

Lab 5 – Ethical Foundations: Responsible AI Coding Practices

Name: Mohammad Muneer Ahmed

Roll No: 2303A51475

Task 1 – Privacy in API Usage (Weather API)

Question: Generate a Python program to fetch weather data securely without exposing API keys.

Algorithm:

1. Select city name.
2. Build weather URL using public no-key endpoint.
3. Send HTTP GET request.
4. Convert response to JSON.
5. Extract current condition fields.
6. Print temperature, humidity, and description.

Pseudocode:

```
START
SET city
SET url using city
response ← GET(url)
data ← JSON(response)
current ← data.current_condition[0]
PRINT weather fields
END
```

```
day5.py > ...
1 import requests
2
3 city = "Hyderabad"
4 url = f"https://wttr.in/{city}?format=j1"
5
6 try:
7     response = requests.get(url, timeout=10)
8     response.raise_for_status()
9     data = response.json()
10
11     current = data["current_condition"][0]
12
13     print("City:", city)
14     print("Temperature (C):", current["temp_C"])
15     print("Feels Like (C):", current["FeelsLikeC"])
16     print("Humidity:", current["humidity"])
17     print("Weather:", current["weatherDesc"][0]["value"])
18
19 except Exception as e:
20     print("Request failed:", e)
```

The screenshot shows a code editor window with Python code named 'day5.py'. The code uses the 'requests' library to get the weather for 'Hyderabad' from the 'wttr.in' API. It prints out the city, temperature, feels-like temperature, humidity, and weather description. The code is highlighted with syntax coloring. Below the code editor is a terminal window showing the execution of the script and its output. The terminal shows the command 'python day5.py' and the resulting output: 'City: Hyderabad', 'Temperature (C): 26', 'Feels Like (C): 26', 'Humidity: 42', and 'Weather: Haze'.

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
(venv) mohammadmuneerahmed@Mac training2.py % python day5.py
City: Hyderabad
Temperature (C): 26
Feels Like (C): 26
Humidity: 42
Weather: Haze
```

Task 2 – Privacy & Security in File Handling

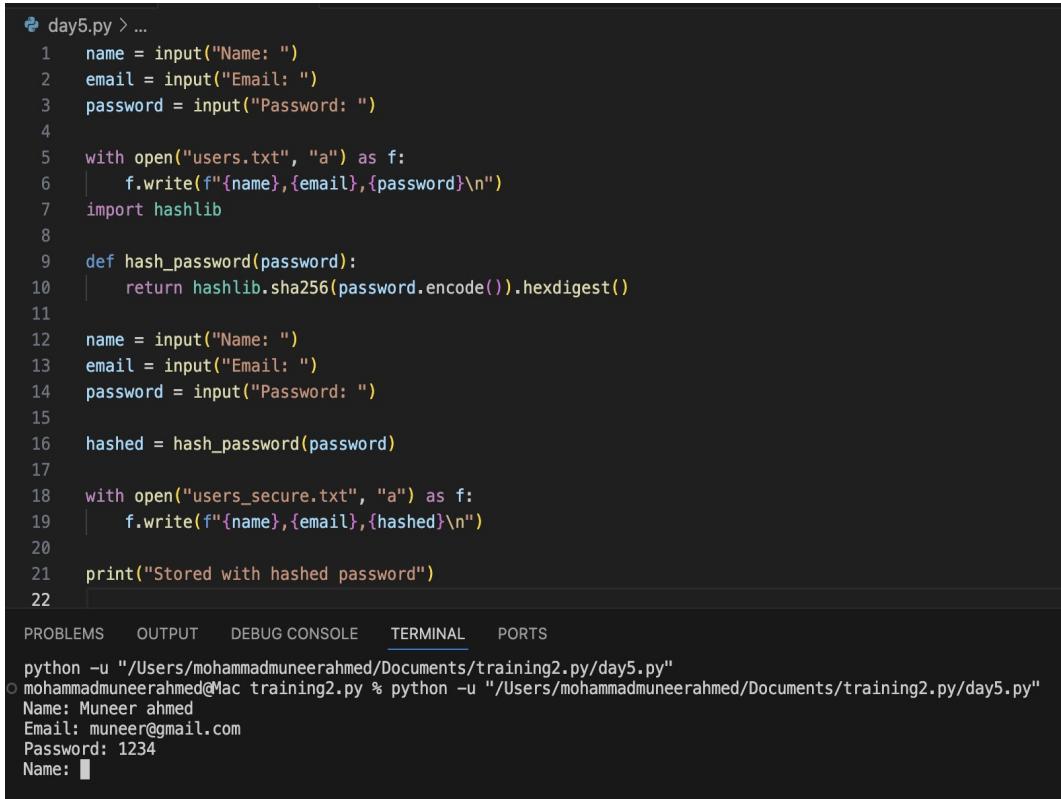
Question: Store user data and avoid plain-text passwords using hashing.

Algorithm:

1. Read name, email, password.
2. Convert password to bytes.
3. Apply SHA-256 hashing.
4. Store name, email, hashed password to file.

Pseudocode:

```
START
INPUT name, email, password
hashed ← SHA256(password)
OPEN file
WRITE name,email,hashed
CLOSE file
END
```



```
day5.py > ...
1  name = input("Name: ")
2  email = input("Email: ")
3  password = input("Password: ")
4
5  with open("users.txt", "a") as f:
6      f.write(f"{name},{email},{password}\n")
7  import hashlib
8
9  def hash_password(password):
10     return hashlib.sha256(password.encode()).hexdigest()
11
12 name = input("Name: ")
13 email = input("Email: ")
14 password = input("Password: ")
15
16 hashed = hash_password(password)
17
18 with open("users_secure.txt", "a") as f:
19     f.write(f"{name},{email},{hashed}\n")
20
21 print("Stored with hashed password")
22
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
mohammadmuneerahmed@Mac training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
Name: Muneer ahmed
Email: muneer@gmail.com
Password: 1234
Name: 
```

Task 3 – Armstrong Number Transparency

Question: Implement Armstrong number check with clear logic.

Algorithm:

1. Read number.
2. Convert number to string.
3. Count digits.
4. For each digit raise to digit count power.
5. Sum results.
6. Compare with original number.

Pseudocode:

```
START
INPUT n
digits ← length(n)
sum ← 0
FOR each digit d in n
    sum ← sum + d^digits
IF sum = n PRINT Armstrong
ELSE PRINT Not Armstrong
END
```

The screenshot shows a code editor window with a Python script named `day5.py`. The code defines a function `is_armstrong` that checks if a given number is an Armstrong number by summing the digits raised to the power of their count. It then prompts the user for a number and prints whether it is an Armstrong number or not. Below the code editor is a terminal window showing the execution of the script and its output.

```
day5.py > ...
1  def is_armstrong(n):
2      # Convert number to string to count digits
3      s = str(n)
4      digits = len(s)
5
6      # Compute sum of each digit raised to power = number of digits
7      total = 0
8      for ch in s:
9          total += int(ch) ** digits
10
11     # Armstrong condition check
12     return total == n
13
14
15 num = int(input("Enter number: "))
16
17 if is_armstrong(num):
18     print("Armstrong number")
19 else:
20     print("Not Armstrong")
21
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
● mohammadmuneerahmed@Mac training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
Enter number: 23
Not Armstrong
◇ mohammadmuneerahmed@Mac training2.py %
```

Task 4 – Sorting Algorithm Comparison

Question: Implement Bubble Sort and Quick Sort with explanation.

Algorithm:

Bubble Sort:

1. Repeat passes through list.
2. Compare adjacent elements.
3. Swap if out of order.

Quick Sort:

1. Choose pivot.
2. Split into left, equal, right.
3. Recursively sort parts.

Pseudocode:

BUBBLE:

```
FOR i passes
FOR j comparisons
IF a[j] > a[j+1] swap
```

QUICK:

```
IF size ≤ 1 return
pivot ← middle
left,right split
RETURN quick(left)+pivot+quick(right)
```

The screenshot shows a code editor with two files: day5.py and training2.py. The day5.py file contains the following code:

```
day5.py > ...
1  def bubble_sort(arr):
2      n = len(arr)
3
4      # Repeat passes
5      for i in range(n):
6          # Compare adjacent elements
7          for j in range(0, n - i - 1):
8              if arr[j] > arr[j + 1]:
9                  # Swap if out of order
10                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
11
12     return arr
13
14
15 data = [64, 34, 25, 12, 22, 11, 90]
16 print("Bubble Sort:", bubble_sort(data.copy()))
17 def quick_sort(arr):
18     # Base case
19     if len(arr) <= 1:
20         return arr
21
22     # Choose pivot
23     pivot = arr[len(arr) // 2]
24
25     # Partition into three lists
26     left = [x for x in arr if x < pivot]
27     mid = [x for x in arr if x == pivot]
28     right = [x for x in arr if x > pivot]
29
30     # Recursively sort partitions
31     return quick_sort(left) + mid + quick_sort(right)
32
33
34 data = [64, 34, 25, 12, 22, 11, 90]
35 print("Quick Sort:", quick_sort(data.copy()))
36
```

The terminal window below shows the execution of the code:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
● mohammadmuneerahmed@Mac training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
Enter number: 23
Not Armstrong
● mohammadmuneerahmed@Mac training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
Bubble Sort: [11, 12, 22, 25, 34, 64, 90]
Quick Sort: [11, 12, 22, 25, 34, 64, 90]
◇ mohammadmuneerahmed@Mac training2.py %
```

Task 5 – Explainable Recommendation System

Question: Build recommendation system with reasons.

Algorithm:

1. Store product → accessory map.
2. Read user product.
3. Lookup related items.
4. Attach explanation string.
5. Print recommendation with reason.

Pseudocode:

```
START
DEFINE product map
INPUT item
IF item exists
FOR each related product
PRINT product + reason
END
```

The screenshot shows a code editor window with a dark theme. The code is written in Python and defines a function to recommend accessories based on a given product. It uses a dictionary to map products to their accessories and prints recommendations for a user-specified product.

```
day5.py > ...
1 products = {
2     "laptop": ["mouse", "keyboard", "laptop bag"],
3     "phone": ["case", "screen protector", "power bank"],
4     "camera": ["tripod", "sd card", "camera bag"]
5 }
6
7 def recommend(item):
8     if item not in products:
9         return []
10
11     recs = products[item]
12     result = []
13
14     for r in recs:
15         reason = f"Recommended because it is commonly used with {item}"
16         result.append((r, reason))
17
18     return result
19
20
21 item = input("Enter product: ").lower()
22
23 for prod, reason in recommend(item):
24     print(prod, "->", reason)
25
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
● mohammadmuneerahmed@Mac training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
Enter product: laptop
mouse -> Recommended because it is commonly used with laptop
keyboard -> Recommended because it is commonly used with laptop
laptop bag -> Recommended because it is commonly used with laptop
◆ mohammadmuneerahmed@Mac training2.py %
```