1. Introduction to Production Systems: Smart Home Automation

Use Case: Develop a rule-based system for automating tasks in a smart home, such as controlling lights, adjusting thermostats, and locking doors based on specific conditions (e.g., time of day, presence of people).

Objective: Implement a basic production system using if-then rules to automate decision-making in a simulated smart home environment.

Experiment Outline:

1. Introduction and Background

- **Brief on Production Systems:** A production system is a type of computational model used for decision-making and automation, based on a set of condition-action rules. Each rule follows an "if-then" structure, where certain actions are triggered when specific conditions are met. These systems are foundational in artificial intelligence and automation, particularly for creating rule-based behaviors.

  Overview of Smart Home Automation: In the context of smart homes, a production system enables automated workflows by defining a set of rules that govern the behavior of devices. For example, in a smart home, a production system could be used to automatically adjust lighting, thermostats, and security systems based on real-time conditions like time of day, occupancy, and temperature.

  For instance:

  - If it's evening and someone is home, then turn on the lights.
  - If no one is home, then lock all doors.
  - If the temperature is below a certain threshold, then increase the thermostat.

  This rule-based approach allows smart homes to respond dynamically to various conditions, improving energy efficiency, security, and comfort without needing manual control.

2. Experiment Setup

- **Simulated Environment:** Use Python code to create a simulated environment with virtual devices that students will control.
- Define a dictionary to represent the current state of each device:

```
# Initial state of the smart home devices
smart_home = {
    "lights": "off",
```

```
    "thermostat": 22,  # in Celsius
    "door_lock": "unlocked",
    "time_of_day": "morning",
    "presence": False  # True if someone is home, False if not
}
```

3. Defining Automation Rules

- **Introduction to Rule-Based Logic:** Rule-based logic is a system of decision-making that uses a set of explicit "if-then" rules to determine actions based on given conditions. Each rule defines a specific condition and a corresponding action that should be executed when that condition is met. This approach allows for predictable, consistent responses to a wide range of situations without complex calculations or deep learning.
- **Example Rules:** Provide a few initial rules, such as:
    - **Rule 1**: If the time of day is evening and presence is True, turn on the lights.
    - **Rule 2**: If the temperature is below 20°C and presence is True, increase the thermostat to 22°C.
    - **Rule 3**: If the presence is False, ensure doors are locked.
    - **Rule 4**: If it’s morning and presence is True, set the thermostat to 20°C.

4. Code Implementation of Rules

- **Function to Apply Rules:** Implement a function to apply these rules to the simulated environment. The function should update the device states based on conditions.

```
def apply_rules(home):
    # Rule 1: Turn on lights in the evening if someone is home
    if home["time_of_day"] == "evening" and home["presence"]:
        home["lights"] = "on"

    # Rule 2: Adjust thermostat if it's cold and someone is home
    if home["thermostat"] < 20 and home["presence"]:
        home["thermostat"] = 22

    # Rule 3: Lock doors if no one is home
    if not home["presence"]:
        home["door_lock"] = "locked"

    # Rule 4: Set morning thermostat if someone is home
    if home["time_of_day"] == "morning" and home["presence"]:
        home["thermostat"] = 20

    return home
```

- **Testing the Rules:** Allow students to change the environment conditions (e.g., time of day, presence) and test how the rules affect the state of each device.

5. Visualization using Grid

1. **Grid Layout**: Each device is represented as a rectangle in a grid layout, with the device name and its current state displayed beside it.
2. **Color Coding**: Colors are mapped to states (green for active/secure states, yellow for inactive/insecure states, and blue for neutral states like the time of day).
3. **Rectangle Patches**: Each device's state is represented by a colored rectangle with text labels for the device name and current state.

This design mimics a dashboard and provides an organized view of the smart home's state before and after applying the automation rules.

6. Analysis and Questions (to be completed by students)

- **Reflection Questions:**
    - How would you modify rules to handle conflicting conditions?
    - How could this rule-based approach scale to more complex systems?
    - Summarize the learning outcomes and discuss how rule-based systems are foundational to more complex automation and AI systems.

Lab Program 1: Smart Home Automation

In [1]:
```
# Import necessary libraries
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
```

In [2]:
```
# Define the initial state of the smart home devices
smart_home = {
    "lights": "off",
    "thermostat": 22,  # in Celsius
    "door_lock": "unlocked",
    "time_of_day": "morning",
    "presence": False  # True if someone is home, False if not
}
```

In [3]:
```
# Function to visualize the smart home state as a dashboard grid
def visualize_state_grid(home_state, title="Smart Home State"):
    # Set up the plot
    fig, ax = plt.subplots(figsize=(8, 6))
    ax.set_xlim(0, 3)
    ax.set_ylim(0, len(home_state))
    ax.set_title(title, fontsize=16, fontweight='bold')
    ax.axis("off")

    # Define colors for each state based on device type
    color_map = {
        "on": "green", "off": "yellow",
        "locked": "green", "unlocked": "yellow",
        True: "green", False: "yellow",
        "morning": "aqua", "evening": "aqua",
        "thermostat": "green"
    }

    # Place each device state as a colored rectangle in the grid
    for i, (device, state) in enumerate(home_state.items()):
        # Define the color based on the state
        color = color_map.get(state, "green")
        rect = Rectangle((0.5, i), 2, 1, color=color, edgecolor="black")
        ax.add_patch(rect)

        # Add device name and state text
        ax.text(1, i + 0.5, device.capitalize(), va="center", ha="center", 
fontweight="bold", color="black")
        ax.text(2, i + 0.5, str(state), va="center", ha="center", fontweigh
t="bold", color="black")

    plt.show()
```

In [4]:

```python
# Display the initial state of the smart home
print("Initial State:", smart_home)
visualize_state_grid(smart_home, title="Initial Smart Home State")
```

```
Initial State: {'lights': 'off', 'thermostat': 22, 'door_lock': 'unlocke
d', 'time_of_day': 'morning', 'presence': False}
```

```
<ipython-input-3-d5084f46e5f9>:23: UserWarning: Setting the 'color' proper
ty will override the edgecolor or facecolor properties.
  rect = Rectangle((0.5, i), 2, 1, color=color, edgecolor="black")
```

In [5]:

```python
# Define the function to apply rules
def apply_rules(home):
    # Rule 1: Turn on lights in the evening if someone is home
    if home["time_of_day"] == "evening" and home["presence"]:
        home["lights"] = "on"

    # Rule 2: Adjust thermostat if it's cold and someone is home
    if home["thermostat"] < 20 and home["presence"]:
        home["thermostat"] = 22

    # Rule 3: Lock doors if no one is home
    if not home["presence"]:
        home["door_lock"] = "locked"

    # Rule 4: Set morning thermostat if someone is home
    if home["time_of_day"] == "morning" and home["presence"]:
        home["thermostat"] = 20

    return home
```

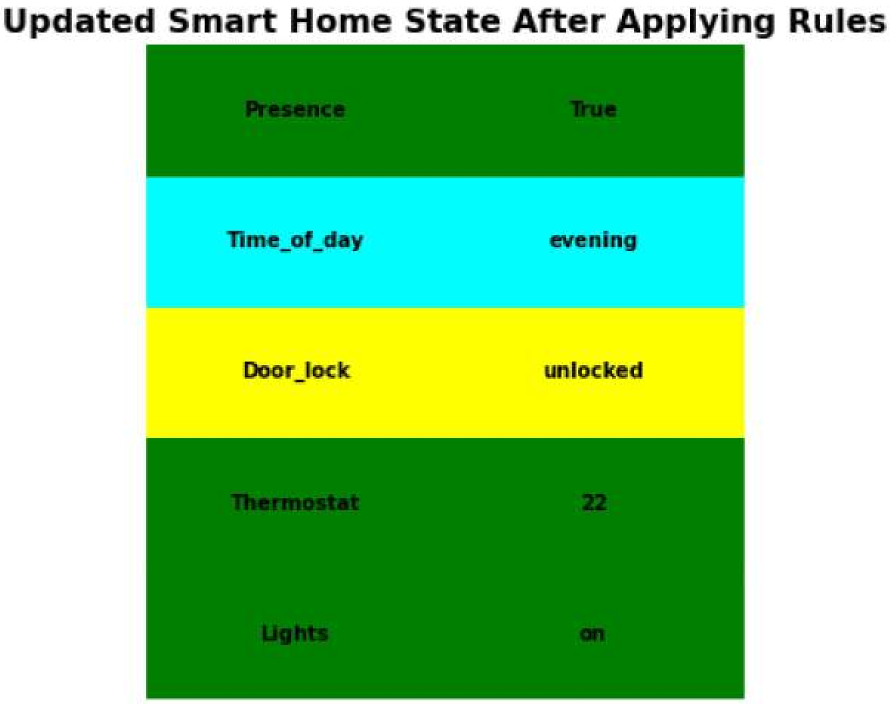
In [6]:

```python
# Update smart home conditions for testing (e.g., presence and time of day)
smart_home["time_of_day"] = "evening"
smart_home["presence"] = True
smart_home["thermostat"] = 18  # Set thermostat to test adjustment rule
```

In [7]:
```python
# Apply rules and show updated state
updated_home = apply_rules(smart_home)
print("Updated State:", updated_home)
visualize_state_grid(updated_home, title="Updated Smart Home State After Ap
plying Rules")
```

```
Updated State: {'lights': 'on', 'thermostat': 22, 'door_lock': 'unlocked',
'time_of_day': 'evening', 'presence': True}

<ipython-input-3-d5084f46e5f9>:23: UserWarning: Setting the 'color' proper
ty will override the edgecolor or facecolor properties.
  rect = Rectangle((0.5, i), 2, 1, color=color, edgecolor="black")
```



In [ ]:

In [ ]: