Assignment 12 (Linked List)

> **Name: Mohammed Varaliya**

> **Roll No: 54**

Questions

1. **Checking if the linked list is sorted.**
2. **Removing duplicates from sorted linked list.**
3. **Reversing a linked list.**

---

1. **Checking if the linked list is sorted.**
    a. **Code:**

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

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

    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 check_sorted(self):
        cur_node = self.head
        prev_data = float('-inf') # Use negative infinity as initial value
        while cur_node:
            if cur_node.data >= prev_data:
                prev_data = cur_node.data
```

```
                cur_node = cur_node.next
            elif cur_node.data < prev_data:
                return False
        return True

    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(10)
    llist.append(20)
    llist.append(20)
    llist.append(30)
    llist.append(40)

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

    # Cheking whether the list is sorted or not
    print(llist.check_sorted())
```

```
# Checking if the linked list is sorted.

def check_sorted(self):
    cur_node = self.head
    prev_data = float('-inf') # Use negative infinity as initial value
    while cur_node:
        if cur_node.data >= prev_data:
            prev_data = cur_node.data
            cur_node = cur_node.next
        elif cur_node.data < prev_data:
            return False
    return True
```

1. **Explanation:**

PAGE No.
DATE / /

Assignment 12 19-Oct-2024

1. Checking if the linked list is sorted.

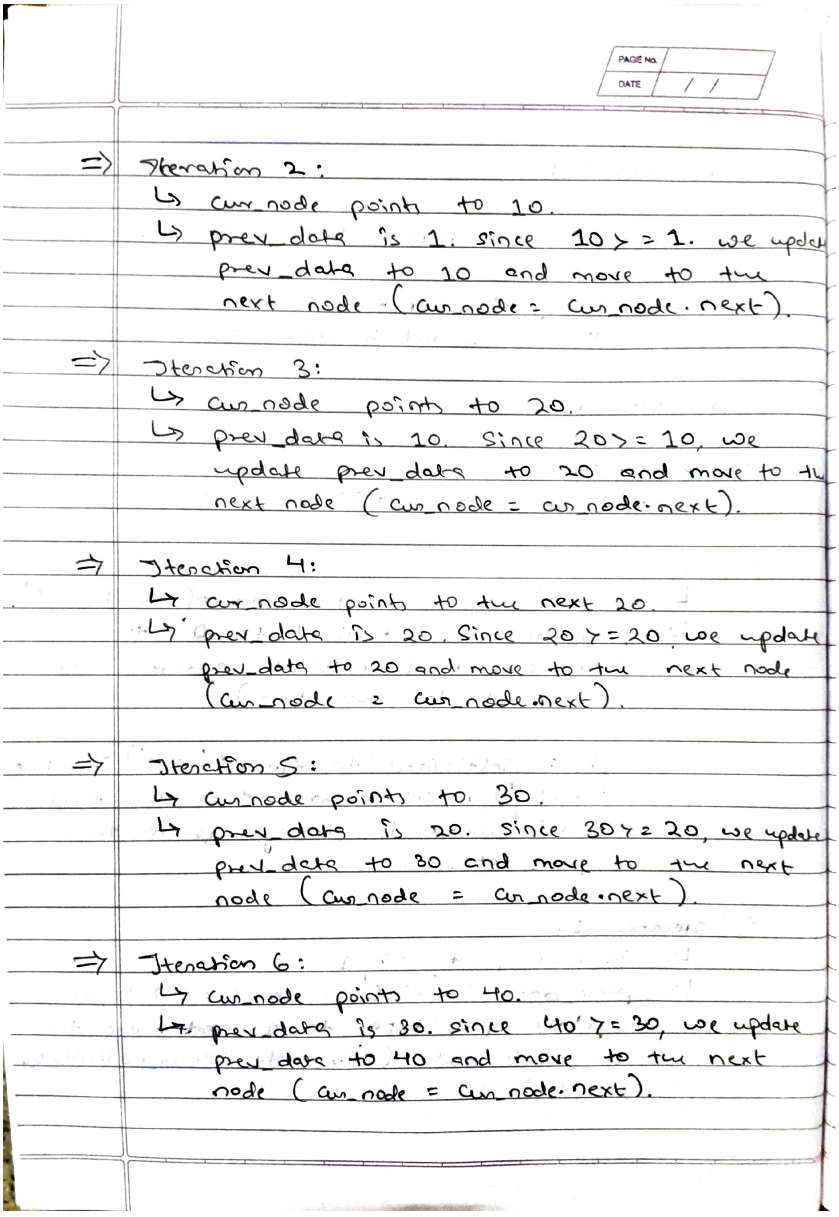
The goal of this function is to check if the linked list is sorted in non-decreasing order.

Walkthrough:

⇒ Initial list:
1 → 10 → 20 → 20 → 30 → 40 → None.

⇒ Variables:
↳ 'cur_node' starts at the head of the list (1).
↳ 'prev_data' is initialized to 0 (to make sure the first node is greater than or equal to prev_data).
or
It is initialized to float('-inf') (negative infinity), which ensures that the first node's value is always greater than prev_data.

⇒ Iteration 1:
↳ cur_node points to 1.
↳ prev_data is '-inf', since 1 >= -inf, we update prev_data to 1 and move to the next node ('cur_node = cur_node.next).



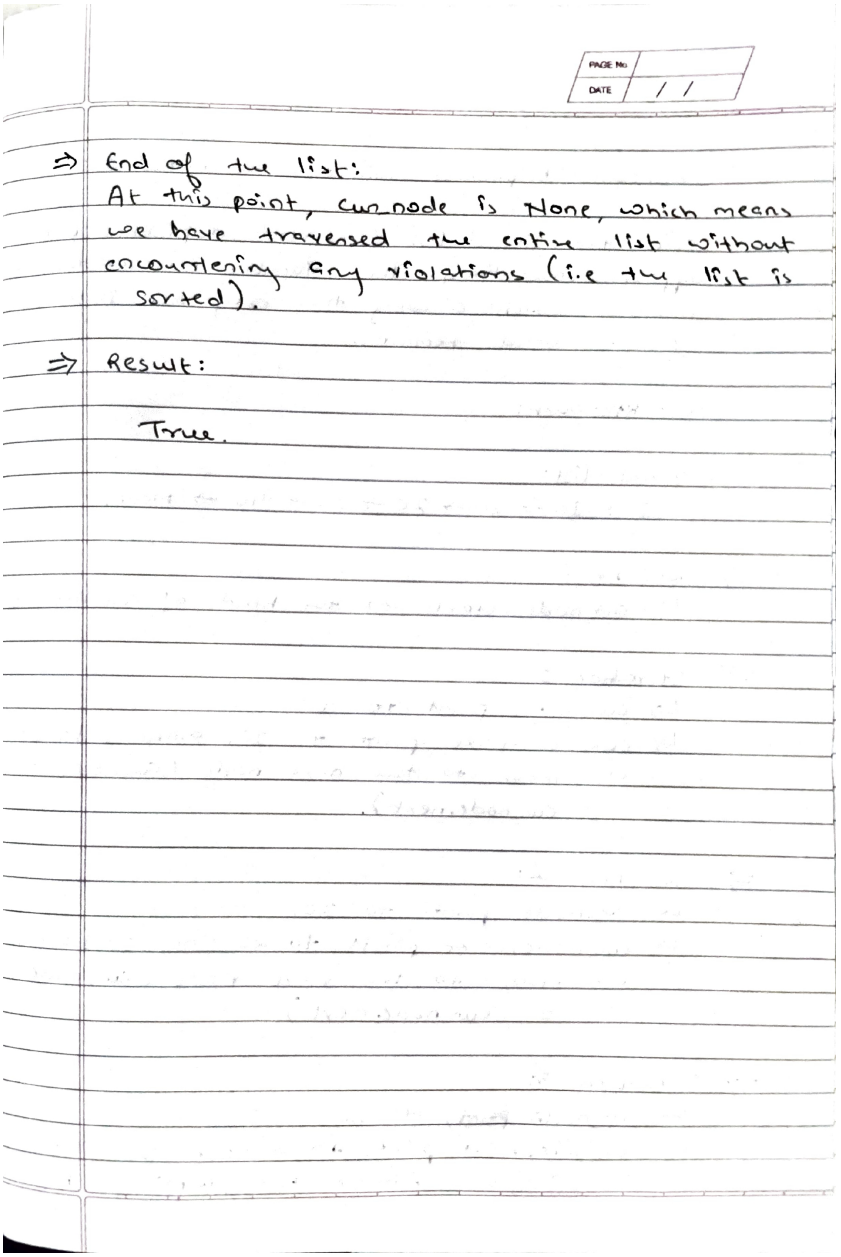
⇒ Iteration 2:
↳ cur_node points to 10.
↳ prev_data is 1. since 10 >= 1. we update prev_data to 10 and move to the next node (cur_node = cur_node.next).

⇒ Iteration 3:
↳ cur_node points to 20.
↳ prev_data is 10. Since 20 >= 10, we update prev_data to 20 and move to the next node (cur_node = cur_node.next).

⇒ Iteration 4:
↳ cur_node points to the next 20.
↳ prev_data is 20. Since 20 >= 20, we update prev_data to 20 and move to the next node (cur_node = cur_node.next).

⇒ Iteration 5:
↳ cur_node points to 30.
↳ prev_data is 20. since 30 >= 20, we update prev_data to 30 and move to the next node (cur_node = cur_node.next).

⇒ Iteration 6:
↳ cur_node points to 40.
↳ prev_data is 30. since 40 >= 30, we update prev_data to 40 and move to the next node (cur_node = cur_node.next).



b. **Output:**

```
1 -> 10 -> 20 -> 20 -> 30 -> 40 -> None
True
```

---

2. **Removing duplicates from sorted linked list.**

a. **Code:**

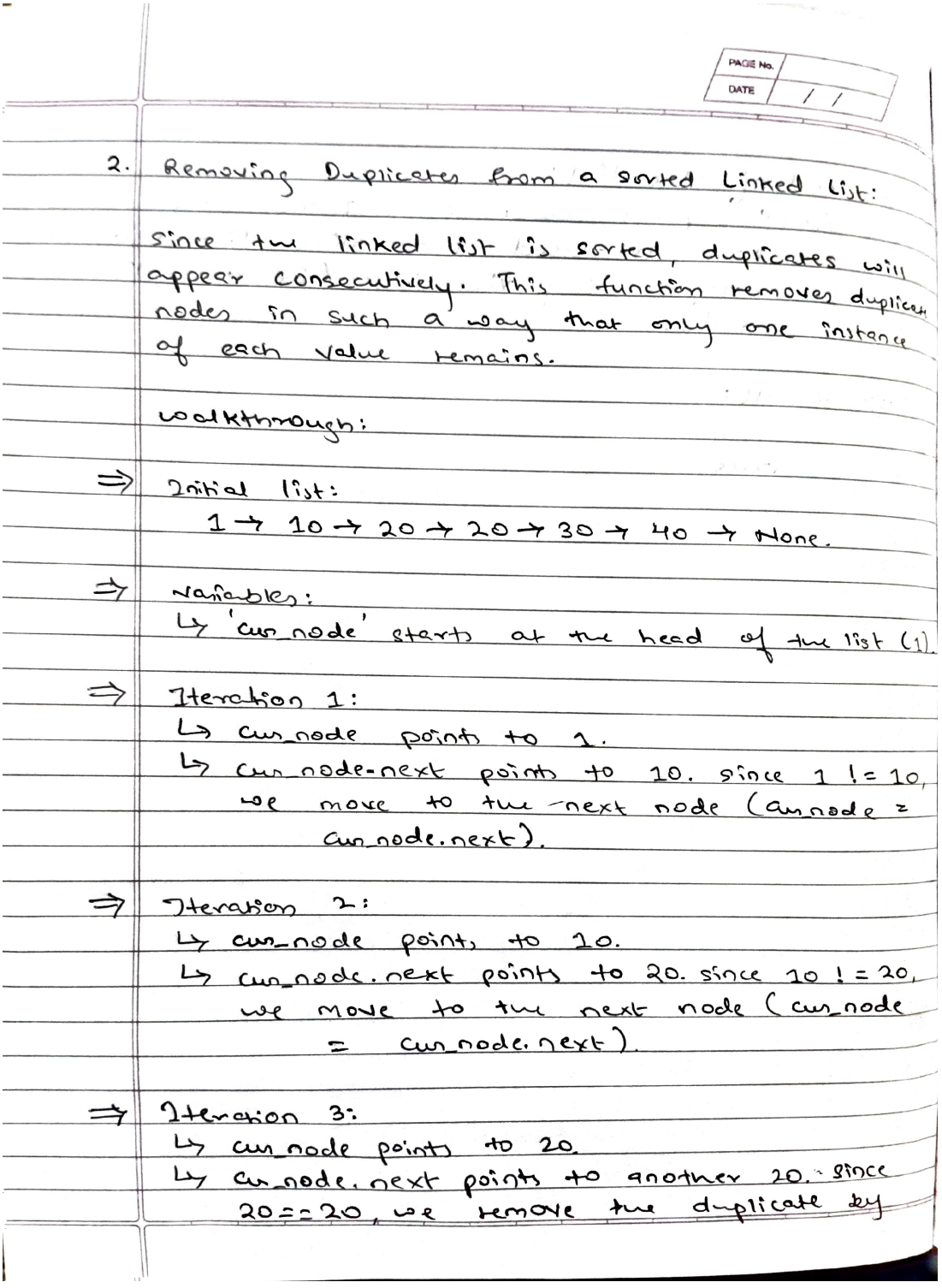
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

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

```
    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 removing_duplicates_from_sorted(self):
        cur_node = self.head
        while cur_node and cur_node.next:
            if cur_node.data == cur_node.next.data:
                cur_node.next = cur_node.next.next
            else:
                cur_node = cur_node.next

    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(10)
    llist.append(20)
    llist.append(20)
    llist.append(30)
    llist.append(40)

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

    # Removing duplicate nodes from sorted list
    llist.removing_duplicates_from_sorted()

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

```
# Removing duplicates from sorted linked list.

def removing_duplicates_from_sorted(self):
    cur_node = self.head
    while cur_node and cur_node.next:
        if cur_node.data == cur_node.next.data:
            cur_node.next = cur_node.next.next
```

```
        else:
            cur_node = cur_node.next
```

1. **Explanation:**



2. Removing Duplicates from a sorted Linked List:

Since the linked list is sorted, duplicates will appear consecutively. This function removes duplicate nodes in such a way that only one instance of each value remains.

Walkthrough:

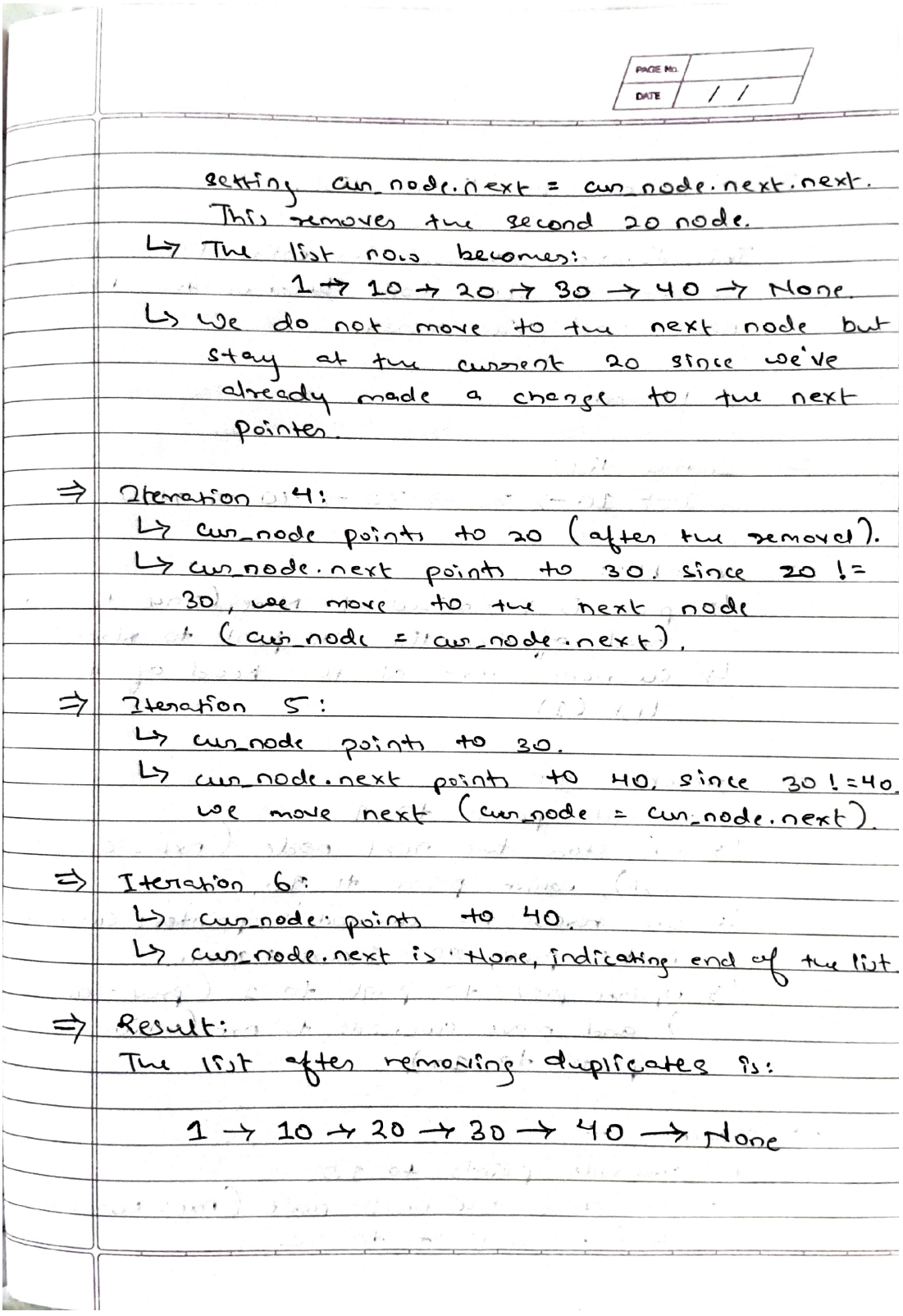
⇒ Initial list:
1 → 10 → 20 → 20 → 30 → 40 → None.

⇒ Variables:
↳ 'cur_node' starts at the head of the list (1).

⇒ Iteration 1:
↳ cur_node points to 1.
↳ cur_node.next points to 10. since 1 != 10, we move to the next node (cur_node = cur_node.next).

⇒ Iteration 2:
↳ cur_node points to 10.
↳ cur_node.next points to 20. since 10 != 20, we move to the next node (cur_node = cur_node.next).

⇒ Iteration 3:
↳ cur_node points to 20.
↳ cur_node.next points to another 20. since 20 == 20, we remove the duplicate by



b. **Output:**

```
1 -> 10 -> 20 -> 20 -> 30 -> 40 -> None
1 -> 10 -> 20 -> 30 -> 40 -> None
```

3. **Reversing a linked list.**

a. **Code:**

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

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

    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 reversing_list(self):
        prev = None
        cur_node = self.head
        while cur_node:
            nxt = cur_node.next
            cur_node.next = prev

            prev = cur_node
            cur_node = nxt

        self.head = prev

    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(10)
    llist.append(20)
    llist.append(30)
    llist.append(40)

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

    # Reversing the entire list
    llist.reversing_list()
```

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

```
# Reversing a linked list.

def reversing_list(self):
    prev = None
    cur_node = self.head
    while cur_node:
        nxt = cur_node.next
        cur_node.next = prev

        prev = cur_node
        cur_node = nxt

    self.head = prev
```

1. **Explanation:**

PAGE No.
DATE / /

3. Reversing the linked list.

This function reverses the entire linked list so that the last node becomes the first node, and vice versa.

Walkthrough:

⇒ Initial list:
1 → 10 → 20 → 30 → 40 → None.

⇒ Variables:
↳ prev is initialized to None (since the new head of the list will point to None).
↳ cur_node starts at the head of the list (1).

⇒ Iteration 1:
↳ cur_node points to 1.
↳ We store the next node (nxt = cur_node.next), which points to 10.
↳ We reverse the cur_node's pointer: cur_node.next = prev, so 1.next = None.
↳ Update prev to point to 1 (prev = cur_node), and move cur_node to nxt (which points to 10).

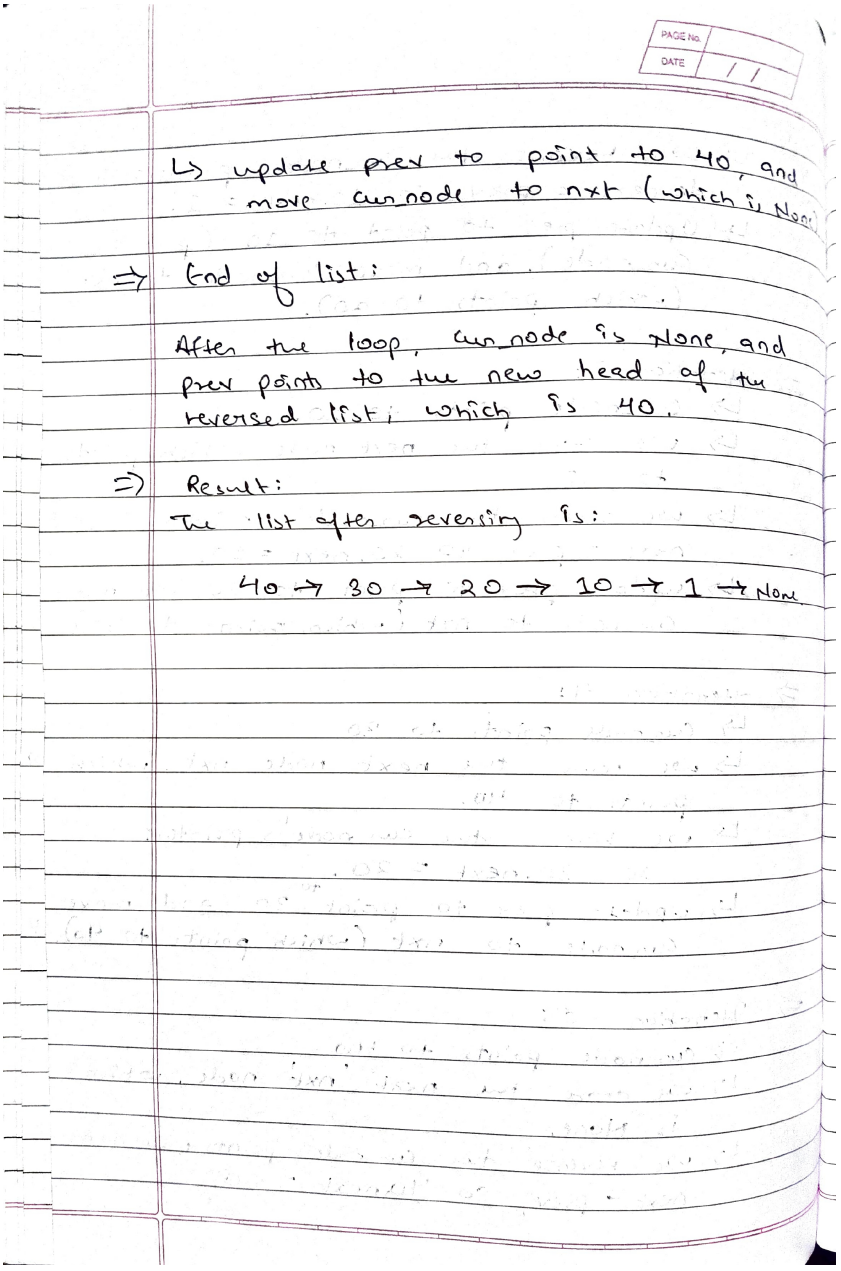
⇒ Iteration 2:
↳ cur_node points to 10.
↳ We store the next node (nxt = cur_node.next), which points to 20.

↳ We reverse the cur_node's pointer: cur_node.next = prev, so 10.next = 1.
↳ Update prev to point to 10 (prev = cur_node), and move cur_node to nxt (which points to 20).

⇒ Iteration 3:
↳ cur_node points to 20.
↳ We store the next node, which points to 30.
↳ We reverse the cur_node's pointer: cur_node.next = prev, so 20.next = 10.
↳ update prev to point to 20, and move cur_node to nxt (which points to 30).

⇒ Iteration 4:
↳ cur_node points to 30.
↳ We store the next node, nxt, which points to 40.
↳ We reverse the cur_node's pointer: so 30.next = 20.
↳ update prev to point to 30, and move cur_node to nxt (which points to 40).

⇒ Iteration 5:
↳ cur_node points to 40.
↳ We store the next 'nxt' node, which is None.
↳ We reverse the cur_node's pointer: cur_node.next = prev, so 40.next = 30.



b. **Output:**

```
1 -> 10 -> 20 -> 30 -> 40 -> None
40 -> 30 -> 20 -> 10 -> 1 -> None
```

---

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

MCA-Coursework/SEM-1/Data-Structure-Using-Python/Assignments/Assignment-12 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-12
