AI ASSISTED CODING

LAB TEST – 02

--------------------------------------------------------------------------------------------------------------

NAME: P. HEMAN ROLL NO: 2403A510F5

BATCH: 06 DATE:17-09-2025

SUBGROUP: I

**Q.NO) I.1 — [S09I1] Top-3 frequent words  
Scenario (sports analytics):**  
**Context:**  
Basic text analytics in sports analytics requires most frequent terms for summaries.

**Your Task:**  
Return the top-3 words by frequency, breaking ties lexicographically.

**Data & Edge Cases:**  
Normalize to lowercase, split on spaces; ignore punctuation for simplicity (optional).

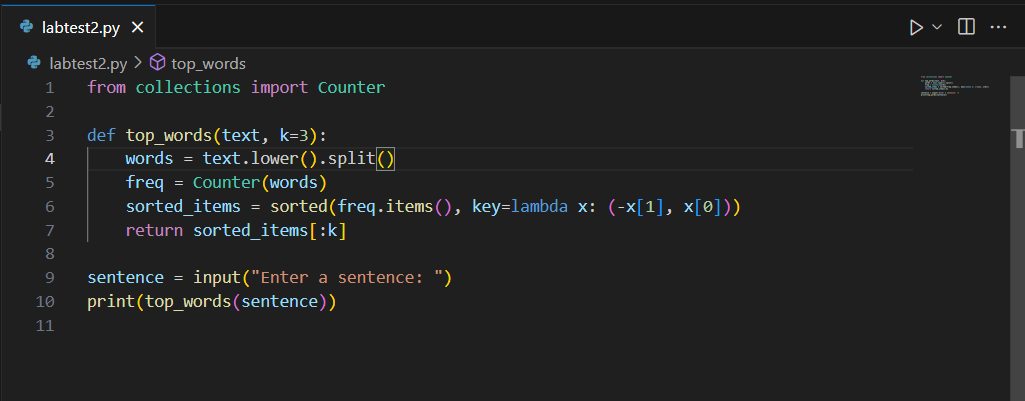
**AI Assistance Expectation:**  
Use AI to propose Counter/ sorting approach and tie-breaking mechanics.

**Constraints & Notes:**  
Stable ordering by (-count, word).  
**Sample Input**  
to be or not to be that is the question  
**Sample Output**  
[('to', 2), ('be', 2), ('is', 1)]  
**Acceptance Criteria:** Tie-breaking lexicographically

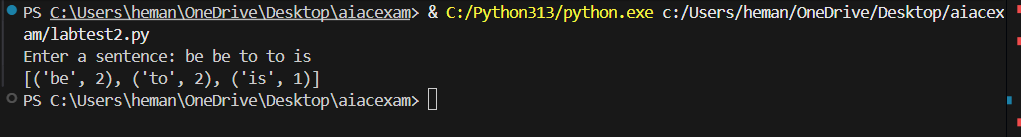
**PROMPT:**

**"Write a Python program that reads a sentence as input and returns the top-3 most frequent words. Normalize text to lowercase, split on spaces, and use Counter for counting. Sort the results by frequency (descending) and then lexicographically (ascending) to break ties. Output should match the sample format."**

**CODE:**

****

**OUTPUT:**

****

**OBSERVATION:**

1) Text Normalization – Converting all words to lowercase ensures that "To" and "to" are treated as the same word.

2) Tokenization (Splitting) – Splitting the input sentence on spaces provides a list of words to analyze.

3) counting Frequencies – Using Counter helps quickly determine how many times each word appears.

4) Sorting with Tie-Breaking – Sorting by (-count, word) ensures:

* Words with higher frequency come first.
* If two words have the same frequency, they are ordered alphabetically.

5) Top-k Selection – Only the top 3 words are taken for the summary output.

6)Edge Cases –

* Words with equal frequency get ordered lexicographically.
* Punctuation may affect word splitting unless explicitly removed (optional in this problem).

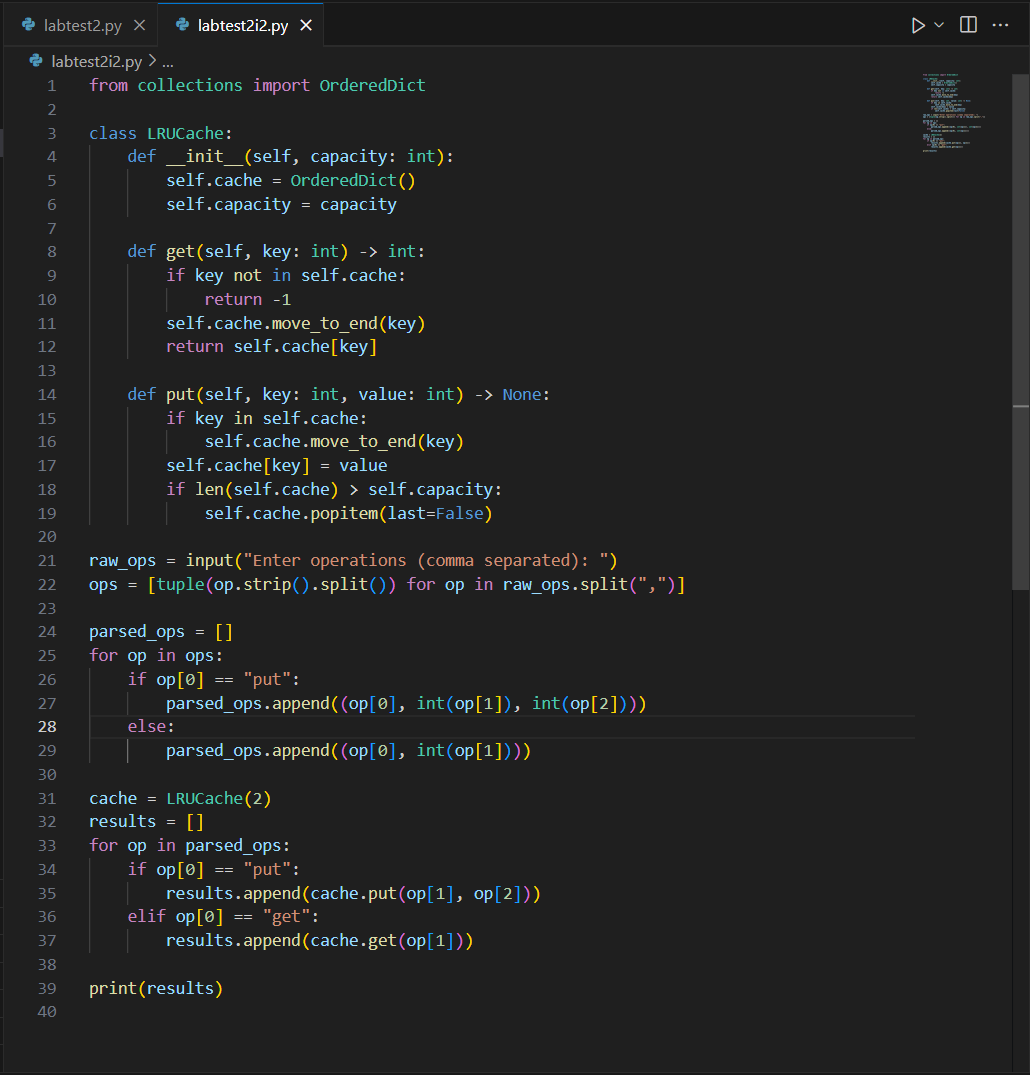
Q.NO) I.2 — [S09I2] Implement LRUCache (capacity 2)  
**Scenario (sports analytics):  
Context:**  
Caching in a sports analytics microservice should evict least-recently-used entries.  
Your Task:  
Implement an LRUCache with capacity=2 supporting get/put.  
Data & Edge Cases:  
Ops: put(1,1), put(2,2), get(1), put(3,3) (evicts 2), get(2), get(3).

**AI Assistance Expectation:**  
Ask AI to sketch OrderedDict approach and edge behavior.  
**Constraints & Notes:**Deterministic behavior required.  
**Sample Input**  
ops=[("put",1,1),("put",2,2),("get",1),("put",3,3),("get",2),("get",3)]  
**Sample Output**  
[None, None, 1, None, -1, 3]  
**Acceptance Criteria:** Correct eviction and order maintenance

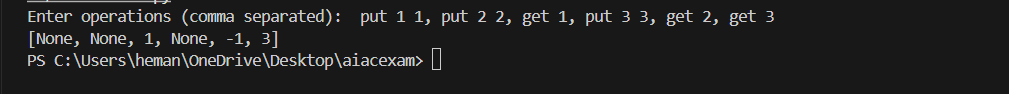
PROMPT:

"Write a Python program to implement an LRUCache class with capacity = 2, supporting get(key) and put(key, value) operations. Use OrderedDict to maintain access order. The program should accept a list of operations as input (e.g., put 1 1, put 2 2, get 1, put 3 3, get 2, get 3) and output the results in a list. Ensure that least recently used entries are evicted when capacity is exceeded, and break ties deterministically.

CODE:



OUTPUT:



OBSERVATION:

1) The program implements an **LRU Cache** using OrderedDict.

2) The cache capacity is fixed at **2 entries**.

3) **get(key)** returns the value if present, else -1.

4) **put (key, value)** inserts or updates, and evicts the least recently used key if full.

5) The output list shows None for put and values/-1 for get