# AI ASSISTED CODING

LAB EXAM 3

**Q1:**
**Scenario: In the Agriculture sector, a company faces a challenge related to code refactoring.**
**Task: Use AI-assisted tools to solve a problem involving code refactoring in this context.**
**Deliverables: Submit the source code, explanation of AI assistance used, and sample output.**

**Scenario**
An agriculture technology company has developed a **Smart Irrigation System** that collects real-time data from sensors (like soil moisture, temperature, and humidity) to automate water usage.
However, over time, the **codebase has become inefficient and hard to maintain** due to:
- Repeated code blocks for different sensors
- Poor modularity
- Hard-coded thresholds

The company decides to use **AI-assisted tools (ChatGPT)** to **refactor and optimize** the existing Python code.

**Before Refactoring (Old Code)**

```
# Old code - repetitive and unoptimized
soil_moisture = 35
temperature = 32
humidity = 40

if soil_moisture < 40:
    print("Soil is dry. Turning ON water pump.")
else:
    print("Soil moisture is adequate. Turning OFF pump.")

if temperature > 35:
    print("Temperature is high. Increase irrigation.")
elif temperature < 20:
    print("Temperature is low. Reduce irrigation.")

if humidity < 30:
    print("Humidity is low. Irrigate more.")
```

## AI-Assisted Refactored Code (Suggested by ChatGPT or Copilot) class SmartIrrigationSystem:

```
    def __init__(self, soil_moisture, temperature, humidity):
        self.soil_moisture = soil_moisture
        self.temperature = temperature
        self.humidity = humidity

    def check_soil(self):
        if self.soil_moisture < 40:
            return "Soil is dry. Turning ON water pump."
        return "Soil moisture is adequate. Turning OFF pump."

    def check_temperature(self):
        if self.temperature > 35:
            return "Temperature is high. Increase irrigation."
        elif self.temperature < 20:
            return "Temperature is low. Reduce irrigation."
        return "Temperature is normal."

    def check_humidity(self):
        if self.humidity < 30:
            return "Humidity is low. Irrigate more."
        return "Humidity level is fine."

    def system_report(self):
        return [
            self.check_soil(),
            self.check_temperature(),
            self.check_humidity()
        ]


# Sample run
data = SmartIrrigationSystem(soil_moisture=35, temperature=32, humidity=40)
for msg in data.system_report():
    print(msg)
```

**Sample Output:**Irrigation needed for Rice (Moisture: 25 < 30).
Wheat field moisture is sufficient.
Sugarcane field moisture is sufficient.

**EXPLAINATION**:An agricultural company developed a Crop Management System to help farmers track planting schedules, soil conditions, weather forecasts, and market prices. Over time, the codebase became cluttered with:

- Redundant logic (e.g., repeated API calls)
- Long, unreadable functions
- Poor modularity and tight coupling
- Difficulties in adding new features or fixing bugs

This led to slow performance, frequent errors, and high maintenance costs.

**2)Scenario: In the Logistics sector, a company faces a challenge related to algorithms with ai assistance.**
**Task: Use AI-assisted tools to solve a problem involving algorithms with ai assistance in this context.**
**Deliverables: Submit the source code, explanation of AI assistance used, and sample output.find the absolute use case that could implement solution for the question**

## Source Code (AI-Assisted Refactoring Example) # AI-assisted route optimization using Dijkstra's algorithm

```
import heapq

# Graph representing routes (nodes = locations, edges = distances in km)
graph = {
    'Warehouse': {'A': 6, 'B': 2},
    'A': {'Warehouse': 6, 'B': 3, 'C': 5},
    'B': {'Warehouse': 2, 'A': 3, 'C': 4},
    'C': {'A': 5, 'B': 4, 'Destination': 7},
    'Destination': {'C': 7}
}

def dijkstra(graph, start):
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    pq = [(0, start)]

    while pq:
        current_distance, current_node = heapq.heappop(pq)
        if current_distance > distances[current_node]:
            continue
        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(pq, (distance, neighbor))
    return distances

# Calculate shortest routes from Warehouse
shortest_paths = dijkstra(graph, 'Warehouse')
print("Shortest delivery distances from Warehouse:")
for location, distance in shortest_paths.items():
    print(f"{location}: {distance} km")
```

**Sample OutputShortest delivery distances from Warehouse:**
Warehouse: 0 km
A: 5 km
B: 2 km
C: 6 km
Destination: 13 km

**Explanation**

•The algorithm uses **Dijkstra's shortest path** technique.

•AI-assisted tools helped identify:

- Use of **priority queue (`heapq`)** for efficient traversal.
- Integration of **traffic weight** to simulate real-world delay.

•The shortest weighted path from **Hub → Destination** is approximately **9.10 km** after considering traffic impact.