

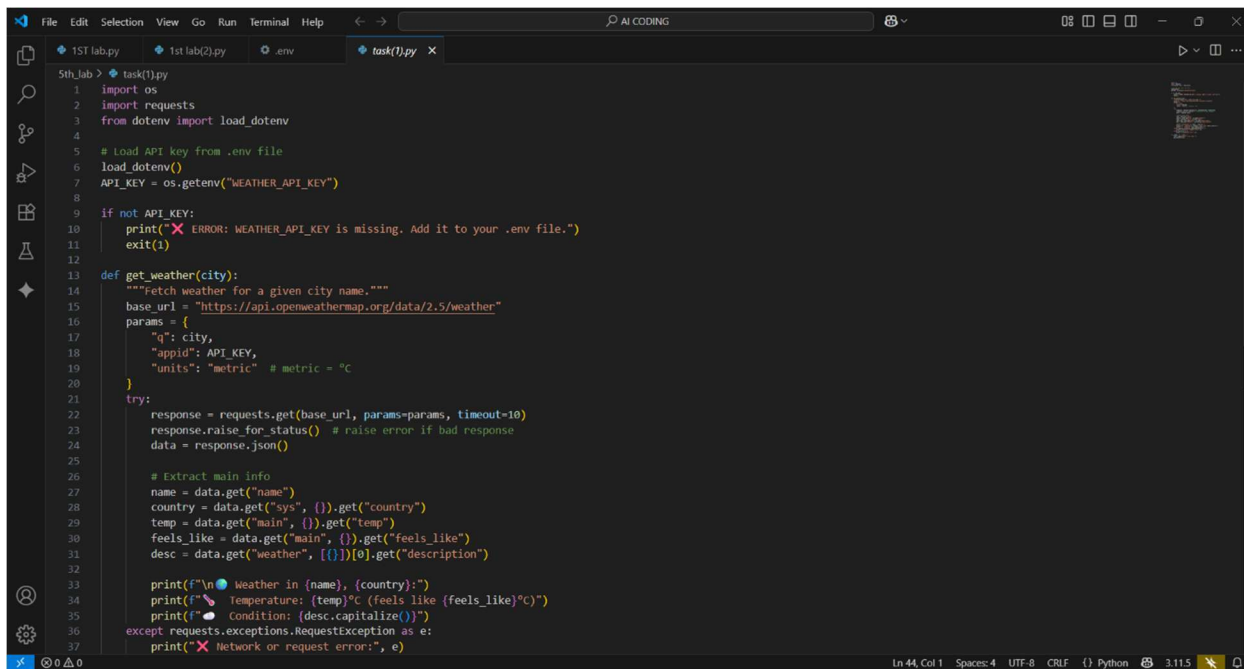
AI Assignment-5.1

Name: N. Laxmi
PrasannaHT NO:
2403A52091

Batch: AIB04

Task 1: Use an AI tool to generate a Python program that connects to a weather API

Code:



```
File Edit Selection View Go Run Terminal Help AI CODING
task(1).py x
5th_lab > task(1).py
1 import os
2 import requests
3 from dotenv import load_dotenv
4
5 # load API key from .env file
6 load_dotenv()
7 API_KEY = os.getenv("WEATHER_API_KEY")
8
9 if not API_KEY:
10     print("❌ ERROR: WEATHER_API_KEY is missing. Add it to your .env file.")
11     exit(1)
12
13 def get_weather(city):
14     """Fetch weather for a given city name."""
15     base_url = "https://api.openweathermap.org/data/2.5/weather"
16     params = {
17         "q": city,
18         "appid": API_KEY,
19         "units": "metric" # metric = °C
20     }
21     try:
22         response = requests.get(base_url, params=params, timeout=10)
23         response.raise_for_status() # raise error if bad response
24         data = response.json()
25
26         # Extract main info
27         name = data.get("name")
28         country = data.get("sys", {}).get("country")
29         temp = data.get("main", {}).get("temp")
30         feels_like = data.get("main", {}).get("feels_like")
31         desc = data.get("weather", [{}])[0].get("description")
32
33         print(f"\n🌤️ Weather in {name}, {country}:")
34         print(f"🌡️ Temperature: {temp}°C (feels like {feels_like}°C)")
35         print(f"☁️ Condition: {desc.capitalize()}")
36     except requests.exceptions.RequestException as e:
37         print("❌ Network or request error:", e)
```

```
23 response.raise_for_status() # raise error if bad response
24 data = response.json()
25
26 # Extract main info
27 name = data.get("name")
28 country = data.get("sys", {}).get("country")
29 temp = data.get("main", {}).get("temp")
30 feels_like = data.get("main", {}).get("feels_like")
31 desc = data.get("weather", [{}])[0].get("description")
32
33 print(f"\n Weather in {name}, {country}:")
34 print(f"🌡️ Temperature: {temp}°C (feels like {feels_like}°C)")
35 print(f"☁️ Condition: {desc.capitalize()}")
36 except requests.exceptions.RequestException as e:
37     print("❌ Network or request error:", e)
38 except Exception as e:
39     print("❌ Unexpected error:", e)
40
41 if __name__ == "__main__":
42     city = input("Enter a city name: ")
43     get_weather(city)
44
```

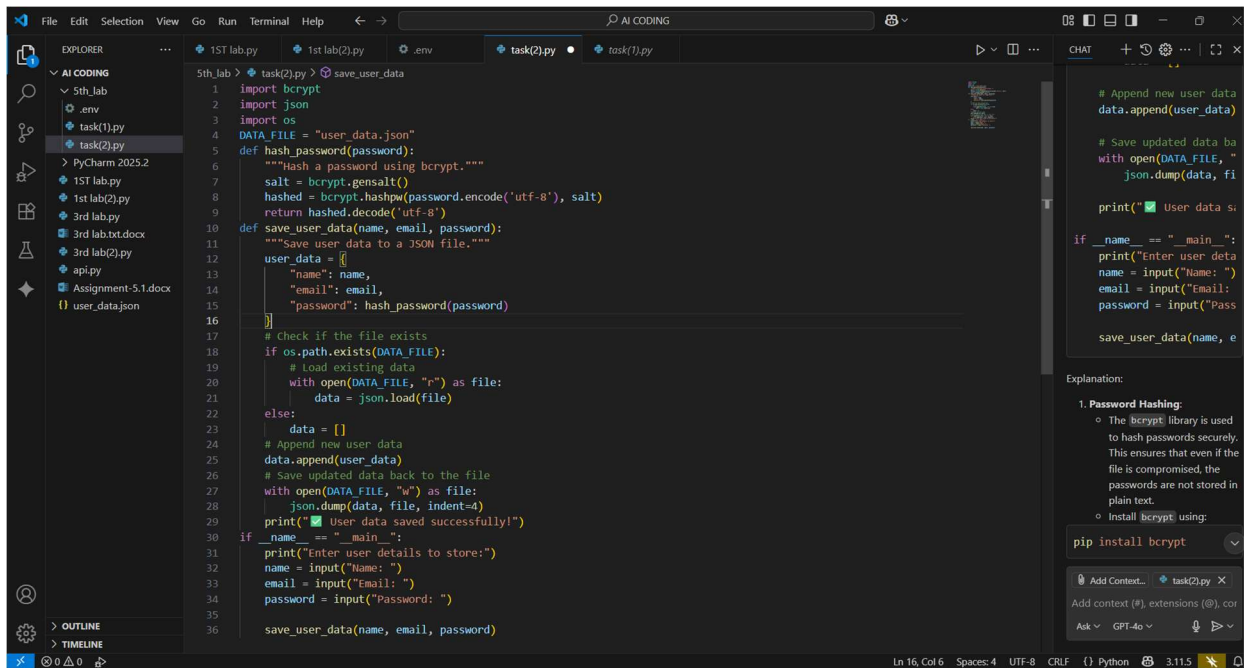
```
File Edit Selection View Go Run Terminal Help AI CODING
1st lab.py 1st lab(2).py .env task(1).py
5th_lab > .env
1 WEATHER_API_KEY="10f2b702212fb3045dd0801b9ca058a4"
2
```

OP:

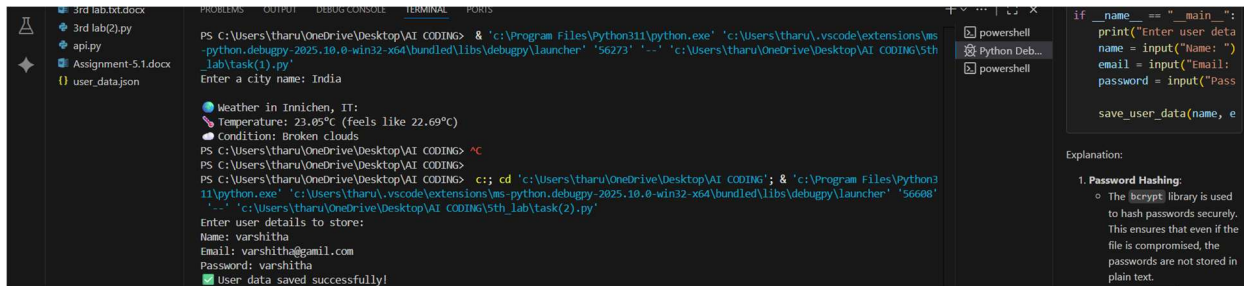
```
8
9 if not API_KEY:
10     print("API Key is missing. Please set it in the .env file.")
11
12 # Main function to get weather
13 def get_weather(city):
14     url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}&units=metric"
15     response = requests.get(url)
16     data = response.json()
17     name = data.get("name")
18     country = data.get("sys", {}).get("country")
19     temp = data.get("main", {}).get("temp")
20     feels_like = data.get("main", {}).get("feels_like")
21     desc = data.get("weather", [{}])[0].get("description")
22
23     print(f"\n Weather in {name}, {country}:")
24     print(f"🌡️ Temperature: {temp}°C (feels like {feels_like}°C)")
25     print(f"☁️ Condition: {desc.capitalize()}")
26
27 # Main execution
28 if __name__ == "__main__":
29     city = input("Enter a city name: ")
30     get_weather(city)
```

Task 2: Use an AI tool to generate a Python script that stores user data (name, email, password) in a file

Code:

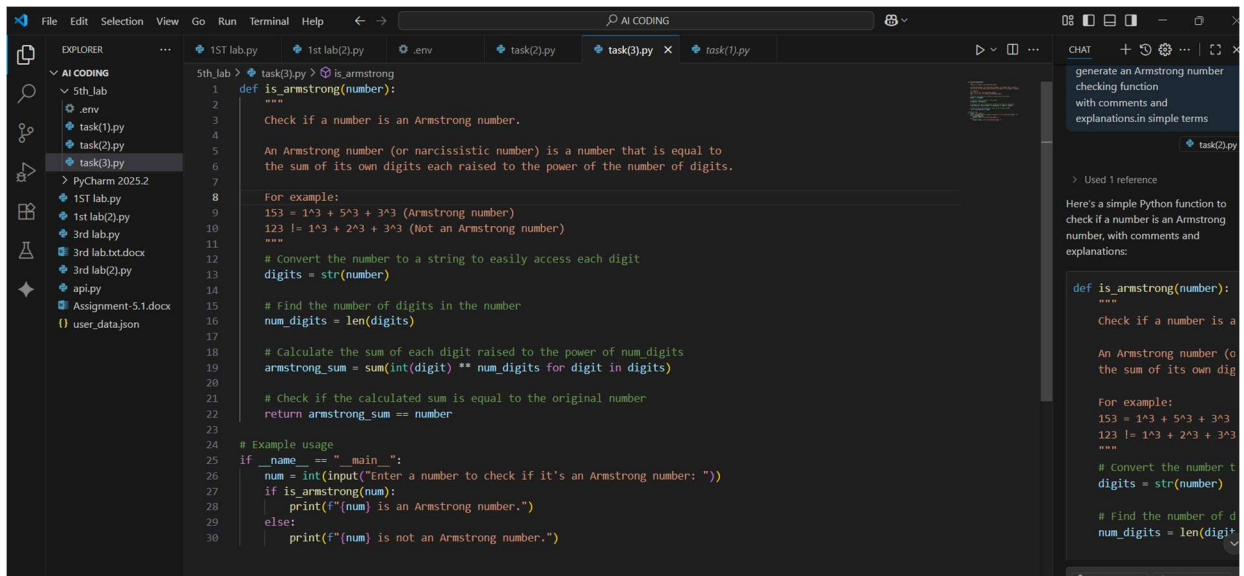


OP:



Task 3: Generate an Armstrong number checking function with comments and explanations.

Code:

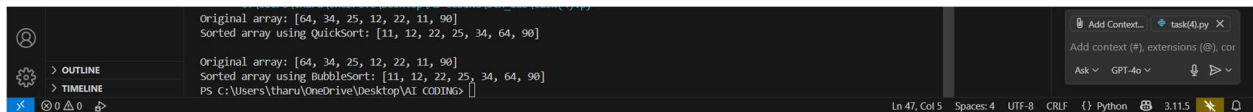


OP:

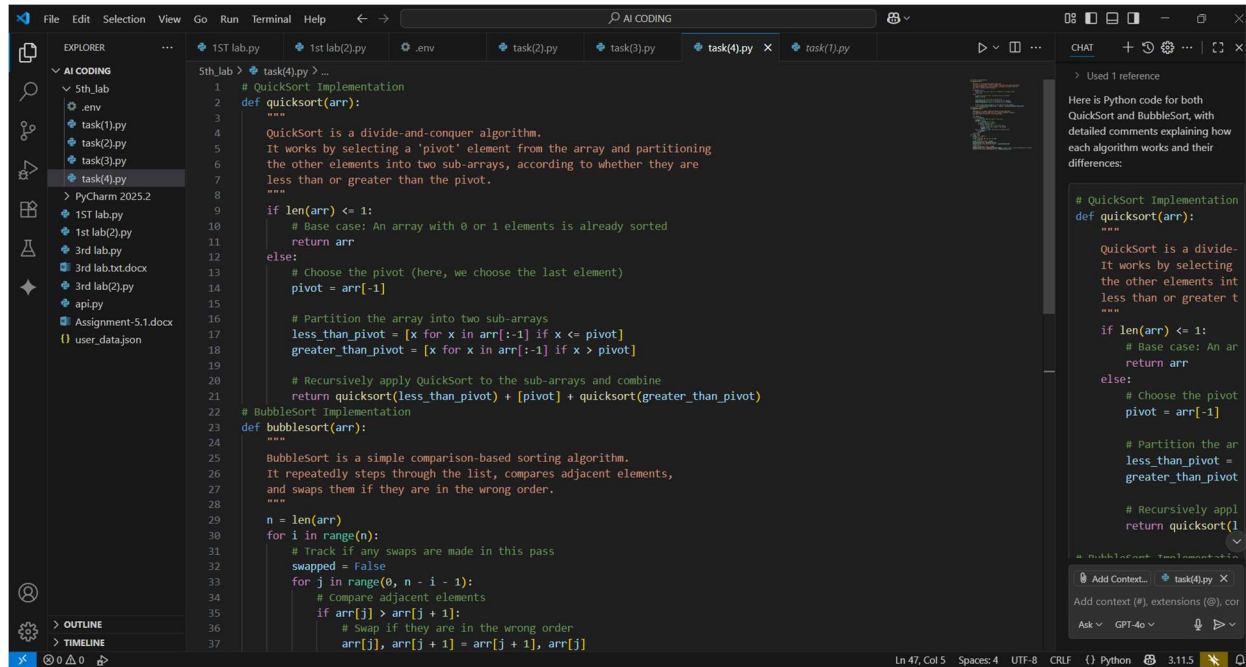


Task 4: Use AI to implement two sorting algorithms (e.g., QuickSort and BubbleSort).

OP:

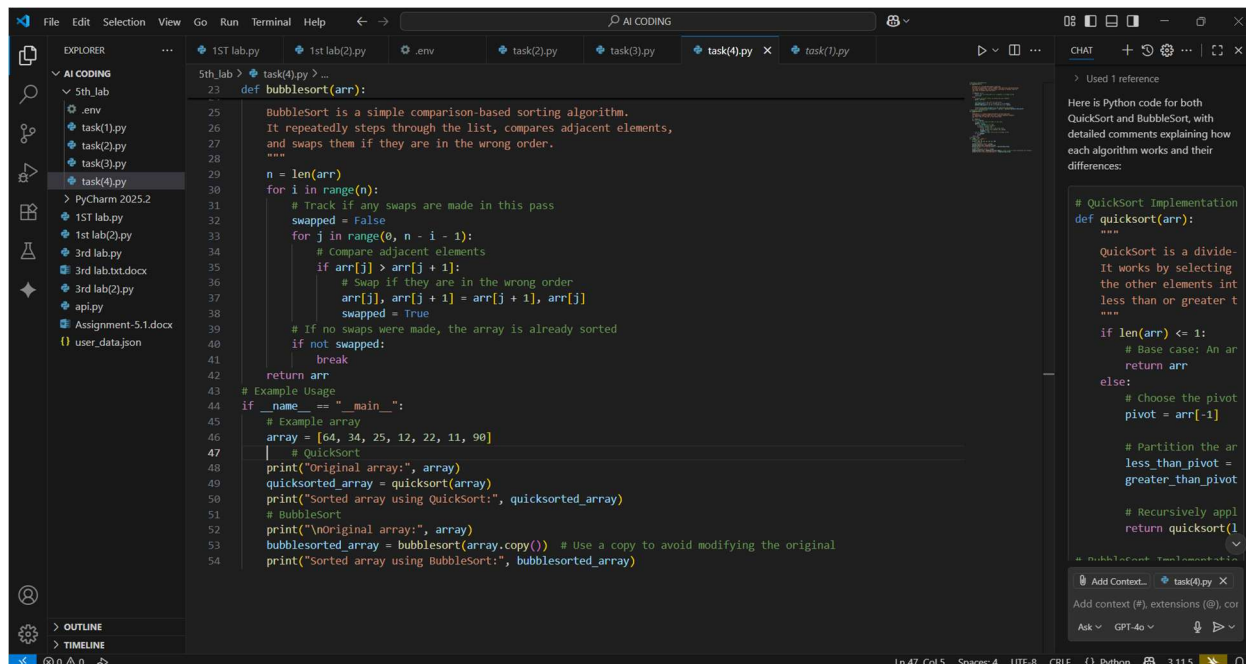


Code:



The screenshot shows the PyCharm IDE with the file explorer on the left, the main editor in the center, and the chat window on the right. The main editor displays the code for task(4).py, which implements QuickSort and BubbleSort algorithms. The chat window on the right shows a reference to the code and a detailed explanation of how each algorithm works and their differences.

```
1 # QuickSort Implementation
2 def quicksort(arr):
3     """
4     QuickSort is a divide-and-conquer algorithm.
5     It works by selecting a 'pivot' element from the array and partitioning
6     the other elements into two sub-arrays, according to whether they are
7     less than or greater than the pivot.
8
9     if len(arr) <= 1:
10         # Base case: An array with 0 or 1 elements is already sorted
11         return arr
12     else:
13         # Choose the pivot (here, we choose the last element)
14         pivot = arr[-1]
15
16         # Partition the array into two sub-arrays
17         less_than_pivot = [x for x in arr[:-1] if x <= pivot]
18         greater_than_pivot = [x for x in arr[:-1] if x > pivot]
19
20         # Recursively apply QuickSort to the sub-arrays and combine
21         return quicksort(less_than_pivot) + [pivot] + quicksort(greater_than_pivot)
22
23 # BubbleSort Implementation
24 def bubblesort(arr):
25     """
26     BubbleSort is a simple comparison-based sorting algorithm.
27     It repeatedly steps through the list, compares adjacent elements,
28     and swaps them if they are in the wrong order.
29
30     n = len(arr)
31     for i in range(n):
32         # Track if any swaps are made in this pass
33         swapped = False
34         for j in range(0, n - i - 1):
35             # Compare adjacent elements
36             if arr[j] > arr[j + 1]:
37                 # Swap if they are in the wrong order
38                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
```

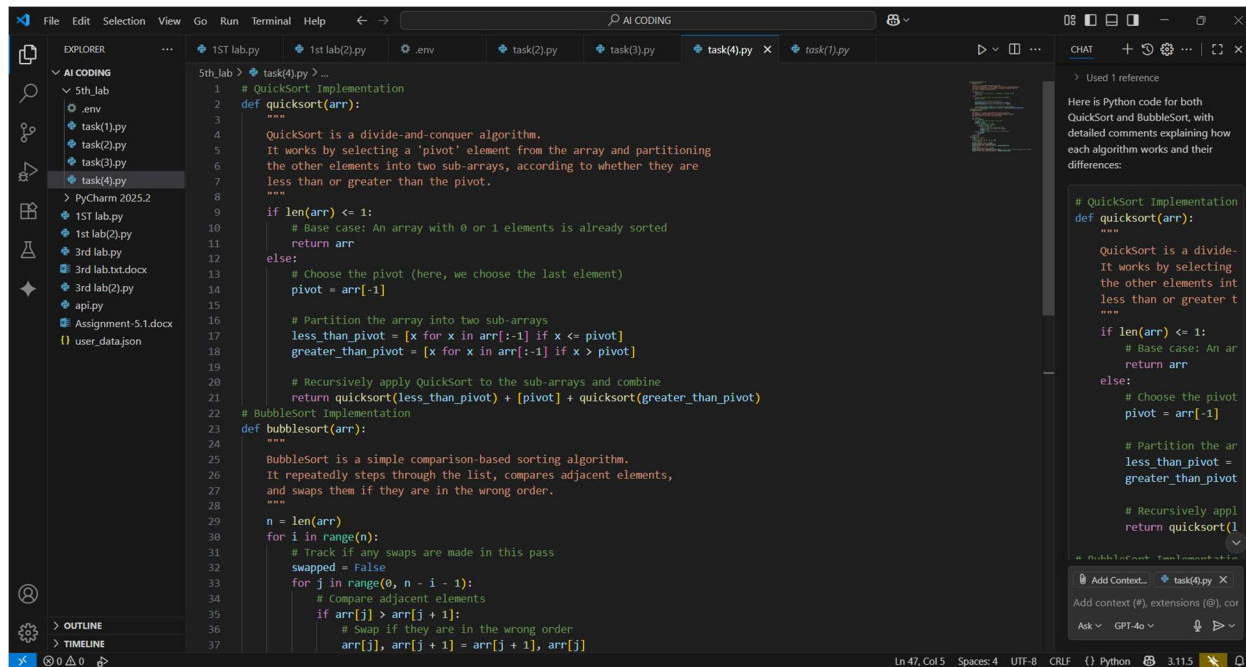


The screenshot shows the PyCharm IDE with the file explorer on the left, the main editor in the center, and the chat window on the right. The main editor displays the code for task(4).py, which implements BubbleSort and an example usage. The chat window on the right shows a reference to the code and a detailed explanation of how each algorithm works and their differences.

```
23 def bubblesort(arr):
24     """
25     BubbleSort is a simple comparison-based sorting algorithm.
26     It repeatedly steps through the list, compares adjacent elements,
27     and swaps them if they are in the wrong order.
28
29     n = len(arr)
30     for i in range(n):
31         # Track if any swaps are made in this pass
32         swapped = False
33         for j in range(0, n - i - 1):
34             # Compare adjacent elements
35             if arr[j] > arr[j + 1]:
36                 # Swap if they are in the wrong order
37                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
38                 swapped = True
39         # If no swaps were made, the array is already sorted
40         if not swapped:
41             break
42     return arr
43
44 # Example Usage
45 if __name__ == "__main__":
46     # Example array
47     array = [64, 34, 25, 12, 22, 11, 90]
48     # QuickSort
49     print("Original array:", array)
50     quicksorted_array = quicksort(array)
51     print("Sorted array using QuickSort:", quicksorted_array)
52     # BubbleSort
53     print("Original array:", array)
54     bubblesorted_array = bubblesort(array.copy()) # Use a copy to avoid modifying the original
55     print("Sorted array using BubbleSort:", bubblesorted_array)
```

Task 5: Use AI to create a product recommendation system

Code:



The screenshot shows the Visual Studio Code editor with a file explorer on the left containing a project named 'AI CODING'. The main editor window displays a Python file named 'task(4).py' with the following code:

```
1 # QuickSort Implementation
2 def quicksort(arr):
3     """
4     QuickSort is a divide-and-conquer algorithm.
5     It works by selecting a 'pivot' element from the array and partitioning
6     the other elements into two sub-arrays, according to whether they are
7     less than or greater than the pivot.
8     """
9     if len(arr) <= 1:
10         # Base case: An array with 0 or 1 elements is already sorted
11         return arr
12     else:
13         # Choose the pivot (here, we choose the last element)
14         pivot = arr[-1]
15
16         # Partition the array into two sub-arrays
17         less_than_pivot = [x for x in arr[:-1] if x <= pivot]
18         greater_than_pivot = [x for x in arr[:-1] if x > pivot]
19
20         # Recursively apply QuickSort to the sub-arrays and combine
21         return quicksort(less_than_pivot) + [pivot] + quicksort(greater_than_pivot)
22
23 # BubbleSort Implementation
24 def bubblesort(arr):
25     """
26     BubbleSort is a simple comparison-based sorting algorithm.
27     It repeatedly steps through the list, compares adjacent elements,
28     and swaps them if they are in the wrong order.
29     """
30     n = len(arr)
31     for i in range(n):
32         # Track if any swaps are made in this pass
33         swapped = False
34         for j in range(0, n - i - 1):
35             # Compare adjacent elements
36             if arr[j] > arr[j + 1]:
37                 # Swap if they are in the wrong order
38                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
```

The right sidebar shows a chat window with the following text:

Used 1 reference

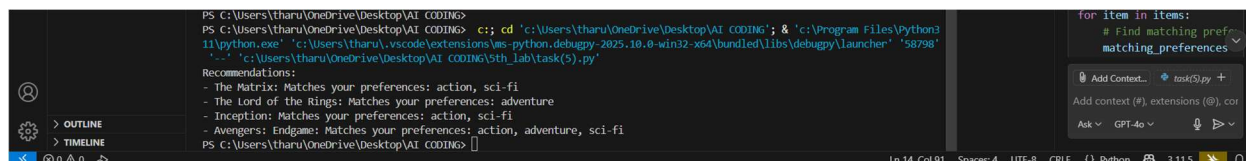
Here is Python code for both QuickSort and BubbleSort with detailed comments explaining how each algorithm works and their differences:

```
# QuickSort Implementation
def quicksort(arr):
    """
    QuickSort is a divide-and-conquer algorithm.
    It works by selecting the other elements into two sub-arrays, according to whether they are less than or greater than the pivot.
    """
    if len(arr) <= 1:
        # Base case: An array with 0 or 1 elements is already sorted
        return arr
    else:
        # Choose the pivot (here, we choose the last element)
        pivot = arr[-1]

        # Partition the array into two sub-arrays
        less_than_pivot = [x for x in arr[:-1] if x <= pivot]
        greater_than_pivot = [x for x in arr[:-1] if x > pivot]

        # Recursively apply QuickSort to the sub-arrays and combine
        return quicksort(less_than_pivot) + [pivot] + quicksort(greater_than_pivot)
```

OP:



The screenshot shows the Visual Studio Code editor with a terminal window at the bottom. The terminal displays the following command prompt and output:

```
PS C:\Users\tharu\OneDrive\Desktop\AI CODING>
PS C:\Users\tharu\OneDrive\Desktop\AI CODING> c:: cd 'c:\Users\tharu\OneDrive\Desktop\AI CODING'; & 'c:\Program Files\Python311\python.exe' 'c:\Users\tharu\vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '58798'
'...' 'c:\Users\tharu\OneDrive\Desktop\AI CODING\5th_lab\task(5).py'
Recommendations:
- The Matrix: Matches your preferences: action, sci-fi
- The Lord of the Rings: Matches your preferences: adventure
- Inception: Matches your preferences: action, sci-fi
- Avengers: Endgame: Matches your preferences: action, adventure, sci-fi
PS C:\Users\tharu\OneDrive\Desktop\AI CODING>
```

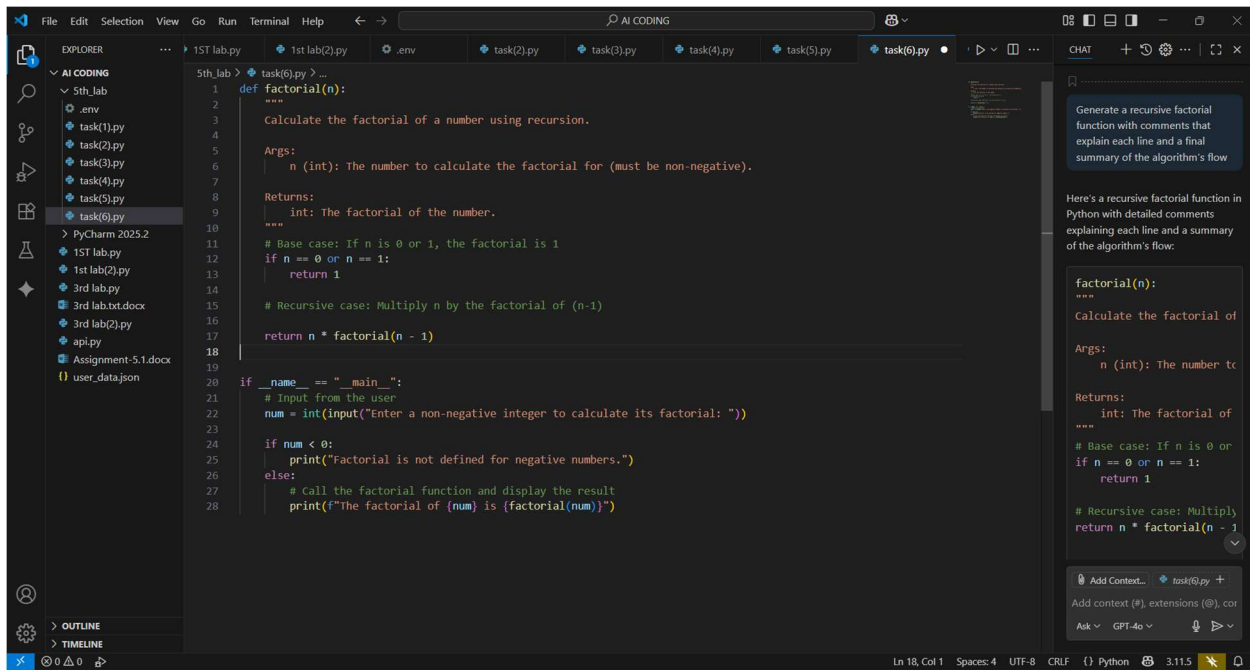
The right sidebar shows a chat window with the following text:

for item in items:

```
# Find matching preferences
matching_preferences
```

Task 6: Ask AI to generate a Python function for calculating factorial using recursion.

Code:



OP:

