only_temp

May 4, 2025

[15]:
```python
class Node:
    def __init__(self,value):
        self.value = value
        self.next = None

class LinkedList:
    def __init__(self,value):
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 1

    def print_list(self):
        temp = self.head
        while temp != None:
            print(temp.value)
            temp = temp.next

    def append(self, value):
        new_node = Node(value)
        # if self.head is None:
        if self.length == 0:
            self.head = new_node
            self.tail = new_node
            self.length = 1
        else:
            self.tail.next = new_node
            self.tail = new_node
            self.length = self.length + 1


my_linked_list = LinkedList(1)
my_linked_list.append(40)


# print(my_linked_list.head)
# LinkedList()
```

```
print(LinkedList.print_list())
```

```
1
40
None
```

[48]:
```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

class LinkedList:
    def __init__(self,value):
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 1

    def print_list(self):
        temp = self.head
        while temp != None :
            print(temp.value)
            temp = temp.next
        # return True

    def append_list(self,value):
        new_node = Node(value)
        if self.length == 0:
            self.head = new_node
            self.tail = new_node
            self.length += 1
            return True
        else:
            self.tail.next = new_node
            self.tail = new_node
        self.length += 1

    def pop(self):
        if self.length == 0 :
            return None
        elif self.length == 1:
            temp = self.head
            self.head = None
            self.tail = None
            self.length -= 1
            return temp
        else:
            temp1 = self.head
            while temp1.next != None:
                temp2 = temp1
                temp1 = temp1.next
            self.tail = temp2
            temp2.next = None
        self.length -= 1
        return temp1

    def pop_first(self):
        if self.length == 0:
            return None
        elif self.length == 1:
            temp = self.head
            self.head = None
            self.tail = None
            self.length -= 1
            return temp
        else:

            temp = self.head
            self.head = self.head.next
            temp.next = None
            self.length -= 1
            return temp

    def prepend(self, value):
        new_node = Node(value)
        if self.length == 0 :
            self.head = new_node
            self.tail = new_node
        else:
            temp = self.head
            self.head = new_node
            self.head.next = temp
        self.length += 1
        return True

    def get (self, index):
        if index<0 or index>=self.length:
            return None
        else:
            temp = self.head
            for _ in range(index):
                temp = temp.next
            return temp

    def set(self, index, value):
        if index<0 or index>=self.length:
            return None
        else:
            temp = self.get(index)
            temp.value = value
        return True

    def insert(self, index, value):
        new_node = Node(value)
        if index<0 or index>=self.length:
            return None
        elif index == 0 :
            self.prepend()
        elif index == self.length:
            self.append()
        else:
            temp = self.get(index-1)
            new_node.next = temp.next
            temp.next = new_node
            self.length +=1
        return True

    def remove(self,index):
        if index<0 or index>=self.length:
            return None
        elif index == 0 :
            self.pop_first()
        elif index == self.length:
            self.pop()
        else:
            prev = self.get(index-1)
            current = self.get(index)
            ahead = self.get(index+1)
            prev.next = ahead
            current.next = None
            self.length -=1
            return current

    def reverse(self):
        prev = None
        current = self.head
        self.head = self.tail
        self.tail = current
        while current != None:
            next_node = current.next
            current.next = prev
            prev = current
            current = next_node
        return True
        # ajhai pani pointer (self.head ra self.tail le pahilakai lai point␣
 ↪gareko xa)




my_linkedList = LinkedList(40)
my_linkedList.append_list(50) ; my_linkedList.append_list(60) ; my_linkedList.
 ↪append_list(70)
my_linkedList.print_list()              ; print(f'----------------')
print(f'------------------------')
print(my_linkedList.remove(1).value)         ; print(f'--------removing␣
 ↪done--------')
my_linkedList.print_list()              ; print(f'---------Printing-------')
print(f'------------------------')
print(my_linkedList.reverse())         ; print(f'--------Reversing--------')
my_linkedList.print_list()              ; print(f'---------Printing-------')
```

```
40
50
60
70
----------------
------------------------
50
--------removing done--------
40
60
70
---------Printing-------
------------------------
True
--------Reversing--------
70
60
40
---------Printing-------
```

1. insert(self, index, value)

Problem: You're calling self.prepend() and self.append() without arguments, but both methods require a value parameter.

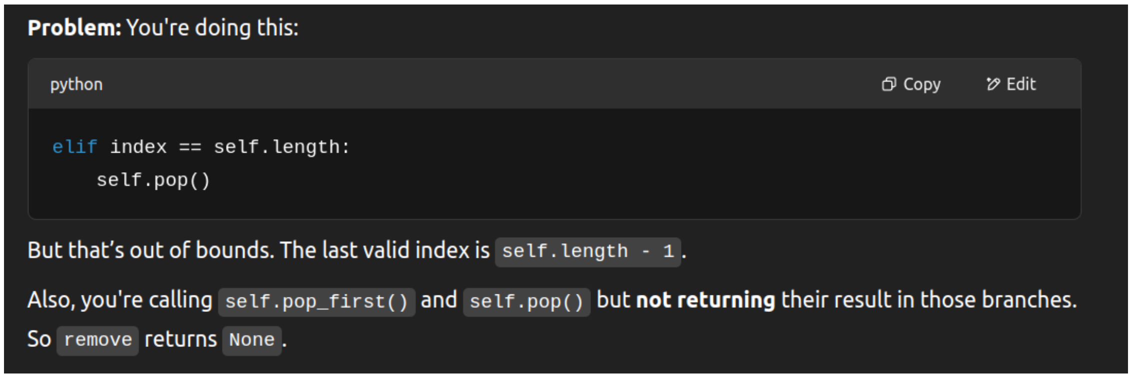
Also: The condition if index < 0 or index >= self.length: makes it impossible to insert at the end of the list (index == self.length), which is usually allowed for insert operations.

```
def insert(self, index, value):
    if index < 0 or index > self.length:
        return None
    if index == 0:
        return self.prepend(value)
    if index == self.length:
        return self.append_list(value)

    new_node = Node(value)
    prev = self.get(index - 1)
    new_node.next = prev.next
    prev.next = new_node
    self.length += 1
    return True
```

2. remove(self, index)

Problem: You're doing this:



Problem: You're doing this:

```python
elif index == self.length:
    self.pop()
```

But that's out of bounds. The last valid index is `self.length - 1`.

Also, you're calling `self.pop_first()` and `self.pop()` but **not returning** their result in those branches. So `remove` returns `None`.

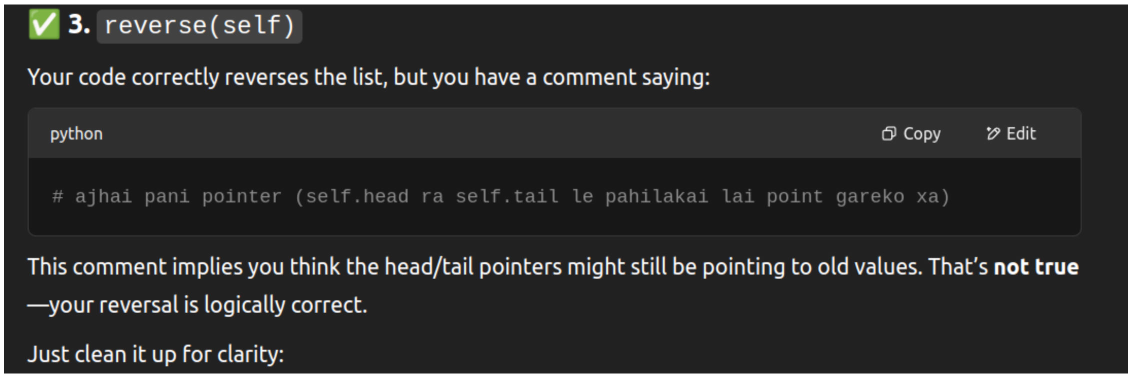
Fix !!

```
def remove(self, index):
    if index < 0 or index >= self.length:
        return None
    if index == 0:
        return self.pop_first()
    if index == self.length - 1:
        return self.pop()

    prev = self.get(index - 1)
    current = prev.next
    prev.next = current.next
    current.next = None
    self.length -= 1
    return current
```

3. reverse(self)

Your code correctly reverses the list, but you have a comment saying:



✅ **3.** `reverse(self)`

Your code correctly reverses the list, but you have a comment saying:

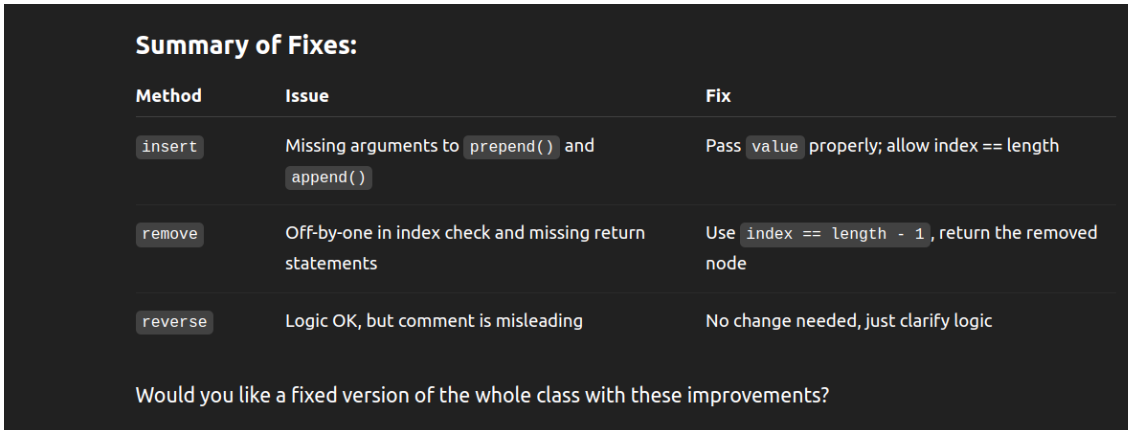
```python
# ajhai pani pointer (self.head ra self.tail le pahilakai lai point gareko xa)
```

This comment implies you think the head/tail pointers might still be pointing to old values. That's **not true**—your reversal is logically correct.

Just clean it up for clarity:

Output Tip:

You're doing print(my_linkedList.remove(1).value) which will throw an error if remove(1) returns None. You may want to check if it's None before printing .value.



Summary of Fixes:

Method	Issue	Fix
`insert`	Missing arguments to `prepend()` and `append()`	Pass `value` properly; allow index == length
`remove`	Off-by-one in index check and missing return statements	Use `index == length - 1`, return the removed node
`reverse`	Logic OK, but comment is misleading	No change needed, just clarify logic

Would you like a fixed version of the whole class with these improvements?