test_1

May 3, 2025

1 3 Under the hood

[1]:

```python
# simple dictionary

dict1 = {'name': 'Biswash', 'age':23}

print(dict1['name'])
print(type(dict1['name']))

print('-------------------')

dict_nest = {'Organization': 'Pulchowk' , 'Student':dict1}
print(type(dict_nest))

print(dict_nest['Organization'])

print('---------------')
print(type(dict_nest['Student']))
print(dict_nest['Student'])

print('------------------------')
print(dict_nest['Student']['name'])
```

```
Biswash
<class 'str'>
-------------------
<class 'dict'>
Pulchowk
---------------
<class 'dict'>
{'name': 'Biswash', 'age': 23}
------------------------
Biswash
```

[2]:

```python
head = {
            "value" : 11,
            "next" : {
                                 "value" : 3,
                                 'next': {
                                                "value" :23,
                                                "next" : {
                                                            "value":7,
                                                            "next" : None
                                                         }

                                         }

                     }

        }
```

[3]:

```
print(head['next']['value'])
```

```
3
```

[4]:

```
print(head['next']['next']['value'])
```

```
23
```

[5]:

```
print(head['next']['next']['next']['value'])
```

```
7
```

[6]:

```
# This is how, linked list is accessed

# This is difference between linked list and dictionaty 'for how to access'

# print(my_linked_list.head.next.next.value)
```

2 4 Linked List (LL) Constructor

[7]:

```
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

my_linked_list = LinkedList(14)
```

```

print(my_linked_list.head.value)
```

```
14
```

[8]:

```
print('Head:', my_linked_list.head.value)
print('Tail:', my_linked_list.tail.value)
print('Length:', my_linked_list.length)
```

```
Head: 14
Tail: 14
Length: 1
```

3 5 Priniting the Linked List

[9]:

```
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 is not None:
            print(temp.value)
            temp = temp.next

value1 = LinkedList(14)

value2 = LinkedList(24)

print(value2.print_list())
```

```
24
None
```

[10]:

```
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

my_linked_list = LinkedList(20)

print(my_linked_list.head) # Address

print(my_linked_list.tail) # Address (pointning to the same address)

print(my_linked_list.head.value)

print(my_linked_list.head.next)
```

```
<__main__.Node object at 0x75d608b4dc90>
<__main__.Node object at 0x75d608b4dc90>
20
None
```

[11]:

```
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

biswash_list = LinkedList(60)

print(biswash_list.head.value)
```

```
60
```

[12]:

```
# Now appending the linked list

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 append_list(self,value):
        new_node = Node(value)
        self.tail.next = new_node
        self.tail = new_node
        self.length = self.length+1

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

[13]:

```
my_linked_list = LinkedList(50)
```

[14]:

```
my_linked_list.append_list(60)
```

[15]:

```
my_linked_list.print_list()
```

```
50
60
```

[16]:

```
print(my_linked_list.head)
```

```
<__main__.Node object at 0x75d608b4c310>
```

[17]:

```
print(my_linked_list.head.value)
```

```
50
```

[18]:

```
print(my_linked_list.head.next)
```

```
<__main__.Node object at 0x75d608b4e830>
```

[19]:

```
print(my_linked_list.head.next.value)
```

```
60
```

[20]:

```
print(my_linked_list.head.next.next)
```

```
None
```

3. self.head and self.tail — what do they store?

Let’s look at this line:

self.head = new_node

self.tail = new_node

Here’s what’s happening:

self.head = new_node

self.head is a reference to the new_node. It stores the memory address of the node, not the actual node data.

When you set self.head = new_node, you’re linking the head to the first node in the list. self.head points to the new_node object.

self.tail = new_node

Similarly, self.tail is a reference to the new_node. It points to the same memory location as new_node, meaning self.tail also stores the memory address of the last node in the list.

So both self.head and self.tail store references to the new_node, not the values themselves.

[21]:

```
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_list = 1

    def append(value):
        new_node = Node(value)
        self.next


my_linked_list = LinkedList(14)

print(my_linked_list.head.value)
print('-------------------------')
print(my_linked_list.head.next)
```

```
14
-------------------------
None
```

4. Why does self.head and self.tail store pointers?

In a linked list, both self.head and self.tail store references (or pointers) to the nodes. They don't store the actual data (the value) but the location of the nodes in memory.

self.head points to the first node.

self.tail points to the last node.

Since both self.head and self.tail are references, they store memory addresses of the Node objects, not the actual value of the node. They are pointers to the nodes.

To visualize:

Attribute What it Stores self.head Reference (pointer) to the first node self.tail Reference (pointer) to the last node

[22]:
```
a = 5
b = a  # b gets the value of a, so b is now also 5

print(f'a = {a} and b = {b}')

print(f"Address of a: {id(a)}")  # Prints the memory address of x
print(f"Address of b: {id(b)}")  # Prints the memory address of y
```

```
a = 5 and b = 5
Address of a: 129562234061168
Address of b: 129562234061168
```

Immutable types in Python, like integers, strings, and tuples, cannot be changed once they're created. When you assign a variable b = a, it copies the value of a into b, so both variables hold their own copy of the value. They do not point to the same memory location.

BUT

If a were a mutable type (like a list), then assigning b = a would create a reference to the same object, not a copy.

[23]:
```
x = [1,2,3]

y = x

print(x,y)

print(f"Address of x: {id(x)}")  # Prints the memory address of x
print(f"Address of y: {id(y)}")  # Prints the memory address of y
```

```
[1, 2, 3] [1, 2, 3]
Address of x: 129562129550912
Address of y: 129562129550912
```

[24]:
```
x.append(4)
print(x,y)
```

```
[1, 2, 3, 4] [1, 2, 3, 4]
```

[25]:

```python
y = x.copy()
y = x[:]
```

To maintain the linked list structure, you would typically want self.next to store a reference to the next node (not a value). If self.next holds an integer result like 30 - self.next, it would break the linked list structure, as next is supposed to store a reference to the next node in the list.

[26]:

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

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

# Create nodes
node1 = Node(10)
node2 = Node(20)

# Create linked list and set head and tail
ll = LinkedList()
# ll.head = node1  # head points to node1
# ll.tail = node2  # tail points to node2

print(ll.head)
print(ll.tail)

print(ll.head.value)
print(ll.tail.value)
```

```
None
None
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[26], line 23
     20 print(ll.head)
     21 print(ll.tail)
---> 23 print(ll.head.value)
     24 print(ll.tail.value)

AttributeError: 'NoneType' object has no attribute 'value'
```

[30]:

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

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

# Create nodes
node1 = Node(10)
node2 = Node(20)

# Create linked list and set head and tail
ll = LinkedList()
ll.head = node1  # head points to node1
ll.tail = node2  # tail points to node2

print(ll.head)
print(ll.tail)

print(ll.head.value)
print(ll.tail.value)
```

```
<__main__.Node object at 0x75d603f026b0>
<__main__.Node object at 0x75d603f745b0>
10
20
```

Each node in a singly-linked list typically consists of two parts:

Data: The actual information stored in the node. Next Pointer: A reference to the next node. The last node's next pointer is usually set to null.

[31]:

```
# Now appending the linked list

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 append_list(self,value):
        new_node = Node(value)
        self.tail.next = new_node
        # self.tail = new_node
        self.length = self.length+1

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

[32]:

```
my_linked_list = LinkedList(13)

print(my_linked_list.head)
print(my_linked_list.tail)

print(my_linked_list.head.value)
print(my_linked_list.tail.value)
```

```
<__main__.Node object at 0x75d608b32fb0>
<__main__.Node object at 0x75d608b32fb0>
13
13
```

[33]:

```
my_linked_list = LinkedList(13)
my_linked_list.append_list(20)
```

[34]:

```
print(my_linked_list.print_list())
```

```
13
20
None
```

[35]:

```
print(my_linked_list.head)
print(my_linked_list.tail)
```

```
<__main__.Node object at 0x75d608b32b60>
<__main__.Node object at 0x75d608b32b60>
```

[36]:

```
print(my_linked_list.head.value)
print(my_linked_list.tail.value)
```

```
13
13
```

[37]:

```
print(my_linked_list.head)
print(my_linked_list.head.next)
```

```
<__main__.Node object at 0x75d608b32b60>
<__main__.Node object at 0x75d608b31870>
```

[38]:

```
print(my_linked_list.head)
print(my_linked_list.head.next.next)
```

```
<__main__.Node object at 0x75d608b32b60>
None
```

[39]:

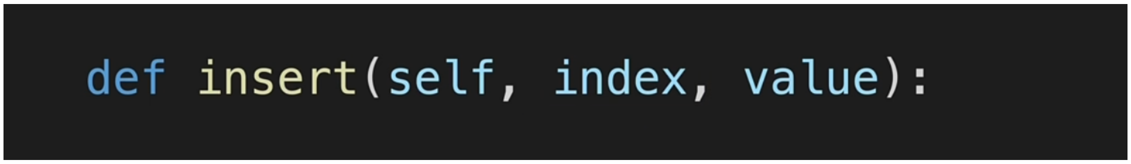
```
print(my_linked_list.head)
print(my_linked_list.head.next.value)
```

```
<__main__.Node object at 0x75d608b32b60>
20
```

[ ]:

4 6

5 7 Append method edge case



```
def insert(self, index, value):
```

[40]:

```
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
```

```python
my_linked_list = LinkedList(20)

print(my_linked_list.print_list())
```

```
20
None
```

[41]:

```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 append(self,value):
        new_node = Node(value)
        self.tail.next = new_node
        self.tail = new_node
        self.length = self.length + 1

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

my_linked_list = LinkedList(20)
my_linked_list.append(30)

print(my_linked_list.print_list())
```

```
20
30
None
```

Now thinking of edge case

test :

```
def insert(self, index, value):
```

[42]:

```
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):
    #     if self.head

# my_linked_list = LinkedList(40)

print(my_linked_list.print_list())

print(my_linked_list.head)
```

```
20
30
None
<__main__.Node object at 0x75d608b4ef80>
```

[ ]:

6 8 Pop intro

[43]:

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

```python
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):
        if self.length == 0:
            new_node = Node(value)
            self.head = new_node
            self.tail = new_node
        else:
            append_node = Node(value)
            self.tail.next = append_node
            self.tail = append_node

        self.length = self.length + 1

    def pop(self):
        temp1 = self.head
        while temp1.next != None:
            temp2 = temp1
            temp1 = temp1.next

        temp2.next = None
        self.tail = temp2
        self.length = self.length -1

my_linked_list = LinkedList(20)

my_linked_list.print_list()

print(f'-------------------')

my_linked_list.append(30)
my_linked_list.append(40)
my_linked_list.append(50)

my_linked_list.print_list()
```

```
20
-------------------
20
30
40
50
```

[44]:

```python
my_linked_list.pop()

print(f'-------------------')
my_linked_list.print_list()
```

```
-------------------
20
30
40
```

[45]:

```python
print(my_linked_list.head)

print(my_linked_list.head.value)

# --------------------------------------

print(my_linked_list.head.next)

print(my_linked_list.head.next.value)

# --------------------------------------

print(my_linked_list.head.next.next)

print(my_linked_list.head.next.next.value)

# --------------------------------------

print(my_linked_list.head.next.next.next)

print(my_linked_list.head.next.next.next.value)


print(my_linked_list.head.next.next.next.next)
```

```
<__main__.Node object at 0x75d60816bdf0>
20
<__main__.Node object at 0x75d60816bee0>
30
<__main__.Node object at 0x75d60816b5e0>
40
```

```
None
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[45], line 21
     17 # ------------------------------------
     19 print(my_linked_list.head.next.next.next)
---> 21 print(my_linked_list.head.next.next.next.value)
     24 print(my_linked_list.head.next.next.next.next)

AttributeError: 'NoneType' object has no attribute 'value'
```

[46]:

```
# Improvised code:

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):
        if self.length == 0:
            new_node = Node(value)
            self.head = new_node
            self.tail = new_node
        else:
            append_node = Node(value)
            self.tail.next = append_node
            self.tail = append_node

        self.length = self.length + 1

    def pop(self):
        if self.length == 0:
            return None  # Nothing to pop
        temp1 = self.head
        temp2 = None
        while temp1.next != None:
            temp2 = temp1
            temp1 = temp1.next

        if temp2:  # If there's a second-to-last node
            temp2.next = None
        self.tail = temp2
        self.length -= 1

# Testing the code
my_linked_list = LinkedList(20)

my_linked_list.print_list()

print(f'-------------------')

my_linked_list.append(30)
my_linked_list.append(40)
my_linked_list.append(50)

my_linked_list.print_list()

my_linked_list.pop()

print(f'-------------------')
my_linked_list.print_list()
```

```
20
-------------------
20
30
40
50
-------------------
20
30
40
```

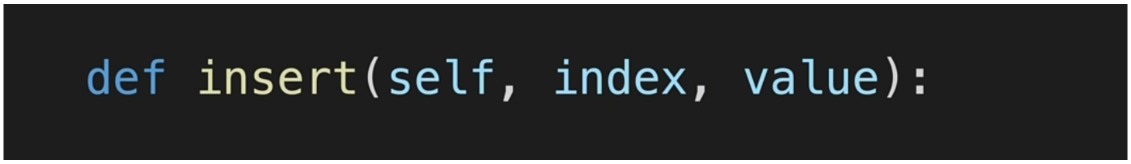
the pop() method is not working correctly as it doesn't properly update the next pointer of the second-to-last node. Here's a corrected version of your pop() method:

the main line is:

```
        temp2.next = None

        first, removing the reference of tail to the 2nd last node
```

6.0.1 Do you consider all the edge cases?

6.0.2 Plus you need to return the object / variable ; Kun hatayeko tei



```
def insert(self, index, value):
```

[47]:

```
# No not ; so try that
```

7 9 Pop First

7.1 This one was missing

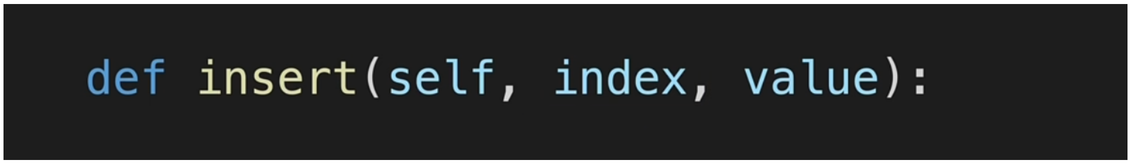
[141]:

```
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):
        # not seeign edge case ; haha
        append_value = Node(value)
        self.tail.next = append_value
        self.tail = append_value
        return True

    def pop_first(self):
        # Not seeing the edge cases
        temp = self.head
        self.head = temp.next
        temp.next = None
        return temp
```

[142]:
```
# Testing the code
my_linked_list = LinkedList(20)

print("Initial list:")
my_linked_list.print_list()

print('-------------------')

my_linked_list.append(30)
my_linked_list.append(40)
my_linked_list.append(50)

print("After appending 30, 40, 50:")
my_linked_list.print_list()

print('-------------------')
my_linked_list.pop_first()
my_linked_list.pop_first()

print('-------------------')
my_linked_list.print_list()
```

```
Initial list:
20
-------------------
After appending 30, 40, 50:
20
30
40
50
-------------------
-------------------
40
50
```

8 10 Now go for prepend



```
def insert(self, index, value):
```

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

```python
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):
        append_value = Node(value)

        if self.length == 0:
            self.head = append_value
            self.tail = append_value
        else:
            self.tail.next = append_value
            self.tail = append_value

        self.length = self.length + 1


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

            self.tail = temp2
            self.tail.next = None
            self.length = self.length - 1

            return temp1.value

    def prepend(self,value):
        temp = self.head

        new_node_head = Node(value)
        self.head = new_node_head
        new_node_head.next = temp

        self.length = self.length +1

        if self.length == 0:
            return None
```

[49]:

```
# Testing the code
my_linked_list = LinkedList(20)

my_linked_list.print_list()

print(f'-------------------')

my_linked_list.append(30)
my_linked_list.append(40)
my_linked_list.append(50)

my_linked_list.print_list()

my_linked_list.pop()
my_linked_list.pop()
my_linked_list.pop()
# my_linked_list.pop()

print(f'-------------------')
my_linked_list.print_list()
```

```
20
-------------------
20
30
40
50
-------------------
20
```

[50]:

```
my_linked_list.prepend(14)

my_linked_list.print_list()
```

```
14
20
```

[51]:

```
my_linked_list.prepend(24)

my_linked_list.prepend(34)
my_linked_list.prepend(44)

my_linked_list.print_list()
```

```
44
34
24
14
20
```

8.0.1 See the edge cases , my boy

when list is empty

```
def insert(self, index, value):
```

[ ]:

[ ]:

9 11 Get Intro

when we pass the index, it will get the (value) of that node and give it to us

def get(self, index)

Also 1 more important thing; for the index

```
def insert(self, index, value):
```

range(3) = 0,1,2 ; while i = 3 ; loop will not run

[115]:
```
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):
        append_node = Node(value)
        if self.length == 0:
            self.head = append_node
            self.tail = append_node
        else:
            self.tail.next = append_node
            self.tail = append_node
        self.length = self.length + 1
        return True

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

        self.tail = temp2
        self.tail.next = None
        self.length = self.length - 1
        return temp1.value

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

    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.value
```

[117]:

```
# Testing the code
my_linked_list = LinkedList(20)

my_linked_list.print_list()

print(f'-------------------')

my_linked_list.append(30)
my_linked_list.append(40)
my_linked_list.append(50)

my_linked_list.print_list()

my_linked_list.pop()
my_linked_list.pop()

print(f'-------------------')
my_linked_list.print_list()

my_linked_list.prepend(7)

print(f'-------------------')
my_linked_list.print_list()

my_linked_list.prepend(14)

print(f'-------------------')
my_linked_list.print_list()

print(f'-------------------')
# my_linked_list.print_list()

print(my_linked_list.head.next.next.next.value)

print(f'-------------------')
print(my_linked_list.get(3))
```

```
20
-------------------
20
```

```
30
40
50
--------------------
20
30
--------------------
7
20
30
--------------------
14
7
20
30
--------------------
30
--------------------
30
```

```
def insert(self, index, value):
```

10 13 Set Method

```

```

```
# testing code in small scale

print(my_linked_list.head.next.value)


print(f'--------------------')

my_linked_list.head.next.value = 77 # Change this 77 to get live result

print(f'--------------------')

my_linked_list.print_list()



# So in this way, we can change it ' modify it to get set function
```

```
77
```

```
-------------------
-------------------
14
77
20
30
```

[113]:

```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):
        append_node = Node(value)
        if self.length == 0:
            self.head = append_node
            self.tail = append_node
        else:
            self.tail.next = append_node
            self.tail = append_node
        self.length = self.length + 1
        return True

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

        self.tail = temp2
        self.tail.next = None
        self.length = self.length - 1
        return temp1.value

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

    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.value

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

[114]:

```
# Testing the code
my_linked_list = LinkedList(20)

my_linked_list.print_list()

print(f'--------------------')

my_linked_list.append(30)
my_linked_list.append(40)
my_linked_list.append(50)

print(my_linked_list.print_list())

print(f'--------------------')
```

```
my_linked_list.set_value(0,22)

print(my_linked_list.print_list())
```

```
20
-------------------
20
30
40
50
None
-------------------
22
30
40
50
None
```

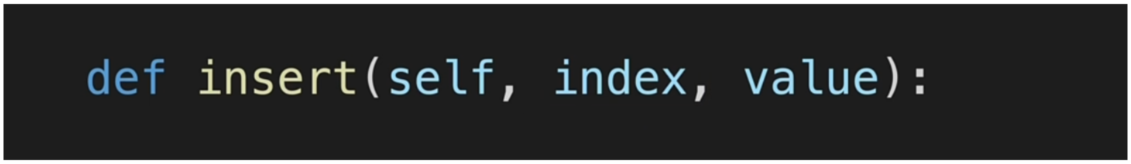
[ ]:

```
# performing set with help of get
```

[ ]:

```

```

11 14 Insert



```
def insert(self, index, value):
```

[134]:

```
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):
        append_value = Node(value)

        # edge cases think
        if self.length == 0 :
            self.head = append_value
            self.tail = append_value
        else:
            self.tail.next = append_value
            self.tail = append_value
        self.length +=1
        return True


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


    def prepend(self,value):
        new_value = Node(value)
        # edge cases
        if self.length == 0 :
            self.head = new_value
            self.tail = new_value
        else:
            new_value.next = self.head
            self.head = new_value
        self.length += 1
        return True

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

    def set(self,index,value):
        # see the edge cases
        if index<0 or index>=self.length:
            return False
        else:
            address = self.get(index)
            address.value = value
        return True

    def insert(self, value, index):
        # see the edge cases
        if index<0 or index>self.length:
            return False
        elif index == 0 :
            self.prepend(value)
        elif index == (self.length):
            self.append(value)
        else:
            new_node = Node(value)
            temp1 = self.head
            for _ in range(index):
                temp2 = temp1
                temp1 = temp1.next
            new_node.next = temp1
            temp2.next = new_node
        return True
```

[135]:

```
# Testing the code
my_linked_list = LinkedList(20)

print("Initial list:")
my_linked_list.print_list()

print('-------------------')

my_linked_list.append(30)
my_linked_list.append(40)
my_linked_list.append(50)
```

```
print("After appending 30, 40, 50:")
my_linked_list.print_list()

print('--------------------')
my_linked_list.insert(25,4)

my_linked_list.print_list()
```

```
Initial list:
20
-------------------
After appending 30, 40, 50:
20
30
40
50
-------------------
20
30
40
50
25
```

[ ]:

```

```

12 15 Remove

[150]:

```
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):
        append_value = Node(value)

        # edge cases think
        if self.length == 0 :
            self.head = append_value
            self.tail = append_value
        else:
            self.tail.next = append_value
            self.tail = append_value
        self.length +=1
        return True


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


    def prepend(self,value):
        new_value = Node(value)
        # edge cases
        if self.length == 0 :
            self.head = new_value
            self.tail = new_value
        else:
            new_value.next = self.head
            self.head = new_value
        self.length += 1
        return True

    def get(self,index):
        # select edge cases
        # index error cases
        if index<0 or index >=self.length:
            return None
```

```python
        temp = self.head
        for _ in range(index):
            temp = temp.next
        return temp

    def set(self,index,value):
        # see the edge cases
        if index<0 or index>=self.length:
            return False
        else:
            address = self.get(index)
            address.value = value
        return True

    def insert(self, value, index):
        # see the edge cases
        if index<0 or index>self.length:
            return False
        elif index == 0 :
            self.prepend(value)
        elif index == (self.length):
            self.append(value)
        else:
            new_node = Node(value)
            temp1 = self.head
            for _ in range(index):
                temp2 = temp1
                temp1 = temp1.next
            new_node.next = temp1
            temp2.next = new_node
        return True

    def pop_first(self):
        # Not seeing the edge cases
        temp = self.head
        self.head = temp.next
        temp.next = None
        return temp

    def remove(self, index):
        # see the edge cases
        if index<0 or index>self.length:
            return False
        elif index == 0 :
            # pop first
            self.pop_first()
        elif index == (self.length):
            # pop
            self.pop()
        else:
            # remove the middle item
            temp1 = self.head
            for _ in range(index):
                temp2 = temp1
                temp1 = temp1.next
            temp2.next = temp1.next
        return temp1
```

[153]:

```
# Testing the code
my_linked_list = LinkedList(20)

print("Initial list:")
my_linked_list.print_list()

print('--------------------')

my_linked_list.append(30)
my_linked_list.append(40)
my_linked_list.append(50)

print("After appending 30, 40, 50:")
my_linked_list.print_list()

print('--------------------')

print(my_linked_list.remove(2).value)
print('--------------------')
my_linked_list.print_list()
print('--------------------')

print(my_linked_list.remove(1).value)
print('--------------------')
my_linked_list.print_list()
```

```
Initial list:
20
--------------------
After appending 30, 40, 50:
20
30
40
50
--------------------
40
```

```
-------------------
20
30
50
-------------------
30
-------------------
20
50
```

13 16 Reverse

[156]:

```
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):
        append_value = Node(value)

        # edge cases think
        if self.length == 0 :
            self.head = append_value
            self.tail = append_value
        else:
            self.tail.next = append_value
            self.tail = append_value
        self.length +=1
        return True


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


    def prepend(self,value):
        new_value = Node(value)
        # edge cases
        if self.length == 0 :
            self.head = new_value
            self.tail = new_value
        else:
            new_value.next = self.head
            self.head = new_value
        self.length += 1
        return True

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

    def set(self,index,value):
        # see the edge cases
        if index<0 or index>=self.length:
            return False
        else:
            address = self.get(index)
            address.value = value
        return True

    def insert(self, value, index):
        # see the edge cases
        if index<0 or index>self.length:
            return False
        elif index == 0 :
            self.prepend(value)
        elif index == (self.length):
            self.append(value)
        else:
            new_node = Node(value)
            temp1 = self.head
            for _ in range(index):
                temp2 = temp1
                temp1 = temp1.next
            new_node.next = temp1
            temp2.next = new_node
        return True

    def pop_first(self):
        # Not seeing the edge cases
        temp = self.head
        self.head = temp.next
        temp.next = None
        return temp

    # def remove(self, index):
    #     # see the edge cases
    #     if index<0 or index>self.length:
    #         return False
    #     elif index == 0 :
    #         # pop first
    #         self.pop_first()
    #     elif index == (self.length):
    #         # pop
    #         self.pop()
    #     else:
    #         # remove the middle item
    #     #     temp1 = self.head
    #     #     for _ in range(index):
    #     #         temp2 = temp1
    #     #         temp1 = temp1.next
    #     #     temp2.next = temp1.next
    #     # return temp1
    #     current = self.head
    #     prev = None
    #     self.tail = self.head
    #     while current != None:
    #         next_node = current.next
    #         current.next = prev
    #         prev = current
    #         current = next_node
    #     self.head = prev

    def reverse(self):
        prev = None
        current = self.head
        self.tail = self.head  # After reversal, original head becomes tail

        while current:
            next_node = current.next  # Save next node
            current.next = prev       # Reverse the pointer
            prev = current            # Move prev forward
            current = next_node       # Move current forward

        self.head = prev  # Finally, set head to the last non-null node
```

```
current = self.head # [14]
prev = None         # None

current.next = prev # None
# [14] --> None
prev = current # [14]
```

Now we must get [7] and again [7] must point with [14]

```
next_node = current.next
```

since, current.next is already none, it must be “assigned” before

current = self.head prev = None next_node = current.next –> [7] current.next = prev None (previous) [14] –> None prev = current ||| prev = [14] Now the list becomes || [14] —-> None

and point current pointer (aaba kaslai reverse garne teslai) current = next node

`Given a list:`

HEAD → [14] → [7] → [20] → [30] → None

`After reversiing you want`

HEAD → [30] → [20] → [7] → [14] → None

13.1 What Needs to Happen?

Each node has a `.next` pointer. Right now, each node points **forward**.

Reversing means: - [14] → None (instead of pointing to [7]) - [7] → [14] - [20] → [7] - [30] → [20]

So you're **reversing the direction of `.next` pointers**.

13.2 How Do We Do That?

We use three variables: 1. `prev` — the node that will come **before** the current one (starts as `None`) 2. `current` — the node we're visiting 3. `next_node` — the node that comes **after** current (we save this before breaking the link)

13.3 Step-by-Step Example

Let's walk through the first iteration:

13.3.1 Initial Setup:

"'python prev = None current = Node(14) # points to Node(7)

13.3.2 Step 1

next_node = current.next # next_node = Node(7)

current.next = prev # Node(14).next = None

prev = current # prev = Node(14)

current = next_node # current = Node(7)

13.3.3 Step 2

next_node = current.next # Node(20)

current.next = prev # Node(7).next = Node(14)

prev = current # Node(7)

current = next_node # Node(20)

```
        Repeat this until current becomes None.
```

Final Step:

When current is None, your prev is pointing to the last node (30), which is the new head of the reversed list.

So, finally set:

```
    self.head = prev
```

Final Structure:

HEAD → [30] → [20] → [7] → [14] → None

[158]:

```python
# Testing the code
my_linked_list = LinkedList(20)

print("Initial list:")
my_linked_list.print_list()

print('--------------------')

my_linked_list.append(30)
my_linked_list.append(40)
my_linked_list.append(50)

print("After appending 30, 40, 50:")
my_linked_list.print_list()

print('--------------------')
my_linked_list.reverse()

my_linked_list.print_list()
```

```
Initial list:
20
--------------------
After appending 30, 40, 50:
20
30
40
50
```

```
-------------------
50
40
30
20
```