AI ASSISTED CODING

LAB-5.1

Katta Lasya

2303A51724

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."

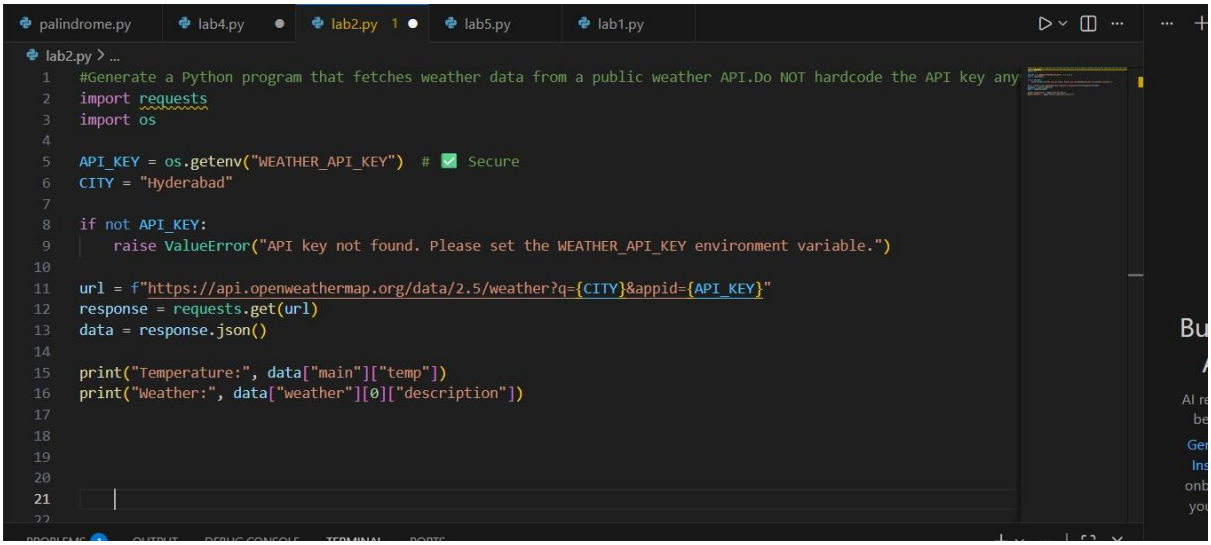
Expected Output:

- Original AI code (check if keys are hardcoded).
- Secure version using enviro**nment 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:



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.

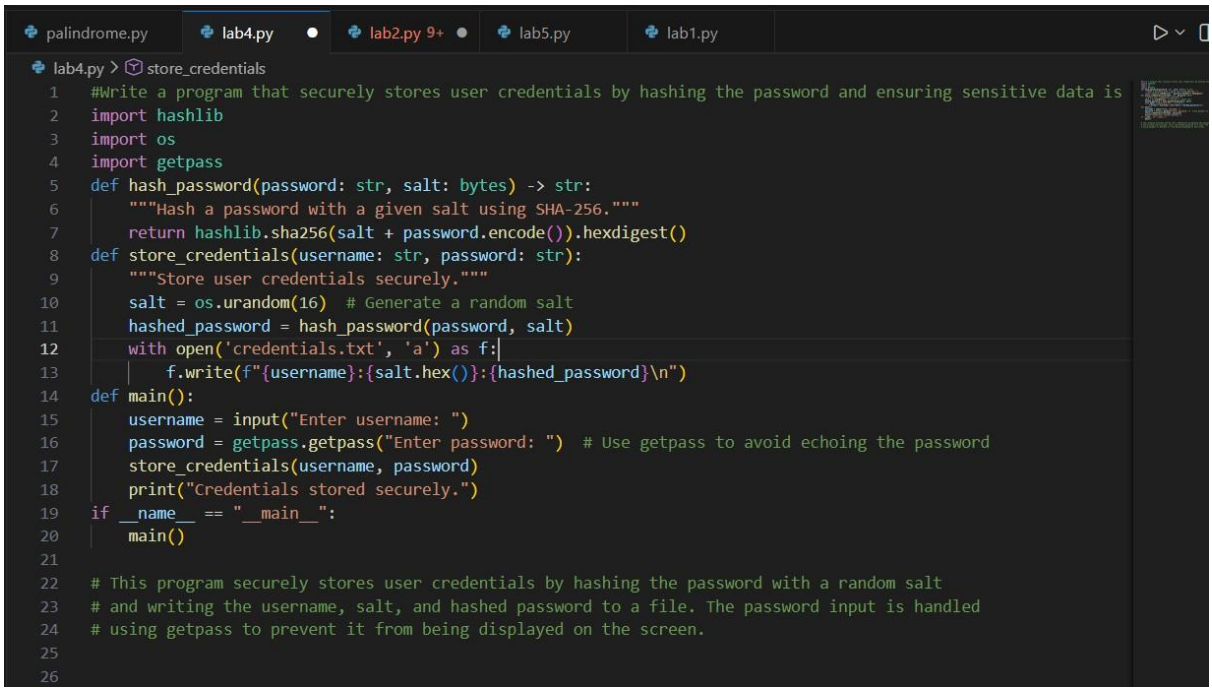
Expected Output:

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

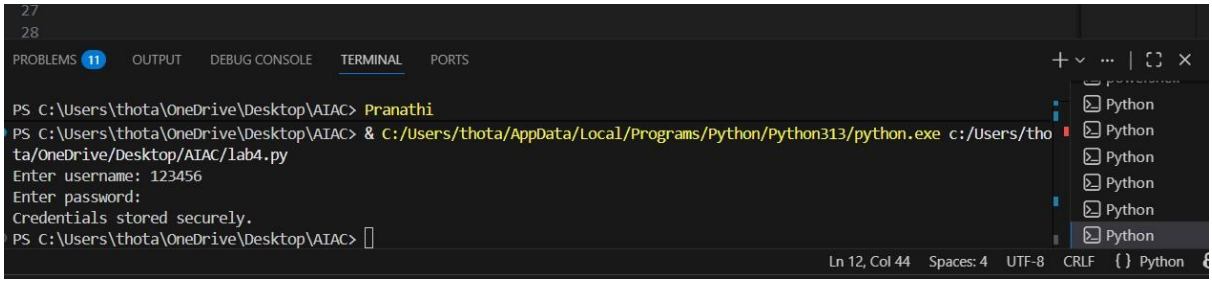
PROMPT:

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

CODE:



OUTPUT:



Task Description #3 (Transparency in Algorithm Design) Objective: Use AI to generate an Armstrong number checking function with comments and explanations.

Instructions:

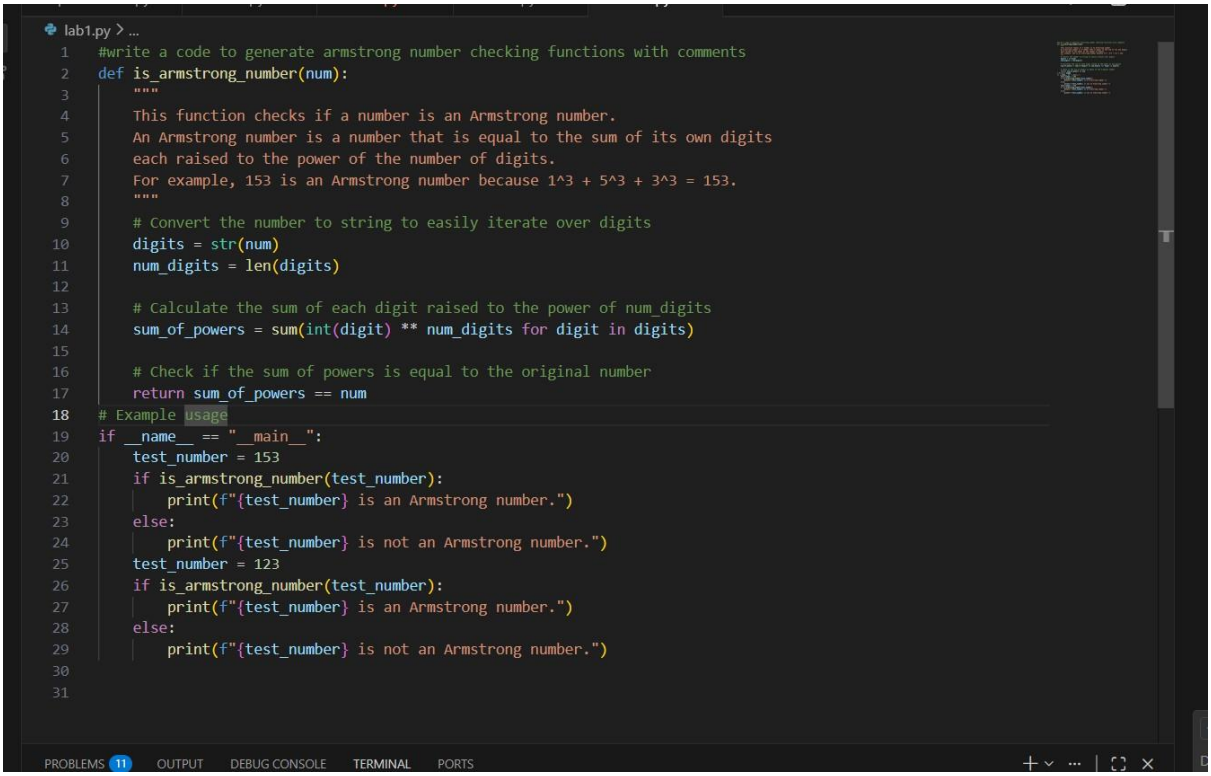
1. Ask AI to explain the code line-by-line.

2. Compare the explanation with code functionality.

Expected Output:

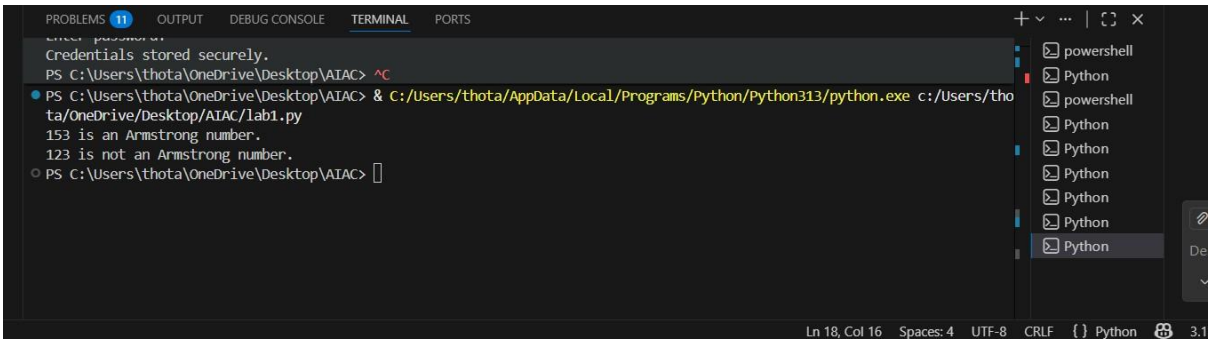
- Transparent, commented code.
- Correct, easy-to-understand explanation **PROMPT:**

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



```python
#write a code to generate armstrong number checking functions with comments
def is_armstrong_number(num):
    """
    This function checks if a number is an Armstrong number.
    An Armstrong number is a number that is equal to the sum of its own digits
    each raised to the power of the number of digits.
    For example, 153 is an Armstrong number because 1^3 + 5^3 + 3^3 = 153.
    """
    # Convert the number to string to easily iterate over digits
    digits = str(num)
    num_digits = len(digits)

    # Calculate the sum of each digit raised to the power of num_digits
    sum_of_powers = sum(int(digit) ** num_digits for digit in digits)

    # Check if the sum of powers is equal to the original number
    return sum_of_powers == num
# Example usage
if __name__ == "__main__":
    test_number = 153
    if is_armstrong_number(test_number):
        print(f"{test_number} is an Armstrong number.")
    else:
        print(f"{test_number} is not an Armstrong number.")
    test_number = 123
    if is_armstrong_number(test_number):
        print(f"{test_number} is an Armstrong number.")
    else:
        print(f"{test_number} is not an Armstrong number.")
```

OUTPUT:



```
Credentials stored securely.
PS C:\Users\thota\OneDrive\Desktop\AIAC> ^C
PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python/Python313/python.exe c:/Users/thota/OneDrive/Desktop/AIAC/lab1.py
153 is an Armstrong number.
123 is not an Armstrong number.
PS C:\Users\thota\OneDrive\Desktop\AIAC>
```

Task Description #4 (Transparency in Algorithm Comparison)

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

Prompt:

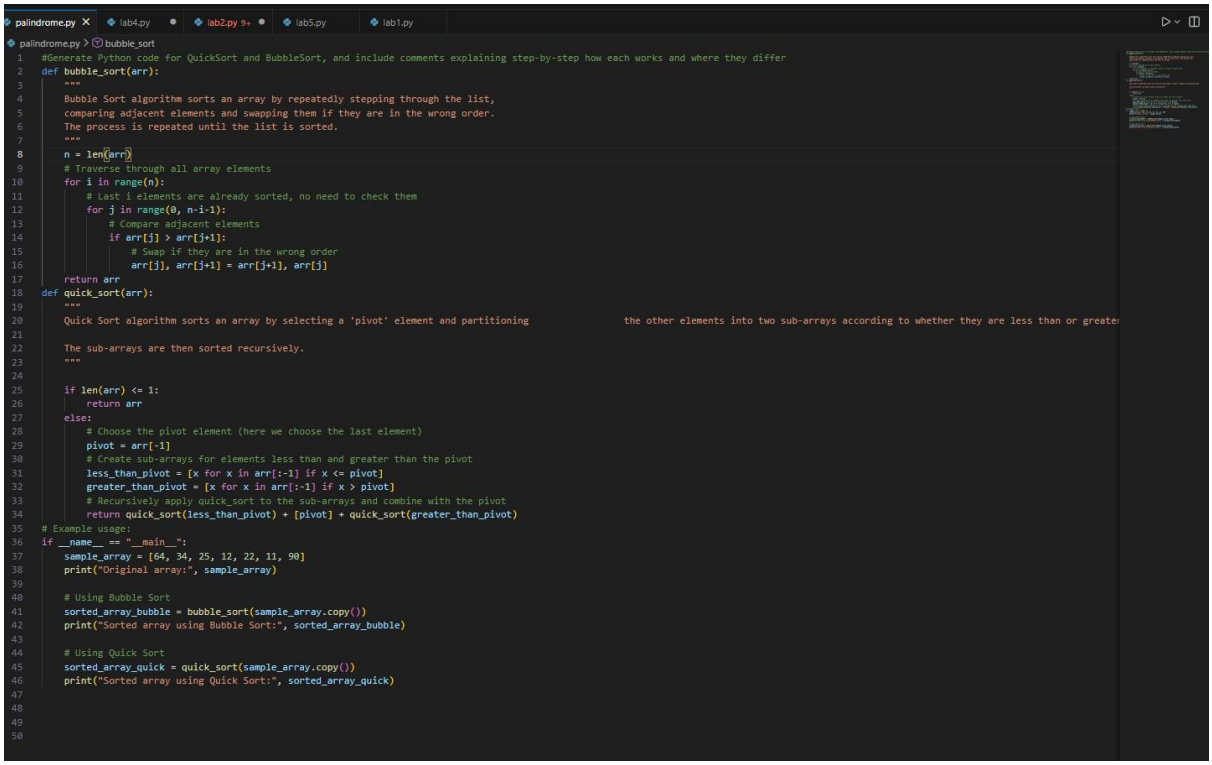
"Generate Python code for QuickSort and BubbleSort, and include comments explaining step-by-step how each works and where they differ." Expected Output:

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

PROMPT:

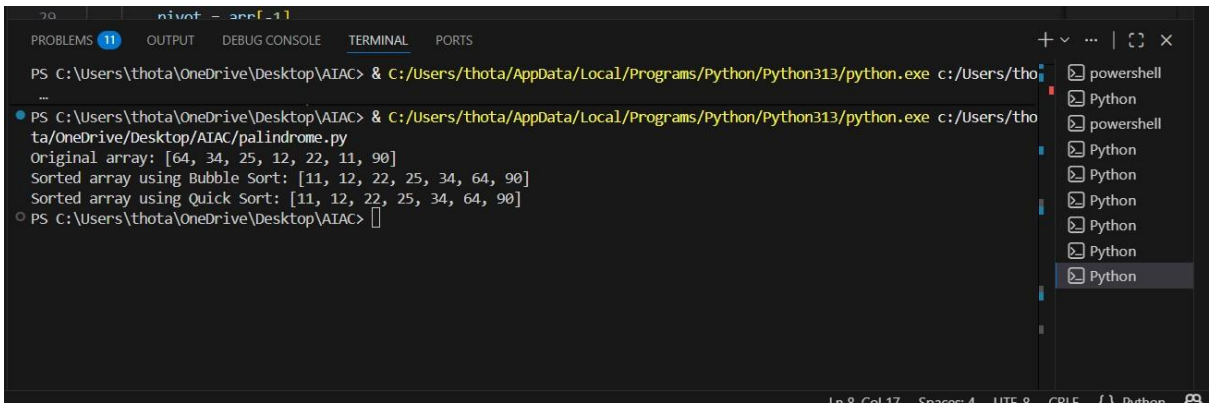
#Generate Python code for QuickSort and BubbleSort, and include comments explaining step-by-step how each works and where they differ.

CODE:



```python
#Generate Python code for QuickSort and BubbleSort, and include comments explaining step-by-step how each works and where they differ
def bubble_sort(arr):
    """
    Bubble Sort algorithm sorts an array by repeatedly stepping through the list,
    comparing adjacent elements and swapping them if they are in the wrong order.
    The process is repeated until the list is sorted.
    """
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):
        # Last i elements are already sorted, no need to check them
        for j in range(0, n-i-1):
            # Compare adjacent elements
            if arr[j] > arr[j+1]:
                # Swap if they are in the wrong order
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
def quick_sort(arr):
    """
    Quick Sort algorithm sorts an array by selecting a 'pivot' element and partitioning               the other elements into two sub-arrays according to whether they are less than or greate

    The sub-arrays are then sorted recursively.
    """

    if len(arr) <= 1:
        return arr
    else:
        # Choose the pivot element (here we choose the last element)
        pivot = arr[-1]
        # Create sub-arrays for elements less than and greater than the pivot
        less_than_pivot = [x for x in arr[:-1] if x <= pivot]
        greater_than_pivot = [x for x in arr[:-1] if x > pivot]
        # Recursively apply quick_sort to the sub-arrays and combine with the pivot
        return quick_sort(less_than_pivot) + [pivot] + quick_sort(greater_than_pivot)
# Example usage:
if __name__ == "__main__":
    sample_array = [64, 34, 25, 12, 22, 11, 90]
    print("Original array:", sample_array)

    # Using Bubble Sort
    sorted_array_bubble = bubble_sort(sample_array.copy())
    print("Sorted array using Bubble Sort:", sorted_array_bubble)

    # Using Quick Sort
    sorted_array_quick = quick_sort(sample_array.copy())
    print("Sorted array using Quick Sort:", sorted_array_quick)
```

OUTPUT:



```
PROBLEMS 11  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python/Python313/python.exe c:/Users/tho
...
PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python/Python313/python.exe c:/Users/tho
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]
PS C:\Users\thota\OneDrive\Desktop\AIAC>
```

Task Description #5 (Transparency in AI Recommendations)

Task: Use AI to create a product recommendation system.

Prompt:

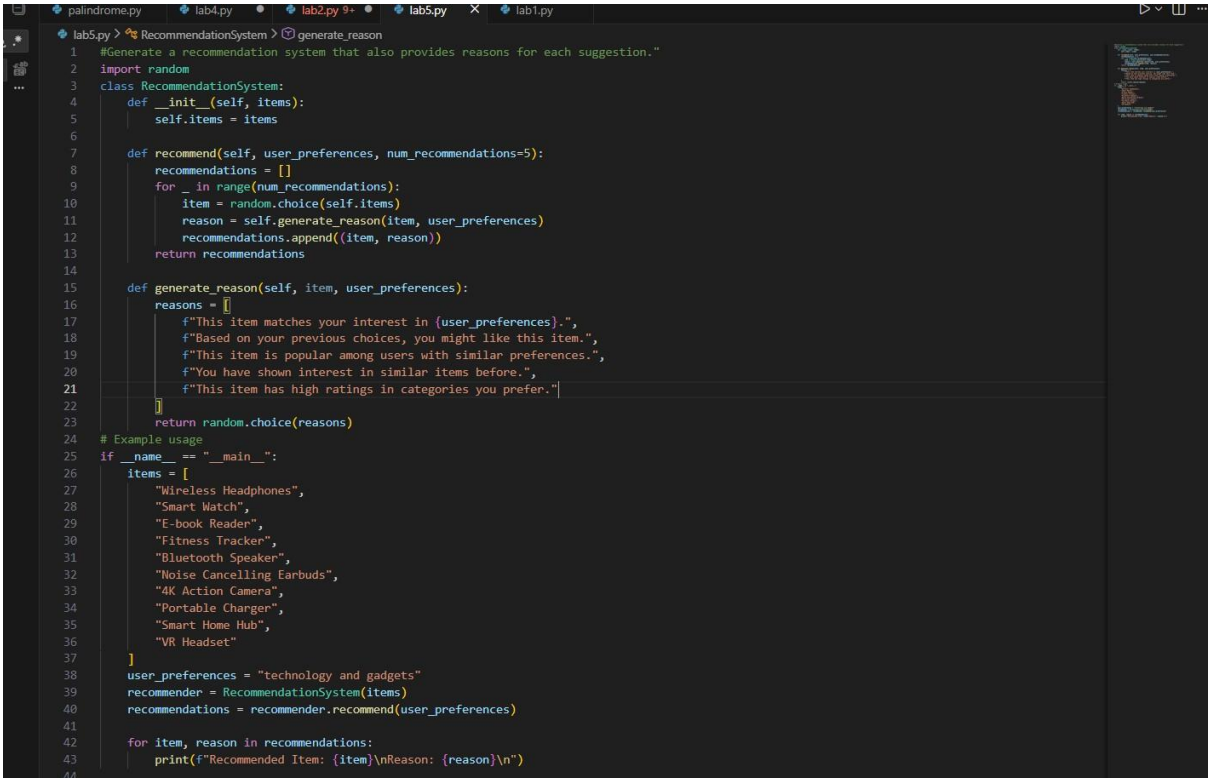
"Generate a recommendation system that also provides reasons for each suggestion."

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:**



OUTPUT: