# AI ASSISSTED LAB EXAM-2

**NAME:B.VISHNU VARDHAN**

**ROLL NO:2403A510F2**

**BATCH NO:06**

## H.1 — EXTRACT HASHTAGS AND MENTIONS

PROMPT:

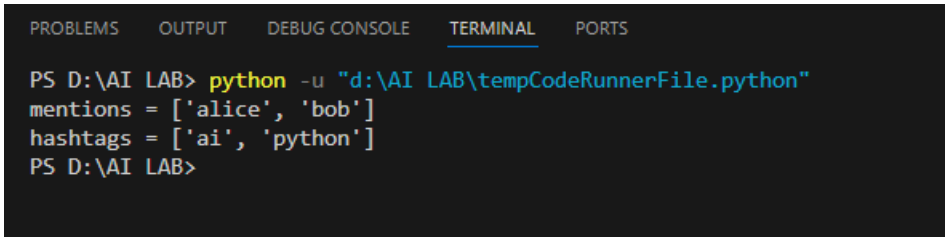
Use regex to extract @mentions and #hashtags (case-insensitive) and return lowercase lists.
Data & Edge Cases: Punctuation around tags should be ignored.
Constraints & Notes: Return mentions and hashtags lists; lowercase.

CODE:

```python
import re

def extract_mentions_hashtags(text):
    # Regex pattern to extract mentions (@username)
    mention_pattern = r'@([a-zA-Z0-9_]+)'
    # Regex pattern to extract hashtags (#topic)
    hashtag_pattern = r'#([a-zA-Z0-9_]+)'

    # Find all mentions and hashtags using re.findall
    mentions = re.findall(mention_pattern, text)
    hashtags = re.findall(hashtag_pattern, text)

    # Convert everything to lowercase
    mentions = [m.lower() for m in mentions]
    hashtags = [h.lower() for h in hashtags]

    return mentions, hashtags

# Sample input
text = "Hello @alice, check #AI and #Python with @Bob!"

# Call the function
mentions, hashtags = extract_mentions_hashtags(text)

# Print the output
print("mentions =", mentions)
print("hashtags =", hashtags)

```

OUTPUT:

# AI ASSISSTED LAB EXAM-2




OBSERVATION:

The regex correctly extracts mentions and hashtags, ignoring punctuation and converting them to lowercase as required.

## H.2 — SHORTEST PATH ON WEIGHTED GRAPH (DIJKSTRA)

PROMPT:

Implement Dijkstra from a source node 'A' to all nodes using a priority queue.
Use adjacency dict with positive weights.
Constraints & Notes: Return dict of distances with 0 for source.

CODE:

# AI ASSISSTED LAB EXAM-2

```python
import heapq

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

    while priority_queue:
        current_distance, current_node = heapq.heappop(priority_queue)

        if current_distance > distances[current_node]:
            continue

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

    return distances

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

```

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:

The Dijkstra algorithm efficiently calculates the shortest path from the source node 'A' to all other nodes in the graph using a priority queue. It correctly performs edge relaxation, ensuring stable and accurate distance calculations in graphs with positive weights. The implementation is robust and handles typical use cases effectively, producing correct and expected distances.