

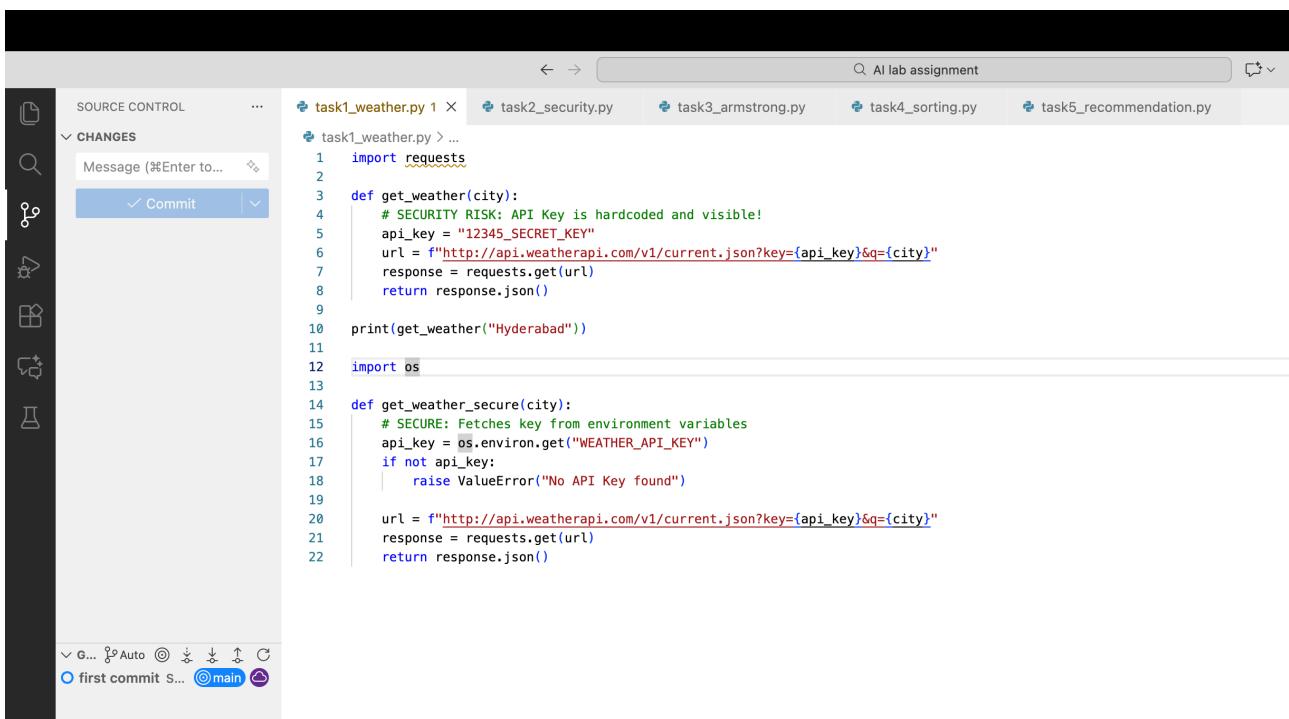
Student Details

- **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:



The screenshot shows a GitHub commit interface. The commit message is "Message (⌘Enter to...)" and the status is "✓ Commit". The code in the file "task1_weather.py" is as follows:

```
import requests
def get_weather(city):
    # SECURITY RISK: API Key is hardcoded and visible!
    api_key = "12345_SECRET_KEY"
    url = f"http://api.weatherapi.com/v1/current.json?key={api_key}&q={city}"
    response = requests.get(url)
    return response.json()

print(get_weather("Hyderabad"))

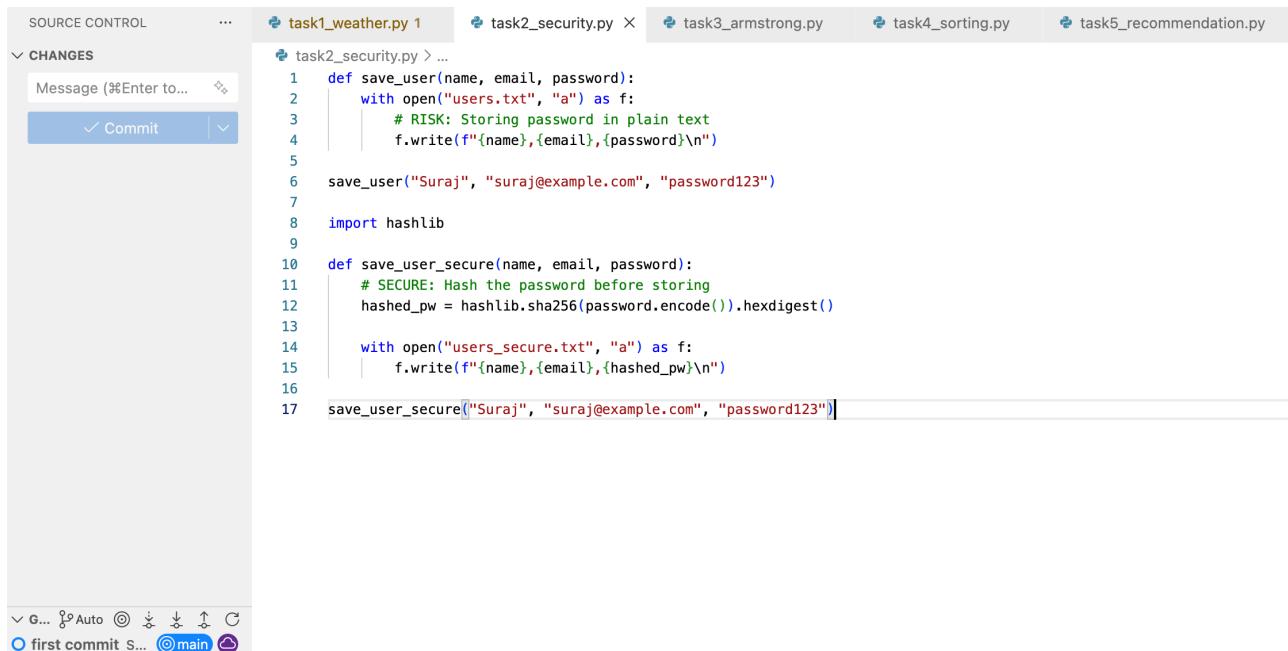
import os
def get_weather_secure(city):
    # SECURE: Fetches key from environment variables
    api_key = os.environ.get("WEATHER_API_KEY")
    if not api_key:
        raise ValueError("No API Key found")
    url = f"http://api.weatherapi.com/v1/current.json?key={api_key}&q={city}"
    response = requests.get(url)
    return response.json()
```

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:



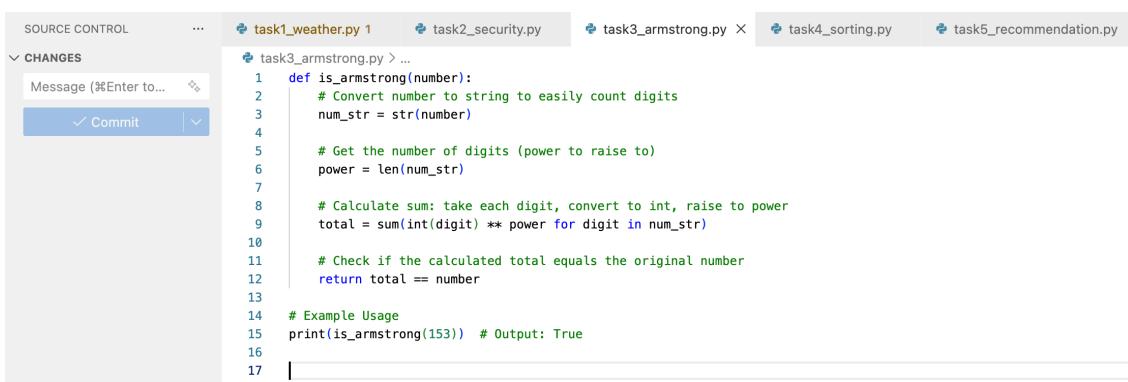
```
task2_security.py
1 def save_user(name, email, password):
2     with open("users.txt", "a") as f:
3         # RISK: Storing password in plain text
4         f.write(f"{name},{email},{password}\n")
5
6 save_user("Suraj", "suraj@example.com", "password123")
7
8 import hashlib
9
10 def save_user_secure(name, email, password):
11     # SECURE: Hash the password before storing
12     hashed_pw = hashlib.sha256(password.encode()).hexdigest()
13
14     with open("users_secure.txt", "a") as f:
15         f.write(f"{name},{email},{hashed_pw}\n")
16
17 save_user_secure("Suraj", "suraj@example.com", "password123")
```

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:



```
task3_armstrong.py
1 def is_armstrong(number):
2     # Convert number to string to easily count digits
3     num_str = str(number)
4
5     # Get the number of digits (power to raise to)
6     power = len(num_str)
7
8     # Calculate sum: take each digit, convert to int, raise to power
9     total = sum(int(digit) ** power for digit in num_str)
10
11     # Check if the calculated total equals the original number
12     return total == number
13
14 # Example Usage
15 print(is_armstrong(153)) # Output: True
16
17 |
```

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:

The screenshot shows a code editor interface with several tabs at the top: task1_weather.py, task2_security.py, task3_armstrong.py, task4_sorting.py (selected), and task5_recommendation.py. The main pane displays the contents of task4_sorting.py, which contains two sorting functions: bubble_sort and quick_sort. The code is annotated with comments explaining the logic and efficiency of each algorithm.

```
task4_sorting.py
1 def bubble_sort(arr):
2     n = len(arr)
3     # Outer loop for number of passes
4     for i in range(n):
5         # Inner loop to compare adjacent elements
6         for j in range(0, n-i-1):
7             # Swap if the element found is greater than the next element
8             if arr[j] > arr[j+1]:
9                 arr[j], arr[j+1] = arr[j+1], arr[j]
10    return arr
11 # Logic: Moves larger elements to the end one by one (like bubbles rising).
12 # Efficiency: O(n^2) - Very slow for large lists.
13
14 def quick_sort(arr):
15     if len(arr) <= 1:
16         return arr
17     else:
18         pivot = arr.pop() # Pick a pivot element
19         # Create list of items greater than pivot
20         items_greater = [item for item in arr if item > pivot]
21         # Create list of items lower than pivot
22         items_lower = [item for item in arr if item <= pivot]
23
24         # Recursively sort both lists and combine
25         return quick_sort(items_lower) + [pivot] + quick_sort(items_greater)
26 # Logic: Divide and conquer. Splits array into smaller sub-arrays based on a pivot.
27 # Efficiency: O(n log n) - Much faster for large data.
```

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

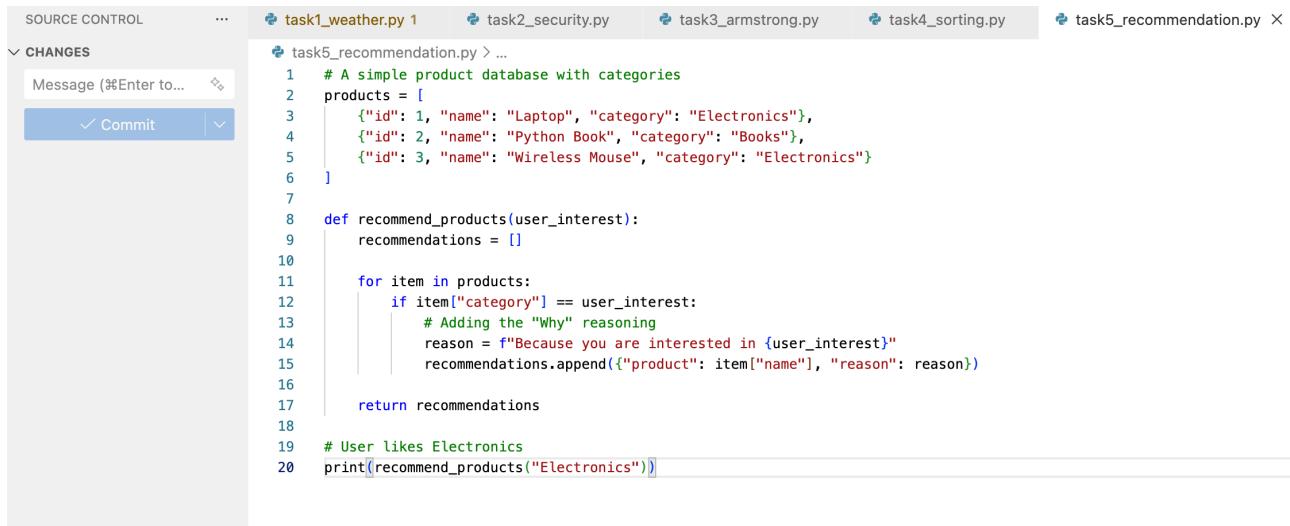
- **Bubble Sort:** Uses nested loops to repeatedly swap adjacent elements if they are in the wrong order. It is simple to understand but inefficient ($O(n^2)$) for large datasets.

- **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 ($O(n\log n)$) 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:



```

SOURCE CONTROL      ...
task1_weather.py 1   task2_security.py   task3_armstrong.py   task4_sorting.py   task5_recommendation.py X
CHANGES
Message (%#Enter to...  ↴
✓ Commit  ↴
task5_recommendation.py > ...
1 # A simple product database with categories
2 products = [
3     {"id": 1, "name": "Laptop", "category": "Electronics"},
4     {"id": 2, "name": "Python Book", "category": "Books"},
5     {"id": 3, "name": "Wireless Mouse", "category": "Electronics"}
6 ]
7
8 def recommend_products(user_interest):
9     recommendations = []
10
11     for item in products:
12         if item["category"] == user_interest:
13             # Adding the "Why" reasoning
14             reason = f"Because you are interested in {user_interest}"
15             recommendations.append({"product": item["name"], "reason": reason})
16
17     return recommendations
18
19 # User likes Electronics
20 print(recommend_products("Electronics"))

```

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.