Lab 5.1

Name:K Pavani

Batch:27

Hall-ticket:2303A51494

Task 1:

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

Help

ChatGPT 5.2 ⌄

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

Output:

ChatGPT 5.2 ⌄

python   Copy code

```python
import os
import requests

def get_weather(city):
    api_key = os.getenv("WEATHER_API_KEY")
    if not api_key:
        raise RuntimeError("API key not found in environment variables")

    url = "https://api.openweathermap.org/data/2.5/weather"
    params = {
        "q": city,
        "appid": api_key,
        "units": "metric"
    }

    response = requests.get(url, params=params, timeout=10)
    response.raise_for_status()
    return response.json()

# Example usage
weather = get_weather("Delhi")
print(weather["main"]["temp"])
```

Secure code:

Windows (PowerShell)

```powershell
powershell

setx WEATHER_API_KEY "your_api_key_here"
```

```python
import os
import requests

def get_weather(city):
    # Read API key securely from environment variable
    api_key = os.getenv("WEATHER_API_KEY")

    # Check if API key exists
    if not api_key:
        raise Exception("API key not found. Set WEATHER_API_KEY as an environment variable.")

    # Weather API endpoint
    url = "https://api.openweathermap.org/data/2.5/weather"

    # Request parameters
    params = {
        "q": city,
        "appid": api_key,
        "units": "metric"
    }

    # Send request securely
    response = requests.get(url, params=params, timeout=10)
    response.raise_for_status()

    return response.json()


# ---- Function Call ----
city = input("Enter city name: ")
weather_data = get_weather(city)

print("Temperature:", weather_data["main"]["temp"])
print("Weather:", weather_data["weather"][0]["description"])
```

- **Explanation:**
  Original AI code is insecure because the API key is hardcoded directly in the source code.

- Hardcoded keys can be easily exposed if the code is shared, pushed to GitHub, or leaked.

- Secure version removes the API key from code and stores it in environment variables instead.

- Environment variables protect sensitive data by loading the API key only at runtime.

- This approach follows secure coding best practices, allowing easy key rotation without code changes

Task 2:

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

Prompt:

Generate a Python script that stores user data (name, email, and password) in a file.

Output:

```python
def store_user_data(name, email, password):
    with open("users.txt", "a") as file:
        file.write(f"Name: {name}\n")
        file.write(f"Email: {email}\n")
        file.write(f"Password: {password}\n")
        file.write("-" * 25 + "\n")

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

store_user_data(name, email, password)
print("User data stored successfully.")
```

Explanation:

- Passwords are stored in plain text, so anyone can read them directly.
- No encryption or hashing is used to protect sensitive data.
- If the file is leaked or stolen, all user accounts are compromised.
- Even authorized users can see passwords, enabling insider misuse.
- This approach violates basic security and privacy standards

✅ Secure Python Script Using Password Hashing

```python
import hashlib

def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

def store_user_data(name, email, password):
    hashed_password = hash_password(password)

    with open("users_secure.txt", "a") as file:
        file.write(f"Name: {name}\n")
        file.write(f"Email: {email}\n")
        file.write(f"Password Hash: {hashed_password}\n")
        file.write("-" * 25 + "\n")

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

store_user_data(name, email, password)
print("User data stored securely.")
```

Why This Version Is Secure

1. Passwords are never stored in plain text.

2. Hashing is one-way, so the original password cannot be retrieved.

3. Even if the file is exposed, real passwords are not revealed.

4. The approach follows secure coding best practices.

Task 3:

Objective: Use AI to generate an Armstrong number checking function

with comments and explanations.

Prompt:

Generate a Python function to check whether a given number is an Armstrong number.

Include clear comments in the code.

After the code, explain the program line by line in simple and easy-to-understand language.

Ensure the explanation accurately matches what each line of the code does.

Output:

```python
def is_armstrong(number):
    # Convert the number to a string to access each digit
    num_str = str(number)

    # Count how many digits the number has
    num_digits = len(num_str)

    # Variable to store the sum of powered digits
    total = 0

    # Loop through each digit in the number
    for digit in num_str:
        # Convert digit to integer and raise it to the power of number of digits
        total += int(digit) ** num_digits

    # Check if the calculated sum equals the original number
    if total == number:
        return True
    else:
        return False


# ---- Function Call ----
num = int(input("Enter a number: "))

if is_armstrong(num):
    print("The number is an Armstrong number.")
else:
    print("The number is not an Armstrong number.")
```

```
Enter a number: 153
The number is an Armstrong number.
```

- **Explanation:**
  The code is transparent and easy to read because each step is clearly written and logically structured.
- Meaningful comments explain what every important line does, making the code beginner-friendly.
- The logic is simple and explicit, so there is no hidden or confusing behavior in the program.
- The function works only on user input and does not store or expose any sensitive data, ensuring safety.
- Clear structure and comments help in easy debugging, maintenance, and secure understanding of the program.

Task 4: 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

🗗 👍 👎 ⬆ …

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

Output:

```python
def bubble_sort(arr):
    n = len(arr)

    # Loop through the array multiple times
    for i in range(n):
        # Compare adjacent elements in each pass
        for j in range(0, n - i - 1):
            # If current element is greater than the next one, swap them
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

    # Return the sorted array
    return arr
def quick_sort(arr):
    # Base case: if array has 1 or no elements, it is already sorted
    if len(arr) <= 1:
        return arr

    # Choose the last element as pivot
    pivot = arr[-1]

    # Elements smaller than pivot
    left = []

    # Elements greater than pivot
    right = []

    # Compare each element (except pivot) with pivot
    for element in arr[:-1]:
        if element < pivot:
            left.append(element)
        else:
            right.append(element)

    # Recursively sort left and right parts and combine them
    return quick_sort(left) + [pivot] + quick_sort(right)
numbers = [5, 3, 8, 4, 2]

print("Bubble Sort Result:", bubble_sort(numbers.copy()))
print("Quick Sort Result:", quick_sort(numbers))
```

```
Bubble Sort Result: [2, 3, 4, 5, 8]
Quick Sort Result: [2, 3, 4, 5, 8]
```

- **Explanation:**
  Bubble Sort is easy to understand but inefficient for large inputs.
- Quick Sort is faster and more efficient using recursion and partitioning.

- Comments make both codes transparent and easy to follow.
- The difference lies in comparison method and performance.

Task 5: Use AI to create a product recommendation system.

Prompt:
Generate a recommendation system that also provides reasons for each

suggestion.

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

Output:

```python
def recommend_movies(user_preferences, movie_database):
    recommendations = []

    # Loop through each movie in the database
    for movie, genres in movie_database.items():
        matched_genres = []

        # Check which user preferences match the movie genres
        for pref in user_preferences:
            if pref in genres:
                matched_genres.append(pref)

        # If at least one preference matches, recommend the movie
        if matched_genres:
            reason = f"Recommended because it matches your interest in {', '.join(matched_genres)}"
            recommendations.append((movie, reason))

    return recommendations


# ----- Sample Data -----
user_preferences = ["Action", "Sci-Fi"]

movie_database = {
    "Inception": ["Action", "Sci-Fi", "Thriller"],
    "Titanic": ["Romance", "Drama"],
    "Avengers": ["Action", "Fantasy"],
    "Interstellar": ["Sci-Fi", "Drama"],
    "Notebook": ["Romance"]
}

# ----- Function Call -----
results = recommend_movies(user_preferences, movie_database)

# Display recommendations with reasons
for movie, reason in results:
    print(f"{movie}: {reason}")
```

```
Inception: Recommended because it matches your interest in Action, Sci-Fi
Avengers: Recommended because it matches your interest in Action
Interstellar: Recommended because it matches your interest in Sci-Fi
```

Explanation:

- The code generates recommendations along with clear reasons for each suggestion.
- Each explanation directly states which user preferences matched the item.
- The recommendation logic is transparent and easy to follow.
- Explanations are written in simple, human-readable language.

- Users can easily understand and trust why each recommendation is made.