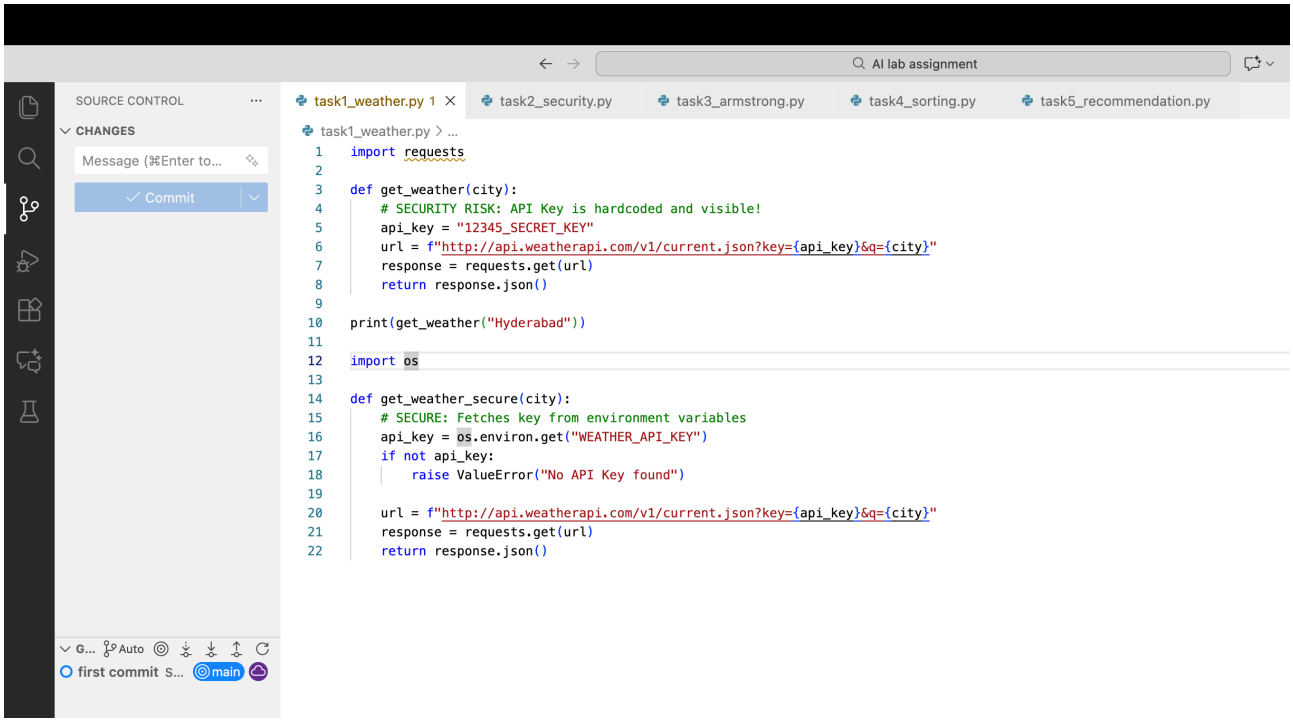
Student Details

- **Name:** Telu Sai Suraj
- **Roll Number:** [2403A54L01]
- **Branch & Year:** B.Tech, 3rd Year
- **College:** SR University
- **Assignment Number:** Lab Assignment 5.1

Task 01: Privacy in API Usage

Prompt Given to AI: "Generate code to fetch weather data securely without exposing API keys in the code."

Code Screenshot:



Explanation of Code: The first part of the code demonstrates an insecure method where the API key is hardcoded, making it visible to anyone who sees the script. The second function, `get_weather_secure`, fixes this vulnerability. Instead of pasting the key directly, it imports the `os` module to fetch the 'WEATHER_API_KEY' from the computer's environment variables. This ensures that sensitive credentials are never exposed in the source code itself, preventing unauthorised access.

Task 02: Privacy & Security in File Handling

Prompt Given to AI: "Generate a Python script that stores user data (name, email, password) in a file. Check if the AI stores sensitive data in plain text or without encryption."

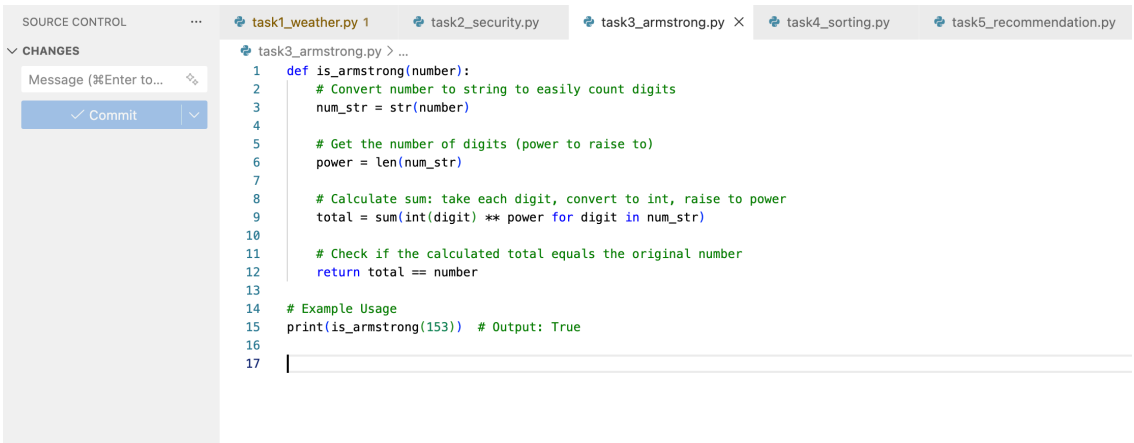
Code Screenshot:

Explanation of Code: The initial script saves user details, including passwords, directly into a text file in plain text. This is a major security risk because anyone with file access can read the credentials. The secure version imports the `hashlib`library. Before saving the password, it converts it into a SHA-256 hash. Hashing is a one-way process, meaning even if a hacker steals the file, they cannot reverse the hash to find the original password, significantly improving security.

Task 03: Transparency in Algorithm Design

Prompt Given to AI: "Generate an Armstrong number checking function with comments and explanations. Explain the code line-by-line."

Code Screenshot:



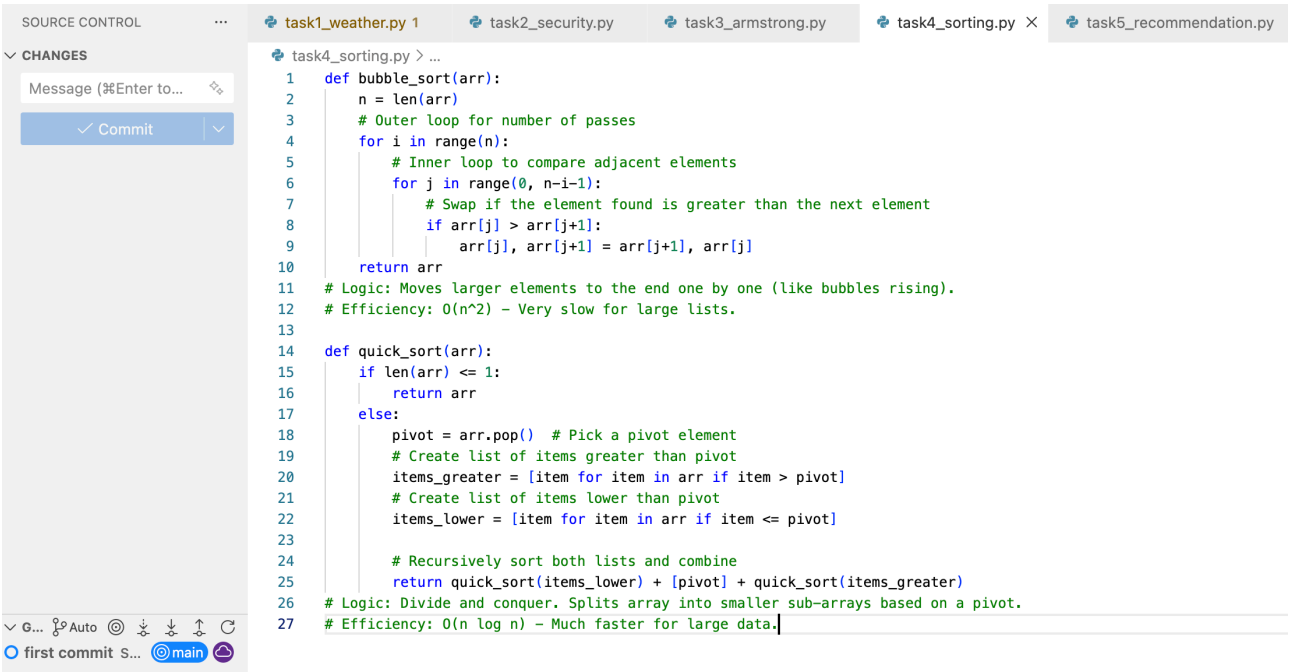
Explanation of Code: This function determines if a number is an Armstrong number (where the sum of its digits raised to the power of the number of digits equals the original number).

1. First, it converts the integer to a string to easily count the number of digits.
2. It calculates the power (length of the number).
3. It uses a loop (or list comprehension) to iterate through each digit, converting it back to an integer and raising it to the calculated power.
4. Finally, it sums these values and compares the result to the original input. If they match, it returns `True`.

Task 04: Transparency in Algorithm Comparison

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

Code Screenshot:



Explanation of Code: The code implements two different sorting algorithms to demonstrate efficiency differences.

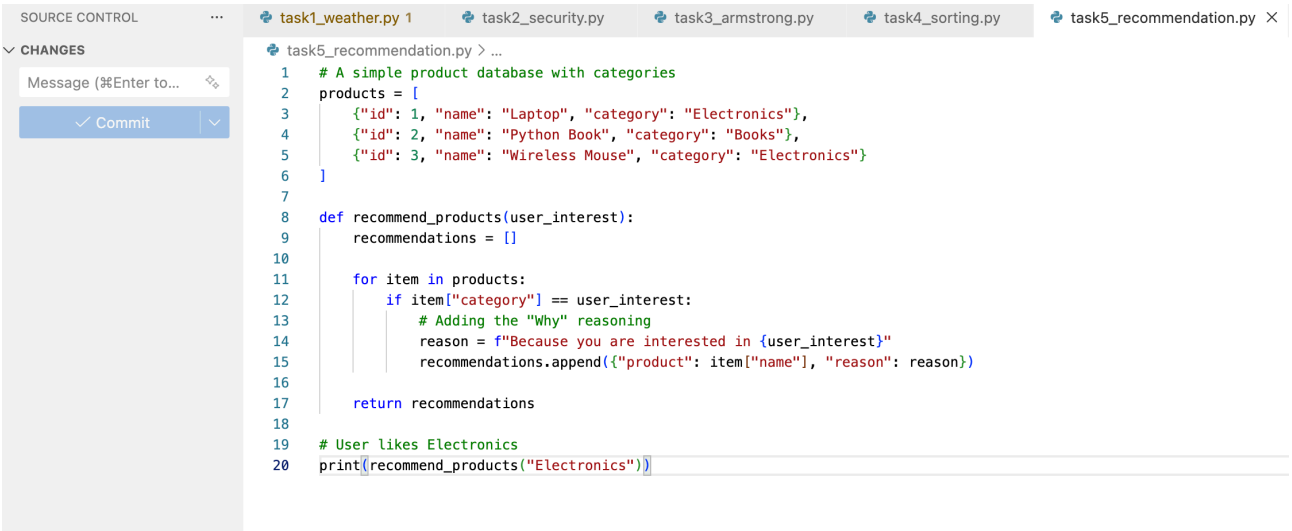
- **Bubble Sort:** Uses nested loops to repeatedly swap adjacent elements if they are in the wrong order. It is simple to understand but inefficient (O(n2)) for large datasets.

- **Quick Sort:** Uses a "divide and conquer" strategy. It selects a 'pivot' element and partitions the array into elements less than and greater than the pivot. It recursively sorts these sub-arrays. This approach is much faster (O(nlogn)) and suitable for real-world data processing.

Task 05: Transparency in AI Recommendation

Prompt Given to AI: "Generate a recommendation system that also provides reasons for each suggestion."

Code Screenshot:



Explanation of Code: This recommendation system prioritizes "explainability." Instead of a "black box" algorithm that gives a result without context, this simple rule-based system iterates through a product database. When it finds a product matching the user's category (e.g., Electronics), it explicitly constructs a "reason" string (e.g., "Because you are interested in Electronics"). This ensures the user understands *why* a specific item was suggested, which builds trust in the AI system.

Thank You.