# Bahria University,

## Karachi Campus



LAB EXPERIMENT NO.

**\_06\_**

LIST OF TASKS

|  |  |
| --- | --- |
| **TASK NO** | **OBJECTIVE** |
| 01 | You're developing a security system for a bank vault. Implement a function that checks if the security camera detects unauthorized access. If the camera detects unauthorized access, trigger the alarm system. |
| 02 | You're creating a temperature monitoring system for a server room. Develop a function that checks if the temperature sensor indicates a temperature above the threshold. If the temperature is not above the threshold, ensure the cooling system remains off. |
| 03 | You're building a navigation app for drivers. Write a function that determines if the GPS signal is available. If the GPS signal is available, calculate the route to the destination. |
| 04 | You're developing a scheduling app for students. Implement a function that checks if the user has selected either a morning or evening class. If the user hasn't selected a morning class, assume they've chosen an evening class. |
| 05 | You're developing a game with powerup mechanics. Write a function that simplifies the logic for activating a power-up, considering factors such as player level and available resources. |
| 06 | You're creating a reservation system for a restaurant. Develop a function that adds a new reservation to the system based on the available time slots and seating capacity. |

Submitted On:

Date: 02/04/2024

**Task No 01:** Modus Ponens Task:

**Scenario**: You're developing a security system for a bank vault. Implement a function that checks if the security camera detects unauthorized access. If the camera detects unauthorized access, trigger the alarm system.

**Objective**: Use modus ponens to activate the alarm system when unauthorized access is detected, demonstrating how logical inference triggers actions in a security system.

**Solution:**

def check\_access(camera\_status):

    return camera\_status == "unauthorized"

def trigger\_alarm():

    print("Alarm system activated! Unauthorized access detected.")

def modus\_ponens(camera\_status):

    if check\_access(camera\_status):

        trigger\_alarm()

camera\_status = "unauthorized"

modus\_ponens(camera\_status)

**Output:**



**Task No 02:** Modus Tollens Task:

**Scenario**: You're creating a temperature monitoring system for a server room. Develop a function that checks if the temperature sensor indicates a temperature above the threshold. If the temperature is not above the threshold, ensure the cooling system remains off.

**Objective**: Apply modus tollens to keep the cooling system off when the temperature is not above the threshold, showcasing how logical inference prevents unnecessary actions based on negated conditions

**Solution:**

def check\_temperature(temperature, threshold):

    return temperature > threshold

def turn\_on\_cooling\_system():

    print("Cooling system turned on. Temperature above threshold.")

def modus\_tollens(temperature, threshold):

    if not check\_temperature(temperature, threshold):

        print("Cooling system remains off. Temperature not above threshold.")

temperature = 25

threshold = 30

modus\_tollens(temperature, threshold)

**Output:**



**Task No 03:** Hypothetical Syllogism Task:

**Scenario**: You're building a navigation app for drivers. Write a function that determines if the GPS signal is available. If the GPS signal is available, calculate the route to the destination.

**Objective**: Utilize hypothetical syllogism to calculate the route to the destination when the GPS signal is available, illustrating how logical inference guides actions in a navigation system.

**Solution:**

def is\_gps\_signal\_available():

    return True

def calculate\_route\_to\_destination():

    return "Route calculated"

def navigation\_system():

    if is\_gps\_signal\_available():

        return calculate\_route\_to\_destination()

    else:

        return "GPS signal not available"

print(navigation\_system())

**Output:**

****

**Task No 04:** Disjunctive Syllogism Task:

**Scenario**: You're developing a scheduling app for students. Implement a function that checks if the user has selected either a morning or evening class. If the user hasn't selected a morning class, assume they've chosen an evening class.

**Objective**: Apply disjunctive syllogism to infer the user's class preference when a choice is not explicitly provided, demonstrating how logical inference handles alternative options.

**Solution:**

def has\_selected\_morning\_class():

    return False

def infer\_class\_preference():

    if not has\_selected\_morning\_class():

        return "User has chosen an evening class"

    else:

        return "User has chosen a morning class"

print(infer\_class\_preference())

**Output:**

****

**Task No 05:** Simplification Task:

**Scenario**: You're developing a game with powerup mechanics. Write a function that simplifies the logic for activating a power-up, considering factors such as player level and available resources. **Objective**: Use simplification to streamline the activation logic for power-ups, demonstrating how logical inference simplifies complex decisionmaking in game development.

**Solution:**

def activate\_power\_up(player\_level, available\_resources):

    if player\_level >= 5 and available\_resources:

        return "Power-up activated!"

    else:

        return "Unable to activate power-up."

player\_level = 7

available\_resources = True

print(activate\_power\_up(player\_level, available\_resources))

**Output:**

**A black and white text

Description automatically generated**

**Task No 06:** Addition Task:

**Scenario**: You're creating a reservation system for a restaurant. Develop a function that adds a new reservation to the system based on the available time slots and seating capacity.

**Objective**: Apply the addition rule to incorporate new reservations into the system, showcasing how logical inference updates data based on incoming information in a reservation system.

**Solution:**

def add\_reservation(reservation\_list, time\_slot, seating\_capacity):

    time\_slot\_available = True

    for reservation in reservation\_list:

        if reservation[0] == time\_slot:

            time\_slot\_available = False

            break

    if time\_slot\_available and seating\_capacity > 0:

        reservation\_list.append((time\_slot, seating\_capacity))

        return "Reservation added successfully!"

    else:

        return "Unable to add reservation. Time slot not available or invalid seating capacity."

reservations = [("12:00 PM", 4), ("1:30 PM", 2), ("7:00 PM", 6)]

new\_reservation\_time = "6:30 PM"

new\_reservation\_capacity = 5

print(add\_reservation(reservations, new\_reservation\_time, new\_reservation\_capacity))

**Output:**

****