**Algaziev Almaz(EBKBVI)**

**Python Assignment 1**

**Smart Home IoT Automation Simulator**

The home automation system is simulated through a Python script utilizing the **tkinter** library for the graphical user interface. The system consists of three main components: a smart light device (**home\_light**), a temperature control device (**TemperatureControl**), and a smart camera device (**SmartCamera**). These devices are managed by the **HomeAutomationSystem** class.

**home\_light Class**

The **home\_light** class represents a smart light device that can be toggled on/off, and its intensity can be adjusted. Additionally, the class provides a method to simulate changes in light intensity when the light is active.

**TemperatureControl Class**

The **TemperatureControl** class represents a temperature control device capable of being toggled on/off and setting a desired temperature. The class also includes a method to simulate changes in temperature when the control is active.

**SmartCamera Class**

The **SmartCamera** class represents a smart camera device that can be toggled on/off. It features methods to simulate motion detection and changes in camera status.

**HomeAutomationSystem Class**

The **HomeAutomationSystem** class acts as the central controller for the smart devices. It allows the addition of devices and controls their simulation. The **simulate** method triggers the simulation of changes in all registered devices.

**SmartHomeApp Class**

The **SmartHomeApp** class utilizes the **tkinter** library to create a graphical user interface for the home automation system. It provides buttons, sliders, and labels for user interaction and real-time feedback. The app is capable of toggling devices, adjusting settings, and simulating changes automatically.

**How to Run the Simulation:**

1. Execute the script.
2. The GUI window will open, displaying the current status of the smart devices and controls.
3. Use the provided buttons and sliders to interact with the devices and controls.
4. The simulation will run automatically, updating the device status and UI every 900 milliseconds.

**Test Cases:**

1. Toggle Device ON/OFF.
2. Adjust Device Value.
3. Toggle Control ON/OFF.
4. Adjust Control Temperature.
5. Toggle Camera ON/OFF.
6. Simulate Camera Motion.
7. Automated Simulation.

These test cases cover various scenarios and user interactions to ensure the proper functioning of the home automation system. Users can refer to the provided documentation for each class to understand their functionalities and usage.

**SmartCamera Class**

Description:

The **SmartCamera** class represents a smart camera device in the home automation system. It has functionalities to toggle the camera on/off and simulate motion detection.

Methods:

* **\_\_init\_\_(self, device\_id: str)**: Initializes the SmartCamera instance with a unique device ID.
* **toggle\_camera(self)**: Toggles the camera's status between ON and OFF.
* **simulate\_motion(self)**: Simulates motion detection, setting the motion\_detected attribute to True.

**TemperatureControl Class**

Description:

The **TemperatureControl** class represents a temperature control device in the home automation system. It has functionalities to toggle the control on/off, set the desired temperature, and simulate temperature changes.

Methods:

* **\_\_init\_\_(self, device\_id: str)**: Initializes the TemperatureControl instance with a unique device ID.
* **toggle\_control(self)**: Toggles the control's status between ON and OFF.
* **set\_temperature(self, temperature: int)**: Sets the desired temperature for the control.
* **simulate\_change(self)**: Simulates a change in temperature when the control is active.

**home\_light Class**

Description:

The **home\_light** class represents a smart light device in the home automation system. It has functionalities to toggle the light on/off and adjust the light intensity.

Methods:

* **\_\_init\_\_(self, device\_id: str)**: Initializes the home\_light instance with a unique device ID.
* **toggle\_device(self)**: Toggles the light's status between ON and OFF.
* **set\_value(self, value: int)**: Adjusts the light intensity.
* **simulate\_change(self)**: Simulates a change in light intensity when the light is active.

**HomeAutomationSystem Class**

Description:

The **HomeAutomationSystem** class represents the overall home automation system. It manages different smart devices and controls in the system.

Methods:

* **\_\_init\_\_(self)**: Initializes an instance of the HomeAutomationSystem.
* **add\_component(self, component)**: Adds a smart device or control component to the system.
* **simulate(self)**: Simulates changes in all registered devices and controls.

**SmartHomeApp Class**

Description:

The **SmartHomeApp** class represents the graphical user interface for the smart home simulation. It utilizes the **tkinter** library to create a window with controls for the user.

Methods:

* **\_\_init\_\_(self)**: Initializes the SmartHomeApp instance, setting up the UI and connecting it with the HomeAutomationSystem.
* **setup\_interface(self)**: Sets up the graphical user interface with text display, sliders, buttons, and labels.
* Various methods like **toggle\_device**, **adjust\_device\_value**, **toggle\_control**, **adjust\_control\_temperature**, **simulate\_camera\_motion**, **toggle\_camera**, and **update** that handle user interactions and update the UI based on device and control status.

**How to Run the Simulation:**

1. Execute the script.
2. The GUI window will open, displaying the current status of the smart devices and controls.
3. Use the provided buttons and sliders to interact with the devices and controls.
4. The simulation will run automatically, updating the device status and UI every 900 milliseconds.

**Test Cases:**

1. **Toggle Device ON/OFF:**
   * Toggle the smart device ON and OFF using the provided button.
   * Verify that the status and labels update accordingly.
2. **Adjust Device Value:**
   * Use the slider to adjust the device value.
   * Verify that the label reflects the updated value.
3. **Toggle Control ON/OFF:**
   * Toggle the temperature control ON and OFF using the provided button.
   * Verify that the status and labels update accordingly.
4. **Adjust Control Temperature:**
   * Use the slider to adjust the control temperature.
   * Verify that the label reflects the updated temperature.
5. **Toggle Camera ON/OFF:**
   * Toggle the camera ON and OFF using the provided button.
   * Verify that the status and labels update accordingly.
6. **Simulate Camera Motion:**
   * Click the "Random Detect Motion" button.
   * Verify that the motion status label updates accordingly.
7. **Automated Simulation:**
   * Observe the automatic simulation updates.
   * Verify that device and control values update as expected.

These test cases cover different scenarios and user interactions to ensure the proper functioning of the home automation system.

**home\_light Class**

Description:

The **home\_light** class represents a smart light device in the home automation system. It has functionalities to toggle the light on/off, adjust the light intensity, and simulate changes in light intensity.

Methods:

* **\_\_init\_\_(self, device\_id, initial\_value=0)**: Initializes a new instance of the **home\_light** class with a unique device ID and an optional initial light intensity value (default is 0).
* **toggle\_device(self)**: Toggles the light's status between ON and OFF.
* **set\_value(self, level)**: Adjusts the light intensity to the specified level, ranging from 0 to 100.
* **simulate\_change(self)**: Simulates a change in light intensity when the light is active. The intensity is set to a random value between 0 and 100.

Attributes:

* **device\_id**: A unique identifier for the light device.
* **status**: Represents the current status of the light (True for ON, False for OFF).
* **value**: Represents the current intensity level of the light, ranging from 0 to 100.

**How to Use the home\_light Class:**

1. **Initialization:**

pythonCopy code

light\_device = home\_light(device\_id='device1', initial\_value=50)

1. **Toggle Device ON/OFF:**

pythonCopy code

light\_device.toggle\_device()

1. **Adjust Light Intensity:**

pythonCopy code

light\_device.set\_value(75) # Set intensity to 75

1. **Simulate Change:**

pythonCopy code

light\_device.simulate\_change() # Simulate a change in light intensity

**Example:**

pythonCopy code

# Create a light device light\_device = home\_light(device\_id='device1', initial\_value=50) # Toggle the light ON light\_device.toggle\_device() # Adjust the light intensity light\_device.set\_value(75) # Simulate a change in light intensity light\_device.simulate\_change()

In the provided example, a **home\_light** instance is created with a device ID of 'device1' and an initial intensity value of 50. The light is then toggled ON, its intensity adjusted to 75, and a simulated change in intensity is performed. Users can integrate this class into their home automation system for controlling and simulating smart light devices.

**TemperatureControl Class**

Description:

The **TemperatureControl** class represents a temperature control device in the home automation system. It has functionalities to toggle the control on/off, set the desired temperature, and simulate changes in temperature.

Methods:

* **\_\_init\_\_(self, device\_id, initial\_temperature=20)**: Initializes a new instance of the **TemperatureControl** class with a unique device ID and an optional initial temperature value (default is 20°C).
* **toggle\_control(self)**: Toggles the control's status between ON and OFF.
* **set\_temperature(self, temp)**: Sets the desired temperature for the control, within the range of 10°C to 30°C.
* **simulate\_change(self)**: Simulates a change in temperature when the control is active. The temperature is set to a random value between 10°C and 30°C.

Attributes:

* **device\_id**: A unique identifier for the temperature control device.
* **status**: Represents the current status of the control (True for ON, False for OFF).
* **temperature**: Represents the current desired temperature set by the control, ranging from 10°C to 30°C.

**How to Use the TemperatureControl Class:**

1. **Initialization:**

pythonCopy code

temp\_control\_device = TemperatureControl(device\_id='control1', initial\_temperature=25)

1. **Toggle Control ON/OFF:**

pythonCopy code

temp\_control\_device.toggle\_control()

1. **Set Desired Temperature:**

pythonCopy code

temp\_control\_device.set\_temperature(22) # Set desired temperature to 22°C

1. **Simulate Change:**

pythonCopy code

temp\_control\_device.simulate\_change() # Simulate a change in temperature

**Example:**

pythonCopy code

# Create a temperature control device temp\_control\_device = TemperatureControl(device\_id='control1', initial\_temperature=25) # Toggle the control ON temp\_control\_device.toggle\_control() # Set the desired temperature temp\_control\_device.set\_temperature(22) # Simulate a change in temperature temp\_control\_device.simulate\_change()

In the provided example, a **TemperatureControl** instance is created with a device ID of 'control1' and an initial temperature of 25°C. The control is then toggled ON, the desired temperature set to 22°C, and a simulated change in temperature is performed. Users can integrate this class into their home automation system for controlling and simulating temperature control devices.

Description:

The **SmartCamera** class represents a smart camera device in the home automation system. It has functionalities to toggle the camera on/off, simulate motion detection, and simulate changes in camera status.

Methods:

* **\_\_init\_\_(self, device\_id)**: Initializes a new instance of the **SmartCamera** class with a unique device ID.
* **toggle\_camera(self)**: Toggles the camera's status between ON and OFF.
* **simulate\_motion(self)**: Simulates motion detection when the camera is active. The motion\_detected attribute is set to a random choice of True or False.
* **simulate\_change(self)**: Simulates a change in camera status by invoking **simulate\_motion** when the camera is active.

Attributes:

* **device\_id**: A unique identifier for the smart camera device.
* **status**: Represents the current status of the camera (True for ON, False for OFF).
* **motion\_detected**: Represents whether motion is detected by the camera (True or False).

**How to Use the SmartCamera Class:**

1. **Initialization:**

pythonCopy code

camera\_device = SmartCamera(device\_id='camera1')

1. **Toggle Camera ON/OFF:**

pythonCopy code

camera\_device.toggle\_camera()

1. **Simulate Motion:**

pythonCopy code

camera\_device.simulate\_motion() # Simulate motion detection

1. **Simulate Change:**

pythonCopy code

camera\_device.simulate\_change() # Simulate a change in camera status

**Example:**

pythonCopy code

# Create a smart camera device camera\_device = SmartCamera(device\_id='camera1') # Toggle the camera ON camera\_device.toggle\_camera() # Simulate motion detection camera\_device.simulate\_motion() # Simulate a change in camera status camera\_device.simulate\_change()

In the provided example, a **SmartCamera** instance is created with a device ID of 'camera1'. The camera is then toggled ON, motion is simulated, and a change in camera status is simulated. Users can integrate this class into their home automation system for controlling and simulating smart camera devices.