AI ASSISTED CODING

LAB-5.1

NAME:CH.RAMCHARAN

ENROLL.NO:2403A52069

BATCH-04

TASK-01:

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

PROMPT:

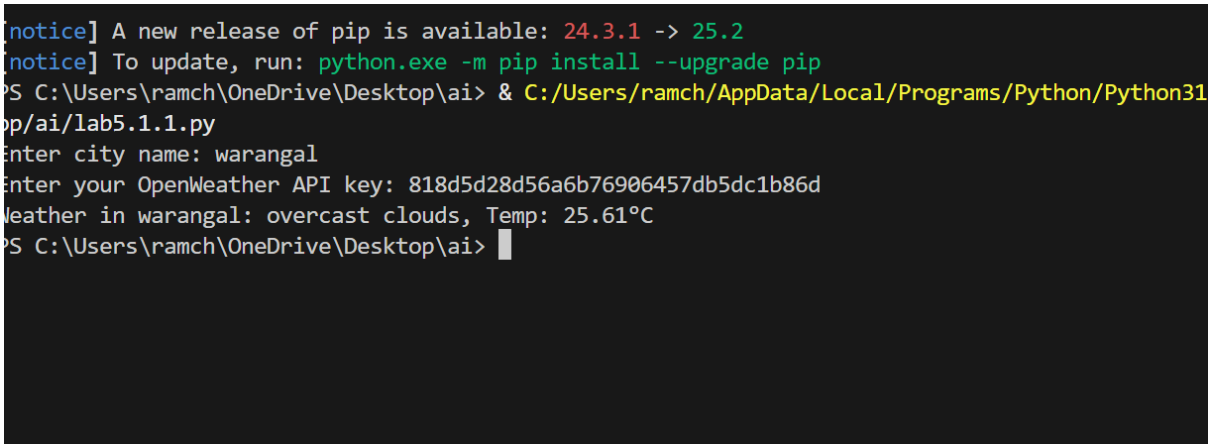
Generate a python program that connects to a weather API and displays the climate of the particular city.

Code:

lab5.1.1.py > ...

```python
import os
import requests

def get_weather(city):
    api_key = os.getenv('WEATHER_API_KEY')
    if not api_key:
        api_key = input("Enter your OpenWeather API key: ").strip()
    url = f'https://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric'
    response = requests.get(url)
    response.raise_for_status()
    return response.json()

if __name__ == "__main__":
    city = input("Enter city name: ")
    try:
        weather = get_weather(city)
        print(f"Weather in {city}: {weather['weather'][0]['description']}, Temp: {weather['main']['temp']}°C")
    except Exception as e:
        print(f"Error: {e}")
```

Output:



Observation:

The code generated by the github copilot is to check the weather of a particular city or village by entering its name and it asks for the API key which is generated in Open WeatherApp.

TASK-02:

Use an AI tool to generate a Python script that stores user data (name, email, password) in a file.

PROMPT:

Write a python program to store the users data in a file.

CODE:

lab5.1.2.py > ...

```python
# Program to store user data

def get_user_data():
    user_data = {}
    user_data['name'] = input("Enter your name: ")
    user_data['age'] = input("Enter your age: ")
    user_data['email'] = input("Enter your email: ")
    return user_data

def main():
    data = get_user_data()
    print("\nStored User Data:")
    for key, value in data.items():
        print(f"{key.capitalize()}: {value}")

if __name__ == "__main__":
    main()
```

OUTPUT:

```
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python313/python.exe c:/Users/ram
ch/OneDrive/Desktop/ai/lab5.1.2.py
Enter your name: ram
Enter your age: 34
Enter your email: ram@gmail.com

Stored User Data:
Name: ram
Age: 34
Email: ram@gmail.com
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

OBSERVATION:

The code takes the input as name , age, email and stores them in a file which is created by it by asking several permissions.

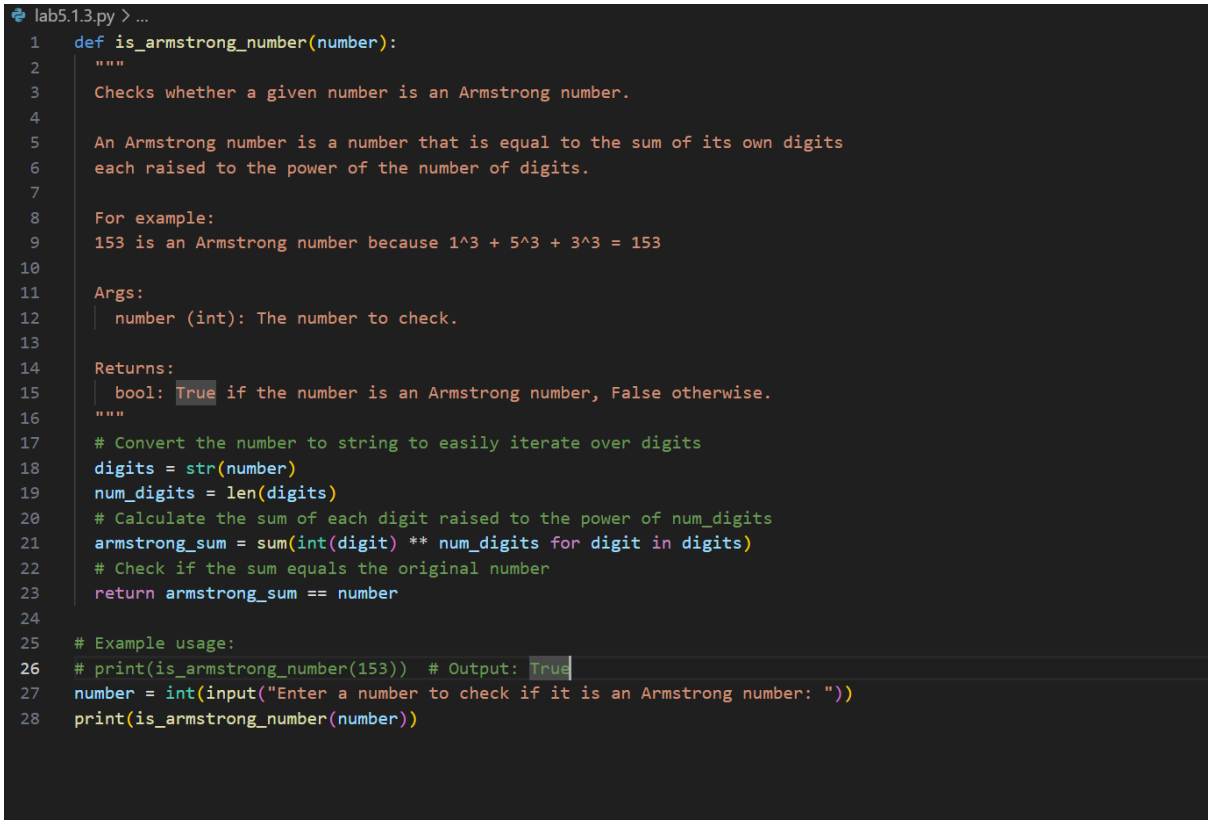
TASK-03:

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

PROMPT:

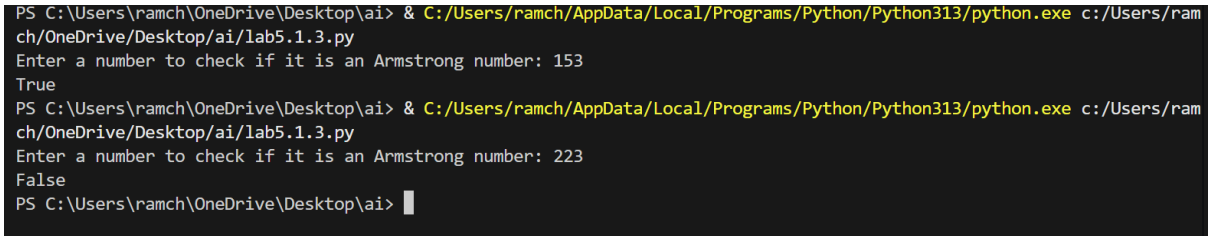
Write a python function to check whether the given number is Armstrong or not also provide clear explanation using comments.

CODE:



```python
def is_armstrong_number(number):
    """
    Checks whether a given 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

    Args:
        number (int): The number to check.

    Returns:
        bool: True if the number is an Armstrong number, False otherwise.
    """
    # Convert the number to string to easily iterate over digits
    digits = str(number)
    num_digits = len(digits)
    # Calculate the sum of each digit raised to the power of num_digits
    armstrong_sum = sum(int(digit) ** num_digits for digit in digits)
    # Check if the sum equals the original number
    return armstrong_sum == number

# Example usage:
# print(is_armstrong_number(153))  # Output: True
number = int(input("Enter a number to check if it is an Armstrong number: "))
print(is_armstrong_number(number))
```

OUTPUT:



```
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python313/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab5.1.3.py
Enter a number to check if it is an Armstrong number: 153
True
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python313/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab5.1.3.py
Enter a number to check if it is an Armstrong number: 223
False
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

OBSERVATIONS:

The code is about the python function which accepts a number as an input and checks whether the given number is Armstrong or not. The copilot gave a clear explanation using comments.

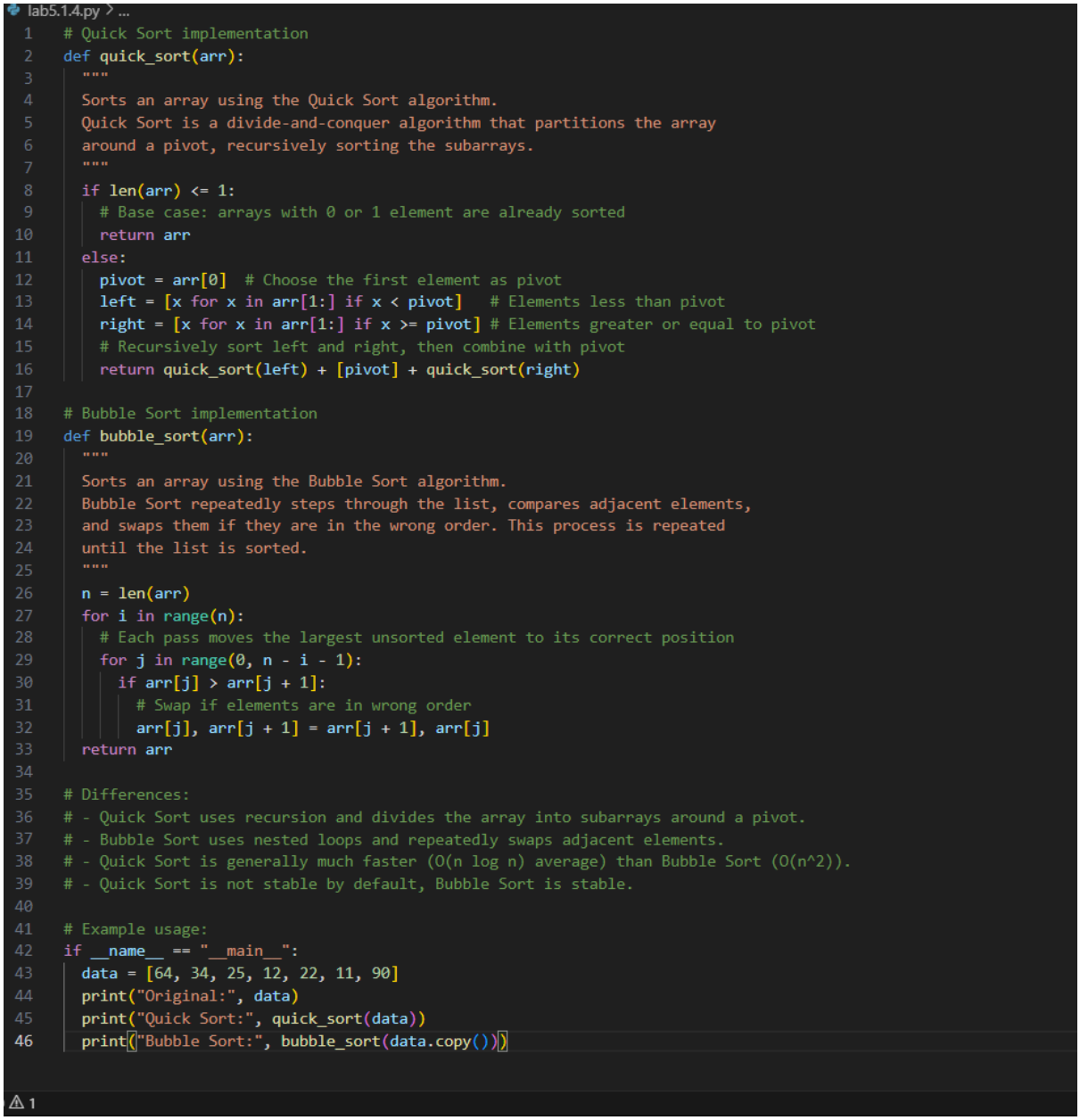
TASK-04:

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.

CODE:



```python
# Quick Sort implementation
def quick_sort(arr):
    """
    Sorts an array using the Quick Sort algorithm.
    Quick Sort is a divide-and-conquer algorithm that partitions the array
    around a pivot, recursively sorting the subarrays.
    """
    if len(arr) <= 1:
        # Base case: arrays with 0 or 1 element are already sorted
        return arr
    else:
        pivot = arr[0]  # Choose the first element as pivot
        left = [x for x in arr[1:] if x < pivot]   # Elements less than pivot
        right = [x for x in arr[1:] if x >= pivot] # Elements greater or equal to pivot
        # Recursively sort left and right, then combine with pivot
        return quick_sort(left) + [pivot] + quick_sort(right)

# Bubble Sort implementation
def bubble_sort(arr):
    """
    Sorts an array using the Bubble Sort algorithm.
    Bubble Sort repeatedly steps through the list, compares adjacent elements,
    and swaps them if they are in the wrong order. This process is repeated
    until the list is sorted.
    """
    n = len(arr)
    for i in range(n):
        # Each pass moves the largest unsorted element to its correct position
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                # Swap if elements are in wrong order
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr

# Differences:
# - Quick Sort uses recursion and divides the array into subarrays around a pivot.
# - Bubble Sort uses nested loops and repeatedly swaps adjacent elements.
# - Quick Sort is generally much faster (O(n log n) average) than Bubble Sort (O(n^2)).
# - Quick Sort is not stable by default, Bubble Sort is stable.

# Example usage:
if __name__ == "__main__":
    data = [64, 34, 25, 12, 22, 11, 90]
    print("Original:", data)
    print("Quick Sort:", quick_sort(data))
    print("Bubble Sort:", bubble_sort(data.copy()))
```

OUTPUT:

OBSERVATION:

The code generated by the copilot is the technique quick sort and bubble sort of the array and the code is explained step-by-step using comments.

TASK-05:

Use AI to create a product recommendation system

PROMPT:

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

CODE:

```python
import random

# Sample data: items and their features
items = {
  "The Matrix": {"genre": "Sci-Fi", "director": "Wachowski", "year": 1999},
  "Inception": {"genre": "Sci-Fi", "director": "Nolan", "year": 2010},
  "Titanic": {"genre": "Romance", "director": "Cameron", "year": 1997},
  "Interstellar": {"genre": "Sci-Fi", "director": "Nolan", "year": 2014},
  "The Notebook": {"genre": "Romance", "director": "Cassavetes", "year": 2004},
}

# User preferences
user_preferences = {
  "genre": "Sci-Fi",
  "director": "Nolan"
}

def recommend(items, preferences):
  recommendations = []
  for item, features in items.items():
    score = 0
    reasons = []
    for key, value in preferences.items():
      if features.get(key) == value:
        score += 1
        reasons.append(f"Matches your preferred {key}: {value}")
    if score > 0:
      recommendations.append((item, score, reasons))
  recommendations.sort(key=lambda x: x[1], reverse=True)
  return recommendations

def main():
  recs = recommend(items, user_preferences)
  if recs:
    print("Recommended for you:")
    for item, score, reasons in recs:
      print(f"- {item} (score: {score})")
      for reason in reasons:
        print(f"  Reason: {reason}")
  else:
    print("No recommendations found based on your preferences.")

if __name__ == "__main__":
  main()
```

OUTPUT:

```
Bubble Sort: [11, 12, 22, 25, 34, 64, 90]
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python313/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab5.1.5.py
Recommended for you:
- Inception (score: 2)
  Reason: Matches your preferred genre: Sci-Fi
  Reason: Matches your preferred director: Nolan
- Interstellar (score: 2)
  Reason: Matches your preferred genre: Sci-Fi
  Reason: Matches your preferred director: Nolan
- The Matrix (score: 1)
  Reason: Matches your preferred genre: Sci-Fi
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

OBSERVATION:

The code is about the product recommendation system and it also explains the reason why the product is recommended for you.

TASK-06:

Ask AI to generate a Python function for calculating factorial using recursion

PROMPT:

Generate a recursive factorial function with comments that explain each line and a final summary of the algorithm's flow.

CODE:

lab5.1.6.py > ...

```python
def factorial(n):
    """
    Recursive function to calculate the factorial of a non-negative integer n.
    The factorial of n (n!) is the product of all positive integers less than or equal to n.
    """
    # Base case: if n is 0 or 1, return 1 (since 0! = 1! = 1)
    if n == 0 or n == 1:
        return 1
    # Recursive case: multiply n by the factorial of (n-1)
    else:
        return n * factorial(n - 1)

# Summary:
# The algorithm checks if n is 0 or 1 and returns 1.
# Otherwise, it calls itself with (n-1), multiplying the result by n.
# This process repeats, reducing n by 1 each time, until the base case is reached.
# The recursive calls then resolve, multiplying the results together to produce n!.

# Dynamic input from user
try:
    num = int(input("Enter a non-negative integer: "))
    if num < 0:
        print("Factorial is not defined for negative numbers.")
    else:
        print(f"Factorial of {num} is {factorial(num)}")
except ValueError:
    print("Please enter a valid integer.")
```

OUTPUT:

```
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python313/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab5.1.6.py
Enter a non-negative integer: 5
Factorial of 5 is 120
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

OBSERVATION:

The code is to find the factorial of the given number by using recursive function. It also explains how the algorithm works in comments.

TASK-07:

```
def support_reply(name,gender):
    if gender.lower() == “male”:
        prefix = “Mr.”
   else:
       prefix = “Mrs.”
return f”Dear{prefix} {name},we have resolved the problem.”
```

Regenerate the code so that support messages use neutral language (e.g.,
“Dear {name}”) and optionally accept preferred titles

PROMPT:

Regenerate the code to accept the neutral comments and give the appropriate output.

CODE:

lab5.1.7.py > ...

```python
def support_reply(name, gender=None, title=None):
    if title:
        prefix = title
    elif gender:
        if gender.lower() == "male":
            prefix = "Mr."
        elif gender.lower() == "female":
            prefix = "Ms."
        else:
            prefix = ""
    else:
        prefix = ""
    if prefix:
        return f"Dear {prefix} {name},"
    else:
        return f"Dear {name},"

name = input("Enter name: ")
gender = input("Enter gender (male/female/other, leave blank if unknown): ") or None
title = input("Enter title (leave blank if none): ") or None
print(support_reply(name, gender, title))
```

OUTPUT:

```
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python313/python.exe c:/Users/ramch/OneDrive/Des
op/ai/lab5.1.7.py
Enter name: ramcharan
Enter gender (male/female/other, leave blank if unknown): male
Enter title (leave blank if none):
Dear Mr. ramcharan,
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

OBSERVATION:

The code is about generating message based on their gender and greet the person in an appropriate way. The copilot generated very well without creating any nuisense.