NAME: T.SAI TANUJ

HALLTICKET NUMBER:2403A52413

BATCH:15

TASK-1

Task:

Top-3 words by frequency; tie-break lexicographically

Sample Input :

to be or not to be that is the question

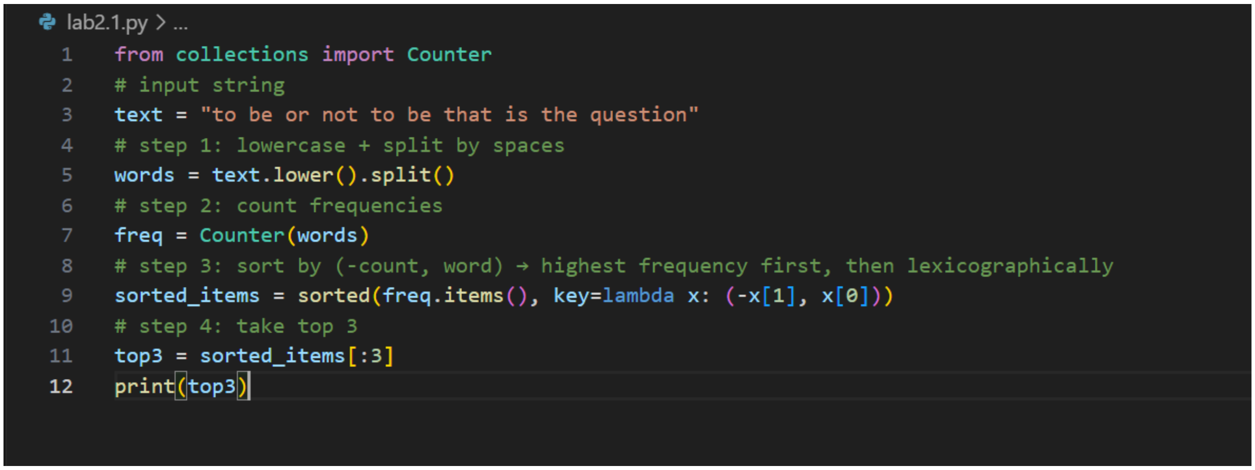
Acceptance Criteria:

Tie-breaking lexicographically

Prompt:

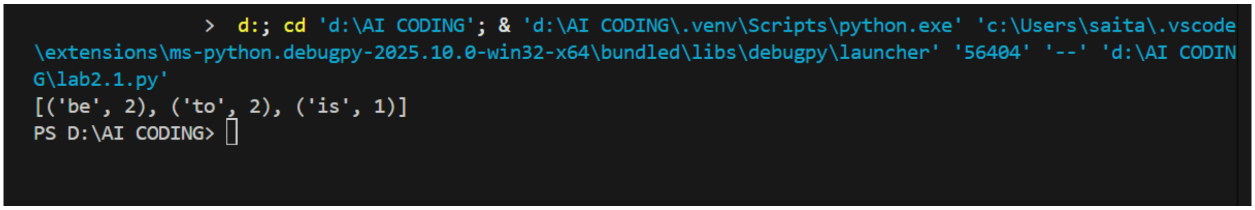
Write a Python program to read a sentence, count word frequencies, and print the top-3 words with their counts.
Use collections.Counter and sorting.
Break ties lexicographically (alphabetical order).

Code:



```python
from collections import Counter
# input string
text = "to be or not to be that is the question"
# step 1: lowercase + split by spaces
words = text.lower().split()
# step 2: count frequencies
freq = Counter(words)
# step 3: sort by (-count, word) → highest frequency first, then lexicographically
sorted_items = sorted(freq.items(), key=lambda x: (-x[1], x[0]))
# step 4: take top 3
top3 = sorted_items[:3]
print(top3)
```

Output:



```
         > d:; cd 'd:\AI CODING'; & 'd:\AI CODING\.venv\Scripts\python.exe' 'c:\Users\saita\.vscode
\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '56404' '--' 'd:\AI CODIN
G\lab2.1.py'
[('be', 2), ('to', 2), ('is', 1)]
PS D:\AI CODING> 
```

OBSERVATION:

1. The program converts the input text into lowercase and splits it into words.

2.It counts word frequencies using Counter.

3. Sorting is done by **frequency (descending)** and then **alphabetically** for tie-breaking.

4. Finally, the top-3 words are selected.

5. For the given input "to be or not to be that is the question", the words **"be"** and **"to"** both occur 2 times. Since there is a tie, they are arranged alphabetically → "be" comes before "to".

TASK-2

Task:

Implement capacity=2 LRU with get/put

Sample Input :

ops=[("put",1,1),("put",2,2),("get",1),("put",3,3),("get",2),("get",3)]

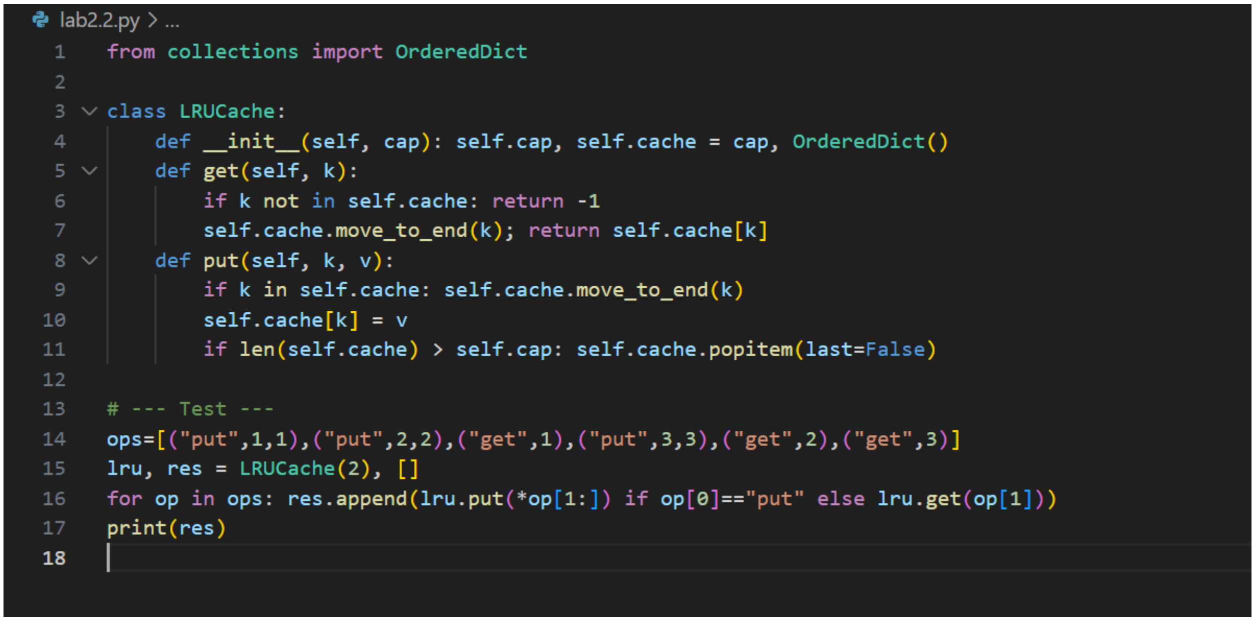
Acceptance Criteria:

Correct eviction

Prompt:

Write a Python program to implement an LRU (Least Recently Used) cache using OrderedDict.
Capacity = 2. Support get(key) and put(key, value) operations.
On get, return the value if present, otherwise -1.
On put, if the cache is full, evict the least recently used item.
Demonstrate the program with the operation sequence.

CODE:



lab2.2.py > ...

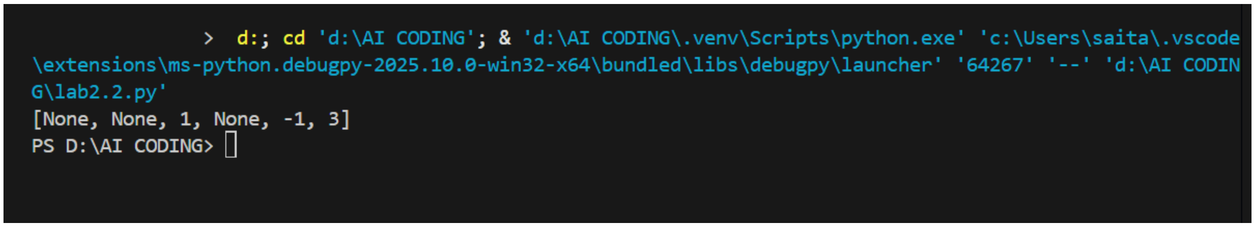
```
from collections import OrderedDict

class LRUCache:
    def __init__(self, cap): self.cap, self.cache = cap, OrderedDict()
    def get(self, k):
        if k not in self.cache: return -1
        self.cache.move_to_end(k); return self.cache[k]
    def put(self, k, v):
        if k in self.cache: self.cache.move_to_end(k)
        self.cache[k] = v
        if len(self.cache) > self.cap: self.cache.popitem(last=False)

# --- Test ---
ops=[("put",1,1),("put",2,2),("get",1),("put",3,3),("get",2),("get",3)]
lru, res = LRUCache(2), []
for op in ops: res.append(lru.put(*op[1:]) if op[0]=="put" else lru.get(op[1]))
print(res)

```

Output:



```
                > d:; cd 'd:\AI CODING'; & 'd:\AI CODING\.venv\Scripts\python.exe' 'c:\Users\saita\.vscode
\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '64267' '--' 'd:\AI CODIN
G\lab2.2.py'
[None, None, 1, None, -1, 3]
PS D:\AI CODING> 
```

OBSERVATION:

 he program uses OrderedDict to maintain insertion order of cache entries.

 get(key) returns the value if present and moves the key to the end (most recently used).

⍰ put(key, value) inserts/updates the key and evicts the **least recently used** entry if capacity is exceeded.

⍰ For the given operations:

- Insert (1,1) and (2,2) → cache = {1:1, 2:2}
- Get(1) → returns 1 and makes 1 most recent → {2:2, 1:1}
- Put(3,3) → evicts 2 (least recently used) → {1:1, 3:3}
- Get(2) → not found → -1
- Get(3) → found → 3

⍰ Thus, the final output is [None, None, 1, None, -1, 3], proving **correct eviction policy**.