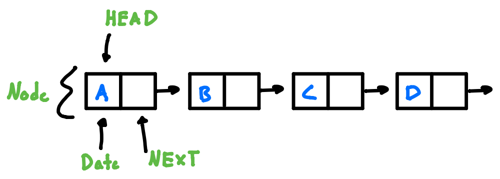
Singly Linked Lists - Length

- # of nodes in a linked list



Logic:

- Start from begining of the list, set a current node to the head of the list and go through each of the nodes until we hit null, and we will keep a running tally of how many nodes we've encountered.

Length Iterative

```
def len_iterative(self):

    count = 0
    cur_node = self.head

    while cur_node:
        count += 1
        cur_node = cur_node.next
    return count
```

Set current node to the front of the list.

While still a valid node object (loop thru linked list)

Keep going thru the nodes

Return the count of nodes

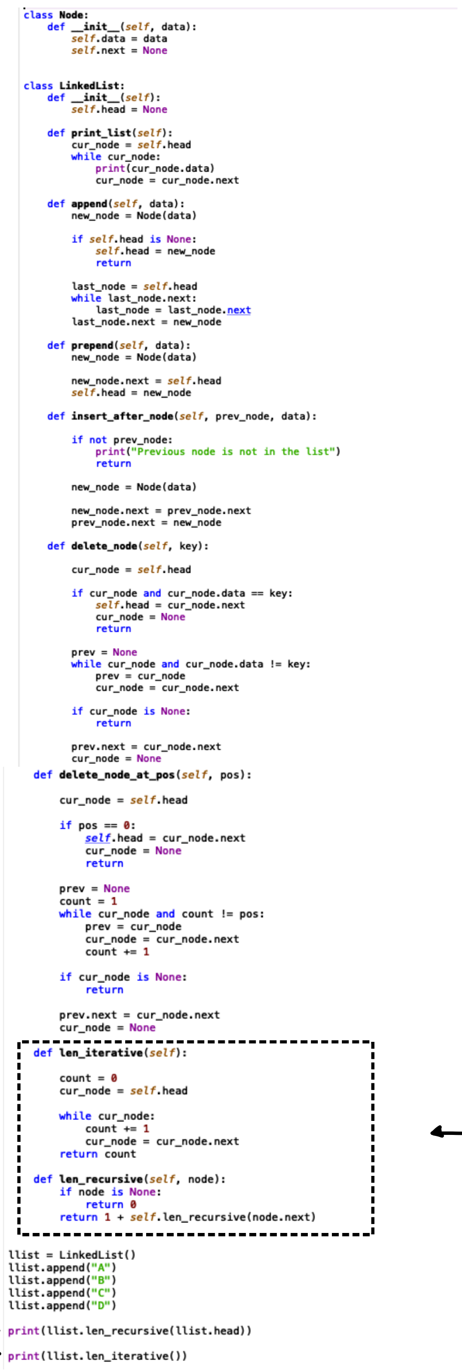
Length Recursive

```
def len_recursive(self, node):

    if node is None:
        return 0

    return 1 + self.len_recursive(node.next)
```

Base Case

Call recursive, then pass in the next node

Full Code:



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


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

    def print_list(self):
        cur_node = self.head
        while cur_node:
            print(cur_node.data)
            cur_node = cur_node.next

    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 prepend(self, data):
        new_node = Node(data)

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

    def insert_after_node(self, prev_node, data):

        if not prev_node:
            print("Previous node is not in the list")
            return

        new_node = Node(data)

        new_node.next = prev_node.next
        prev_node.next = new_node

    def delete_node(self, key):

        cur_node = self.head

        if cur_node and cur_node.data == key:
            self.head = cur_node.next
            cur_node = None
            return

        prev = None
        while cur_node and cur_node.data != key:
            prev = cur_node
            cur_node = cur_node.next

        if cur_node is None:
            return

        prev.next = cur_node.next
        cur_node = None
    def delete_node_at_pos(self, pos):

        cur_node = self.head

        if pos == 0:
            self.head = cur_node.next
            cur_node = None
            return

        prev = None
        count = 1
        while cur_node and count != pos:
            prev = cur_node
            cur_node = cur_node.next
            count += 1

        if cur_node is None:
            return

        prev.next = cur_node.next
        cur_node = None

    def len_iterative(self):

        count = 0
        cur_node = self.head

        while cur_node:
            count += 1
            cur_node = cur_node.next
        return count

    def len_recursive(self, node):
        if node is None:
            return 0
        return 1 + self.len_recursive(node.next)

llist = LinkedList()
llist.append("A")
llist.append("B")
llist.append("C")
llist.append("D")

print(llist.len_recursive(llist.head))

print(llist.len_iterative())
```

← Added functions

```
4
4
```