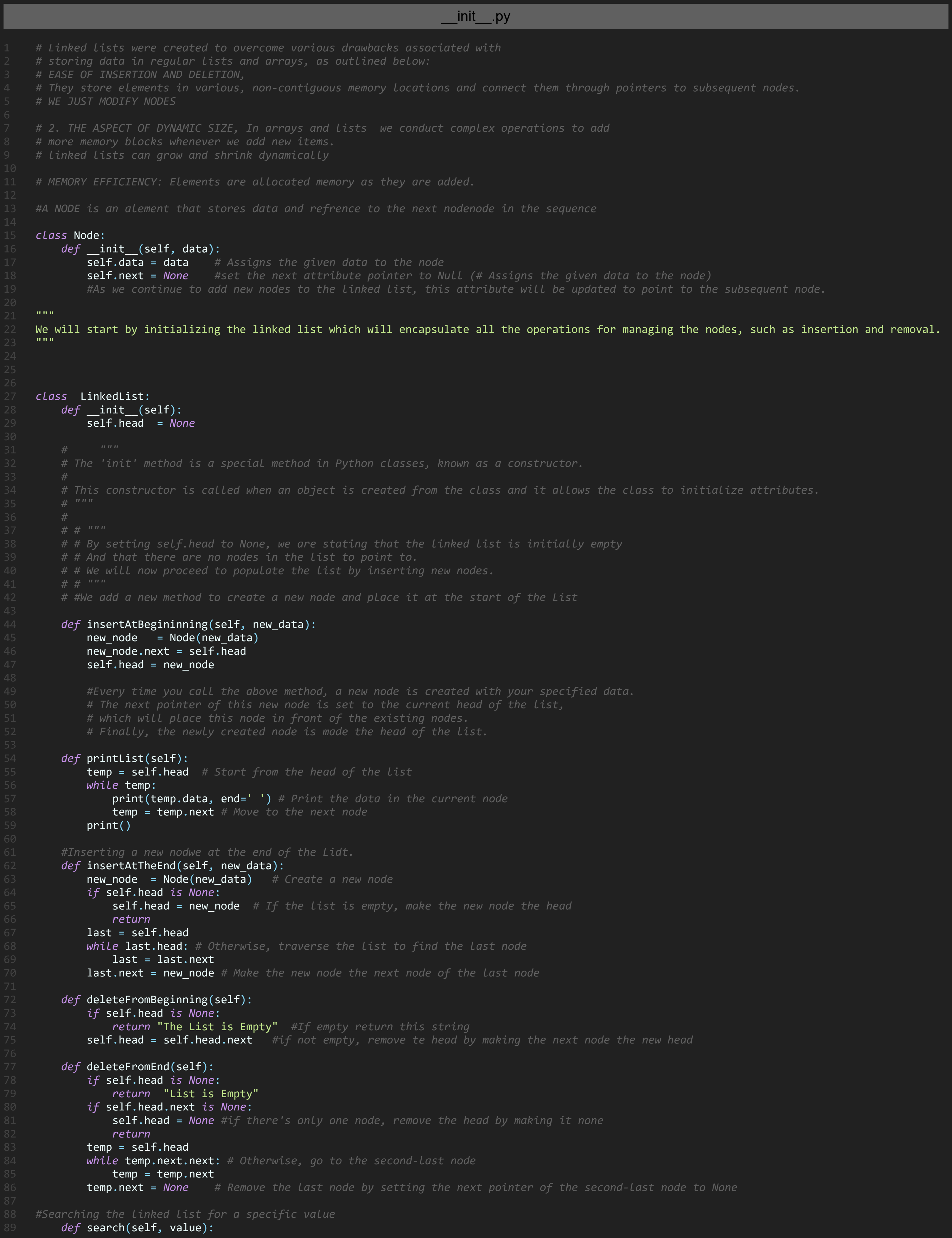

__init__.py

```python
# Linked lists were created to overcome various drawbacks associated with
# storing data in regular lists and arrays, as outlined below:
# EASE OF INSERTION AND DELETION,
# They store elements in various, non-contiguous memory locations and connect them through pointers to subsequent nodes.
# WE JUST MODIFY NODES

# 2. THE ASPECT OF DYNAMIC SIZE, In arrays and lists  we conduct complex operations to add
# more memory blocks whenever we add new items.
# linked lists can grow and shrink dynamically

# MEMORY EFFICIENCY: Elements are allocated memory as they are added.

#A NODE is an alement that stores data and refrence to the next nodenode in the sequence

class Node:
    def __init__(self, data):
        self.data = data    # Assigns the given data to the node
        self.next = None    #set the next attribute pointer to Null (# Assigns the given data to the node)
        #As we continue to add new nodes to the linked list, this attribute will be updated to point to the subsequent node.

"""
We will start by initializing the linked list which will encapsulate all the operations for managing the nodes, such as insertion and removal.
"""



class  LinkedList:
    def __init__(self):
        self.head  = None

    #     """
    # The 'init' method is a special method in Python classes, known as a constructor.
    #
    # This constructor is called when an object is created from the class and it allows the class to initialize attributes.
    # """
    #
    # # """
    # # By setting self.head to None, we are stating that the linked list is initially empty
    # # And that there are no nodes in the list to point to.
    # # We will now proceed to populate the list by inserting new nodes.
    # # """
    # #We add a new method to create a new node and place it at the start of the List

    def insertAtBegininning(self, new_data):
        new_node   = Node(new_data)
        new_node.next = self.head
        self.head = new_node

        #Every time you call the above method, a new node is created with your specified data.
        # The next pointer of this new node is set to the current head of the list,
        # which will place this node in front of the existing nodes.
        # Finally, the newly created node is made the head of the list.

    def printList(self):
        temp = self.head  # Start from the head of the list
        while temp:
            print(temp.data, end=' ') # Print the data in the current node
            temp = temp.next # Move to the next node
        print()

    #Inserting a new nodwe at the end of the Lidt.
    def insertAtTheEnd(self, new_data):
        new_node  = Node(new_data)   # Create a new node
        if self.head is None:
            self.head = new_node  # If the list is empty, make the new node the head
            return
        last = self.head
        while last.head: # Otherwise, traverse the list to find the last node
            last = last.next
        last.next = new_node # Make the new node the next node of the last node

    def deleteFromBeginning(self):
        if self.head is None:
            return "The List is Empty"  #If empty return this string
        self.head = self.head.next   #if not empty, remove te head by making the next node the new head

    def deleteFromEnd(self):
        if self.head is None:
            return  "List is Empty"
        if self.head.next is None:
            self.head = None #if there's only one node, remove the head by making it none
            return
        temp = self.head
        while temp.next.next: # Otherwise, go to the second-last node
            temp = temp.next
        temp.next = None    # Remove the last node by setting the next pointer of the second-last node to None

#Searching the linked list for a specific value
    def search(self, value):
        current = self.head  # Start with the head of the list
        position = 0  # Counter to keep track of the position
        while current:  # Traverse the list
            if current.data == value:  # Compare the list's data to the search value
                return f"Value '{value}' found at position {position}"  # Print the value if a match is found
            current = current.next
            position += 1
        return f"Value '{value}' not found in the list"

if __name__ == '__main__':
    # Create a new LinkedList instance
    llist = LinkedList()

    # Insert each letter at the beginning using the method we created
    llist.insertAtBegininning('fox')
    llist.insertAtBegininning('brown')
    llist.insertAtBegininning('quick')
    llist.insertAtBegininning('the')

    # Now 'the' is the head of the list, followed by 'quick', then 'brown' and 'fox'

    # Print the list
    llist.printList()
    # Insert a word at the end
    llist.insertAtEnd('jumps')

    llist.printList()

    # Deleting nodes from the beginning and end
    llist.deleteFromBeginning()
    print("List after deletion:")
    llist.printList()
    # Search for 'quick' and 'lazy' in the list
    print(llist.search('quick'))  # Expected to find
    print(llist.search('lazy'))  # Expected not to find
```