Length

In this lesson, you will learn how to calculate the length of a linked list.

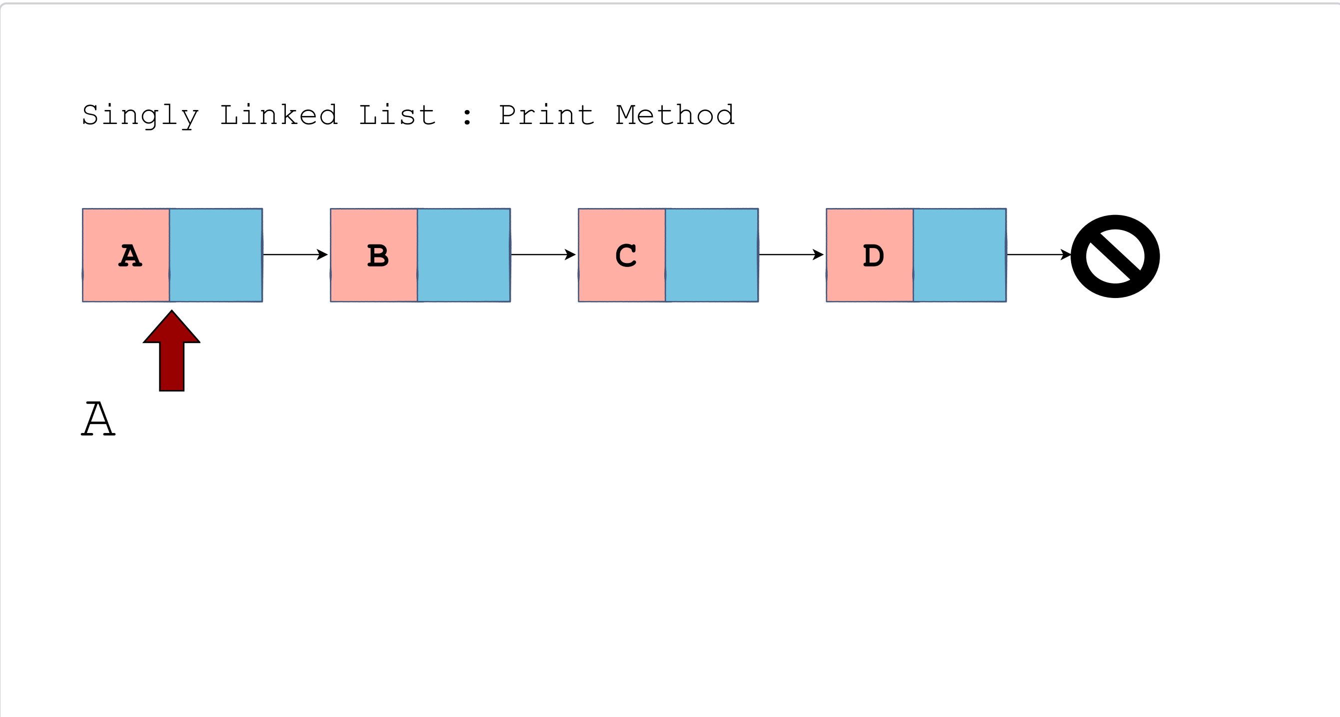
We'll cover the following

- Algorithm
- Iterative Implementation
- Recursive Implementation

In this lesson, we'll calculate the length or the number of nodes in a given linked list. We'll be doing this in both an iterative and recursive manner.

Algorithm

Let's look at a linked list and recall how we managed to print out the elements of a linked list. We iterate through every element of the linked list. We start from the head node and while we don't reach `None`, we print the data field of the node that we point to and increment the while loop by setting the current node equal to the next node.



Iterative Implementation

The above algorithm is going to help us construct an iterative method to calculate the length of a linked list. Let's go ahead and create a method `len_iterative` and step through it.

```
def len_iterative(self):
    count = 0
    cur_node = self.head
    while cur_node:
        count += 1
        cur_node = cur_node.next
    return count
```

len_iterative(self)

`len_iterative` takes `self` since it's a class method. As we start from the beginning of the linked list, we set `cur_node` equal to the head of the linked list on **line 3**. Then we go through each of the nodes until we hit `None`, which will terminate the `while` loop on **line 4**. We keep a count of how many nodes by setting a `count` variable equal to zero at the beginning of the method on **line 2**. `count` will keep track of the number of nodes we've encountered as long as the `cur_node` is not `None` by incrementing itself on **line 5**.

Let's go ahead and verify this code:

```
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 does not exist.")
            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


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


print(llist.len_iterative())
```

class Node and class LinkedList

In the code above, we have a linked list object `llist` and we insert four entries into the linked list (**lines 100-103**).

The statement on **line 106** `print(llist.len_iterative())` gives an output of 4 which proves that our implementation is correct.

Recursive Implementation

Let's move on to the recursive implementation of calculating the length of a linked list:

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

len_recursive(self, node)

In the implementation of `len_recursive`, we pass in a `node` to the method.
Now if we want to calculate the length of the whole linked list, we have to pass

Now if we want to calculate the length of the whole linked list, we have to pass

the start of the linked list as the `node` on **line 1**. On **line 4**, we have a recursive call to `self.len_recursive` where we pass `node.next` to it.

Now, whenever we have a recursive function, we need a base case. For the `len_recursive` method, the base case is whether or not we've encountered the end of the linked list. If we reach the end of the linked list, meaning the `node` is `None`, we return zero on **line 3**. Otherwise, if the `node` is not `None`, we call `len_recursive` on **line 4** and pass in the next node. Also on **line 4**, we return `1` plus what we're going to return from `self.len_recursive(node.next)`.

Now we'll call this method in a way similar to the iterative method, but we're going to pass the node that corresponds to the head of the linked list to this method.

```
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 does not exist.")
            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):
        if self.head:
            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()
print("The length of an empty linked list is:")
print(llist.len_recursive(llist.head))
llist.append("A")
llist.append("B")
llist.append("C")
llist.append("D")

print("The length of the linked list calculated recursively after inserting 4 elements is:")
print(llist.len_recursive(llist.head))
print("The length of the linked list calculated iteratively after inserting 4 elements is:")
print(llist.len_iterative())
```

class Node and class LinkedList

As you can see from the code above, we get output equal to `4` from the `len_recursive` method. If the linked list is empty, this method returns zero.

In conclusion, it doesn't matter if we calculate the length of a linked list iteratively or recursively, we will always get the same answer.

I hope you enjoyed the lesson!

In the next lesson, we'll learn how to swap two nodes in a linked list. See you there!