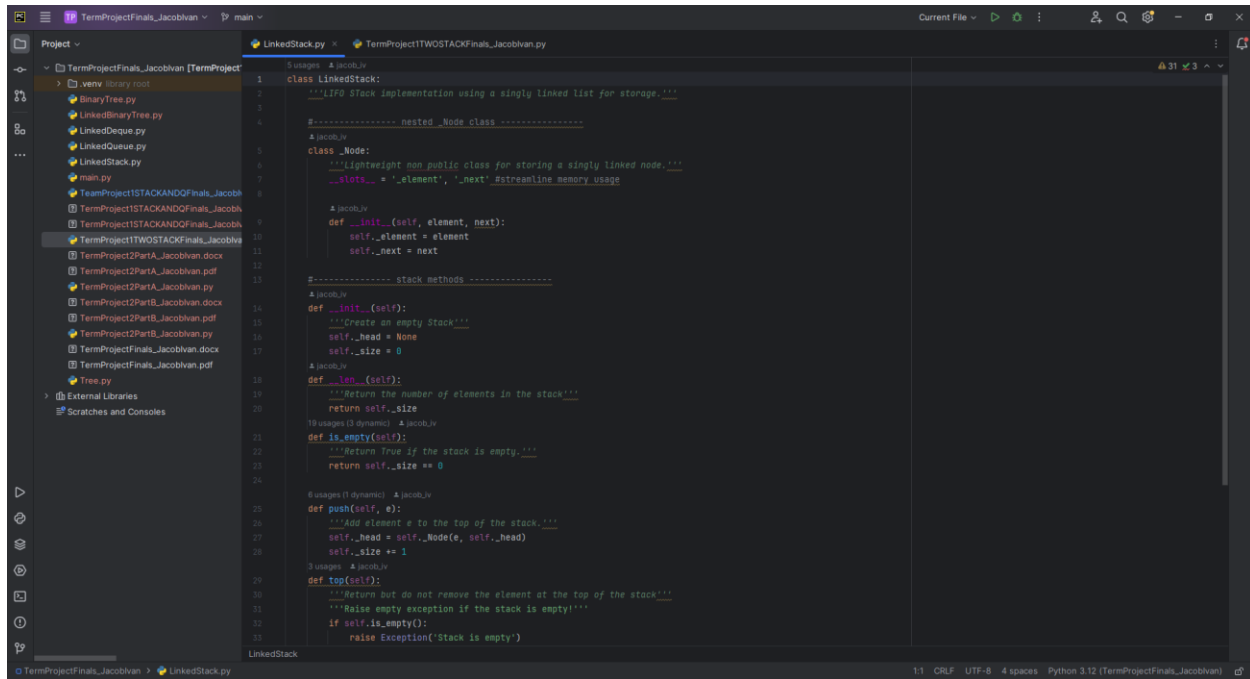
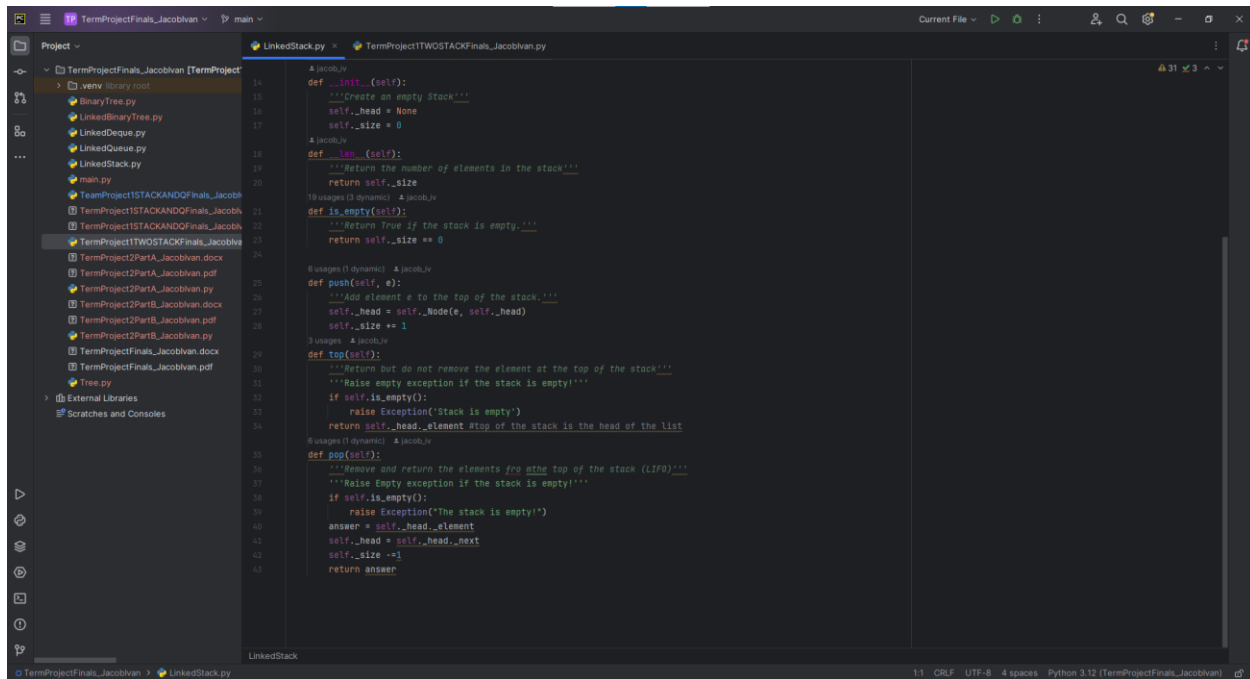


CODE:

LINKEDSTACK

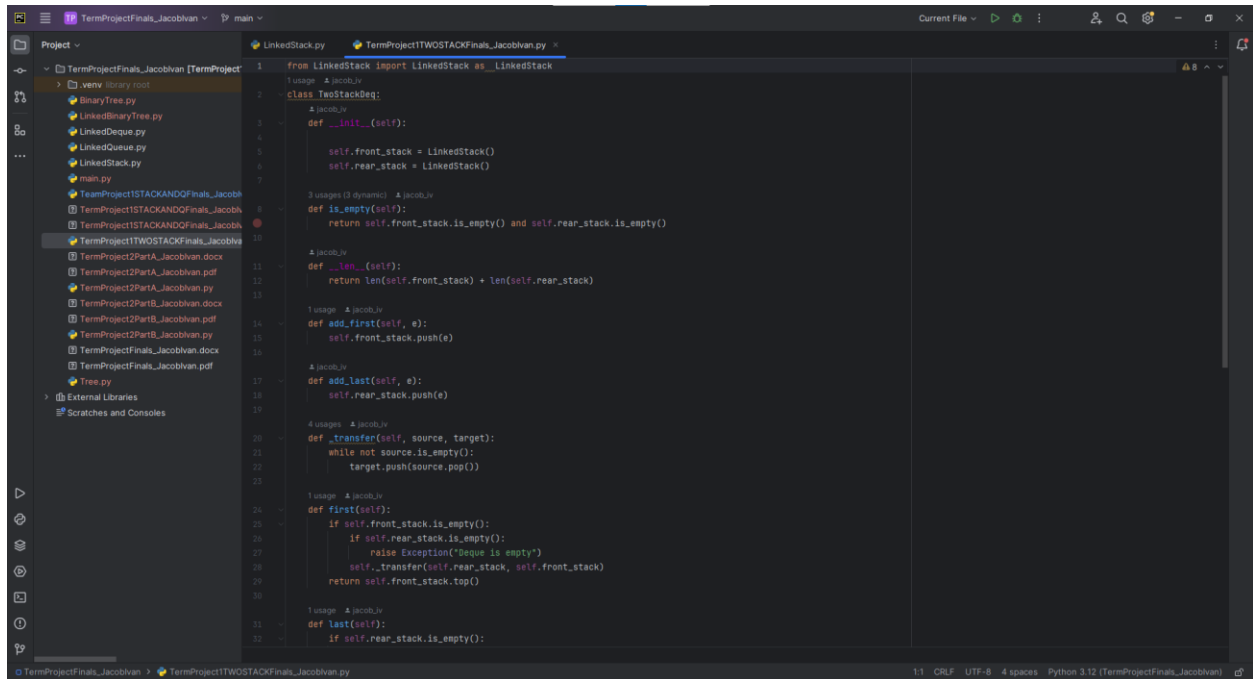


```
1 class LinkedStack:
2     """LIFO Stack implementation using a singly linked list for storage."""
3     #----- nested _Node class -----
4     class _Node:
5         """lightweight non public class for storing a singly linked node"""
6         __slots__ = 'element', 'next' #streamline memory usage
7
8         def __init__(self, element, next):
9             self.element = element
10            self.next = next
11
12    #----- stack methods -----
13    def __init__(self):
14        """Create an empty Stack"""
15        self._head = None
16        self._size = 0
17
18    def __len__(self):
19        """Return the number of elements in the stack"""
20        return self._size
21
22    def is_empty(self):
23        """Return True if the stack is empty."""
24        return self._size == 0
25
26    def push(self, e):
27        """Add element e to the top of the stack"""
28        self._head = self._Node(e, self._head)
29        self._size += 1
30
31    def top(self):
32        """Return but do not remove the element at the top of the stack"""
33        """Raise empty exception if the stack is empty"""
34        if self.is_empty():
35            raise Exception('Stack is empty')
```



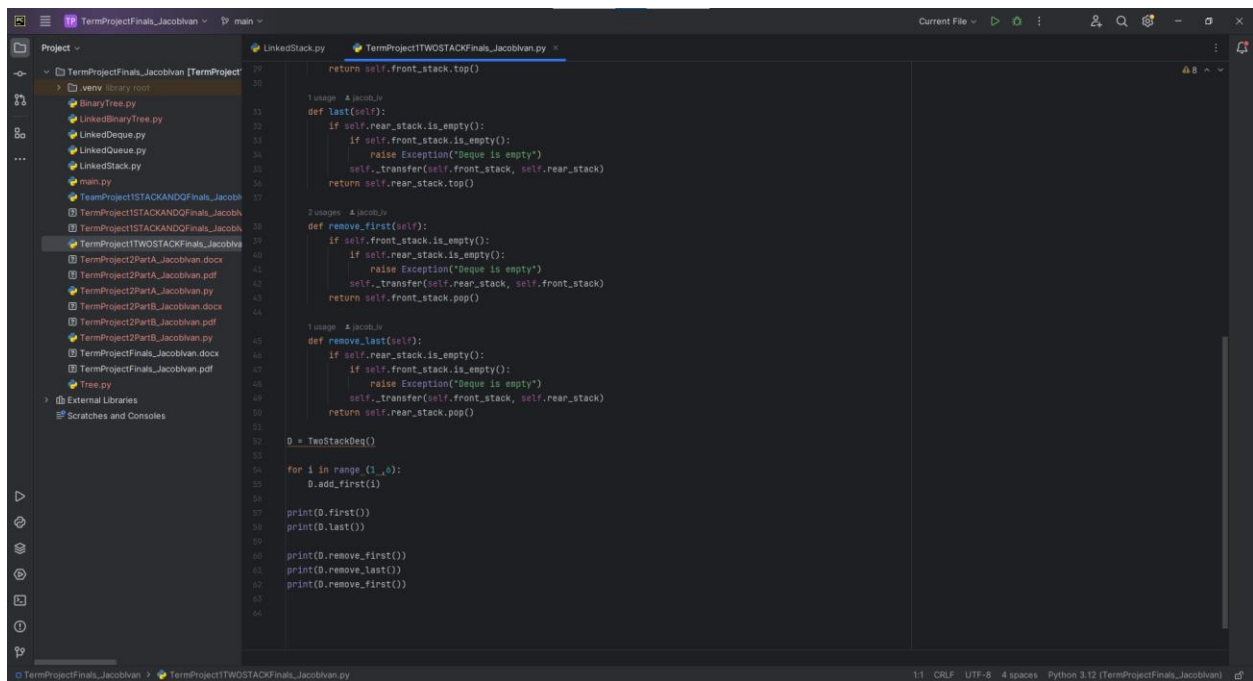
```
14 def __init__(self):
15     """Create an empty Stack"""
16     self._head = None
17     self._size = 0
18
19     def __len__(self):
20         """Return the number of elements in the stack"""
21         return self._size
22
23     def is_empty(self):
24         """Return True if the stack is empty."""
25         return self._size == 0
26
27     def push(self, e):
28         """Add element e to the top of the stack"""
29         self._head = self._Node(e, self._head)
30         self._size += 1
31
32     def top(self):
33         """Return but do not remove the element at the top of the stack"""
34         """Raise empty exception if the stack is empty"""
35         if self.is_empty():
36             raise Exception('Stack is empty')
37         return self._head._element #top of the stack is the head of the list
38
39     def pop(self):
40         """Remove and return the elements from the top of the stack (LIFO)"""
41         """Raise Empty exception if the stack is empty"""
42         if self.is_empty():
43             raise Exception('The stack is empty!')
44         answer = self._head._element
45         self._head = self._head._next
46         self._size -= 1
47         return answer
```

MAINCODE



The screenshot shows a code editor with a project named 'TermProjectFinals_Jacobvian'. The file 'TermProject1TWOSTACKFinals_Jacobvian.py' is open. The code defines a 'TwoStackDeque' class that uses two 'LinkedList' objects to implement a deque. The class methods include: 'init' (initializes front_stack and rear_stack), 'is_empty' (checks if both stacks are empty), 'len' (returns the total number of elements), 'add_first' (adds to the front_stack), 'add_last' (adds to the rear_stack), 'transfer' (moves elements from one stack to the other), 'first' (returns the first element), and 'last' (returns the last element). The 'first' method uses a 'transfer' call to move elements from the rear_stack to the front_stack if the front_stack is empty. The 'last' method uses a 'transfer' call to move elements from the front_stack to the rear_stack if the rear_stack is empty.

```
1 from LinkedList import LinkedList as _LinkedList
2 class TwoStackDeque:
3     def __init__(self):
4         self.front_stack = _LinkedList()
5         self.rear_stack = _LinkedList()
6
7     def is_empty(self):
8         return self.front_stack.is_empty() and self.rear_stack.is_empty()
9
10    def len(self):
11        return len(self.front_stack) + len(self.rear_stack)
12
13    def add_first(self, e):
14        self.front_stack.push(e)
15
16    def add_last(self, e):
17        self.rear_stack.push(e)
18
19    def transfer(self, source, target):
20        while not source.is_empty():
21            target.push(source.pop())
22
23    def first(self):
24        if self.front_stack.is_empty():
25            if self.rear_stack.is_empty():
26                raise Exception("Deque is empty")
27            self.transfer(self.rear_stack, self.front_stack)
28        return self.front_stack.top()
29
30    def last(self):
31        if self.rear_stack.is_empty():
32            if self.front_stack.is_empty():
33                raise Exception("Deque is empty")
34            self.transfer(self.front_stack, self.rear_stack)
35        return self.rear_stack.top()
```



The screenshot shows the same code editor with the 'TermProject1TWOSTACKFinals_Jacobvian.py' file. The code continues with the main execution logic. It creates an instance of 'TwoStackDeque' named 'D'. It then performs a series of operations: adding elements to the front and rear, and removing elements from both ends. The operations are: 'D.add_first(1)', 'D.add_first(2)', 'D.add_last(3)', 'D.add_last(4)', 'D.remove_first()', 'D.remove_last()', 'D.remove_first()', and 'D.remove_last()'. The code also includes print statements to display the results of these operations.

```
36 D = TwoStackDeque()
37 for i in range(1, 5):
38     D.add_first(i)
39
40 print(D.first())
41 print(D.last())
42
43 print(D.remove_first())
44 print(D.remove_last())
45 print(D.remove_first())
46 print(D.remove_last())
```

OUTPUT:

