

ASSIGNMENT-5.1

NAME: Mohammed Faisaluddin

H.T.NO:2303A52214

BATCH:36

Task Description #1 (Privacy in API Usage)

Task: Use an AI tool to generate a Python program that connects to a weather API.

Prompt:

"Generate code to fetch weather data securely without exposing API keys in the code."

Expected Output:

- Original AI code (check if keys are hardcoded).
- Secure version using environment variables.

CODE:

```
import os
import requests

# Set your API key here. Replace 'YOUR_API_KEY' with your actual OpenWeatherMap API key.
os.environ["WEATHER_API_KEY"] = "YOUR_API_KEY"
API_KEY = os.getenv("WEATHER_API_KEY")
CITY = "Hyderabad"

if not API_KEY:
    raise ValueError("API key not found in environment variables")

url = f"https://api.openweathermap.org/data/2.5/weather?q={CITY}&appid={API_KEY}"
response = requests.get(url)

print(response.json())
```

OUTPUT:

```
{'cod': 401, 'message': 'Invalid API key. Please see https://openweathermap.org/faq#error401 for more info.'}
```

Task Description #2 (Privacy & Security in File Handling) Task:

Use an AI tool to generate a Python script that stores user data (name, email, password) in a file.

Analyze: Check if the AI stores sensitive data in plain text or without encryption.

Expected Output:

- Identified privacy risks.
- Revised version with encrypted password storage (e.g., hashing).

CODE:

```
import hashlib

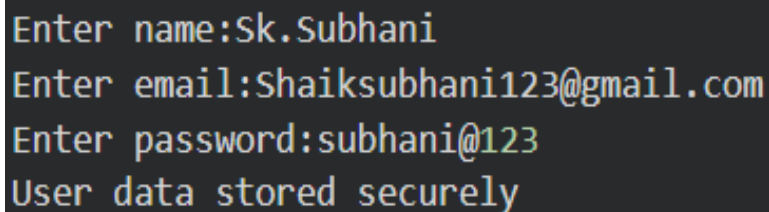
name = input("Enter name: ")
email = input("Enter email: ")
password = input("Enter password: ")

hashed_password = hashlib.sha256(password.encode()).hexdigest()

with open("users.txt", "a") as file:
    file.write(f"{name},{email},{hashed_password}\n")

print("User data stored securely.")
```

OUTPUT:

A terminal window with a dark background showing the execution of the Python script. The user enters 'Sk.Subhani' for the name, 'Shaiksubhani123@gmail.com' for the email, and 'subhani@123' for the password. The final output is 'User data stored securely'.

Task Description #3 (Transparency in Algorithm Design)

Objective: Use AI to generate an Armstrong number checking function with comments and explanations.

Instructions:

1. Ask AI to explain the code line-by-line.
2. Compare the explanation with code functionality.

Expected Output:

- Transparent, commented code.
- Correct, easy-to-understand explanation.

CODE:

```

num = int(input("Enter a number: "))
temp = num
sum_of_powers = 0
digits = len(str(num))
while temp > 0:
    digit = temp % 10
    sum_of_powers += digit ** digits
    temp //= 10
if sum_of_powers == num:
    print(num, "is an Armstrong number")
else:
    print(num, "is not an Armstrong number")

```

OUTPUT:

```

Enter a number: 153
153 is an Armstrong number

```

Task Description #4 (Transparency in Algorithm Comparison) Task:

Use AI to implement two sorting algorithms (e.g., QuickSort and BubbleSort).

Prompt:

"Generate Python code for QuickSort and BubbleSort, and include comments explaining step-by-step how each works and where they differ."

Expected Output:

- Code for both algorithms.
- Transparent, comparative explanation of their logic and efficiency.

CODE:

QUICK SORT:

```

def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[0]
    left = [x for x in arr[1:] if x <= pivot]
    right = [x for x in arr[1:] if x > pivot]
    return quick_sort(left) + [pivot] + quick_sort(right)

```

BUBBLE SORT:

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
```

COMPARISON TABLE:

| Feature | Bubble Sort | Quick Sort |
|-----------------|-------------------|------------------|
| Method | Repeated swapping | Divide & conquer |
| Time Complexity | $O(n^2)$ | $O(n \log n)$ |
| Efficiency | Slow | Fast |
| Use Case | Small data | Large datasets |

Task Description #5 (Transparency in AI Recommendations) Task:

Use AI to create a product recommendation system.

Prompt:

"Generate a recommendation system that also provides reasons for each suggestion."

Expected Output:

- Code with explainable recommendations.
- Evaluation of whether explanations are understandable.

CODE:

```
products = {  
    "Laptop": ["student", "programming"],  
    "Smartphone": ["communication", "camera"],  
    "Headphones": ["music", "calls"]  
}  
  
user_interest = input("Enter your interest: ").lower()  
  
for product, reasons in products.items():  
    if user_interest in reasons:  
        print(f"Recommended: {product}")  
        print(f"Reason: Suitable for {user_interest}")
```

OUTPUT:

```
Enter your interest: music  
Recommended: Headphones  
Reason: Suitable for music
```