Munir's Smart-Home Program

Made by: Munir Abood – Student of Computer Science BSc at Eötvös Loránd University

E-mail: munircsdev@gmail.com

Relevant Course: Python

11/16/2023

User documentation

Task

In this project, I am tasked with creating a real-life Smart-Home program system using the high-level Python programming language. During my time in my Python course, I did learn a few things, but not quite the full picture. When I started making this program, that's when my knowledge and learning really took a rise and I had to teach myself a lot of new things like integrating dynamic functions, methods and instances in my main programming, learning to separate the main program with the script so that I can arrange my code systematically and learn to be organized, I learned the basics of the Tkinter GUI by python, which I used to compose my GUI and integrate it with the bank-end. Overall, this task was fun.

In a nutshell, my program is a system which users can control the Music, Lights, and the Air Conditioning in their smart home, on top of this, they can also control the properties of these devices, such as light brightness, music volume level, and the air-conditioning temperature. It is not ready to deploy to a real-life smart-home system, but the logic and codebase is enough to implement it in testing-environment.

Usage

Structure of the smarthome.py file (main file)

The program can be found in the archived file by the name "smarthome.py" in my folder. You can start the program by simply running the code in the editor or by hitting CNTRL + F5 (Assuming you are using Visual Studio)

AutomationSystem Class

This class represents the main automation system that manages different devices. To fulfil the Central Automation System requirement, this class is created for that.

Properties:

devices: A list to store instances of various devices.

Methods:

init(self): Constructor method, initializes an empty list for devices.

discover_device(self, device): Adds a device to the list of devices.

execute_automation_tasks(self): Iterates through the list of devices and executes automation tasks specific to each device type. For Music devices, it plays a random song and prints the current song. For Lights devices, it simulates turning lights on/off based on the time of day. For AC devices, it updates the temperature.

run_simulation(self, interval=5): Runs the automation simulation in a loop. It repeatedly executes automation tasks with a specified time interval.

Music Class

This class represents a Music device.

Properties:

current_song: The currently playing song.

playlist: List of songs available to play.

play_music: Boolean indicating whether music is playing.

volume: The volume level.

next_song: Placeholder for next song.

previous_song: Placeholder for the previous song.

Methods:

init(self, playlist, current_song=None, volume=50, play_music=False, next_song=None, previous_song=None): Constructor method, initializes the Music device properties.

random_song(self): Plays a random song from the playlist.

print_current_song(self): Prints the currently playing song.

Lights Class

This class represents a Lights device.

Properties:

lights_on: Boolean indicating whether lights are on.

lights_off: Boolean indicating whether lights are off.

brightness: The brightness level.

Methods:

init(self, brightness, lights_on=False, lights_off=True): Constructor method, initializes the Lights device properties.

after_sunset(self, current_time): Simulates turning on the lights after sunset by comparing the current time with a predefined sunset time.

after_sunrise(self, current_time): Simulates turning off the lights after sunrise by checking the current time.

print_state(self): Prints the current state of the lights (on/off).

Ac Class

This class represents an Air Conditioning (AC) device.

Properties:

on: Boolean indicating whether the AC is on.

off: Boolean indicating whether the AC is off.

temperature: The current temperature setting.

Methods:

init(self, on=True, off=False, temperature=22): Constructor method, initializes the AC device properties.

updatetemperature(self): Updates the temperature of the AC in a loop while it is on. The temperature is set to a random value within a range, and the current temperature is printed. The loop continues until the AC is turned off.

Main Block

The code in the if __name__ == "__main__": block creates instances of the AutomationSystem, Music, Lights, and AC classes. It then adds these devices to the automation system and runs the simulation loop.

This script simulates a basic smart home system with music playback, lights control, and air conditioning functionality.

Sample Program Output

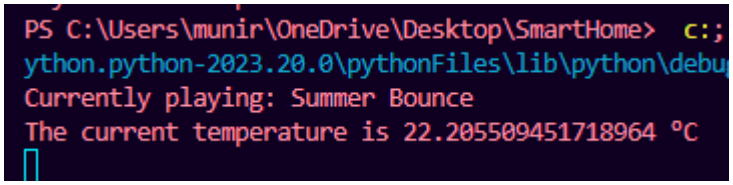
Iteration 1:

- Music System: “Critical – Cyberspeed mix”.
- Lights Controller: Lights turned on after sunset.
- AC Controller: Current temperature is 23.5 °C.

Iteration 2:

- Music System: “SELF LUH”.
- Lights Controller: Lights turned off after sunrise.
- AC Controller: Current temperature is 21.8 °C.

Sample output



Possible errors

Undefined Classes:

If the classes Music, Lights, Ac, and AutomationSystem are not defined or imported correctly, you will encounter NameError when trying to create instances of these classes.

Incorrect Usage of Classes:

If the properties or methods of the classes are not used correctly, it could lead to unexpected behavior or errors. For example, if a property is accessed before it is initialized, it may result in AttributeError.

Incorrect Conditions:

In the after_sunset method of the Lights class, the comparison if now >= sunset_time may not work correctly if the now and sunset_time are not in the same time zone or format. This could lead to lights not turning on when expected.

Division by Zero:

If the interval parameter passed to time.sleep(interval) in the run_simulation method of the AutomationSystem class is set to zero or a negative value, it will result in a ValueError.

Infinite Loop:

If there is an issue with the simulation loop in run_simulation, such as not breaking out of the loop under certain conditions, it may result in an infinite loop.

Using the GUI

Overview

The Smart Home GUI provides a user interface to control and monitor various smart home devices, including music playback, lights, and air conditioning. The GUI is connected to the smart home system, allowing users to toggle devices and adjust their properties in real-time.

Components

1. Toggle Buttons

Toggle Music: Pressing this button toggles the music system on and off. When the music system is on, it plays a random song.

Toggle Lights: Pressing this button toggles the lights on and off.

Toggle AC: Pressing this button toggles the air conditioning (AC) on and off. When the AC is on, it updates the temperature periodically.

2. Sliders

Brightness Slider: Adjusts the brightness of the lights. Move the slider to the right to increase brightness and to the left to decrease it.

Temperature Slider: Sets the desired temperature for the air conditioning. Move the slider to the right to increase the temperature and to the left to decrease it.

Volume Slider: Controls the volume of the music system. Move the slider to the right to increase volume and to the left to decrease it.

Usage

Toggle Devices:

Use the "Toggle Music," "Toggle Lights," and "Toggle AC" buttons to turn the respective devices on or off.

Adjust Brightness:

Move the "Brightness Slider" to adjust the brightness level of the lights in real-time.

Set Temperature:

Use the "Temperature Slider" to set the desired temperature for the air conditioning. The AC will adjust its temperature accordingly.

Control Volume:

Adjust the "Volume Slider" to control the volume of the music system.

Notes

Real-time Updates: Changes made using the GUI are reflected in real-time in the smart home system. For example, adjusting the brightness slider will immediately impact the lights.

Console Feedback: The console provides feedback on the actions taken, such as toggling devices, setting brightness, temperature, and volume. The console output helps track the state of the smart home system.

Termination: Close the GUI window to terminate the program. This will stop the simulation and close the application.

Developer environment

An operating system capable of running exe files (eg. Windows 7). mingw32-g++.exe c++ compiler (v4.7), Code::Blocks (v13.12) developer tool.

The structure of the program

The modules used by the program, and their locations:

Smarthome.py – the main program, in the source folder.

script.py – the file where we run the smart home script to test our output.

mysmarthomeGUI.py – the GUI code of the program, running this file will open the GUI interface.

Testing

Valid test cases

Valid test cases for the smart home program can cover various scenarios to ensure the correct functioning of the system. Here are some valid test cases:

Toggle Devices:

Test toggling each device individually (Music, Lights, AC).

Test toggling multiple devices simultaneously.

Adjust Brightness:

Set the brightness slider to the minimum value (0).

Set the brightness slider to the maximum value (100).

Gradually increase and decrease the brightness using the slider.

Set Temperature:

Set the temperature slider to the minimum value (15°C).

Set the temperature slider to the maximum value (30°C).

Gradually increase and decrease the temperature using the slider.

Control Volume:

Set the volume slider to the minimum value (0).

Set the volume slider to the maximum value (100).

Gradually increase and decrease the volume using the slider.

Toggle Music and AC Interaction:

Toggle the music system on and set the AC temperature to a specific value.

Observe how the music system and AC interact when both are turned on.

Toggle Lights and AC Interaction:

Toggle the lights on and set the AC temperature to a specific value.

Observe how the lights and AC interact when both are turned on.

Simulate Real-time Updates:

Toggle devices and adjust settings in real-time using the GUI.

Observe how the console feedback reflects the changes.

Edge Cases:

Test with an empty playlist for the music system.

Test with extreme values for brightness, temperature, and volume sliders.

Long-Term Simulation:

Run the simulation for an extended period to ensure stability and check for memory leaks.