Deque in Python

Definition:

Deque, short for double-ended queue, is an optimized sequence container that provides efficient insertion and deletion of elements from both ends. It supports O(1) time complexity operations for append and pop from both ends of the deque.

Advantages:

- **Efficient Insertion and Deletion:** O(1) time complexity for append and pop operations from both ends.
- **Versatility:** Can be used as a stack (Last In, First Out) or a queue (First In, First Out) depending on the requirement.
- **Memory Efficiency:** Space-efficient compared to dynamic arrays when frequently inserting or removing elements from the front and back.

Disadvantages:

- **Random Access:** Accessing elements by index is not as efficient as with lists due to its underlying data structure.
- **Complexity of Use:** Requires understanding of its double-ended nature for optimal use in different scenarios.

Use Cases:

- **Algorithm Design:** Used in algorithms that require efficient insertion and deletion from both ends, such as breadth-first search.
- **Data Structures:** Implementation of more complex data structures like double-ended queues, stacks, or queues.
- **Real-time Data Processing:** Efficiently manage sliding windows or event streams.

Applications:

- **Graph Algorithms:** Breadth-first search (BFS) and algorithms that require bidirectional processing.
- **Sliding Window Problems:** Manage a window of elements with efficient dynamic resizing.

- **Job Scheduling:** Handle jobs with varying priorities or time constraints.

Time and Space Complexity:

- **Time Complexity:**
    - Append and pop operations: O(1)
    - Insert and delete operations from arbitrary positions: O(n)
- **Space Complexity:** O(n)

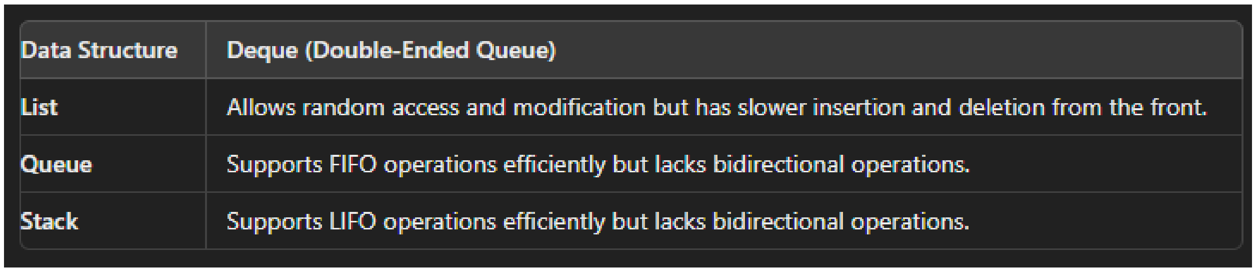
Methods:

- `append(item)`: Adds an item to the right end of the deque.
- `appendleft(item)`: Adds an item to the left end of the deque.
- `pop()`: Removes and returns the item from the right end of the deque.
- `popleft()`: Removes and returns the item from the left end of the deque.
- `clear()`: Removes all elements from the deque.
- `count(item)`: Returns the number of occurrences of `item` in the deque.
- `extend(iterable)`: Extends the deque by appending elements from the iterable to the right.
- `extendleft(iterable)`: Extends the deque by appending elements from the iterable to the left.
- `rotate(n)`: Rotates the deque `n` steps to the right (if `n` is positive) or left (if `n` is negative).

Operations:

- **Insertion:** Add an element to the front or back of the deque.
- **Deletion:** Remove an element from the front or back of the deque.
- **Access:** Retrieve elements from the front or back of the deque.
- **Manipulation:** Rotate or extend the deque based on specific requirements.

Comparison with Other Data Structures:



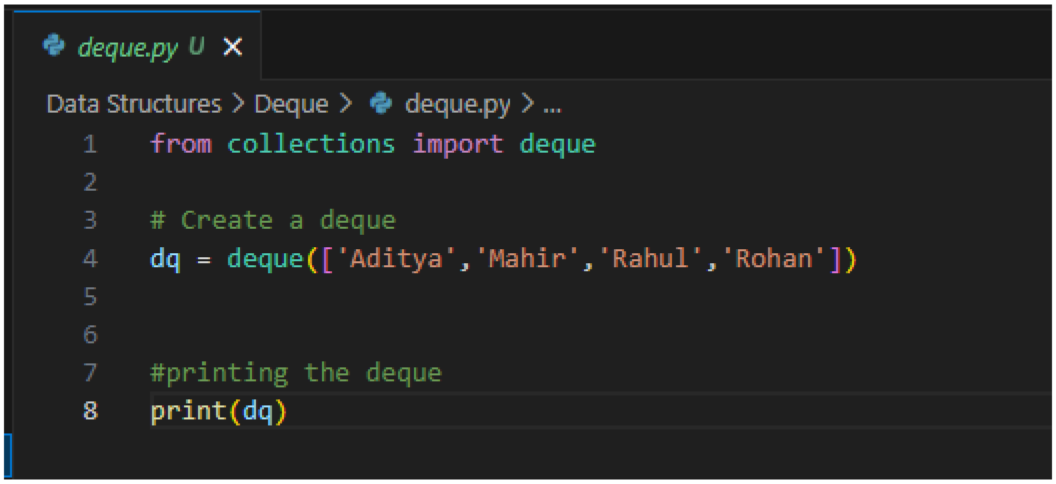
Data Structure	Deque (Double-Ended Queue)
List	Allows random access and modification but has slower insertion and deletion from the front.
Queue	Supports FIFO operations efficiently but lacks bidirectional operations.
Stack	Supports LIFO operations efficiently but lacks bidirectional operations.

Interesting Facts:

- The deque is implemented as a doubly-linked list under the hood in Python, providing efficient O(1) time complexity for append and pop operations from both ends.
- It's pronounced "deck," reflecting its double-ended nature.

Example Use Cases with Problem-Solving Questions:

1. **Sliding Window Maximum:**
    - Problem: Find the maximum value in every sliding window of size k in an array.
    - Solution: Use a deque to maintain indices of elements in the current window, efficiently managing sliding window updates and maximum value queries.
2. **Palindrome Checker:**
    - Problem: Determine if a given string is a palindrome using a deque to check characters from both ends.
    - Solution: Utilize a deque to compare characters starting from the outermost ends towards the center.
3. **Bidirectional Search:**
    - Problem: Implement a bidirectional search algorithm for finding the shortest path between two nodes in a graph.
    - Solution: Utilize two deques to perform BFS simultaneously from both the start and end nodes, meeting in the middle.

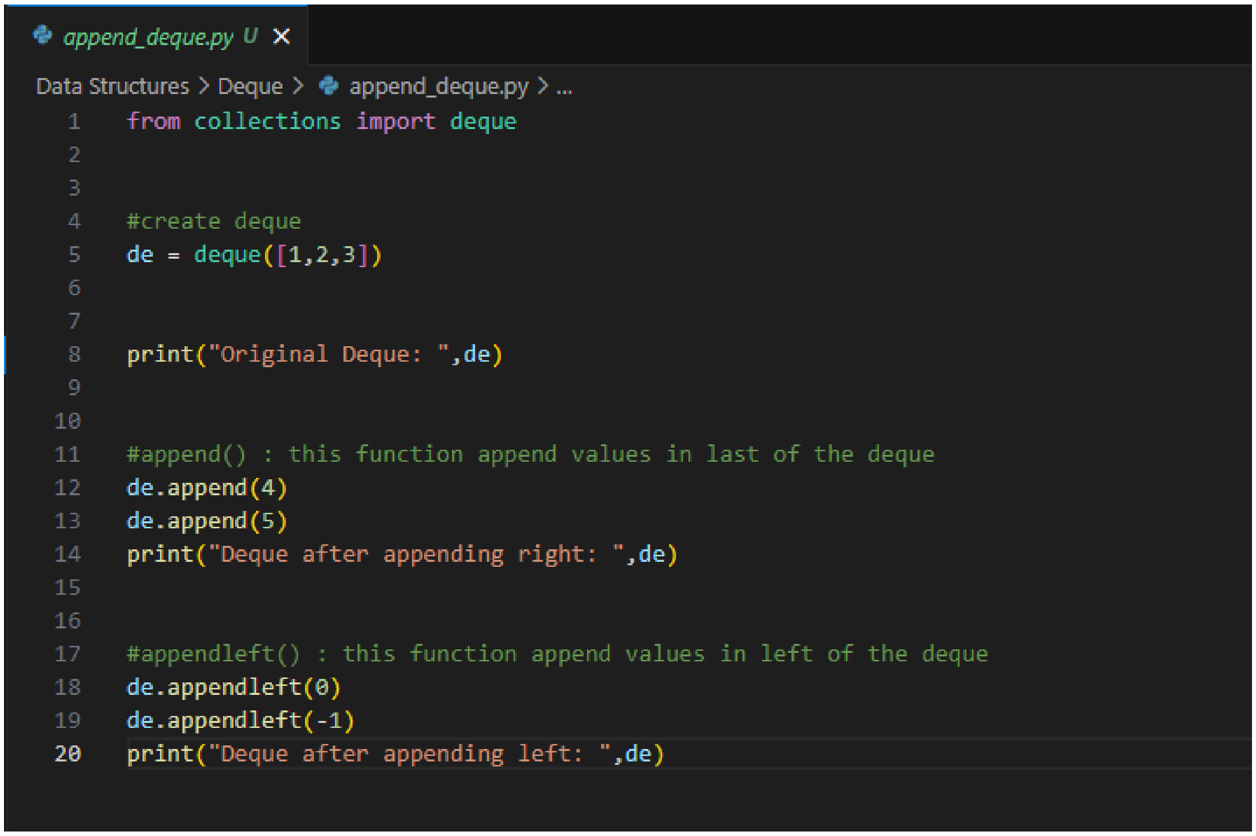

deque.py U ×

Data Structures > Deque > deque.py > ...

```python
from collections import deque

# Create a deque
dq = deque(['Aditya','Mahir','Rahul','Rohan'])


#printing the deque
print(dq)
```


append_deque.py U ×

Data Structures > Deque > append_deque.py > ...

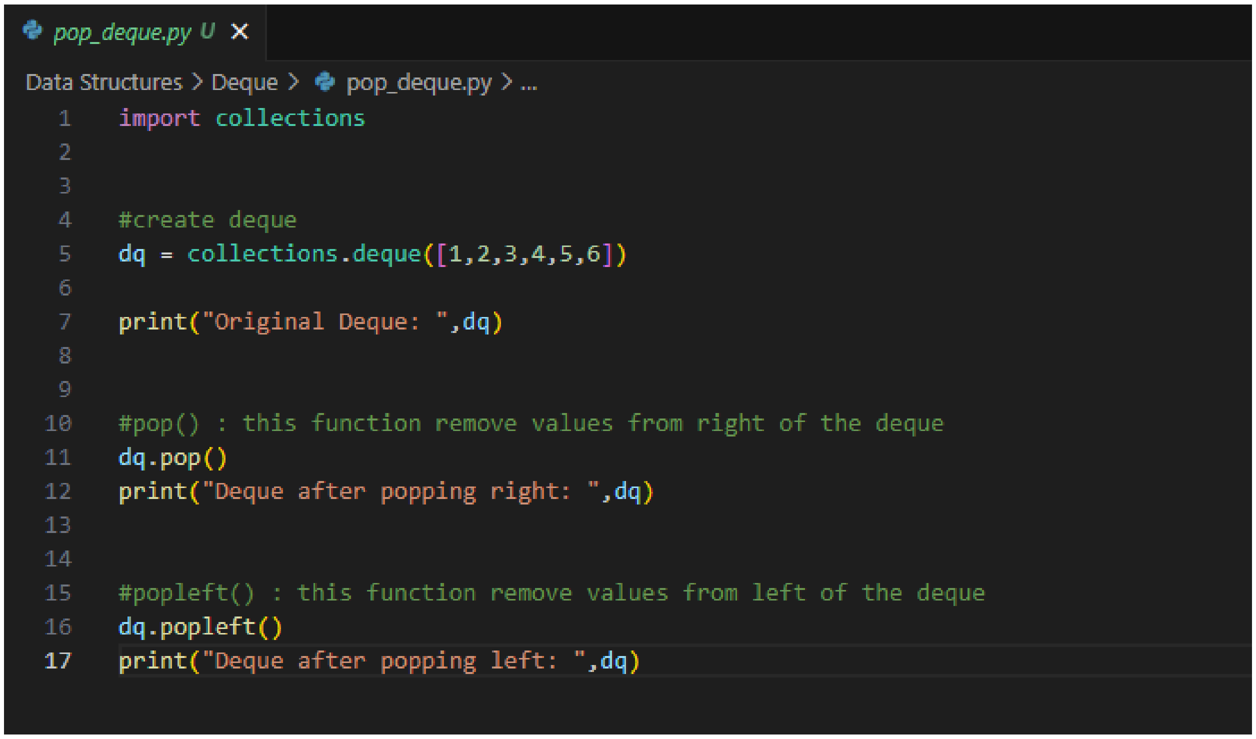
```python
from collections import deque


#create deque
de = deque([1,2,3])


print("Original Deque: ",de)


#append() : this function append values in last of the deque
de.append(4)
de.append(5)
print("Deque after appending right: ",de)


#appendleft() : this function append values in left of the deque
de.appendleft(0)
de.appendleft(-1)
print("Deque after appending left: ",de)
```



```python
import collections


#create deque
dq = collections.deque([1,2,3,4,5,6])

print("Original Deque: ",dq)


#pop() : this function remove values from right of the deque
dq.pop()
print("Deque after popping right: ",dq)


#popleft() : this function remove values from left of the deque
dq.popleft()
print("Deque after popping left: ",dq)
```