AIASSISTEDCODING

LABTEST-02

NAME:M.Suchith

ENROLL.NO:2403A52068

BATCH:04

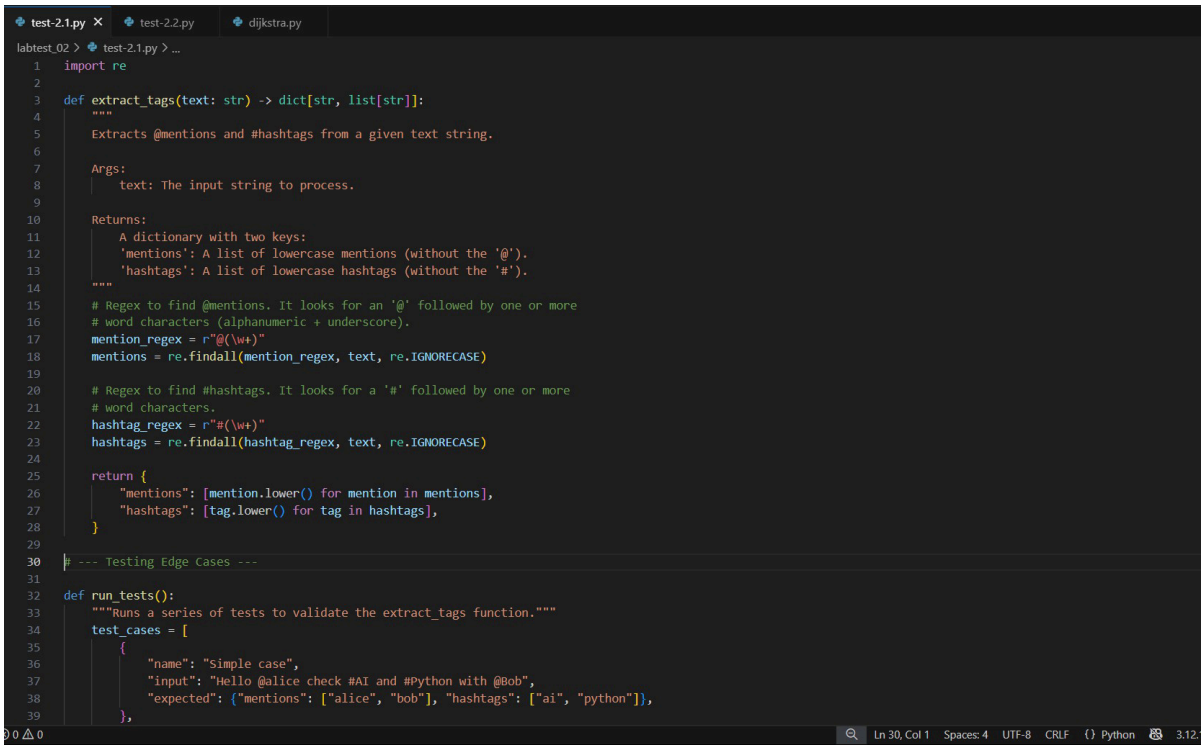
TASK-01:

Useregrextoextract@mentionsand#hashtags(case - insensitive) and return lower case list.

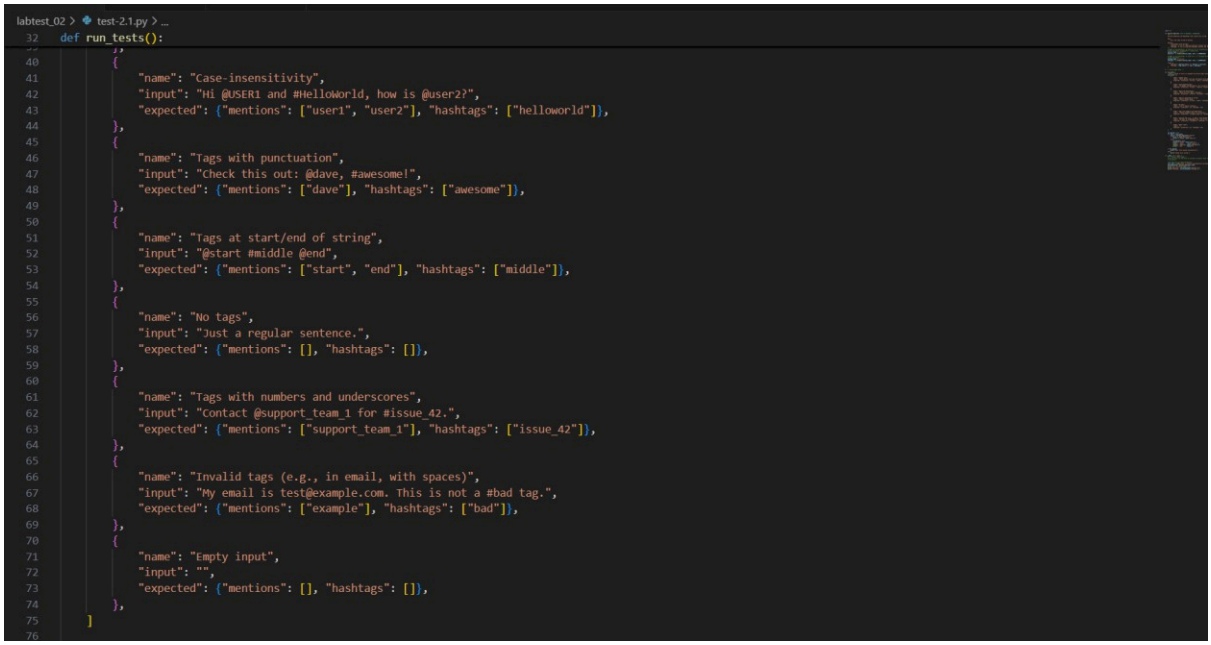
PROMPT:

Generateapythoncodewhichacceptsainputstringas input and performs the operation like segregating the wordswhichstartswiththehashtags“#”andmentions “@” and place them in a list in lower case.

CODE:



```python
import re

def extract_tags(text: str) -> dict[str, list[str]]:
    """
    Extracts @mentions and #hashtags from a given text string.

    Args:
        text: The input string to process.

    Returns:
        A dictionary with two keys:
        'mentions': A list of lowercase mentions (without the '@').
        'hashtags': A list of lowercase hashtags (without the '#').
    """
    # Regex to find @mentions. It looks for an '@' followed by one or more
    # word characters (alphanumeric + underscore).
    mention_regex = r"@(\w+)"
    mentions = re.findall(mention_regex, text, re.IGNORECASE)

    # Regex to find #hashtags. It looks for a '#' followed by one or more
    # word characters.
    hashtag_regex = r"#(\w+)"
    hashtags = re.findall(hashtag_regex, text, re.IGNORECASE)

    return {
        "mentions": [mention.lower() for mention in mentions],
        "hashtags": [tag.lower() for tag in hashtags],
    }

# --- Testing Edge Cases ---

def run_tests():
    """Runs a series of tests to validate the extract_tags function."""
    test_cases = [
        {
            "name": "Simple case",
            "input": "Hello @alice check #AI and #Python with @Bob",
            "expected": {"mentions": ["alice", "bob"], "hashtags": ["ai", "python"]},
        },
```



```python
def run_tests():
        {
            "name": "Case-insensitivity",
            "input": "Hi @USER1 and #HelloWorld, how is @user2?",
            "expected": {"mentions": ["user1", "user2"], "hashtags": ["helloworld"]},
        },
        {
            "name": "Tags with punctuation",
            "input": "Check this out: @dave, #awesome!",
            "expected": {"mentions": ["dave"], "hashtags": ["awesome"]},
        },
        {
            "name": "Tags at start/end of string",
            "input": "@start #middle @end",
            "expected": {"mentions": ["start", "end"], "hashtags": ["middle"]},
        },
        {
            "name": "No tags",
            "input": "Just a regular sentence.",
            "expected": {"mentions": [], "hashtags": []},
        },
        {
            "name": "Tags with numbers and underscores",
            "input": "Contact @support_team_1 for #issue_42.",
            "expected": {"mentions": ["support_team_1"], "hashtags": ["issue_42"]},
        },
        {
            "name": "Invalid tags (e.g., in email, with spaces)",
            "input": "My email is test@example.com. This is not a #bad tag.",
            "expected": {"mentions": ["example"], "hashtags": ["bad"]},
        },
        {
            "name": "Empty input",
            "input": "",
            "expected": {"mentions": [], "hashtags": []},
        },
    ]
```

```python
def run_tests():

    all_passed = True
    for test in test_cases:
        result = extract_tags(test["input"])
        if result == test["expected"]:
            print(f"✅ PASSED: {test['name']}")
        else:
            all_passed = False
            print(f"❌ FAILED: {test['name']}")
            print(f"  Input:    '{test['input']}'")
            print(f"  Expected: {test['expected']}")
            print(f"  Got:      {result}")

    if all_passed:
        print("\nAll tests passed successfully!")
    else:
        print("\nSome tests failed.")

if __name__ == "__main__":
    # --- Dynamic Input Usage ---
    # You can still run the tests by calling run_tests() here if you want.
    # run_tests()

    # Ask the user for input dynamically
    user_input = input("Enter a sentence with @mentions and #hashtags: ")
    extracted_data = extract_tags(user_input)
    print("\n--- Extracted Tags ---")
    print(f"Mentions: {extracted_data['mentions']}")
    print(f"Hashtags: {extracted_data['hashtags']}")

```

OUTPUT:

```
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/labtest_02/test-2.1.py
Enter a sentence with @mentions and #hashtags: Hello @alice check #AI and #Python with @Bob

--- Extracted Tags ---
Mentions: ['alice', 'bob']
Hashtags: ['ai', 'python']
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

OBSERVATION:

ThisPythonscriptactslikeasocialmediatagfinderforany piece of text. It's designed to read a sentence and intelligentlypulloutallthewordsthatstartwith a#(hashtags)oran@(mentions).Thescriptissmartenough to ignore punctuation attached to the tags and isn't case -

sensitive.Afteryouprovideasentence,itneatlyorganizesthe findings into two separate, lowercase lists—one for all the mentions and one for all the hashtags—and displays them.

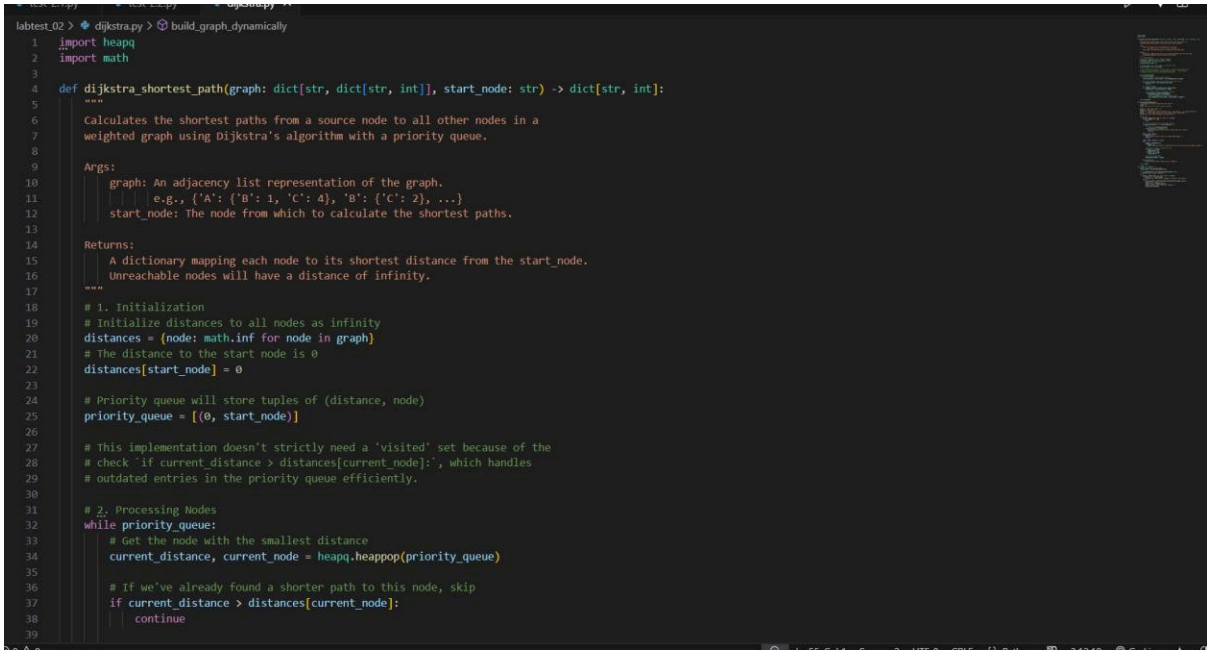
TASK-02:

ImplementDijkstrafromasourcenode'A'toallnodes using a priority queue

PROMPT:

Generateapythonscriptwhichfindstheshortest distancebytakingtheinputasnodeswhicharegraphs and asks from which node you want the shortest path
andgivetheresultinthestructureddictionaryformat.

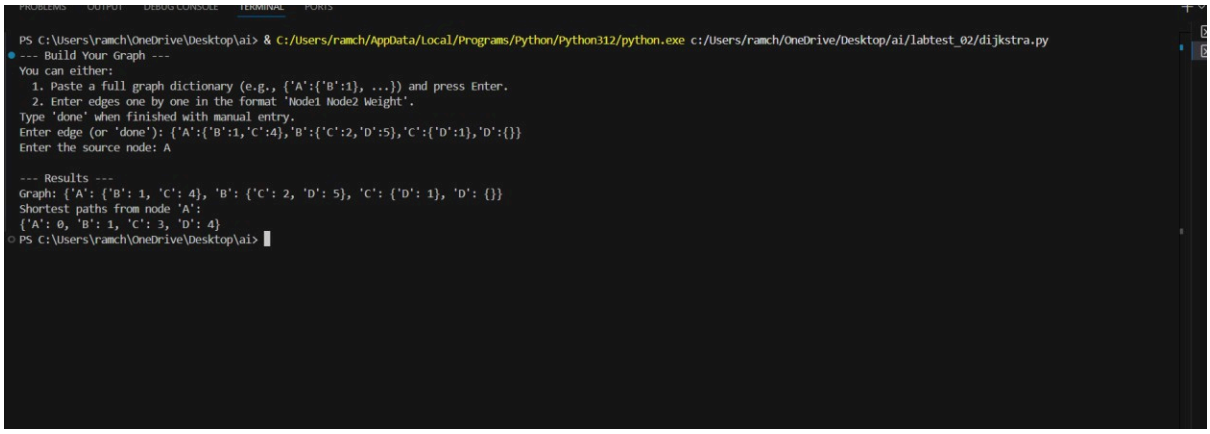
CODE:



```python
def dijkstra_shortest_path(graph: dict[str, dict[str, int]], start_node: str) -> dict[str, int]:

        # 3. Edge Relaxation
        for neighbor, weight in graph[current_node].items():
            new_distance = current_distance + weight

            # If a shorter path to the neighbor is found
            if new_distance < distances[neighbor]:
                distances[neighbor] = new_distance
                # Push the updated path to the priority queue
                heapq.heappush(priority_queue, (new_distance, neighbor))

    return distances

def build_graph_dynamically():
    """Interactively builds a graph from user input."""
    graph = {}
    import ast # Use ast for safe literal evaluation

    print("--- Build Your Graph ---")
    print("You can either:")
    print("  1. Paste a full graph dictionary (e.g., {'A':{'B':1}, ...}) and press Enter.")
    print("  2. Enter edges one by one in the format 'Node1 Node2 Weight'.")
    print("Type 'done' when finished with manual entry.")

    while True:
        entry = input("Enter edge (or 'done'): ").strip()
        if entry.lower() == 'done':
            break

        # Try to parse the input as a dictionary literal
        if entry.startswith('{') and entry.endswith('}'):
            try:
                return ast.literal_eval(entry)
            except (ValueError, SyntaxError):
                print("Invalid dictionary format. Please check your syntax.")
                continue

        parts = entry.split()
```

```python
def build_graph_dynamically():
                print("Invalid format. Please use 'Node1 Node2 Weight'.")
                continue

            node1, node2, weight_str = parts
            try:
                weight = int(weight_str)
                if weight < 0:
                    print("Warning: Dijkstra's algorithm may not work correctly with negative weights.")

                # Add nodes to graph if they don't exist
                if node1 not in graph:
                    graph[node1] = {}
                if node2 not in graph:
                    graph[node2] = {}

                # Add the directed edge
                graph[node1][node2] = weight

            except ValueError:
                print("Invalid weight. Please enter an integer.")

    return graph
if __name__ == "__main__":
    # Build the graph from user input
    network_graph = build_graph_dynamically()

    if not network_graph and isinstance(network_graph, dict):
        print("Graph is empty. Exiting.")
    else:
        source = input("Enter the source node: ").strip()
        if source not in network_graph:
            print(f"Error: Source node '{source}' not found in the graph.")
        else:
            shortest_paths = dijkstra_shortest_path(network_graph, source)
            print(f"\n--- Results ---")
            print(f"Graph: {network_graph}")
            print(f"Shortest paths from node '{source}':")
            print(shortest_paths)
```

OUTPUT:



OBSERVATION:

This Python script acts like a GPS for a network you define. It usesDijkstra'salgorithm,aclassicmethodtofindtheshortest path from a single starting point to all other locations. The
script first asks you to build the network map, either by pasting a fulldictionary or by adding connections one by one.
Afteryouspecifyyourstarting"source"node,itcalculatesthe minimum "distance" (or cost) to every other node. The resultsshowasimplelistofeachdestinationandtheshortest possible route cost to get there from your start point.