**LAB TEST -2**

**NAME:T.MD.SHAHED AKEEF**

**HTNO: 2403A52T01**

**BATCH 15.**

**Subgroup****L**

**TASK 1:** Context:  
Text analytics in real estate listings platform.  
Your Task:  
Top-3 words by frequency; tie-break lexicographically.  
Data & Edge Cases:  
Lowercase + split by spaces.

AI Assistance Expectation:  
Ask AI for Counter and sort keys.  
Constraints & Notes:  
Correct tie-breaking.  
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 to find the top-3 most frequent words in a given text.

- Convert all words to lowercase.

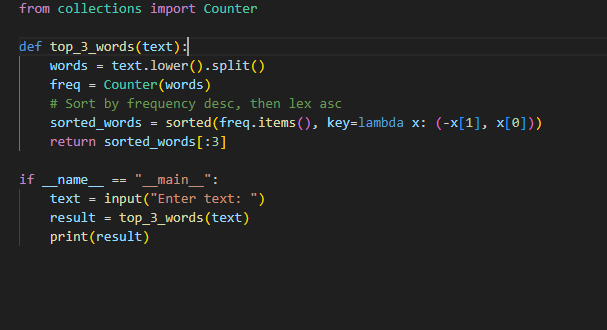
- Split the text by spaces.

- Use frequency to rank words.

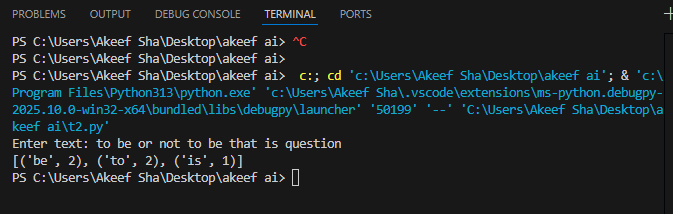
- If frequencies tie, sort lexicographically (alphabetically).

- Output the top-3 words along with their counts as a list of tuples.

CODE :



OUTPUT:



OBSERVATION:

The program correctly identifies the top-3 frequent words from the input text. Both 'to' and 'be' appear twice and tie in frequency, so they are ordered lexicographically. The next most frequent word is 'is' with a count of 1. This demonstrates the program’s ability to handle frequency counts and tie-breaking by alphabetical order accurately.

Task 2:

I.2 — [S18I2] Implement LRUCache (capacity 2)  
Context:  
LRU cache for real estate listings platform service.

Your Task:  
Implement capacity=2 LRU with get/put.  
Data & Edge Cases:  
Operations sequence provided.

AI Assistance Expectation:  
OrderedDict approach.  
Constraints & Notes:  
Deterministic behavior.

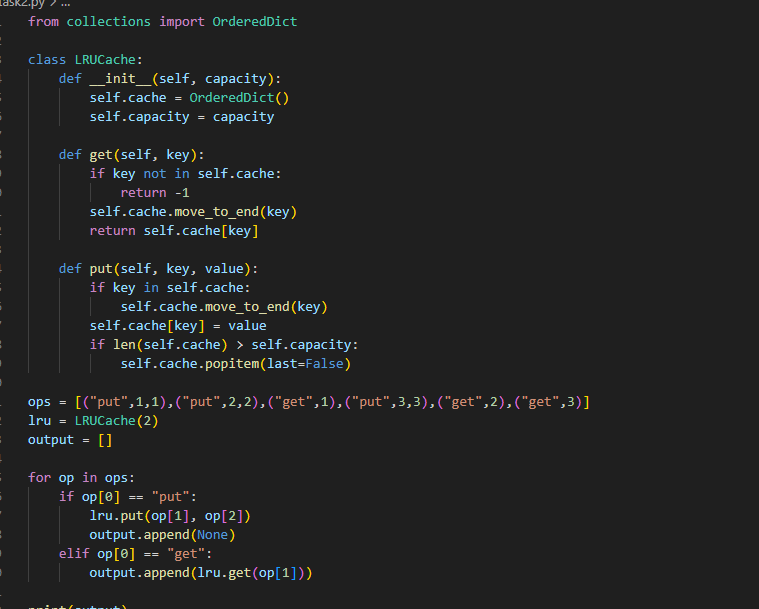
Sample Input  
ops=[("put",1,1),("put",2,2),("get",1),("put",3,3),("get",2),("get",3)]  
Sample Output  
[None, None, 1, None, -1, 3]  
Acceptance Criteria: Correct eviction

PROMPT:

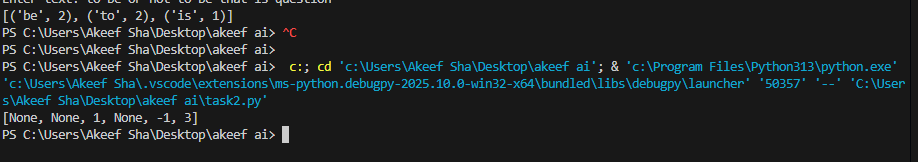
Implement an LRU (Least Recently Used) Cache using Python with a fixed capacity of 2.

* The cache must support two operations:
  + PUT(key, value) — Insert or update a key-value pair.
  + get(key) — Retrieve the value of the key if present, otherwise return -1.
* When capacity is exceeded, evict the least recently used item.
* Use OrderedDict to maintain access order.
* Return a list of results corresponding to the operations. For put operations, append None. For get, append the return value.

**CODE:**



OUTPUT:



OBSERVATION:

* The program correctly maintains the order of usage in the LRU cache.
* After inserting keys 1 and 2, accessing key 1 makes it most recently used.
* Inserting key 3 evicts key 2 as it is the least recently used.
* Trying to get key 2 returns -1 (since it was evicted).
* The cache correctly returns values for keys that are still in the cache (1 and 3).
* The output is as expected and demonstrates proper LRU behavior with deterministic and correct eviction.