School of Computer Science and Artificial Intelligence

Lab Assignment-5.1

Course Title : **AI Assistant Coding**
Name of Student : **Shaik Naved Ahmed**
Enrollment No. : **2303A54053**
Batch No. : **47-B**

Task Description #1 (Privacy in API Usage)

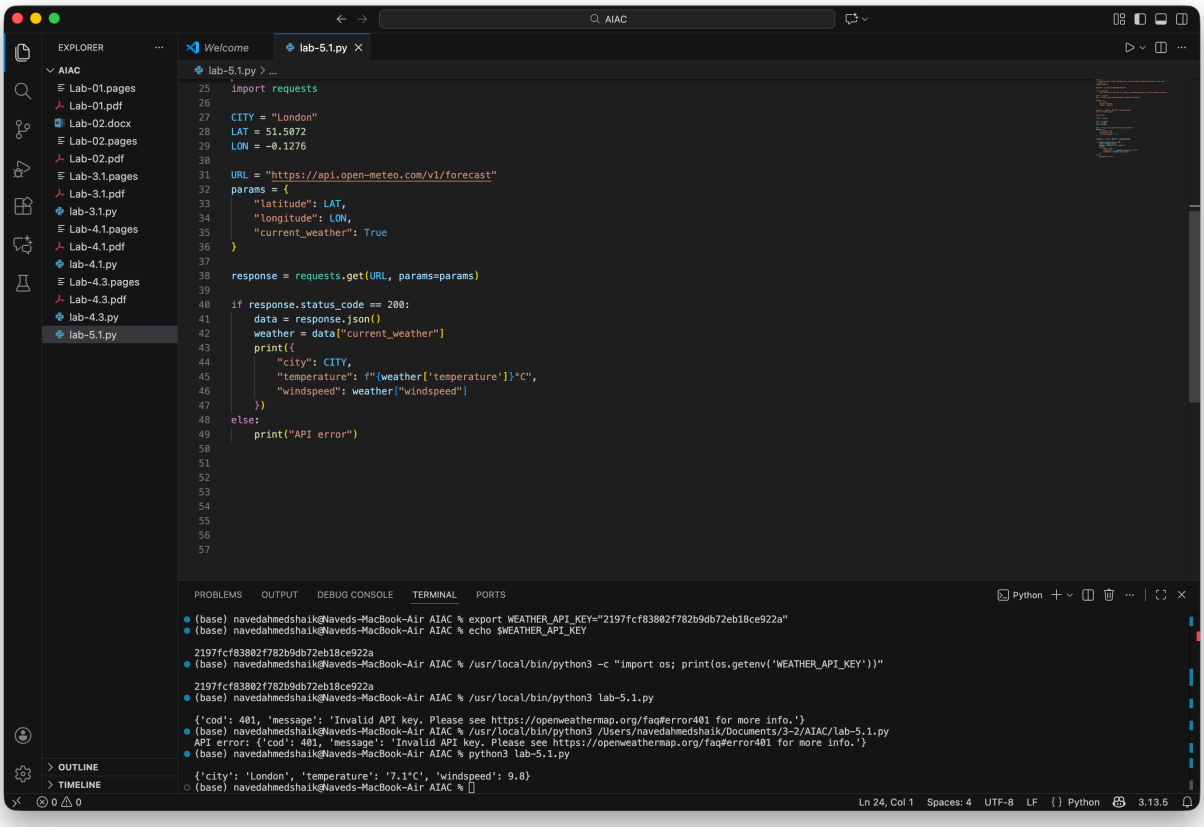
Task: Use an AI tool to generate a Python program that connects to a weather API.

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.



Explanation:

The program fetches current weather data for a specified city using the OpenWeatherMap API. It first reads the API key from an environment variable for security. If the key is missing, it raises an error. It then sends an HTTP GET request with the city name, API key, and metric units, converts the response to JSON format, and prints the weather data.

Input & Output:

Input	Output	Reason
CITY = "London" + valid API key	JSON weather data	API request succeeds and returns current weather information for London.
CITY = "London" + missing API key	Error raised	API key is not found in environment variables.
Invalid CITY name	Error message in JSON	API cannot find weather data for the given city.

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.

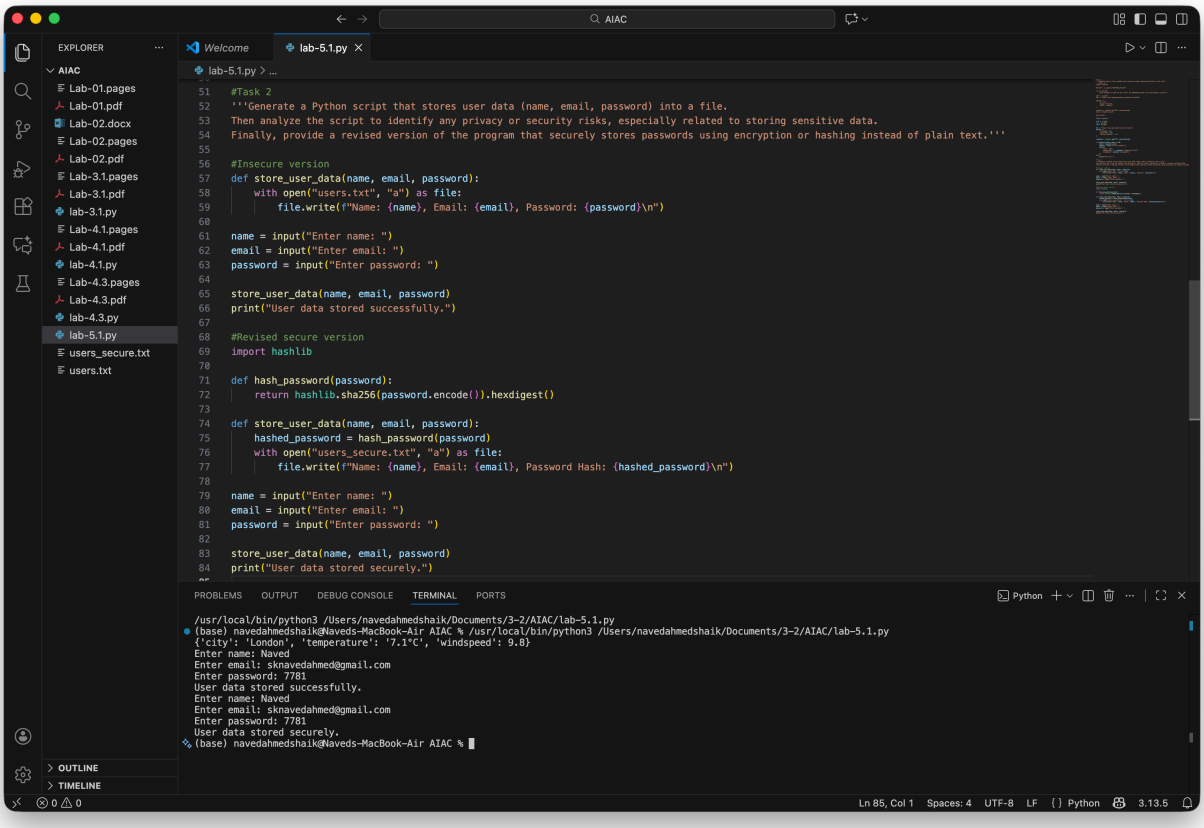
Analyze: Check if the AI stores sensitive data in plain text or without encryption.

Expected Output:

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

Prompt:

"Generate a Python script that stores user data (name, email, password) into a file.
Then analyze the script to identify any privacy or security risks, especially related to storing sensitive data.
Finally, provide a revised version of the program that securely stores passwords using encryption or hashing instead of plain text."



Explanation:

The first (insecure) script collects user details and stores them directly in a text file, including the password in plain text. This makes the data easy to read and misuse if the file is accessed by an unauthorized person. The revised version improves security by hashing the password using the SHA-256 algorithm before storing it. Instead of saving the actual password, only its hash is stored, which helps protect user credentials even if the file is exposed.

Privacy & Security Risks in the Insecure Version:

- Passwords are stored in **plain text**, making them readable to anyone with file access.
- If the file is leaked or stolen, user credentials can be easily misused.
- No protection against insider threats or accidental data exposure.

Improvement in the Secure Version:

- Passwords are **hashed** using SHA-256 before storage.
- Original passwords cannot be easily retrieved from the hash.
- Reduces risk in case of file compromise.

Input & Output:

Version	Input	Output File	Reason
Insecure	Name, Email, Password	users.txt	Stores sensitive data directly, including plain-text password.
Secure (Revised)	Name, Email, Password	users_secure.txt	Stores password as a hash, improving data security.

Task Description #3 (Transparency in Algorithm Design)

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

Instructions:

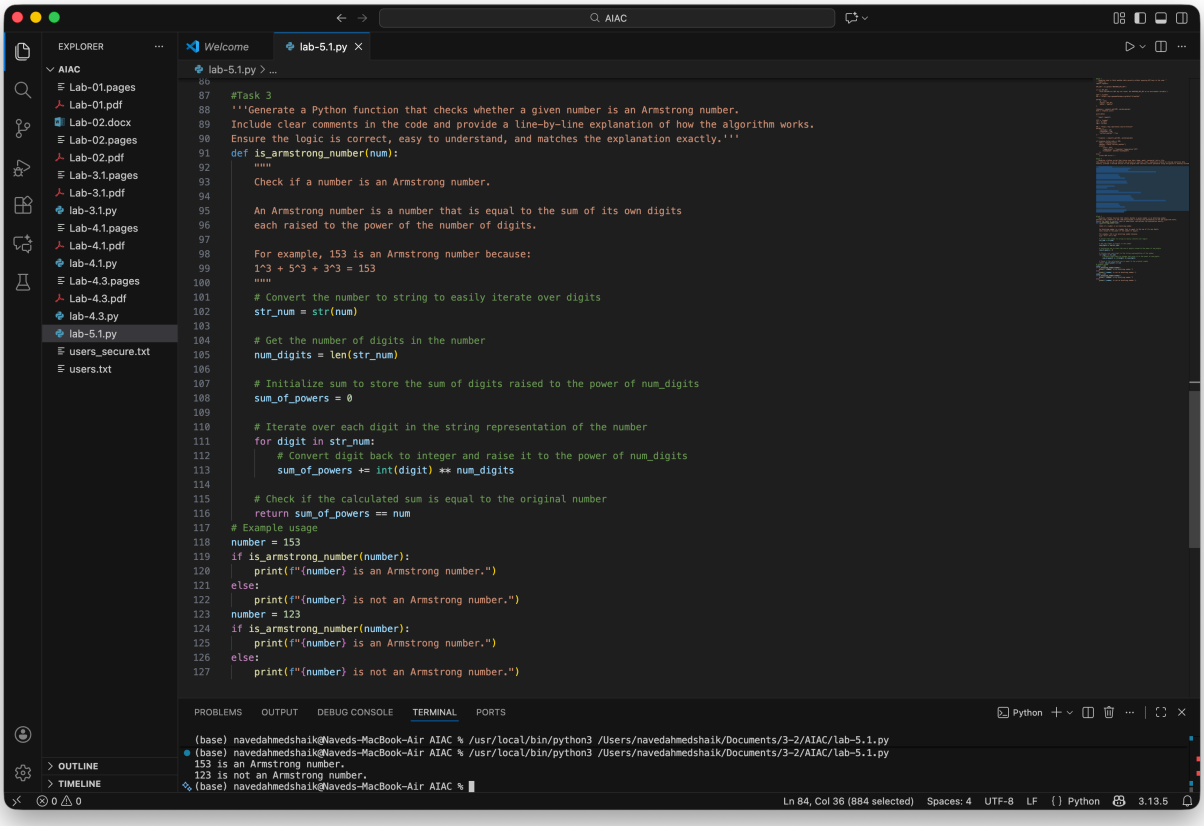
1. Ask AI to explain the code line-by-line.
2. Compare the explanation with code functionality.

Expected Output:

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

Prompt:

Generate a Python function that checks whether a given number is an Armstrong number.
Include clear comments in the code and provide a line-by-line explanation of how the algorithm works.
Ensure the logic is correct, easy to understand, and matches the explanation exactly.



Explanation:

- The function is_armstrong_number(num) checks whether a given number is an Armstrong number.
- The number is first converted into a string so that each digit can be accessed easily.
- The total number of digits is calculated using len(), which is required for the Armstrong condition.
- A variable sum_of_powers is initialized to store the sum of each digit raised to the power of the total number of digits.
- The function loops through each digit in the string, converts it back to an integer, raises it to the required power, and adds it to sum_of_powers.
- Finally, the function compares sum_of_powers with the original number and returns True if they are equal; otherwise, it returns False.

Input & Output:

Input	Output	Reason
153	153 is an Armstrong number.	(1^3 + 5^3 + 3^3 = 153), equal to the original number.
123	123 is not an Armstrong number.	(1^3 + 2^3 + 3^3 = 36), not equal to 123.

Task Description #4 (Transparency in Algorithm Comparison)

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

Prompt:

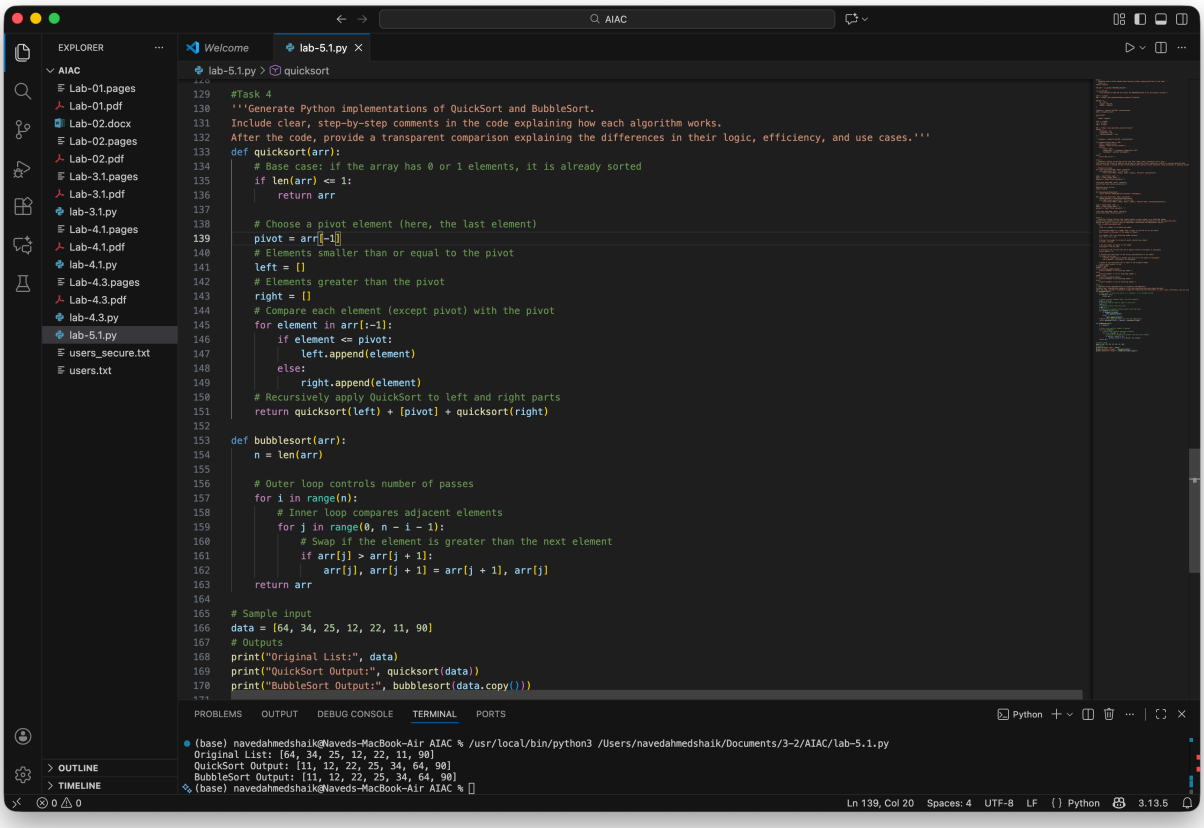
"Generate Python code for QuickSort and BubbleSort, and include comments explaining step-by-step how each works and where they differ."

Expected Output:

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

Prompt:

Generate Python implementations of QuickSort and BubbleSort.
Include clear, step-by-step comments in the code explaining how each algorithm works.
After the code, provide a transparent comparison explaining the differences in their logic, efficiency, and use cases.



Explanation:

QuickSort works using a divide-and-conquer strategy. It begins by selecting a pivot element from the list and then partitions the remaining elements into two sublists: one containing elements smaller than or equal to the pivot and the other containing elements greater than the pivot. These sublists are then sorted recursively using the same approach. Finally, the sorted left sublist, pivot, and sorted right sublist are combined to produce the fully sorted array. This method significantly reduces the number of comparisons for large datasets, making QuickSort efficient in most practical scenarios.

BubbleSort, on the other hand, follows a much simpler approach based on repeated comparisons of adjacent elements. In each pass through the list, it compares neighboring elements and swaps them

if they are in the wrong order. This process is repeated multiple times until no more swaps are needed, indicating that the list is sorted. While BubbleSort is easy to understand and implement, it performs unnecessary comparisons and swaps, which makes it inefficient and slow for large datasets.

Comparison:

Aspect	QuickSort	BubbleSort
Algorithm Type	Divide and Conquer	Comparison-based
Time Complexity (Average)	O(n log n)	$O(n^2)$
Time Complexity (Worst)	$O(n^2)$	$O(n^2)$
Space Complexity	O(log n)	O(1)
Efficiency	High for large datasets	Low for large datasets
Use Cases	Large datasets, performance-critical tasks	Small datasets, educational purposes

Input & Output:

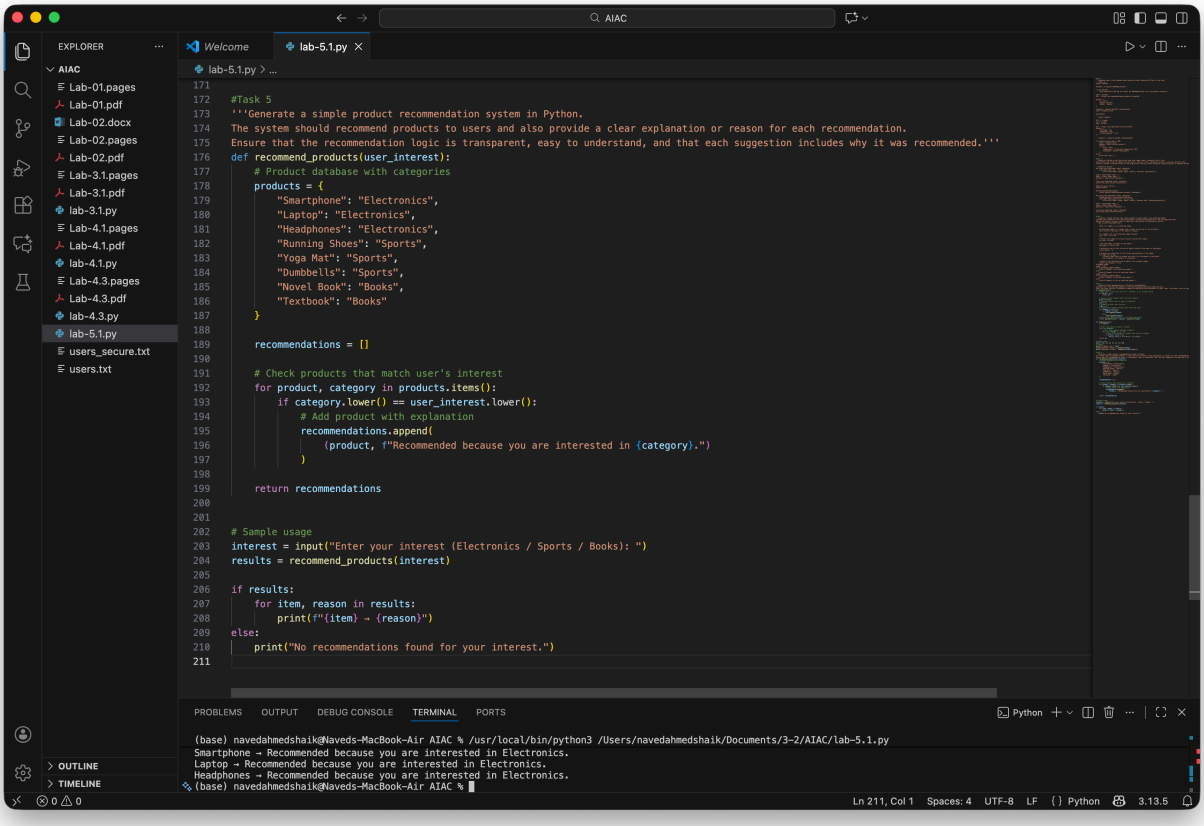
Input List	Algorithm	Output List	Reason
[64, 34, 25, 12, 22, 11, 90]	QuickSort	[11, 12, 22, 25, 34, 64, 90]	List is recursively partitioned around pivots and sorted.
[64, 34, 25, 12, 22, 11, 90]	Bubble Sort	[11, 12, 22, 25, 34, 64, 90]	Adjacent elements are repeatedly swapped until the list is sorted.

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 simple product recommendation system in Python.
The system should recommend products to users and also provide a clear explanation or reason for each recommendation.
Ensure that the recommendation logic is transparent, easy to understand, and that each suggestion includes why it was recommended.



Explanation:

The function recommends products based on the user's area of interest. A predefined product database maps each product to a category. The user's input is compared with these categories in a case-insensitive manner. If a product's category matches the user's interest, it is added to the recommendation list along with a clear reason explaining why that product was suggested. If no category matches, the system informs the user that no recommendations are available.

Input & Output:

User Interest	Recommended Product	Reason
Electronics	Smartphone	Recommended because you are interested in Electronics.
Electronics	Laptop	Recommended because you are interested in Electronics.
Electronics	Headphones	Recommended because you are interested in Electronics.
Sports	Running Shoes	Recommended because you are interested in Sports.
Sports	Yoga Mat	Recommended because you are interested in Sports.
Books	Novel Book	Recommended because you are interested in Books.
Books	Textbook	Recommended because you are interested in Books.