

AI- ASSISTANT CODING

ASSIGNMENT 5.1

2303A51836

BATCH 27

**Task Description #1 – Privacy in API Usage**

**TASK**

Use an AI tool to generate a Python program that connects to a weather API securely without exposing API keys.

**PROMPT**

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

**CODE**

**✗ Insecure Version (Hardcoded API Key)**

```
import requests  
  
API_KEY = "12345ABCDE" # Hardcoded API key (NOT SAFE)  
  
CITY = "Hyderabad"
```

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

```
print(response.json())
```

**✓ Secure Version (Using Environment Variables)**

```
import requests  
  
import os  
  
API_KEY = os.getenv("WEATHER_API_KEY") # Read from environment variable  
CITY = "Hyderabad"
```

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

## OUTPUT

- Weather details of the given city are displayed in JSON format.
- API key is **not visible** in the code.

## EXPLANATION

- Hardcoding API keys is a **privacy risk**.
  - Environment variables keep keys hidden and secure.
  - This approach prevents accidental exposure on GitHub or public files.
- 

## Task Description #2 – Privacy & Security in File Handling

### TASK

Store user data securely by avoiding plain-text password storage.

### PROMPT

Generate a Python script that stores user data securely instead of plain text.

### CODE

#### **X Insecure Version (Plain Text Password)**

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

```
with open("users.txt", "a") as file:
```

```
    file.write(f"{name},{email},{password}\n")
```

#### **✓ Secure Version (Hashed Password)**

```
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")
```

#### OUTPUT

- User data is stored.
- Password appears as a **hashed value**, not readable text.

#### EXPLANATION

- Plain text passwords can be stolen easily.
  - Hashing converts passwords into irreversible values.
  - Even if the file is leaked, passwords remain protected.
- 

### Task Description #3 – Transparency in Algorithm Design

#### TASK

Create an Armstrong number checker with transparent explanation.

#### PROMPT

Generate Python code to check an Armstrong number and explain it line by line.

#### CODE

```
def is_armstrong(number):  
    digits = list(map(int, str(number)))  
    power = len(digits)  
  
    total = sum(d ** power for d in digits)  
  
    return total == number
```

```
num = int(input("Enter a number: "))  
print(is_armstrong(num))
```

#### OUTPUT

Enter a number: 153

True

#### EXPLANATION

- Number is converted into digits.
- Each digit is raised to the power of total digits.

- If the sum equals the original number, it is an Armstrong number.
  - Code and explanation match clearly, ensuring transparency.
- 

## Task Description #4 – Transparency in Algorithm Comparison

### TASK

Implement and compare QuickSort and BubbleSort.

### PROMPT

Generate Python code for QuickSort and BubbleSort with step-by-step explanations.

### CODE

```
# Bubble Sort

def bubble_sort(arr):

    for i in range(len(arr)):

        for j in range(0, len(arr)-i-1):

            if arr[j] > arr[j+1]:

                arr[j], arr[j+1] = arr[j+1], arr[j]

    return arr
```

### # 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)
```

### OUTPUT

Bubble Sort: [1, 2, 4, 5, 8]

Quick Sort: [1, 2, 4, 5, 8]

## EXPLANATION

- BubbleSort compares adjacent elements repeatedly.
- QuickSort divides the list using a pivot.
- BubbleSort is slower ( $O(n^2)$ ).
- QuickSort is faster ( $O(n \log n)$ ).

Aspect	Bubble Sort	Quick Sort
Method	Swapping adjacent elements	Divide and conquer
Time Complexity	$O(n^2)$	$O(n \log n)$
Efficiency	Slow	Fast
Use Case	Small lists	Large datasets

---

## Task Description #5 – Transparency in AI Recommendations

### TASK

Create a recommendation system with explainable suggestions.

### PROMPT

Generate a recommendation system that explains why each product is suggested.

### CODE

```
def recommend_products(user_interest):  
    products = {  
        "fitness": ["Dumbbells", "Yoga Mat"],  
        "technology": ["Laptop", "Smartphone"],  
        "books": ["Fiction Novel", "Self-help Book"]  
    }
```

```
recommendations = products.get(user_interest, [])
```

```
for item in recommendations:
```

```
    print(f"Recommended: {item} because it matches your interest in {user_interest}.")
```

### OUTPUT

Recommended: Laptop because it matches your interest in technology.

Recommended: Smartphone because it matches your interest in technology.

#### **EXPLANATION**

- Recommendations are based on user interests.
  - Each suggestion includes a clear reason.
  - Improves transparency and user trust.
-