Assignment 11 (Linked List)

> **Name: Mohammed Varaliya**

> **Roll No: 54**

Questions

1. **A method is_empty() to check if the list is empty.**
2. **A method prepend() to insert a new element at the start of the list**
3. **A method append() to insert a new element at the end of the list.**
4. **A method insert_after() to insert a new element after a specified node.**
5. **A method insert_at_sorted_pos() to insert a new element in a sorted list at sorted position.**

---

1. **A method is_empty() to check if the list is empty.**
    a. **Code:**

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

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

    def is_empty(self):
        if self.head is None:
            return True
        else:
            return False


if __name__ == "__main__":

    llist = SinglyLinkedList()

    # A method is_empty() to check if the list is empty.
    print("\nCheck whether the singly linked list is empty or not: ", llist.is_empty
())
```

```
# A method is_empty() to check if the list is empty.

def is_empty(self):
    if self.head is None:
        return True
    else:
        return False
```

1. **Explanation:**
    a. The `is_empty` method checks if the singly linked list is empty. Here's how it works:
    b. **Check Head**: It checks if the `head` of the list is `None`. If the `head` is `None`, it means the list is empty (since there are no nodes).
    c. **Return Result**: If the `head` is `None`, it returns `True`, indicating that the list is empty. Otherwise, it returns `False`, meaning the list is not empty.

b. **Output:**

```
Check whether the singly linked list is empty or not:  True
```

---

2. **A method prepend() to insert a new element at the start of the list**

a. **Code:**

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

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

    def prepend(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return

        cur_node = self.head
        self.head = new_node
        new_node.next = cur_node

    def display_list_iterative(self):
        cur_node = self.head
        while cur_node:
            print(cur_node.data, end=" -> ")
            cur_node = cur_node.next
        print("None")


if __name__ == "__main__":

    llist = SinglyLinkedList()
    llist.prepend(2)
    llist.prepend(5)
```

```
    llist.prepend(8)
    llist.prepend(1)

    # A method display() to print all elements in the list.
    llist.display_list_iterative()
```

```
# A method prepend() to insert a new element at the start of the list

def prepend(self, data):
    new_node = Node(data)
    if self.head is None:
        self.head = new_node
        return

    cur_node = self.head
    self.head = new_node
    new_node.next = cur_node
```

1. **Explanation:**
    a. The `prepend` method inserts a new node at the start of the singly linked list. Here's how it works:
    b. **Create New Node**: It first creates a new node with the given data.
    c. **Check if List is Empty**: If the list is empty (i.e., the `head` is `None`), the new node becomes the `head` of the list.
    d. **Insert at Start**: If the list is not empty, it makes the new node the `head` and points the new node's `next` to the current `head` (which is the first node in the list). This effectively inserts the new node at the start of the list.

b. **Output:**

```
1 -> 8 -> 5 -> 2 -> None
```

---

3. **A method append() to insert a new element at the end of the list.**
    a. **Code:**

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

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

    def is_empty(self):
        if self.head is None:
            return True
        else:
            return False

    def prepend(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
```

```
        cur_node = self.head
        self.head = new_node
        new_node.next = cur_node

    def append(self, data):
        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            return

        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def display_list_iterative(self):
        cur_node = self.head
        while cur_node:
            print(cur_node.data, end=" -> ")
            cur_node = cur_node.next
        print("None")


if __name__ == "__main__":

    llist = SinglyLinkedList()
    llist.append(2)
    llist.append(5)
    llist.append(8)
    llist.append(1)

    # A method display() to print all elements in the list.
    llist.display_list_iterative()

    # A method append() to insert a new element at the end of the list.
    llist.append(10)

    # A method display() to print all elements in the list after inserting node from
end.
    llist.display_list_iterative()
```

```
# A method append() to insert a new element at the end of the list.

def append(self, data):
    new_node = Node(data)

    if self.head is None:
        self.head = new_node
        return

    last_node = self.head
    while last_node.next:
        last_node = last_node.next
    last_node.next = new_node
```

1. **Explanation:**
    a. The `append` method inserts a new node at the end of the singly linked list. Here's how it works:
    b. **Create New Node**: It creates a new node with the given data.
    c. **Check if List is Empty**: If the list is empty (i.e., the `head` is `None`), the new node becomes the `head` of the list.
    d. **Traverse to Last Node**: If the list is not empty, it traverses the list to find the last node (the node whose `next` is `None`).
    e. **Insert at End**: Once the last node is found, it sets the `next` of the last node to the new node, effectively appending the new node to the end of the list.

b. **Output:**

```
0 -> 2 -> 5 -> 8 -> 1 -> None
0 -> 2 -> 5 -> 8 -> 1 -> 10 -> None
```

---

4. **A method insert_after() to insert a new element after a specified node.**
    a. **Code:**

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

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

    def is_empty(self):
        if self.head is None:
            return True
        else:
            return False

    def prepend(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return

        cur_node = self.head
        self.head = new_node
        new_node.next = cur_node

    def append(self, data):
        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            return

        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node
```

```
    def insert_after(self, key, data):
        cur_node = self.head
        while cur_node:
            if cur_node.next is None and cur_node.data == key:
                self.append(data)
                return
            elif cur_node.data == key:
                new_node = Node(data)
                nxt = cur_node.next
                new_node.next = nxt
                cur_node.next = new_node
            cur_node = cur_node.next

        if cur_node is None:
            print("Previous Node is not present in the list")
            return

    def display_list_iterative(self):
        cur_node = self.head
        while cur_node:
            print(cur_node.data, end=" -> ")
            cur_node = cur_node.next
        print("None")


if __name__ == "__main__":

    llist = SinglyLinkedList()
    llist.append(2)
    llist.append(5)
    llist.append(8)
    llist.append(1)

    # A method display() to print all elements in the list.
    llist.display_list_iterative()

    # A method is_empty() to check if the list is empty.
    print("\nCheck whether the singly linked list is empty or not: ", llist.is_empty
())

    # A method prepend() to insert a new element at the start of the list
    llist.prepend(0)

    # A method display() to print all elements in the list after inserting node from
start.
    llist.display_list_iterative()

    # A method append() to insert a new element at the end of the list.
    llist.append(10)

    # A method display() to print all elements in the list after inserting node from
end.
    llist.display_list_iterative()

    # A method insert_after() to insert a new element after a specified node.
    llist.insert_after(5, 6)
```

```
    # A method display() to print all elements in the list after inserting node afte
r other node.
    llist.display_list_iterative()
```

```
# A method insert_after() to insert a new element after a specified node.

def insert_after(self, key, data):
    cur_node = self.head
    while cur_node:
        if cur_node.next is None and cur_node.data == key:
            self.append(data)
            return
        elif cur_node.data == key:
            new_node = Node(data)
            nxt = cur_node.next
            new_node.next = nxt
            cur_node.next = new_node
            return
        cur_node = cur_node.next

    if cur_node is None:
        print("Previous Node is not present in the list")
        return
```

1. **Explanation:**
    a. The `insert_after()` method inserts a new node after a specified node (identified by the `key`) in the singly linked list. Here's how it works:
    b. **Traverse List**: It traverses through the list, checking each node's data.
    c. **Find Node with Key**: If a node with the given `key` is found, a new node with the specified `data` is created and inserted after it by updating the `next` pointers.
    d. **Append at End**: If the `key` is found at the last node, the new node is appended at the end of the list using the `append()` method.
    e. **Edge Case**: If the `key` is not found in the list, an error message is displayed.

b. **Output:**

```
0 -> 2 -> 5 -> 8 -> 1 -> None
0 -> 2 -> 5 -> 8 -> 1 -> 10 -> None
0 -> 2 -> 5 -> 6 -> 8 -> 1 -> 10 -> None
```

---

5. **A method insert_at_sorted_pos() to insert a new element in a sorted list at sorted position.**

a. **Code:**

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

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

    def is_empty(self):
```

```
        if self.head is None:
            return True
        else:
            return False

    def prepend(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return

        cur_node = self.head
        self.head = new_node
        new_node.next = cur_node

    def append(self, data):
        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            return

        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def insert_at_sorted_pos(self, data):
        new_node = Node(data)

        # Case 1: If the list is empty, insert the new node as the head
        if self.head is None:
            self.head = new_node
            return

        # Case 2: If the new node should be inserted before the head (sorted order)
        if self.head.data >= data:
            new_node.next = self.head
            self.head = new_node
            return

        cur_node = self.head
        while cur_node.next and cur_node.next.data < data:
            cur_node = cur_node.next

        new_node.next = cur_node.next
        cur_node.next = new_node

    def display_list_iterative(self):
        cur_node = self.head
        while cur_node:
            print(cur_node.data, end=" -> ")
            cur_node = cur_node.next
        print("None")


if __name__ == "__main__":
```

```
    llist = SinglyLinkedList()
    llist.append(1)
    llist.append(5)
    llist.append(10)
    llist.append(15)
    llist.append(17)

    # A method display() to print all elements in the list.
    llist.display_list_iterative()

    # A method insert_at_sorted_pos() to insert a new element in a sorted list at so
rted position.
    llist.insert_at_sorted_pos(16)

    # A method display() to print all elements in the list after inserting node at a
sorted position.
    llist.display_list_iterative()
```

```
# A method insert_at_sorted_pos() to insert a new element in a sorted list at sorted
position.

def insert_at_sorted_pos(self, data):
    new_node = Node(data)

    # Case 1: If the list is empty, insert the new node as the head
    if self.head is None:
        self.head = new_node
        return

    # Case 2: If the new node should be inserted before the head (sorted order)
    if self.head.data >= data:
        new_node.next = self.head
        self.head = new_node
        return

    cur_node = self.head
    while cur_node.next and cur_node.next.data < data:
        cur_node = cur_node.next

    new_node.next = cur_node.next
    cur_node.next = new_node
```

1. **Explanation:**
    a. The `insert_at_sorted_pos()` method inserts a new element into a sorted singly linked list at the correct sorted position. Here's how it works:
    b. **Create New Node**: It creates a new node with the provided data.
    c. **Handle Empty List**: If the list is empty, the new node becomes the head of the list.
    d. **Insert Before Head**: If the new node's data is smaller than the head's data, it is inserted at the beginning of the list.
    e. **Traverse to Correct Position**: If the new node's data is larger, it traverses the list until it finds the correct spot (where the next node's data is greater than the new node's data).
    f. **Insert Node**: It inserts the new node at the found position by adjusting the `next` pointers accordingly.

b. **Output:**

```
1 -> 5 -> 10 -> 15 -> 17 -> None
1 -> 5 -> 10 -> 15 -> 16 -> 17 -> None
```

---

1. **GitHub Code Explanation with example walkthrough.**

MCA-Coursework/SEM-1/Data-Structure-Using-Python/Assignments/Assignment-11 at main · Mohammedvaraliya/MCA-Coursework
Contribute to Mohammedvaraliya/MCA-Coursework development by creating an account on GitHub.

https://github.com/Mohammedvaraliya/MCA-Coursework/tree/main/SEM-1/Data-Structure-Using-Python/Assignments/Assignment-11
