

AI ASSISTED CODING

LAB-5.1

NAME: S.MOHANTH SIDDARTHA

ENROLL.NO:2403A52047

BATCH-03

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 > ...
1  import os
2  import requests
3
4  def get_weather(city):
5      api_key = os.getenv('WEATHER_API_KEY')
6      if not api_key:
7          api_key = input("Enter your OpenWeather API key: ").strip()
8      url = f'https://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric'
9      response = requests.get(url)
10     response.raise_for_status()
11     return response.json()
12
13 if __name__ == "__main__":
14     city = input("Enter city name: ")
15     try:
16         weather = get_weather(city)
17         print(f"Weather in {city}: {weather['weather'][0]['description']}, Temp: {weather['main']['temp']}°C")
18     except Exception as e:
19         print(f"Error: {e}")
```

Output:

```
[notice] A new release of pip is available: 24.3.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python311/Python311.exe C:/Users/ramch/AppData/Local/Programs/Python/Python311/Scripts/pip/ai/lab5.1.1.py
Enter city name: warangal
Enter your OpenWeather API key: 818d5d28d56a6b76906457db5dc1b86d
Weather in warangal: overcast clouds, Temp: 25.61°C
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

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 > ...
1  # Program to store user data
2
3  def get_user_data():
4      user_data = {}
5      user_data['name'] = input("Enter your name: ")
6      user_data['age'] = input("Enter your age: ")
7      user_data['email'] = input("Enter your email: ")
8      return user_data
9
10 def main():
11     data = get_user_data()
12     print("\nStored User Data:")
13     for key, value in data.items():
14         print(f"{key.capitalize()}: {value}")
15
16 if __name__ == "__main__":
17     main()

```

OUTPUT:

```

PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python313/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab5.1.2.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:

```
lab5.1.3.py > ...
1  def is_armstrong_number(number):
2      """
3      Checks whether a given number is an Armstrong number.
4
5      An Armstrong number is a number that is equal to the sum of its own digits
6      each raised to the power of the number of digits.
7
8      For example:
9      153 is an Armstrong number because  $1^3 + 5^3 + 3^3 = 153$ 
10
11     Args:
12         number (int): The number to check.
13
14     Returns:
15         bool: True if the number is an Armstrong number, False otherwise.
16     """
17     # Convert the number to string to easily iterate over digits
18     digits = str(number)
19     num_digits = len(digits)
20     # Calculate the sum of each digit raised to the power of num_digits
21     armstrong_sum = sum(int(digit) ** num_digits for digit in digits)
22     # Check if the sum equals the original number
23     return armstrong_sum == number
24
25 # Example usage:
26 # print(is_armstrong_number(153)) # Output: True
27 number = int(input("Enter a number to check if it is an Armstrong number: "))
28 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:

```
lab5.1.4.py > ...
1  # Quick Sort implementation
2  def quick_sort(arr):
3      """
4      Sorts an array using the Quick Sort algorithm.
5      Quick Sort is a divide-and-conquer algorithm that partitions the array
6      around a pivot, recursively sorting the subarrays.
7      """
8      if len(arr) <= 1:
9          # Base case: arrays with 0 or 1 element are already sorted
10         return arr
11     else:
12         pivot = arr[0] # Choose the first element as pivot
13         left = [x for x in arr[1:] if x < pivot] # Elements less than pivot
14         right = [x for x in arr[1:] if x >= pivot] # Elements greater or equal to pivot
15         # Recursively sort left and right, then combine with pivot
16         return quick_sort(left) + [pivot] + quick_sort(right)
17
18 # Bubble Sort implementation
19 def bubble_sort(arr):
20     """
21     Sorts an array using the Bubble Sort algorithm.
22     Bubble Sort repeatedly steps through the list, compares adjacent elements,
23     and swaps them if they are in the wrong order. This process is repeated
24     until the list is sorted.
25     """
26     n = len(arr)
27     for i in range(n):
28         # Each pass moves the largest unsorted element to its correct position
29         for j in range(0, n - i - 1):
30             if arr[j] > arr[j + 1]:
31                 # Swap if elements are in wrong order
32                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
33     return arr
34
35 # Differences:
36 # - Quick Sort uses recursion and divides the array into subarrays around a pivot.
37 # - Bubble Sort uses nested loops and repeatedly swaps adjacent elements.
38 # - Quick Sort is generally much faster ( $O(n \log n)$  average) than Bubble Sort ( $O(n^2)$ ).
39 # - Quick Sort is not stable by default, Bubble Sort is stable.
40
41 # Example usage:
42 if __name__ == "__main__":
43     data = [64, 34, 25, 12, 22, 11, 90]
44     print("Original:", data)
45     print("Quick Sort:", quick_sort(data))
46     print("Bubble Sort:", bubble_sort(data.copy()))
```

OUTPUT:

```
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python313/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab5.1.4.py
Original: [64, 34, 25, 12, 22, 11, 90]
Quick Sort: [11, 12, 22, 25, 34, 64, 90]
Bubble Sort: [11, 12, 22, 25, 34, 64, 90]
PS C:\Users\ramch\OneDrive\Desktop\ai> 
```

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:

```

lab5.1.5.py > ...
1  import random
2
3  # Sample data: items and their features
4  items = {
5      "The Matrix": {"genre": "Sci-Fi", "director": "Wachowski", "year": 1999},
6      "Inception": {"genre": "Sci-Fi", "director": "Nolan", "year": 2010},
7      "Titanic": {"genre": "Romance", "director": "Cameron", "year": 1997},
8      "Interstellar": {"genre": "Sci-Fi", "director": "Nolan", "year": 2014},
9      "The Notebook": {"genre": "Romance", "director": "Cassavetes", "year": 2004},
10 }
11
12 # User preferences
13 user_preferences = {
14     "genre": "Sci-Fi",
15     "director": "Nolan"
16 }
17
18 def recommend(items, preferences):
19     recommendations = []
20     for item, features in items.items():
21         score = 0
22         reasons = []
23         for key, value in preferences.items():
24             if features.get(key) == value:
25                 score += 1
26                 reasons.append(f"Matches your preferred {key}: {value}")
27         if score > 0:
28             recommendations.append((item, score, reasons))
29     recommendations.sort(key=lambda x: x[1], reverse=True)
30     return recommendations
31
32 def main():
33     recs = recommend(items, user_preferences)
34     if recs:
35         print("Recommended for you:")
36         for item, score, reasons in recs:
37             print(f"- {item} (score: {score})")
38             for reason in reasons:
39                 print(f"    Reason: {reason}")
40     else:
41         print("No recommendations found based on your preferences.")
42
43 if __name__ == "__main__":
44     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 > ...
1  def factorial(n):
2      """
3      Recursive function to calculate the factorial of a non-negative integer n.
4      The factorial of n (n!) is the product of all positive integers less than or equal to n.
5      """
6      # Base case: if n is 0 or 1, return 1 (since 0! = 1! = 1)
7      if n == 0 or n == 1:
8          return 1
9      # Recursive case: multiply n by the factorial of (n-1)
10     else:
11         return n * factorial(n - 1)
12
13     # Summary:
14     # The algorithm checks if n is 0 or 1 and returns 1.
15     # Otherwise, it calls itself with (n-1), multiplying the result by n.
16     # This process repeats, reducing n by 1 each time, until the base case is reached.
17     # The recursive calls then resolve, multiplying the results together to produce n!.
18
19     # Dynamic input from user
20     try:
21         num = int(input("Enter a non-negative integer: "))
22         if num < 0:
23             print("Factorial is not defined for negative numbers.")
24         else:
25             print(f"Factorial of {num} is {factorial(num)}")
26     except ValueError:
27         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 > ...
1  def support_reply(name, gender=None, title=None):
2      if title:
3          prefix = title
4      elif gender:
5          if gender.lower() == "male":
6              prefix = "Mr."
7          elif gender.lower() == "female":
8              prefix = "Ms."
9          else:
10             prefix = ""
11     else:
12         prefix = ""
13     if prefix:
14         return f"Dear {prefix} {name},"
15     else:
16         return f"Dear {name},"
17
18
19 name = input("Enter name: ")
20 gender = input("Enter gender (male/female/other, leave blank if unknown): ") or None
21 title = input("Enter title (leave blank if none): ") or None
22 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/Desktop/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 nuisance.