AI LAB EXAM 2

NAME:D.SUMITH

ROLL NO:2403A510A3

BATCHNO:06

H.1 — [S09H1] Extract hashtags and mentions

Prompt

Write a Python function that extracts all @mentions and #hashtags from a given text.
The function should return two separate lists — mentions and hashtags — both converted to lowercase.
Ignore any surrounding punctuation next to mentions or hashtags.

Code

```python
import re

def extract_tags(text):
    mentions = re.findall(r'@([a-zA-Z0-9_]+)', text)
    hashtags = re.findall(r'#([a-zA-Z0-9_]+)', text)
    return {
        'mentions': [m.lower() for m in mentions],
        'hashtags': [h.lower() for h in hashtags]
    }

# Sample Input
sample_text = "Hello @alice check #AI and #Python with @Bob"
output = extract_tags(sample_text)
print(output)

# This script extracts mentions and hashtags from a given text.
```

Output

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\AI LAB> python -u "d:\AI LAB\tempCodeRunnerFile.python"
{'mentions': ['alice', 'bob'], 'hashtags': ['ai', 'python']}
PS D:\AI LAB>
```

Observation

The regex correctly extracts mentions and hashtags, ignores punctuation, and normalizes them to lowercase.

AI LAB EXAM 2

H.2 — [S09H2] Shortest path on weighted graph (Dijkstra)

Prompt

Write a Python function that counts the total number of @mentions and #hashtags in a given text.
The function should return a dictionary in the format: {'mentions': count_of_mentions, 'hashtags': count_of_hashtags}.
Mentions and hashtags should be detected using regex and be case-insensitive.

Code

import heapq Untitled-2

```python
import heapq

def dijkstra(graph, start):
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    pq = [(0, start)]

    while pq:
        current_dist, current_node = heapq.heappop(pq)

        if current_dist > distances[current_node]:
            continue

        for neighbor, weight in graph[current_node].items():
            distance = current_dist + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(pq, (distance, neighbor))

    return distances

# Sample Input
graph = {'A':{'B':1,'C':4},'B':{'C':2,'D':5},'C':{'D':1},'D':{}}
output = dijkstra(graph, 'A')
print(output)

# Sample Output: {'A': 0, 'B': 1, 'C': 3, 'D': 4}
```

Output

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\AI LAB> python -u "d:\AI LAB\tempCodeRunnerFile.python"
{'A': 0, 'B': 1, 'C': 3, 'D': 4}
PS D:\AI LAB>
```

Observation

Dijkstra's algorithm computes the shortest paths correctly from source 'A' using edge relaxation and a priority queue.