**Lab 6: Write a Rule Base System in Python for the following rule systems:**

**Task 1: Weather Forecasting**

**Objectives:**

Implement a Python program to predict weather conditions based on rules that consider the sky condition, temperature, and wind presence, determining if it might rain, snow, be a hot day, or a pleasant day.

**Task Name: Weather Forecasting**

Rule 1: If sky is cloudy and there is no wind, then it might rain.

Rule 2: If temperature is below 0°C and the sky is clear, then it might snow.

Rule 3: If temperature is above 30°C and there is no wind, then it might be a hot day.

Rule 4: If sky is clear and there is wind, then it might be a pleasant day.

**Code:**

def weather\_forecast(sky, temperature, wind):

    # *Rule 1: If the sky is cloudy and there is no wind, then it might rain.*

    if sky == "cloudy" and wind == "none":

        return "It might rain."

    # *Rule 2: If the temperature is below 0°C and the sky is clear, then it might snow.*

    if temperature < 0 and sky == "clear":

        return "It might snow."

    # *Rule 3: If the temperature is above 30°C and there is no wind, then it might be a hot day.*

    if temperature > 30 and wind == "none":

        return "It might be a hot day."

    # *Rule 4: If the sky is clear and there is wind, then it might be a pleasant day.*

    if sky == "clear" and wind == "windy":

        return "It might be a pleasant day."

    return "Weather conditions unclear, please check the forecast later."

def main():

    # *Collect user inputs*

    sky = input("Enter the sky condition (cloudy/clear): ")

    temperature = float(input("Enter the temperature in °C: "))

    wind = input("Enter the wind condition (none/windy): ")

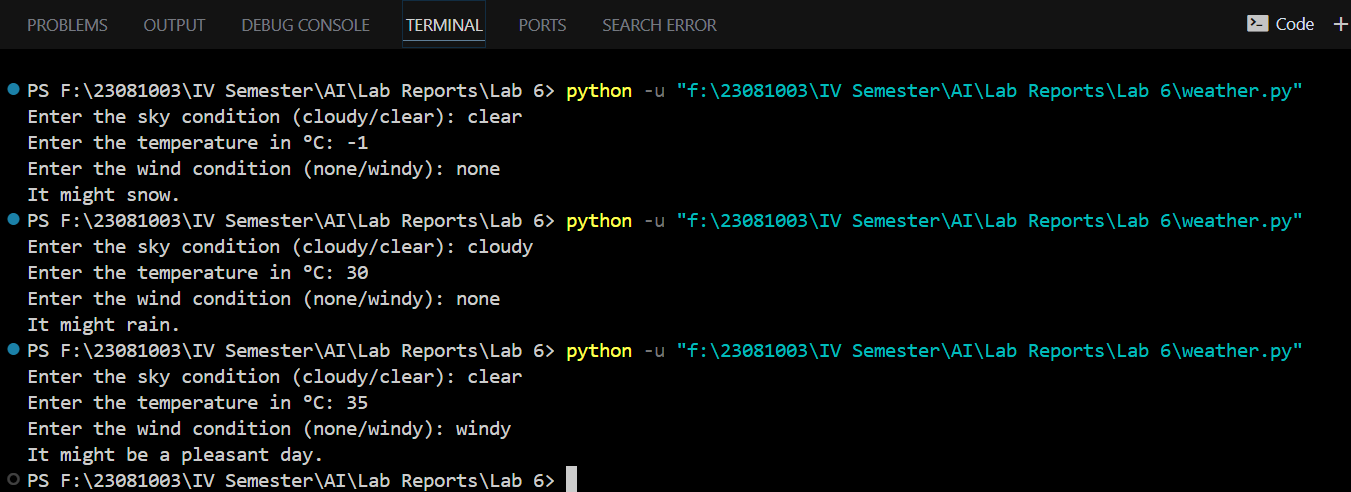
    # *Determine and print the weather forecast*

    forecast = weather\_forecast(sky, temperature, wind)

    print(forecast)

main()

**Output:**

****

**Task 2: Eligibility for a Loan**

**Objectives:**

Implement a Python program to assess an applicant's eligibility for a loan by evaluating their age, income stability, credit score, criminal record, and history of loan defaults.

**Task Name: Eligibility for a Loan**

Rule 1:If applicant's age is between 18 and 65 and they have a stable income, then

they are eligible for a loan.

Rule 2:If applicant has a credit score above 700, then they are eligible for a loan.

Rule 3:If applicant has a criminal record, then they are not eligible for a loan.

Rule 4:If applicant has defaulted on a loan before, then they are not eligible for

a loan.

**Code:**

def check\_loan\_eligibility(age, stable\_income, credit\_score, criminal\_record, loan\_default):

    # *Rule 1: If the applicant's age is between 18 and 65 and they have a stable income, then they are eligible for a loan.*

    if 18 <= age <= 65 and stable\_income == "yes":

        return "Eligible for a loan based on age and stable income."

    # *Rule 2: If the applicant has a credit score above 700, then they are eligible for a loan.*

    if credit\_score > 700:

        return "Eligible for a loan based on credit score.

    # *Rule 3: If the applicant has a criminal record, then they are not eligible for a loan.*

    if criminal\_record == "yes":

        return "Not eligible for a loan due to criminal record."

    # *Rule 4: If the applicant has defaulted on a loan before, then they are not eligible for a loan.*

    if loan\_default == "yes":

        return "Not eligible for a loan due to previous loan default."

    # *If none of the conditions are met, the applicant is not eligible for a loan.*

    return "Not eligible for a loan based on the given criteria."

def main():

    # *Collect user inputs*

    age = int(input("Enter your age: "))

    stable\_income = input("Do you have a stable income? (yes/no): ")

    credit\_score = int(input("Enter your credit score: "))

    criminal\_record = input("Do you have a criminal record? (yes/no): ")

    loan\_default = input("Have you defaulted on a loan before? (yes/no): ")

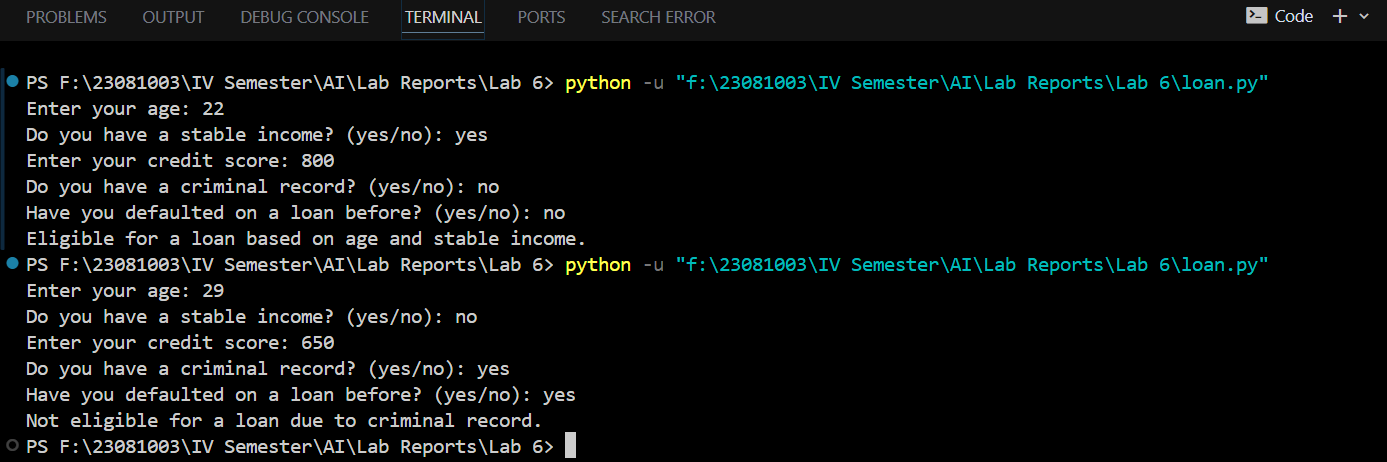
    # *Determine and print the loan eligibility*

    eligibility = check\_loan\_eligibility(age, stable\_income, credit\_score, criminal\_record, loan\_default)

    print(eligibility)

main()

**Output:**



**Task 3: Simple Decision Making**

**Objectives:**

Implement a Python program to predict weather conditions based on rules that consider the sky condition, temperature, and wind presence, determining if it might rain, snow, be a hot day, or a pleasant day.

**Task Name: Simple Decision Making**

Rule 1: If the time is between 6 AM and 8 AM and it's a weekday, then it's time to go to work.

Rule 2: If the time is between 12 PM and 1 PM, then it's time for lunch.

Rule 3: If the time is between 9 PM and 10 PM, then it's time to go to bed.

Rule 4: If it's the weekend and the weather is sunny, then go for a walk. **Code:**

def convert\_to\_24\_hour(time\_str):

    """Converts time from 12-hour format (e.g., '2 PM') to 24-hour format."""

    time\_str = time\_str.strip().upper()

    if "AM" in time\_str or "PM" in time\_str:

        time, period = time\_str.split()

        time = int(time)

        if period == "PM" and time != 12:

            time += 12

        if period == "AM" and time == 12:

            time = 0

    else:

        time = int(time\_str)  # *Assume it's already in 24-hour format*

    return time

def simple\_decision\_making(time, is\_weekday, is\_sunny):

    # *Rule 1: If the time is between 6 AM and 8 AM and it's a weekday, then it's time to go to work.*

    if 6 <= time < 8 and is\_weekday == "yes":

        return "Time to go to work."

    # *Rule 2: If the time is between 12 PM and 1 PM, then it's time for lunch.*

    if 12 <= time <= 13:

        return "Time for lunch."

    # *Rule 3: If the time is between 9 PM and 10 PM, then it's time to go to bed.*

    if 21 <= time <=22:

        return "Time to go to bed."

    # *Rule 4: If it's the weekend and the weather is sunny, then go for a walk.*

    if is\_weekday == "no" and is\_sunny == "yes":

        return "Go for a walk."

    # *If none of the conditions are met, return no specific action.*

    return "No specific action for this time."

def main():

    try:

        # *Collect user inputs*

        time\_str = input("Enter the current time:")

        time = convert\_to\_24\_hour(time\_str)

        # *Validate the time input*

        if time < 0 or time > 23:

            raise ValueError("Time must be between 0 and 23.")

        is\_weekday = input("Is it a weekday? (yes/no): ")

        is\_sunny = input("Is the weather sunny? (yes/no): ")

        # *Ensure inputs are either 'yes' or 'no'*

        if is\_weekday not in ['yes', 'no'] or is\_sunny not in ['yes', 'no']:

            raise ValueError("Input for 'is\_weekday' and 'is\_sunny' must be 'yes' or 'no'.")

        # *Determine and print the decision*

        decision = simple\_decision\_making(time, is\_weekday, is\_sunny)

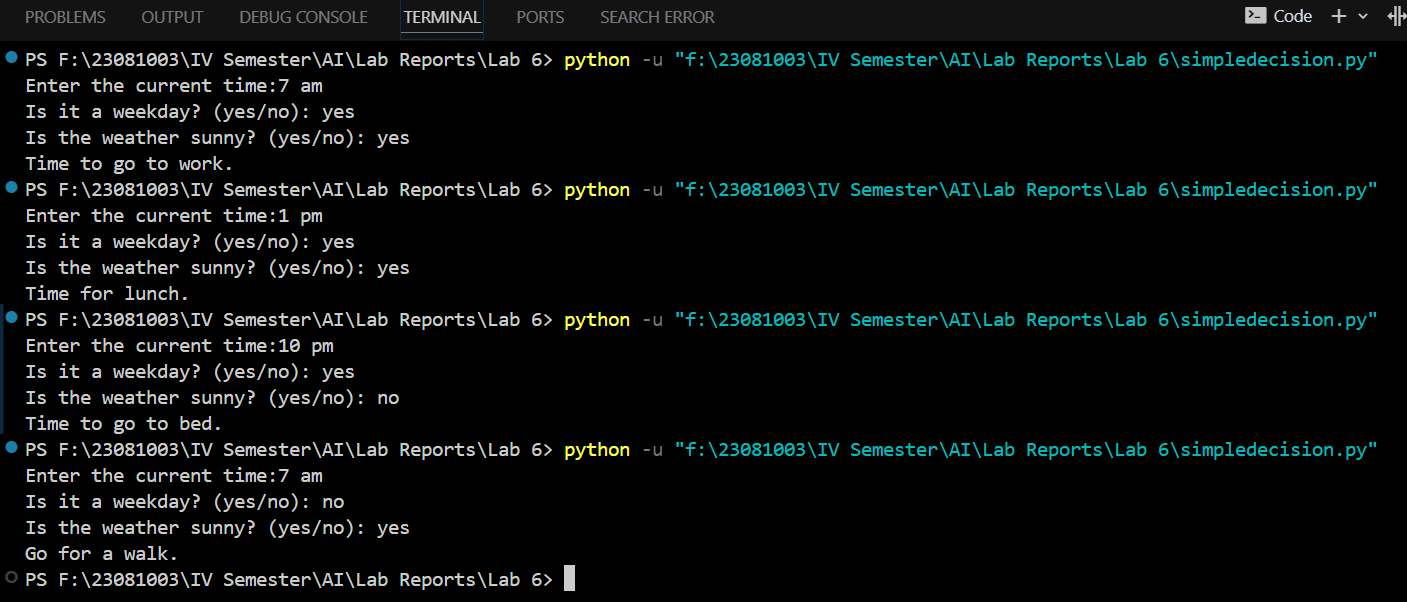
        print(decision)

    except ValueError as e:

        print(f"Error: {e}")

main()

**Output:**

****

**Task 4: Traffic Light Control**

**Objectives:**

Build a rule-based system to control traffic lights at an intersection. The system should manage the flow of traffic by changing lights based on predefined rules and handle special conditions like pedestrian button presses.

**Task Name: Traffic Light Control**

Rule 1: If the light is red, then cars must stop.

Rule 2: If the light is green, then cars can go.

Rule 3: If the light is yellow, then cars must slow down and prepare to stop.

Rule 4: If the pedestrian button is pressed, then the light will turn red after a short delay.

**Code:**

def traffic\_light\_action(light\_color, pedestrian\_button):

"""Determine the action based on the light color and pedestrian button status."""

# Rule 1: If the light is red, then cars must stop.

if light\_color == "red":

return "Cars must stop."

# Rule 2: If the light is green, then cars can go.

elif light\_color == "green":

return "Cars can go."

# Rule 3: If the light is yellow, then cars must slow down and prepare to stop.

elif light\_color == "yellow":

return "Cars must slow down and prepare to stop."

# Rule 4: If the pedestrian button is pressed, then the light will turn red after a short delay.

elif pedestrian\_button == "yes":

return "The light will turn red after a short delay."

# If none of the conditions are met, return no specific action.

return "No specific action for the given light state."

def main():

try:

# Collect user inputs

light\_color = input("Enter the traffic light color (red, green, yellow): ")

pedestrian\_button = input("Is the pedestrian button pressed? (yes/no): ")

# Validate light color input

if light\_color != 'red' and light\_color != 'green' and light\_color != 'yellow':

print("Error: Light color must be 'red', 'green', or 'yellow'.")

return

# Validate pedestrian button input

if pedestrian\_button != 'yes' and pedestrian\_button != 'no':

print("Error: Pedestrian button input must be 'yes' or 'no'.")

return

# Determine and print the traffic light action

action = traffic\_light\_action(light\_color, pedestrian\_button)

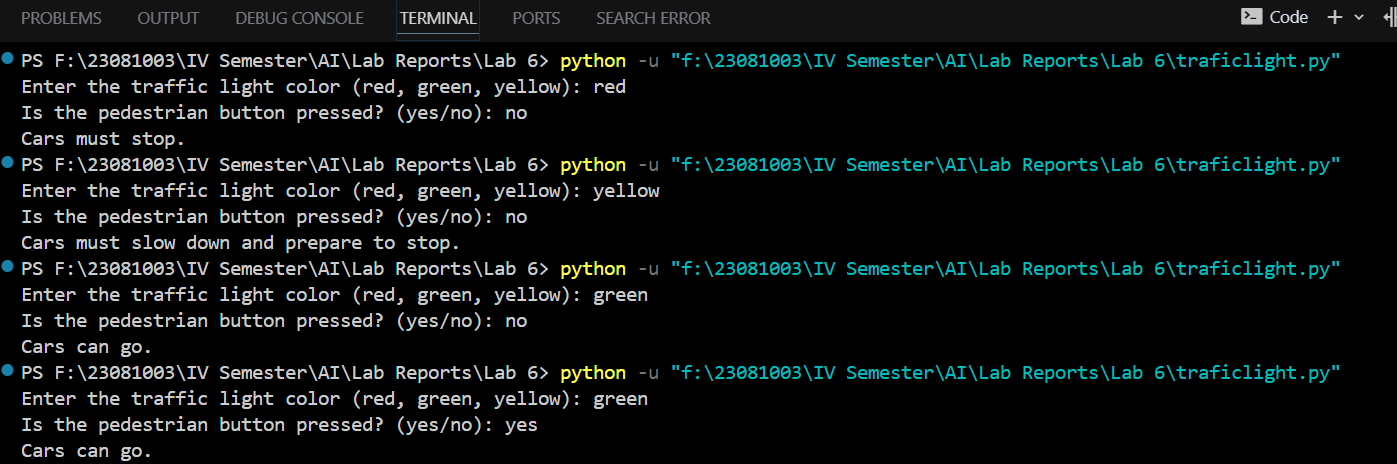
print(action)

except ValueError as e:

print(f"Error: {e}")

main()

**Output:**

****

**Task 5: Smart Home Automation**

**Objectives:**

Implement a Python program to automate home devices by controlling heating, cooling, lighting, and security alarms based on environmental conditions and user activities.

**Task Name: Smart Home Automation**

Rule 1: If the temperature is below 18°C, then turn on the heater.

Rule 2: If the temperature is above 25°C, then turn on the air conditioner.

Rule 3: If it is dark outside and someone is at home, then turn on the lights.

Rule 4: If the security system is armed and a door is opened, then sound the alarm.

**Code:**

def smart\_home\_action(temperature, is\_dark, at\_home, security\_armed, door\_open):

    """Determine the action based on temperature, lighting, and security system status."""

    actions = []

    # *Rule 1: If the temperature is below 18°C, then turn on the heater.*

    if temperature < 18:

        actions.append("Turn on the heater.")

    # *Rule 2: If the temperature is above 25°C, then turn on the air conditioner.*

    if temperature > 25:

        actions.append("Turn on the air conditioner.")

    # *Rule 3: If it is dark outside and someone is at home, then turn on the lights.*

    if is\_dark == "yes" and at\_home == "yes":

        actions.append("Turn on the lights.")

    # *Rule 4: If the security system is armed and a door is opened, then sound the alarm.*

    if security\_armed == "yes" and door\_open == "yes":

        actions.append("Sound the alarm.")

    # *If no actions are needed, return this message*

    if not actions:

        actions.append("No specific action required.")

    return actions

def main():

    try:

        # *Collect user inputs*

        temperature = float(input("Enter the current temperature in °C: "))

        is\_dark = input("Is it dark outside? (yes/no): ")

        at\_home = input("Is someone at home? (yes/no): ")

        security\_armed = input("Is the security system armed? (yes/no): ")

        door\_open = input("Is a door opened? (yes/no): ")

        # *Validate inputs*

        if is\_dark != 'yes' and is\_dark != 'no':

            print("Error: 'Is it dark outside?' input must be 'yes' or 'no'.")

            return

        if at\_home != 'yes' and at\_home != 'no':

            print("Error: 'Is someone at home?' input must be 'yes' or 'no'.")

            return

        if security\_armed != 'yes' and security\_armed != 'no':

            print("Error: 'Is the security system armed?' input must be 'yes' or 'no'.")

            return

        if door\_open != 'yes' and door\_open != 'no':

            print("Error: 'Is a door opened?' input must be 'yes' or 'no'.")

            return

        # *Determine and print the smart home action*

        actions = smart\_home\_action(temperature, is\_dark, at\_home, security\_armed, door\_open)

        for action in actions:

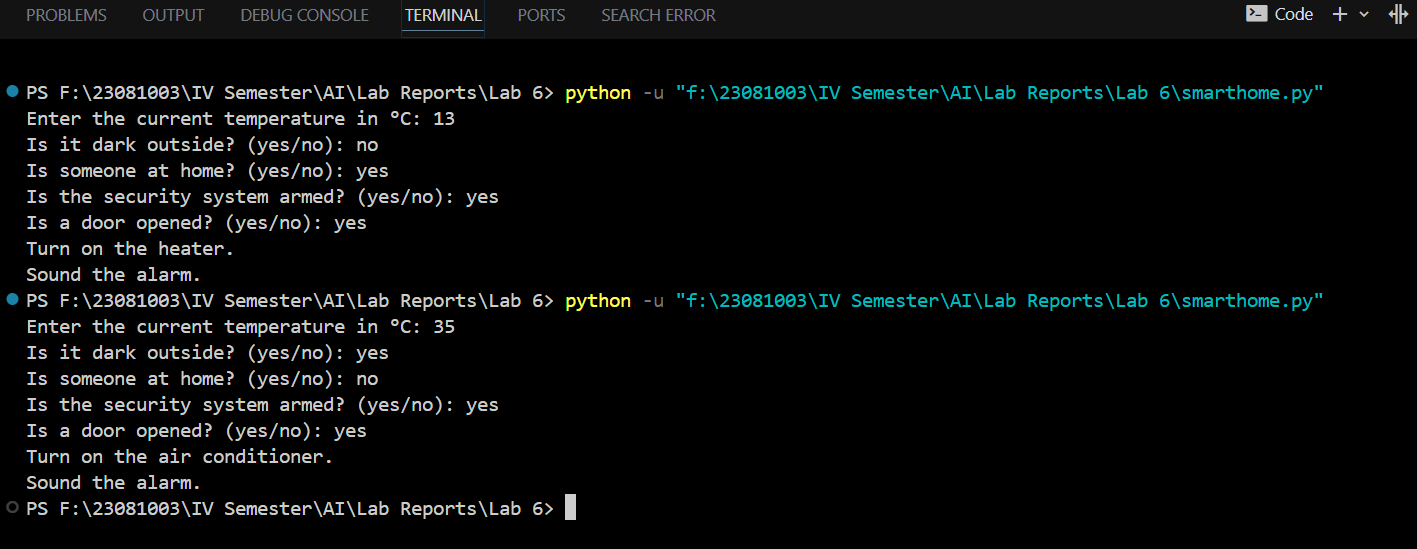
            print(action)

    except ValueError as e:

        print(f"Error: {e}")

main()

**Output:**

****

**Conclusion:**

The program effectively finds the sequence of steps to get exactly 2 gallons of water in the 4-gallon jug (X) using systematic application of predefined rules. The Breadth-First Search ensures that the shortest path to the solution is found. By following the steps in the printed solution path, one can understand how to reach the desired goal starting from empty jugs.