**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):

rules = {

("cloudy", None, "none"): "It might rain.",

("clear", lambda temp: temp < 0, None): "It might snow.",

(None, lambda temp: temp > 30, "none"): "It might be a hot day.",

("clear", None, "windy"): "It might be a pleasant day."

}

for (sky\_condition, temp\_condition, wind\_condition), forecast in rules.items():

if (sky\_condition is None or sky == sky\_condition) and \

(temp\_condition is None or temp\_condition(temperature)) and \

(wind\_condition is None or wind == wind\_condition):

return forecast

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

def main():

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

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

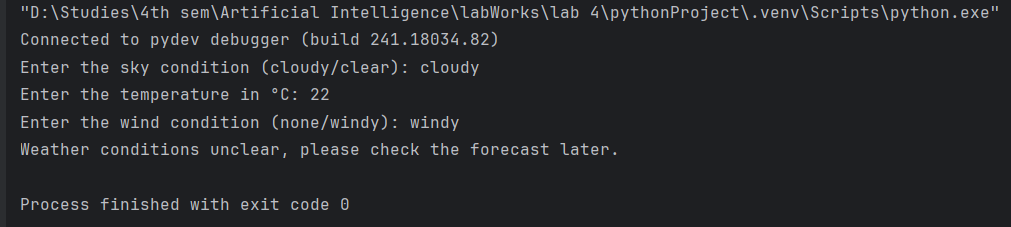
wind = input("Enter the wind condition (none/windy): ")

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):

rules = {

(lambda age, income: 18 <= age <= 65 and income == "yes", None, None, None, None): "Eligible for a loan based on age and stable income.",

(None, None, lambda score: score > 700, None, None): "Eligible for a loan based on credit score.",

(None, None, None, lambda record: record == "yes", None): "Not eligible for a loan due to criminal record.",

(None, None, None, None, lambda default: default == "yes"): "Not eligible for a loan due to previous loan default."

}

for (age\_condition, income\_condition, credit\_condition, record\_condition, default\_condition), result in rules.items():

if (age\_condition is None or age\_condition(age, stable\_income)) and \

(income\_condition is None or income\_condition(stable\_income)) and \

(credit\_condition is None or credit\_condition(credit\_score)) and \

(record\_condition is None or record\_condition(criminal\_record)) and \

(default\_condition is None or default\_condition(loan\_default)):

return result

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

def main():

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): ")

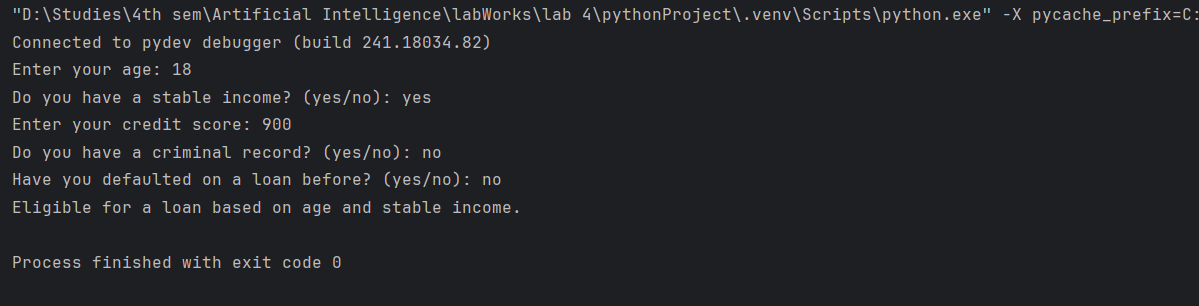
loan\_default = input("Have you defaulted on a loan before? (yes/no): ")

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):

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)

return time

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

rules = {

(lambda t, w: 6 <= t < 8 and w == "yes", None): "Time to go to work.",

(lambda t, w: 12 <= t <= 13, None): "Time for lunch.",

(lambda t, w: 21 <= t <= 22, None): "Time to go to bed.",

(lambda t, w, s: w == "no" and s == "yes"): "Go for a walk."

}

for (time\_condition, sunny\_condition), result in rules.items():

if (time\_condition is None or time\_condition(time, is\_weekday)) and \

(sunny\_condition is None or sunny\_condition(time, is\_weekday, is\_sunny)):

return result

return "No specific action for this time."

def main():

try:

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

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

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): ")

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'.")

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

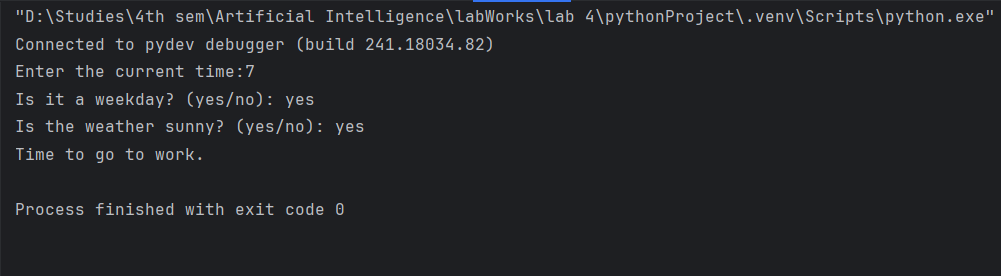
print(decision)

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):

rules = {

(lambda color: color == "red", None): "Cars must stop.",

(lambda color: color == "green", None): "Cars can go.",

(lambda color: color == "yellow", None): "Cars must slow down and prepare to stop.",

(None, lambda button: button == "yes"): "The light will turn red after a short delay."

}

for (color\_condition, button\_condition), result in rules.items():

if (color\_condition is None or color\_condition(light\_color)) and \

(button\_condition is None or button\_condition(pedestrian\_button)):

return result

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

def main():

try:

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

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

if light\_color not in ['red', 'green', 'yellow']:

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

return

if pedestrian\_button not in ['yes', 'no']:

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

return

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

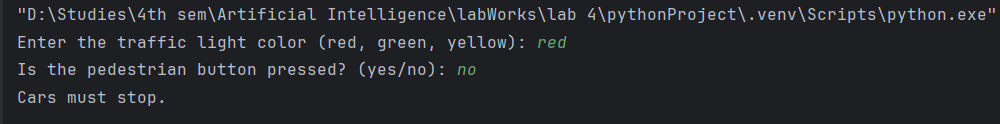
print(action)

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):

actions = []

rules = {

(lambda temp: temp < 18, None, None, None, None): "Turn on the heater.",

(lambda temp: temp > 25, None, None, None, None): "Turn on the air conditioner.",

(None, lambda dark, home: dark == "yes" and home == "yes", None, None, None): "Turn on the lights.",

(None, None, None, lambda armed, door: armed == "yes" and door == "yes"): "Sound the alarm."

}

for (temp\_cond, light\_cond, home\_cond, sec\_cond, door\_cond), action in rules.items():

if (temp\_cond is None or temp\_cond(temperature)) and \

(light\_cond is None or light\_cond(is\_dark, at\_home)) and \

(home\_cond is None or home\_cond(at\_home)) and \

(sec\_cond is None or sec\_cond(security\_armed, door\_open)) and \

(door\_cond is None or door\_cond(door\_open)):

actions.append(action)

if not actions:

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

return actions

def main():

try:

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): ")

if is\_dark not in ['yes', 'no'] or at\_home not in ['yes', 'no'] or \

security\_armed not in ['yes', 'no'] or door\_open not in ['yes', 'no']:

print("Error: All inputs must be 'yes' or 'no'.")

return

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

for action in actions:

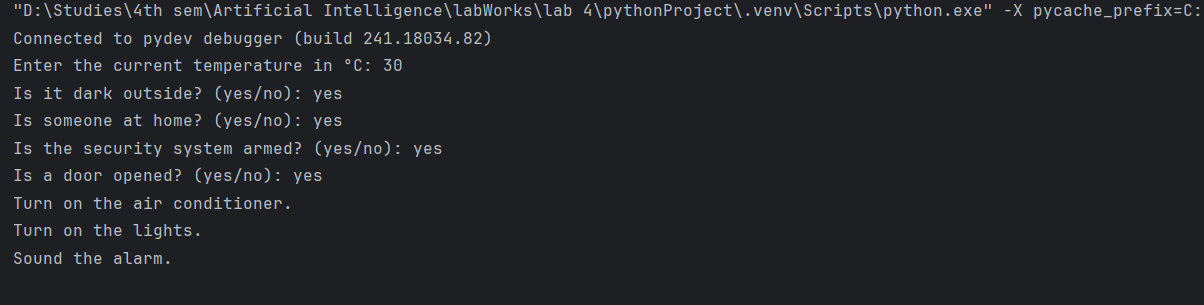
print(action)

except ValueError as e:

print(f"Error: {e}")

main()

**Output:**

****

**Conclusion:**

In this lab exercise, we implemented and analyzed five distinct programs, each designed to handle specific scenarios using conditional logic and rules-based evaluations. Here is a summary of the key outcomes and insights from each program:

Weather Forecast Program:

Purpose: Determine the weather forecast based on sky conditions, temperature, and wind status.

Outcome: The program successfully uses conditional statements to assess multiple weather conditions and provide appropriate forecasts, demonstrating how simple rules can guide decision-making.

Loan Eligibility Checker:

Purpose: Evaluate an applicant's eligibility for a loan based on age, income stability, credit score, criminal record, and previous loan defaults.

Outcome: The program checks various conditions using conditional statements to determine eligibility. It highlights how combining multiple criteria can lead to a comprehensive evaluation of an applicant’s suitability.

Time-Based Decision Making:

Purpose: Make decisions based on the current time, whether it’s a weekday, and if the weather is sunny.

Outcome: This program demonstrates the use of time-based rules to guide daily activities. It shows how specific time windows and contextual factors can influence decision-making in a structured manner.

Traffic Light Action Determiner:

Purpose: Determine actions based on traffic light color and the status of a pedestrian button.

Outcome: The program uses straightforward condition checks to determine the appropriate action based on traffic light status and pedestrian button inputs. It illustrates how traffic management systems can be controlled through simple rule-based logic.

Smart Home System:

Purpose: Control home automation actions based on temperature, lighting, home occupancy, security system status, and door status.

Outcome: This program employs a combination of conditions to manage various smart home functions. It highlights the flexibility of rule-based systems in automating responses based on environmental and security inputs.