thermostats and interactive whiteboards to create a better learning environment. However, managing these devices manually can become difficult, especially when staff need to know where a device is located, whether it is working correctly or when it was last used.

Bright Minds Academy is a private tutoring centre that is introducing smart classroom technology but currently relies on a manual process to manage its devices. This makes it difficult for staff to keep accurate records of each device, monitor its status and quickly identify devices that require maintenance. As the number of smart devices increases, a more organised and efficient solution is needed.

The aim of this project is to develop a **Smart Home Device Manager** using Python to help staff manage smart devices more effectively. The system provides a simple menu-driven interface that allows users to add new devices, update their status, search for a specific device and display all stored devices. The application stores information using Python dictionaries and applies programming concepts such as variables, functions, loops, selection statements and input validation. Although the system is a prototype and does not use a database, it demonstrates how a simple software solution can improve device management and support the daily operation of Bright Minds Academy.

2.0 REQUIREMENT ANALYSIS

The Smart Home Device Manager has been developed to provide Bright Minds Academy with a simple and effective way to manage smart classroom devices. The current process relies on staff manually keeping track of devices, making it difficult to monitor their location, status and last used date. As more smart devices are introduced into the academy, this manual approach becomes less efficient and increases the possibility of

errors. The proposed system aims to improve the organisation of device information by storing it in one place and making it easier for staff to access and update records.

The system includes several **functional requirements** to meet the needs of the academy. Users can **view all registered devices**, allowing them to see the device name, location, status and last used date. Staff can **add a new device** by entering the required information, ensuring that new equipment can easily be recorded. The system also allows users to **update the status** of an existing device by changing it to **Online**, **Offline** or **Under Maintenance**. In addition, users can **search for a specific device** using its name, making it quicker to find device information when needed. To improve the functionality of the application, additional features have been included that allow users to **delete a device**, **view only online devices** and **display devices according to their current status**.

The system also has several **non-functional requirements**. It should be simple and easy to use, with a clear menu that allows staff to navigate the program without difficulty. The application should respond quickly to user input, display information clearly and provide input validation to reduce user errors. It should also be reliable and easy to maintain, allowing future improvements to be added if required.

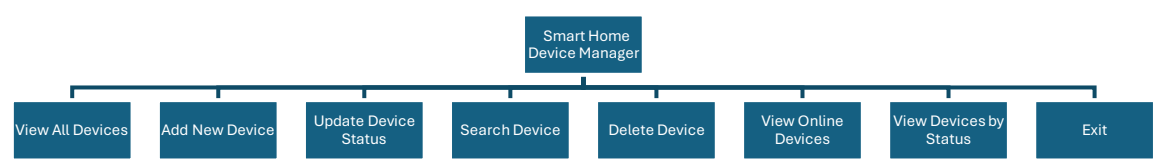
Table 2.1 MoSCoW Prioritisation Table for Home Smart Device Manager

Must Have	Should Have	Could Have	Won't Have
View all devices	Delete device	View online devices	Database integration
Add new device		View devices by status	User login system
Update device status			Cloud storage
Search device			Mobile application

The **MoSCoW prioritisation technique** was used to identify and organise the features of the Smart Home Device Manager according to their level of importance. The **Must Have** features are the core functions required for the system to operate successfully, including viewing devices, adding new devices, updating device status and searching for devices. The **Should Have** features improve the usability of the system but are not essential for its basic operation. The **Could Have** features provide additional functionality, allowing users

to filter devices more efficiently. Finally, the **Won't Have** features, such as database integration, user login, cloud storage and a mobile application, were excluded because they are outside the scope of this prototype and are not required by the assignment.

Table 2.2 Structure Diagram of the Smart Home Device Manager



The structure diagram illustrates the overall organisation of the Smart Home Device Manager. The application is organised into several user-defined functions, each responsible for a specific task. These functions include viewing all devices, adding new devices, updating device status, searching for a device, deleting a device, viewing online devices, viewing devices by status and exiting the program. This modular design improves the readability of the code, makes the program easier to test and maintain, and allows each function to perform a single responsibility.

3.0 SOFTWARE DESIGN AND DEVELOPMENT

Software design and development are important stages of the Software Development Life Cycle (SDLC) because they provide a structured approach to creating a reliable and functional software solution. Before writing any code, it is important to understand the system requirements and plan how the application will work. A well-designed system helps reduce programming errors, improves code organisation and makes the software easier to test, maintain and update. Planning the design before implementation also

ensures that the final application meets the needs of the users and fulfils the project requirements.

For this project, the Smart Home Device Manager was designed to provide Bright Minds Academy with a simple and effective solution for managing smart classroom devices. During the design stage, the program structure, user interface and system functions were planned before development began. Flowcharts and pseudocode were used to organise the program logic and demonstrate how each function operates. The application was then implemented in Python using dictionaries to store device information together with user-defined functions, selection statements, loops and input validation. Throughout the development process, each function was tested to ensure that it worked correctly and met the requirements identified during the analysis stage. Following this structured approach helped produce a program that is organised, easy to use and suitable for the needs of Bright Minds Academy.

3.1 Design

The design stage was used to plan how the Smart Home Device Manager would work before the Python code was written. This helped to organise the program into clear sections and made it easier to understand how the user would interact with the system. The design focuses on creating a simple menu-driven application where each option performs a specific task.

The program starts by displaying the main menu. The user can then choose to view all devices, add a new device, update a device status, search for a device, delete a device, view online devices, view devices by status or exit the system. Each menu option is linked to a separate user-defined function, which supports modular programming and makes the code easier to test and maintain.

Flowcharts and pseudocode were used during the design stage to represent the logic of the program. The flowchart provides a visual representation of the steps followed by the system, while the pseudocode explains the program logic in a simple written format before it is translated into Python code.

3.1.1 Flowchart

A flowchart is a graphical representation of an algorithm or process that uses standard symbols to illustrate the sequence of steps required to solve a problem [1].

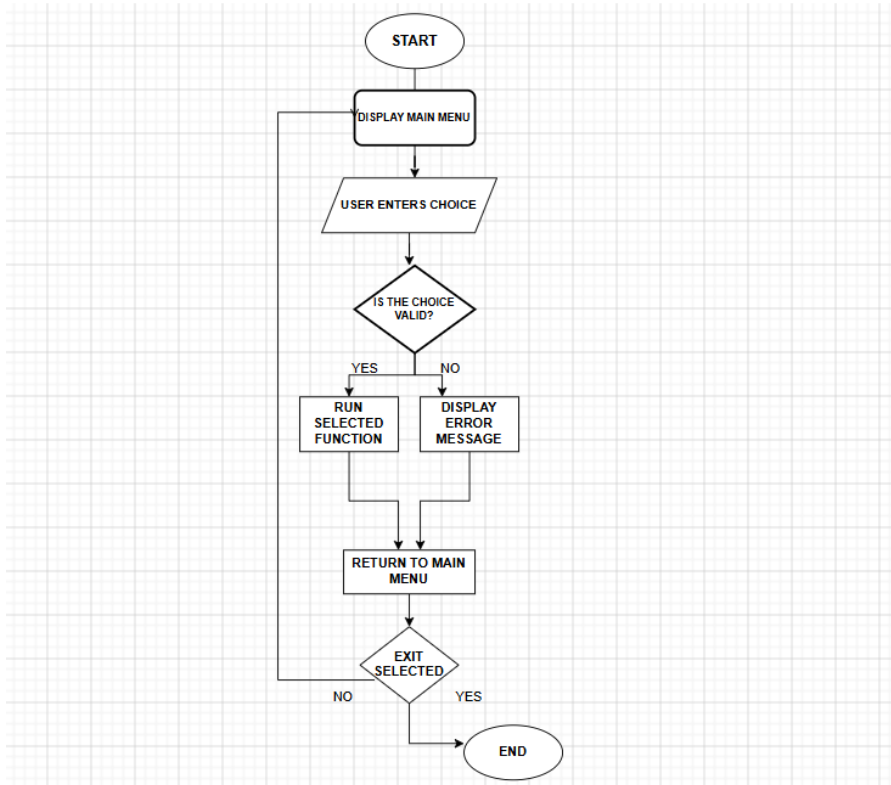


Figure 3.1.1 Flowchart for Smart Home Device Manager

For this project, the flowchart was created to support the design of the **Smart Home Device Manager** developed for Bright Minds Academy. The purpose of the flowchart is to present a clear visual representation of how the program operates from the moment it starts until it is closed. It shows the sequence of actions performed by the system, how user input is processed and how different outputs are generated depending on the option selected. Using a flowchart before writing the Python code made it easier to organise the program into separate user-defined functions and ensured that the application followed a logical structure.

The flowchart begins with the **Start** symbol, which represents the beginning of the program. The system first initialises the device dictionary, which contains information about the smart devices, including their location, status and last used date. After the data has been loaded, the program displays the **main menu**, allowing the user to choose one of the available options. The menu includes viewing all devices, adding a new device, updating a device status, searching for a device, deleting a device, viewing online devices, viewing devices according to their status and exiting the application.

The next step in the flowchart is the **user input**. The user enters a number between **1 and 8** to select the required function. The program then checks whether the input is valid. This decision stage is represented by a decision symbol in the flowchart. If an invalid option is entered, the system displays an **error message** informing the user that the choice is incorrect. The program then returns to the main menu, allowing the user to enter another option without closing the application.

If the user enters a valid option, the corresponding function is executed. Depending on the selected menu option, the program performs different tasks. For example, it may display all stored devices, allow a new device to be added, update the status of an existing device, search for a specific device, delete a device from the system or display only devices that match a selected status. Once the selected function has finished, the program returns to the main menu, allowing the user to continue managing the smart devices.

The final decision in the flowchart checks whether the user has selected the **Exit** option. If the answer is **No**, the program continues by returning to the main menu and waiting for another user input. If the answer is **Yes**, the system displays a closing message and reaches the **End** symbol, which terminates the program.

Overall, the flowchart provides a clear and organised representation of the Smart Home Device Manager. It demonstrates how user inputs are processed, how decisions are made and how appropriate outputs are produced. It also helped simplify the development process by ensuring that the program logic was planned before implementation, making the application easier to develop, test and maintain.

3.1.2 Pseudocode

Pseudocode is an informal description of a computer program or algorithm that uses structured English to describe the steps of a solution without following the syntax of a specific programming language [2]

START

Create dictionary called devices

REPEAT

 Display Main Menu

 Read user choice

 IF choice = 1 THEN

 Display all devices

 Display room, status and last used date

 ELSE IF choice = 2 THEN

 Read device name

 IF device name is empty THEN

 Display error message

 ELSE IF device already exists THEN

 Display device already exists

 ELSE

 Read room

 Read last used date

 Add new device to dictionary

 Set status to Offline

```

        Display success message
    END IF

ELSE IF choice = 3 THEN
    Read device name

    IF device exists THEN

        Display status options

        Read status choice

        IF status choice = 1 THEN
            Set status to Online
        ELSE IF status choice = 2 THEN
            Set status to Offline
        ELSE IF status choice = 3 THEN
            Set status to Under Maintenance
        ELSE
            Display invalid choice
        END IF

        Read last used date

        Update device information

        Display success message

    ELSE
        Display device not found
    END IF

ELSE IF choice = 4 THEN
    Read device name

    IF device exists THEN
        Display device name
        Display room
    
```

```

        Display status
        Display last used date

    ELSE
        Display device not found
    END IF

ELSE IF choice = 5 THEN
    Read device name

    IF device exists THEN
        Delete device
        Display success message

    ELSE
        Display device not found
    END IF

ELSE IF choice = 6 THEN
    Display all online devices

ELSE IF choice = 7 THEN
    Read selected status

    Display all devices with selected status

ELSE IF choice = 8 THEN
    Display exit message

    End program

ELSE
    Display invalid menu choice

END IF

UNTIL user selects Exit

END

```

Figure 3.1.2: Pseudocode for the Smart Home Device Manager

Purpose of the pseudocode

The pseudocode was created before writing the Python program to plan the logic of the Smart Home Device Manager. It provides a structured outline of the program's behaviour and helps organise the application into separate user-defined functions. Writing pseudocode before implementation made it easier to understand how information would be processed and how each menu option would operate. It also reduced the likelihood of programming errors because the program logic could be reviewed before the Python code was developed.

The pseudocode focuses on the main features of the application, including displaying the menu, processing user input and carrying out the selected function. Each major task is represented using simple English statements that are easy to understand. This makes the program design clearer and provides a useful guide during implementation.

Explanation of the pseudocode logic

The pseudocode begins by **starting the program** and loading the smart device information into memory. The system then enters a loop where the **main menu** is displayed. The user is asked to enter a menu option between **1 and 8**. This input is read by the program and checked using selection statements.

If the user selects **View All Devices**, the system displays every smart device together with its location, status and last used date. If **Add New Device** is selected, the user enters the device name, room location and last used date before the information is stored in the dictionary. When **Update Device Status** is selected, the system asks for the device name and allows the user to change its status to **Online**, **Offline** or **Under Maintenance**.

If the user chooses **Search Device**, the program searches the dictionary for the requested device. If the device is found, its details are displayed. If it cannot be found, an appropriate message is shown. Similar logic is used for deleting devices and displaying

devices that match a particular status. Each function finishes by returning the user to the main menu, allowing additional tasks to be completed without restarting the program.

The program continues to repeat this process until the user selects the **Exit** option. When this happens, the system displays a closing message and terminates the application. This loop ensures that users can perform multiple operations during a single session.

The pseudocode translates directly into the Python implementation. Each section of the pseudocode becomes a separate user-defined function, while the decision statements are implemented using **if**, **elif** and **else** statements. The repeating behaviour of the menu is achieved using a **while** loop, and the device information is stored using a Python **dictionary**. As a result, the pseudocode provides a clear bridge between the initial design and the final Python program, ensuring that the implementation follows a logical and well-organised structure.

3.2 Implementation Decision

3.2.1 Data Structures

Data structures are used to organise and store data in a way that makes it easier for a program to access, update and process information efficiently. Choosing the right data structure can improve a program's performance and make the code easier to read and maintain. In Python, common data structures include lists, tuples, dictionaries and sets, each of which is suitable for different programming tasks [3].

For the Smart Home Device Manager, a **dictionary** was selected as the main data structure because it provides a simple and efficient way to organise information about each smart device. A dictionary stores information using **key-value pairs**, where each device name acts as a unique key. The associated values contain additional information about the device, including its **room**, **status** and **last_used** date. This allows all related information for a single device to be stored together, making the data easier to access and manage throughout the program.

The dictionary was chosen instead of using several separate lists because it allows information to be accessed directly by using the device name. This makes searching for a device, updating its status and deleting a device much more efficient. For example, when a user searches for "**Smart Light A1**", the program can immediately retrieve the corresponding information from the dictionary without searching through multiple lists. Similarly, updating the status of a device only requires changing one value within the dictionary while keeping the remaining information unchanged. This approach simplifies the program logic and makes the code easier to read, understand and maintain.

Several variables are used throughout the program to process user input and control the application's behaviour. The **devices** variable stores the main dictionary containing all smart device information. The **choice** variable stores the menu option selected by the user and determines which function will be executed. Within several functions, the **name** variable temporarily stores the device name entered by the user. The **room** variable stores the location of the device, while **last_used** records the date when the device was last used. The **selected_status** variable stores the status selected when filtering devices, and the **found** Boolean variable is used to check whether matching devices exist before displaying the results. During loops, the variables **device** and **details** are used to retrieve information from the dictionary and display it to the user.

Overall, using a dictionary was an appropriate choice for this project because it provides a clear, organised and efficient way of storing smart device information. Together with meaningful variable names, the selected data structure supports all of the main functions of the Smart Home Device Manager, including adding, updating, searching, deleting and displaying devices. This approach also makes the program easier to maintain and provides flexibility for future improvements, such as adding more information about each smart device without significantly changing the overall structure of the application.

3.2.2 Programming Constructs

Programming constructs are the fundamental building blocks used to control the execution and behaviour of a computer program [4]. They provide the logical structure

needed to perform tasks, make decisions and repeat actions. Common programming constructs include sequence, selection, iteration and modular programming through functions. These constructs enable developers to write organised, efficient and maintainable code while ensuring that a program performs the required tasks correctly.

The Smart Home Device Manager uses several programming constructs to provide a structured and user-friendly application. The first construct used is **sequence**, where statements are executed in a logical order. When the program starts, the smart device dictionary is created, the main menu is displayed, and the system waits for the user to select an option. Each function then performs a series of instructions in the correct order before returning control to the main menu. Using sequence ensures that every task is completed systematically and that the program behaves predictably.

The program also makes extensive use of **selection** through **if**, **elif** and **else** statements. These decision-making statements allow the application to respond differently depending on the user's input. For example, after the user enters a menu option, the program checks the value stored in the **choice** variable and executes the corresponding function. Selection statements are also used throughout the application to validate user input. For example, the **add_device()** function checks whether the device name is empty or already exists before adding a new record to the dictionary. Similarly, the **update_status()**, **search_device()** and **delete_device()** functions first check whether the selected device exists before performing the requested operation. This helps prevent errors and ensures that only valid actions are carried out.

Another important programming construct used in the application is **iteration**. A **while True** loop is used in the **main()** function to keep the program running until the user selects the **Exit** option. This allows users to perform multiple operations during a single session without restarting the application. In addition, **for** loops are used when displaying information stored in the dictionary. For example, the program loops through the **devices** dictionary to display all smart devices, list only online devices or display devices with a selected status. Using loops reduces code repetition and allows the program to process multiple records efficiently.

The application also follows the principles of **modular programming** by using **user-defined functions**. Each major feature has been placed into a separate function, including

display_menu(), view_devices(), add_device(), update_status(), search_device(), delete_device(), view_online_devices(), view_devices_by_status() and exit_program().

Dividing the program into smaller functions improves readability, simplifies testing and makes the code easier to maintain. It also allows individual functions to be modified without affecting the rest of the application.

Basic **error handling and input validation** have also been implemented throughout the program. Validation checks prevent users from entering an empty device name, adding duplicate devices or selecting invalid menu options. Additional checks ensure that a device exists before it can be updated, searched or deleted. When invalid input is detected, the program displays an appropriate error message and allows the user to try again rather than terminating unexpectedly. These validation techniques improve the reliability and usability of the Smart Home Device Manager.

Overall, the combination of sequence, selection, iteration, modular programming and input validation provides a clear and logical structure for the application. These programming constructs make the software easier to understand, maintain and extend while ensuring that users can manage smart devices accurately and efficiently.

3.2.3 Python Modules and Libraries

Python modules are files that contain reusable code, including functions, classes and variables, which can be imported into a program to perform specific tasks [5]. A Python library is a collection of related modules that provides additional functionality for software development. Using modules and libraries helps programmers save time, improve code organisation and reduce duplication because existing functions can be reused instead of being written from scratch. Python includes a large **Standard Library**, which contains many built-in modules that are automatically installed with Python and are available without requiring any additional downloads.

The Smart Home Device Manager was developed using only built-in modules from the Python Standard Library. External libraries were not required because the project focuses on demonstrating fundamental programming concepts, including dictionaries, functions, loops, selection statements and input validation. The built-in modules provided all the functionality needed to develop a fully working prototype while keeping the application simple and easy to maintain.

The **os** module was used to improve the appearance of the user interface. Within the **clear_screen()** function, the `os.system()` command clears the console window after each operation. The program automatically executes the appropriate command depending on the operating system by using **cls** for Windows or **clear** for Linux and macOS. Clearing the screen removes previous information, making the program easier to read and allowing users to focus on the current menu or output.

The **time** module was imported to create short delays by using the `time.sleep()` function. These delays give users enough time to read confirmation and error messages before the next screen is displayed. For example, after adding, updating or deleting a smart device, the application pauses for two seconds before returning to the main menu. This improves the overall usability of the program and creates a smoother user experience.

The **sys** module was used to close the application safely when the user selects the **Exit** option. The `sys.exit()` function terminates the program immediately after displaying a closing message. This provides a reliable and controlled way to end the application instead of requiring the user to close the terminal window manually.

The Smart Home Device Manager was developed using **PyCharm Community Edition**, an Integrated Development Environment (IDE) designed for Python programming. Although PyCharm is not a Python library, it provided a suitable environment for writing, editing, testing and debugging the application. Features such as syntax highlighting, automatic code formatting, and error detection helped improve code quality and made the development process more efficient.

Overall, the built-in Python modules selected for this project were appropriate because they improved the functionality and usability of the application without increasing its complexity. Using the Python Standard Library together with the PyCharm development

environment enabled the Smart Home Device Manager to be developed efficiently while producing a well-structured and maintainable solution.

4.0 TESTING AND DEBUGGING

4.1 Role of Testing in Software Environment

Program testing is the process of evaluating software to determine whether it performs as expected and meets the specified requirements [6]. Testing is an important stage of the software development process because it helps identify errors, verify program functionality and ensure that the application produces the correct output for different types of user input. It also allows developers to confirm that the software is reliable, stable and suitable for its intended purpose.

Testing plays a significant role in software development because it helps improve the quality and usability of an application before it is released to users. By testing both valid and invalid inputs, developers can identify programming errors, improve input validation and ensure that the system responds appropriately in different situations. Effective testing also increases user confidence because it demonstrates that the software performs consistently and produces accurate results.

For the Smart Home Device Manager, testing was carried out throughout the development process to ensure that each function operated correctly. The application was tested using both successful and unsuccessful user inputs, including adding new devices, searching for existing and non-existing devices, updating device status, deleting devices and entering invalid menu options. The testing process confirmed that the program met the functional requirements and that the implemented validation checks handled incorrect input appropriately. Overall, testing helped improve the reliability, accuracy and usability of the application before the final version was completed.

4.2 Types of Errors in Programming

Programming errors, also known as bugs, are mistakes in source code that cause a program to produce incorrect results or fail to operate as intended [7]. Identifying and correcting these errors is an important part of software development because it improves the reliability and performance of an application.

One common type of error is a syntax error, which occurs when the rules of the programming language are not followed. Examples include incorrect indentation, missing brackets or misspelt keywords. These errors are usually detected by Python before the program runs.

A logical error occurs when the program runs successfully but produces incorrect results because the program logic is wrong. Logical errors are usually found through testing and debugging. Runtime errors occur while the program is running and are often caused by unexpected user input. In the Smart Home Device Manager, basic input validation was implemented to reduce runtime errors by checking for empty device names, duplicate devices, invalid menu choices and devices that do not exist.

4.3 Example of Test Cases

Test cases were developed to check whether each function of the Smart Home Device Manager worked correctly with both valid and invalid user input. Each test case included the test description, input data, expected result, actual result, and pass/fail outcome.

Testing was carried out in PyCharm by running the program and checking whether the output matched the expected behaviour. The test cases covered key functions such as viewing devices, adding a new device, updating status, searching for devices, deleting devices and handling invalid input.

Example test cases included adding a new device with valid details, which should display a success message, and entering an invalid menu option, which should display an error message.

Test Case ID	TC001
Test Case Title	Add New Smart Device
Objective	Verify that the system successfully adds a new smart device to the device list.
Preconditions	The Smart Home Device Manager is running.
Test Steps	1. Select Option -2 2. Add New Device – Enter a unique device name. 3. Enter the device location. 4. Enter the last used date.
Test Data	Device Name: Smart Camera Location: Library Last Used: 07/07/2026
Expected Result	The system displays the message "Device added successfully."
Pass/Fail	Pass
Comments	The device was added successfully, confirming that the system correctly stores new device information.

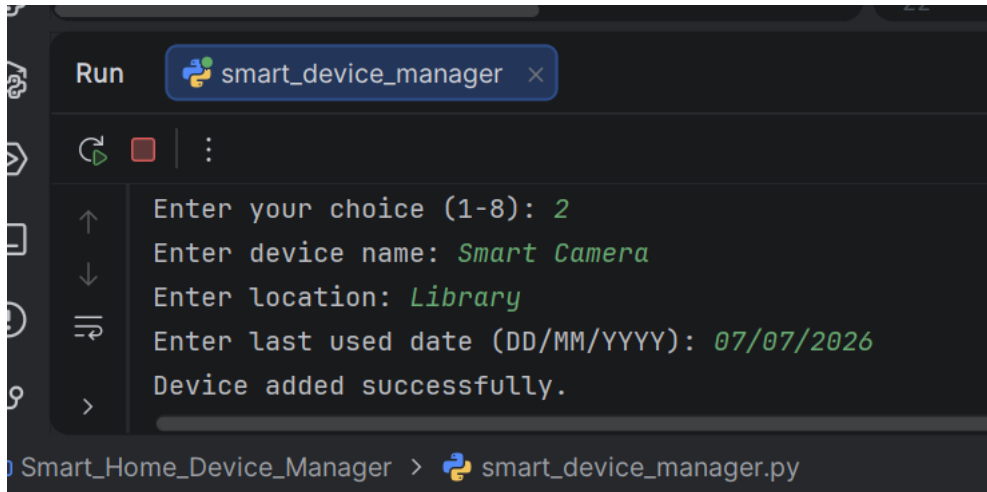
A screenshot of a terminal window titled 'Run smart_device_manager'. The terminal shows a series of prompts and user inputs: 'Enter your choice (1-8): 2', 'Enter device name: Smart Camera', 'Enter location: Library', and 'Enter last used date (DD/MM/YYYY): 07/07/2026'. The final output is 'Device added successfully.' The terminal window is part of an IDE, with a file path 'Smart_Home_Device_Manager > smart_device_manager.py' visible at the bottom.

Figure 4.1: Test Case TC001 demonstrating the successful addition of a new smart device to the Smart Home Device Manager.

Test Case ID TC002

Test Case Title Invalid Menu Choice

Objective Verify that the system correctly handles an invalid menu option.

Preconditions The Smart Home Device Manager is running.

Test Steps

1. Enter an invalid menu option.
2. Check that an error message is displayed.

Test Data 9

Expected Result The system displays "Invalid choice. Please try again."

Pass/Fail Pass

Comments The system displayed the correct error message, confirming that invalid menu input is handled correctly.

A screenshot of a Python IDE terminal window titled "smart_device_manager". The terminal shows a menu with options 7 and 8, followed by a separator line. A user input of '9' is shown, and the program responds with "Invalid choice. Please try again." The terminal text is as follows:

```
7. View Devices by Status
8. Exit
=====
Enter your choice (1-8): 9
Invalid choice. Please try again.
```

Figure 4.2: Test Case TC002 demonstrating how the program handles an invalid menu selection

5.0 EVALUATION

Challenges Faced

Developing the Smart Home Device Manager presented several challenges that helped improve my understanding of software development. One of the main challenges was selecting an appropriate data structure to store the smart device information. The program needed to store multiple pieces of information, including the device name, location, status and last used date. After researching different approaches, I decided to use a nested Python dictionary because it provided an organised and efficient way to store and retrieve data.

Another challenge was implementing the program functions correctly. Each feature, including viewing devices, adding new devices, updating device status, searching, deleting and filtering devices, had to work independently while using the same dictionary. Careful planning using a flowchart and pseudocode helped me organise the program logic before writing the code, reducing errors during implementation.

Input validation was also an important challenge. The application needed to prevent users from entering invalid information, such as empty device names, duplicate devices or invalid menu options. These problems were solved using conditional statements and

validation checks, ensuring that clear error messages were displayed whenever incorrect input was entered. Testing each function with both valid and invalid inputs helped identify minor logical issues and confirmed that the final application worked as expected.

Future Improvements

Although the Smart Home Device Manager successfully meets the objectives of this project, there are several improvements that could be made in future versions. One significant improvement would be replacing the in-memory dictionary with a database such as SQLite or MySQL. This would allow device information to be stored permanently instead of being lost when the application is closed.

Another improvement would be implementing a user login and authentication system to improve security and restrict access to authorised users. Different user roles, such as administrators and staff members, could also be introduced to provide different permissions within the application.

The current version uses a text-based interface. Developing a graphical user interface (GUI) using Python libraries such as Tkinter or PyQt would make the application more user-friendly and easier to navigate. Additional features could include automatic date validation, advanced search filters, sorting devices by location or status, and generating reports. Cloud storage or a mobile application could also be added to allow staff to access the system from different locations. These improvements would increase the usability, security and scalability of the Smart Home Device Manager while maintaining its existing functionality.

Self-Reflection

This project has significantly improved my understanding of Python programming and the software development process. Throughout the assignment, I gained practical experience in analysing user requirements, creating a MoSCoW prioritisation table,

designing flowcharts and pseudocode, implementing a working Python application and testing the final solution.

One of the most valuable lessons I learned was the importance of planning before coding. Preparing the design documents made it easier to understand the program structure and reduced mistakes during implementation. I also developed a better understanding of Python dictionaries, functions, loops, conditional statements and input validation by applying these concepts to solve a real problem.

Testing the application increased my confidence in debugging and identifying logical errors. It also showed me the importance of validating user input to improve software reliability. Overall, this project has strengthened my programming, problem-solving and analytical skills. It has also increased my confidence in developing software solutions and has provided valuable experience that will support my future studies and software development projects.

References

- [1] T. Gaddis, *Starting Out with Python*, 5th ed. Harlow, UK: Pearson, 2021.
- [2] M. Downey, *Think Python: How to Think Like a Computer Scientist*, 3rd ed. Needham, MA, USA: Green Tea Press, 2024.
- [3] E. Matthes, *Python Crash Course*, 3rd ed. San Francisco, CA, USA: No Starch Press, 2023.
- [4] A. Sweigart, *Automate the Boring Stuff with Python*, 3rd ed., 2025.
- [5] Python Software Foundation, *Python 3.13 Documentation*. Available: <https://docs.python.org/3/>.
- [6] I. Sommerville, *Software Engineering*, 11th ed. Harlow, UK: Pearson, 2021.
- [7] B. Lubanovic, *Introducing Python: Modern Computing in Simple Packages*, 3rd ed. Sebastopol, CA, USA: O'Reilly Media, 2023.
- [8] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.
- [9] J. Wengrow, *A Common-Sense Guide to Data Structures and Algorithms*, 2nd ed. Raleigh, NC, USA: Pragmatic Bookshelf, 2021.
- [10] L. Ramalho, *Fluent Python*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2022.
- [11] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, Pearson Modern Classics ed. Boston, MA, USA: Pearson, 2021.
- [12] H. Percival and B. Gregory, *Architecture Patterns with Python: Enabling Test-Driven Development, Domain-Driven Design, and Event-Driven Microservices*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2024.
- [13] A. Arya, *Anyone Can Code: Algorithmic Thinking*. Cham, Switzerland: Springer, 2023.
- [14] Python Software Foundation, *Python Tutorial*. Available: <https://docs.python.org/3/tutorial/>. [Accessed: 7-Jul-2026].

- [15] Python Software Foundation, *The Python Standard Library*. Available: <https://docs.python.org/3/library/>. [Accessed: 7-Jul-2026].
- [16] D. Beazley and B. K. Jones, *Python Cookbook*, 4th ed. Sebastopol, CA, USA: O'Reilly Media, 2025.
- [17] J. Noble, D. Streader and I. O. Gariano, "More Programming Than Programming: Teaching Formal Methods in a Software Engineering Programme," 2022.

