

AI ASSISTED CODING

LAB ASS-5.1

J.Varshini

2303A51758

BATCH-11

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."

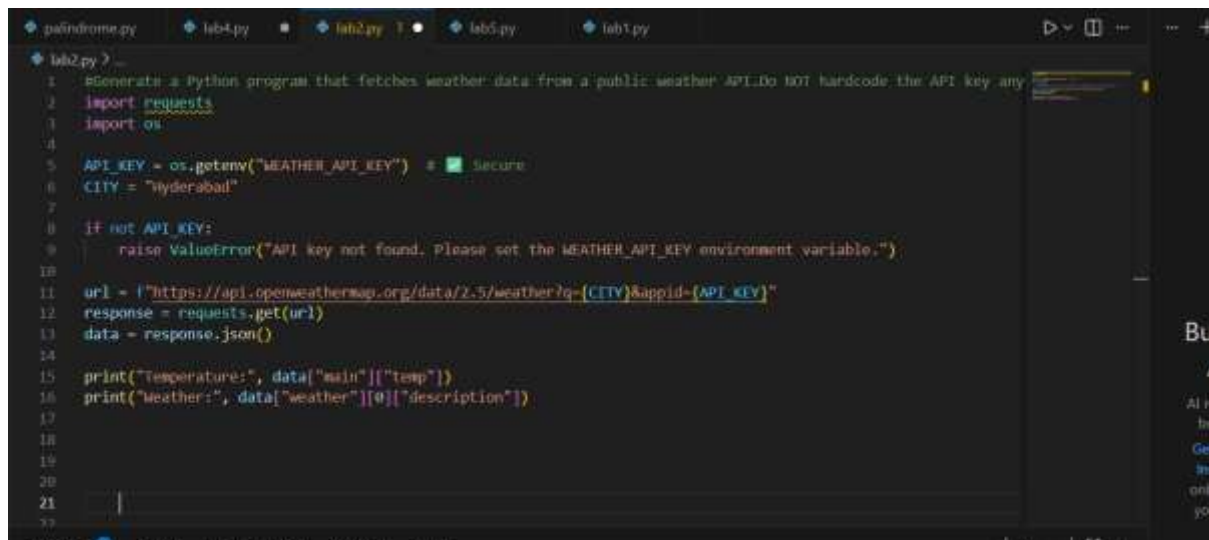
Expected Output:

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

PROMPT:

#Generate a Python program that fetches weather data from a public weather API.Do NOT hardcode the API key anywhere in the source code.The API key MUST be read only from an environment variable using the os module.If the API key is missing,the program should raise an error.Do not include placeholder API keys in the code.

CODE:

A screenshot of a code editor showing a Python script named lab2.py. The script is designed to fetch weather data from a public API. It starts with a comment: "#Generate a Python program that fetches weather data from a public weather API. Do NOT hardcode the API key any". The code imports 'requests' and 'os'. It then sets 'API_KEY' to 'os.getenv("WEATHER_API_KEY")' with a 'Secure' icon next to it, and 'CITY' to 'Hyderabad'. A conditional check 'if not API_KEY:' raises a 'ValueError' with the message "API key not found. Please set the WEATHER_API_KEY environment variable.". The script constructs a URL: 'url = f"https://api.openweathermap.org/data/2.5/weather?q={CITY}&appid={API_KEY}"', makes a GET request, and parses the JSON response. Finally, it prints the temperature and weather description. The editor has a dark theme and shows several tabs at the top: 'palindrome.py', 'lab4.py', 'lab2.py' (active), 'lab5.py', and 'lab1.py'.

```
1 #Generate a Python program that fetches weather data from a public weather API. Do NOT hardcode the API key any
2 import requests
3 import os
4
5 API_KEY = os.getenv("WEATHER_API_KEY") # Secure
6 CITY = "Hyderabad"
7
8 if not API_KEY:
9     raise ValueError("API key not found. Please set the WEATHER_API_KEY environment variable.")
10
11 url = f"https://api.openweathermap.org/data/2.5/weather?q={CITY}&appid={API_KEY}"
12 response = requests.get(url)
13 data = response.json()
14
15 print("Temperature:", data["main"]["temp"])
16 print("Weather:", data["weather"][0]["description"])
17
18
19
20
21
```

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.

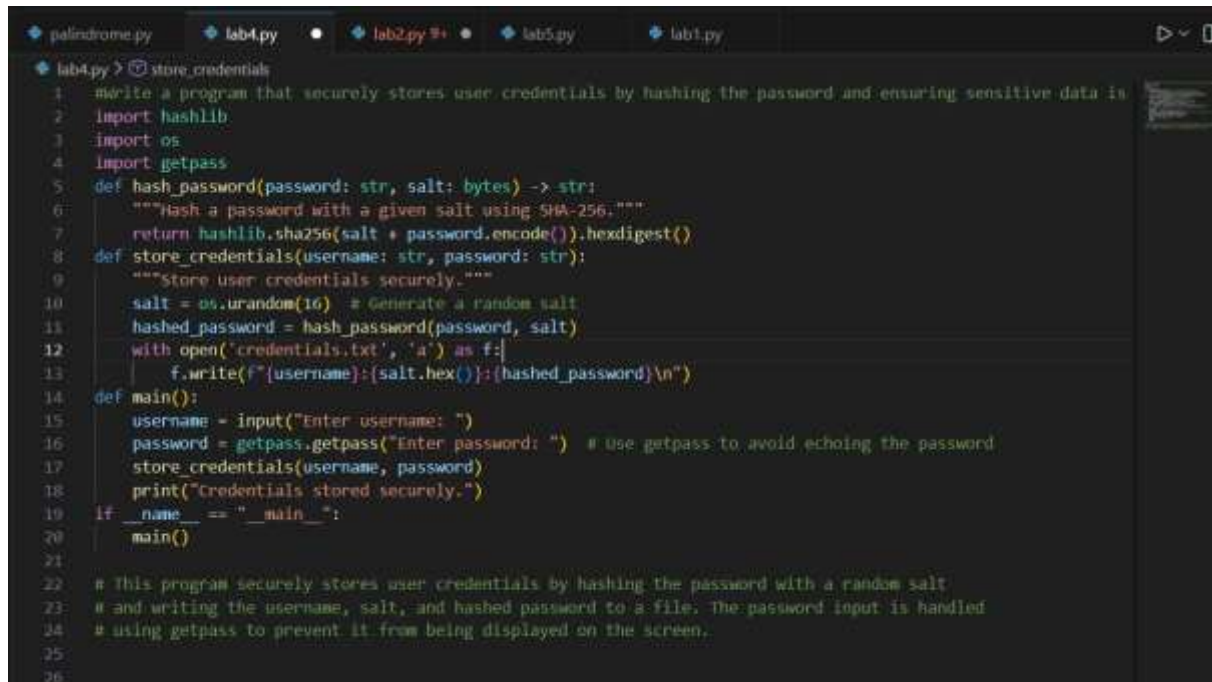
Expected Output:

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

PROMPT:

#Write a program that securely stores user credentials by hashing the password and ensuring sensitive data is not stored in plain text.

CODE:



```
lab4.py > store_credentials
1  #Write a program that securely stores user credentials by hashing the password and ensuring sensitive data is
2  import hashlib
3  import os
4  import getpass
5  def hash_password(password: str, salt: bytes) -> str:
6      """Hash a password with a given salt using SHA-256."""
7      return hashlib.sha256(salt + password.encode()).hexdigest()
8  def store_credentials(username: str, password: str):
9      """Store user credentials securely."""
10     salt = os.urandom(16) # Generate a random salt
11     hashed_password = hash_password(password, salt)
12     with open('credentials.txt', 'a') as f:
13         f.write(f"{username}:{salt.hex()}:{hashed_password}\n")
14 def main():
15     username = input("Enter username: ")
16     password = getpass.getpass("Enter password: ") # Use getpass to avoid echoing the password
17     store_credentials(username, password)
18     print("Credentials stored securely.")
19 if __name__ == "__main__":
20     main()
21
22 # This program securely stores user credentials by hashing the password with a random salt
23 # and writing the username, salt, and hashed password to a file. The password input is handled
24 # using getpass to prevent it from being displayed on the screen.
25
26
```

OUTPUT:



```
Enter username: 123456
Enter password:
Credentials 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 **PROMPT:**

#write a code to generate armstrong number checking functions with comments **CODE:**

```
lab1.py > ...
1  write a code to generate armstrong number checking functions with comments
2  def is_armstrong_number(num):
3      """
4      This function checks if a number is an Armstrong number.
5      An Armstrong number is a number that is equal to the sum of its own digits
6      each raised to the power of the number of digits.
7      For example, 153 is an Armstrong number because 1^3 + 5^3 + 3^3 = 153.
8      """
9      # Convert the number to string to easily iterate over digits
10     digits = str(num)
11     num_digits = len(digits)
12
13     # Calculate the sum of each digit raised to the power of num_digits
14     sum_of_powers = sum(int(digit) ** num_digits for digit in digits)
15
16     # Check if the sum of powers is equal to the original number
17     return sum_of_powers == num
18
19 # Example usage
20 if __name__ == "__main__":
21     test_number = 153
22     if is_armstrong_number(test_number):
23         print(f"{test_number} is an Armstrong number.")
24     else:
25         print(f"{test_number} is not an Armstrong number.")
26     test_number = 123
27     if is_armstrong_number(test_number):
28         print(f"{test_number} is an Armstrong number.")
29     else:
30         print(f"{test_number} is not an Armstrong number.")
31
32
```

OUTPUT:

```
153 is an Armstrong number.
123 is not 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.

PROMPT:

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

CODE:

```

1 # Bubble Sort
2 # This function sorts the array in ascending order, and includes comments explaining how it works, how each step works, and where they differ.
3 def bubble_sort(arr):
4     """Bubble sort algorithm sorts an array by repeatedly stepping through the list,
5     comparing adjacent elements and swapping them if they are in the wrong order.
6     The process is repeated until the list is sorted.
7     """
8     n = len(arr)
9     # Traverse through all array elements
10    for i in range(n):
11        # Last i elements are already sorted, no need to check them
12        for j in range(0, n-i-1):
13            # Compare adjacent elements
14            if arr[j] > arr[j+1]:
15                # Swap if they are in the wrong order
16                arr[j], arr[j+1] = arr[j+1], arr[j]
17    return arr
18
19 # Quick Sort
20 """Quick sort algorithm sorts an array by selecting a 'pivot' element and partitioning
21 the other elements into two sub-arrays according to whether they are less than or greater
22 than the pivot. The sub-arrays are then sorted recursively.
23 """
24 def quick_sort(arr):
25     # Base case: if the array has one or zero elements, it's already sorted
26     if len(arr) <= 1:
27         return arr
28     # Choose the pivot element (here we choose the last element)
29     pivot = arr[-1]
30     # Partition the array into two sub-arrays: one less than the pivot and one greater than the pivot
31     less_than_pivot = [x for x in arr[:-1] if x < pivot]
32     greater_than_pivot = [x for x in arr[:-1] if x > pivot]
33     # Recursively sort both sub-arrays and combine with the pivot
34     return quick_sort(less_than_pivot) + [pivot] + quick_sort(greater_than_pivot)
35
36 # Sample usage
37 if __name__ == '__main__':
38     sample_array = [64, 34, 25, 12, 22, 11, 90]
39     print("Original Array:", sample_array)
40
41     # Using Bubble Sort
42     sorted_array_bubble = bubble_sort(sample_array.copy())
43     print("Sorted array using Bubble Sort:", sorted_array_bubble)
44
45     # Using Quick Sort
46     sorted_array_quick = quick_sort(sample_array.copy())
47     print("Sorted array using Quick Sort:", sorted_array_quick)

```

OUTPUT:

```
ta/OneDrive/Desktop/AIAC/palindrome.py
Original array: [64, 34, 25, 12, 22, 11, 90]
Sorted array using Bubble Sort: [11, 12, 22, 25, 34, 64, 90]
Sorted array using Quick Sort: [11, 12, 22, 25, 34, 64, 90]
```

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.

PROMPT:

#Generate a recommendation system that also provides reasons for each suggestion." **CODE:**

```
1 # Generate a recommendation system that also provides reasons for each suggestion.
2 import random
3
4 class RecommendationSystem:
5     def __init__(self, items):
6         self.items = items
7
8     def recommend(self, user_preferences, num_recommendations=5):
9         recommendations = []
10        for _ in range(num_recommendations):
11            item = random.choice(self.items)
12            reason = self.generate_reason(item, user_preferences)
13            recommendations.append((item, reason))
14        return recommendations
15
16    def generate_reason(self, item, user_preferences):
17        reasons = []
18        # This item matches your interest in {user_preferences}.
19        # Based on your previous choices, you might like this item.
20        # This item is popular among users with similar preferences.
21        # You have shown interest in similar items before.
22        # This item has high ratings in categories you prefer.
23        return random.choice(reasons)
24
25 # Example usage
26 if __name__ == "__main__":
27     items = [
28         "Wireless Headphones",
29         "Smart Watch",
30         "T-Sport Tracker",
31         "Fitness Tracker",
32         "Bluetooth Speaker",
33         "Mini Portable Charger",
34         "4K Action Camera",
35         "Portable Charger",
36         "Smart Home Hub",
37         "VR Headset"
38     ]
39     user_preferences = "technology and gadgets"
40     recommender = RecommendationSystem(items)
41     recommendations = recommender.recommend(user_preferences)
42
43     for item, reason in recommendations:
44         print(f'Recommended Item: {item} Reason: {reason}\n')
```

OUTPUT:

Reason: This item matches your interest in technology and gadgets.

Recommended Item: Smart Home Hub

Reason: This item matches your interest in technology and gadgets.

Recommended Item: Noise Cancelling Earbuds

Reason: Based on your previous choices, you might like this item.

Recommended Item: Noise Cancelling Earbuds

Reason: Based on your previous choices, you might like this item.

powershe
Python
Python
Python
Python
Python
Python