Laboratory 2 - Singly-Linked Lists

CS 2302 – Data structures summer 2019

Dilan Ramirez

INSTRUCTOR: OLAC FUENTES

TEACHING ASSISTANTS (TA): ISMAEL VILLANUEVA-MIRANDA

JULY 1, 2019

Spring 2019

Contents

[Introduction 2](#_Toc12906509)

[Implementation 2](#_Toc12906510)

[Running Time 3](#_Toc12906511)

[Experimental results 4](#_Toc12906512)

[Conclusion 6](#_Toc12906513)

[Appendix 7](#_Toc12906514)

[Honesty Certification 11](#_Toc12906515)

# Introduction

1. **Copy**

Implement the following function “Copy(list)” Builds and returns a copy of list.

1. **ItemAt**

Implement the following function that returns the data item at position i in list.

1. **Pop**

Implement the following function that Remove item at position i in list. If i is not specified, it removes the first item in list.

1. **Count**

Implement the following function that returns the number of times x appears in list.

1. **Index**

Implement the following function that returns the index of the first item whose value is equal to x list.

1. **Clear**

Implement the following function removes all items from list.

1. **Sublist**

Implement the following function builds and returns a sublist of list, from element start to

element end (not inclusive).

1. **Reverse**

Implement the following function that reverses the elements in list (in place)

# Implementation

1. **Copy**

I traversed the whole list as we do when printing. The difference was that in this case instead of printing the item, I appended all the elements in another list. Finally, I just returned the list.

1. **ItemAt**

First, I check if the list is empty. If it is empty, I return None. Then I check if the item desired is at position 0. If so, I return L.head.item. if both cases are not required, I traverse the list using a count until the counter I equal to the position i.

1. **Pop**

First, I check if the list is empty. If it is empty, I return None. Then I check if the value to delete is L.head.item. if so, I delete that node. then, I traverse the whole list until finding the value to pop and delete it.

1. **Count**

First, I check if the list is empty. If it is empty, I return None. I set a counter, Then, I traverse the whole list. f the element of the list is equal to x, the counter is increased. Finally, I return the counter

1. **Index**

First, I check if the list is empty. If it is empty, I return None. I set a counter, Then, I traverse the whole list. The counter is increased until the item is found. Finally, I return the counter

1. **Clear**

I traverse the whole list and using the pop function, I delete every node.

1. **Sublist**
2. **Reverse**

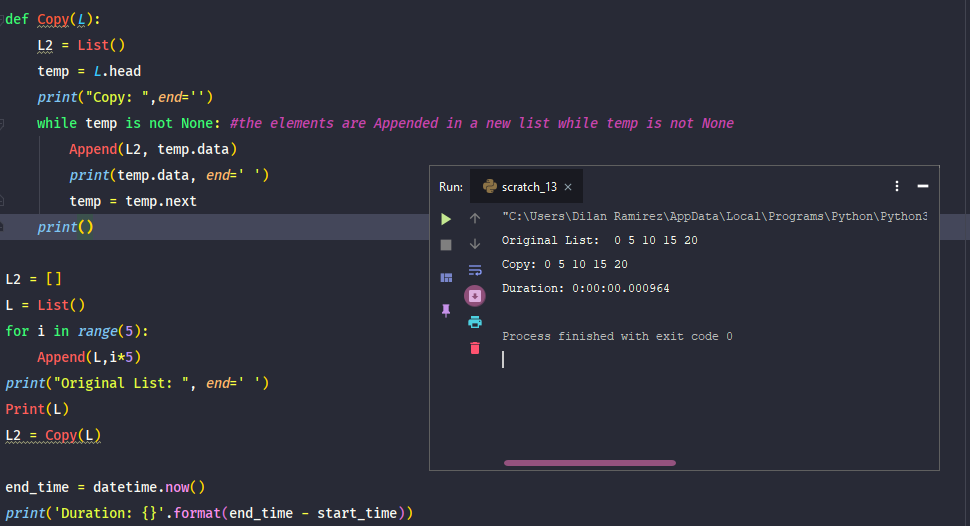
First, I check if the list is empty. If it is empty, I return None. Then, a new pointer is created to then connect them. Finally the list is traverse and the elements are printed in reverse order.

# Running Time

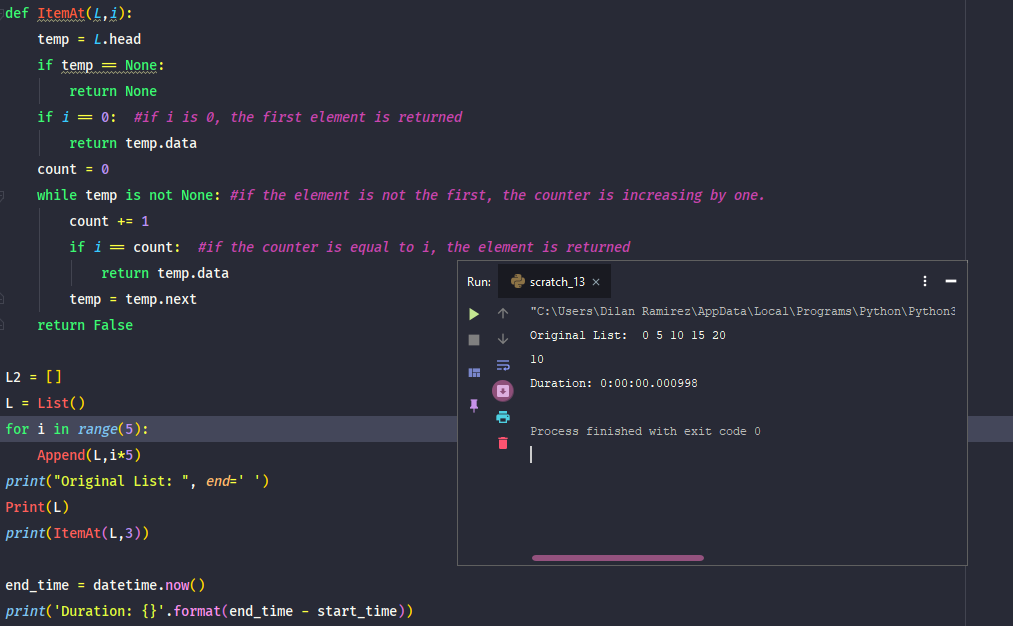
1. Big-O: O (n)
2. Big-O: O (n)
3. Big-O: O (n)
4. Big-O: O (n)
5. Big-O: O (n)
6. Big-O: O (n^2)
7. Big-O: O (n)
8. Big-O: O (n)

# Experimental results

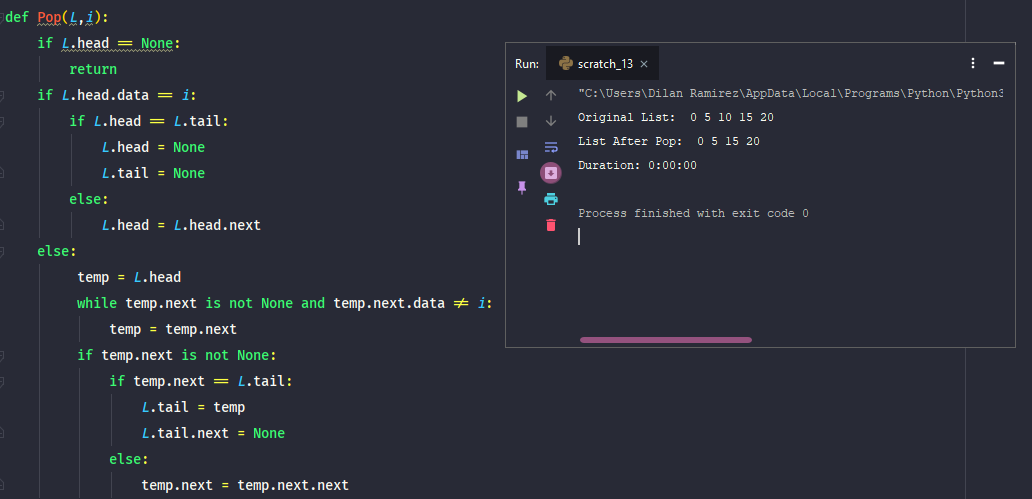
**a.**



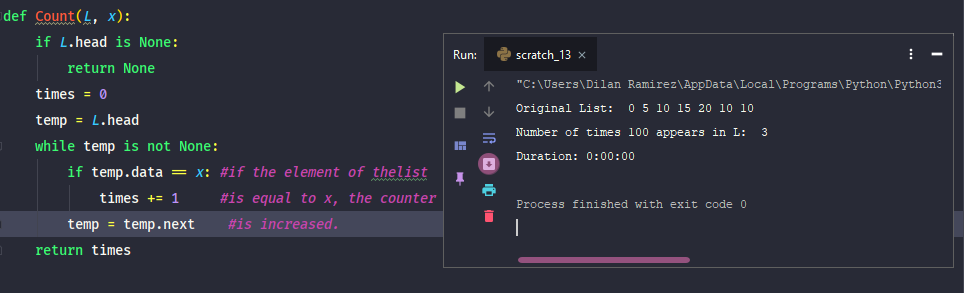
**b.**



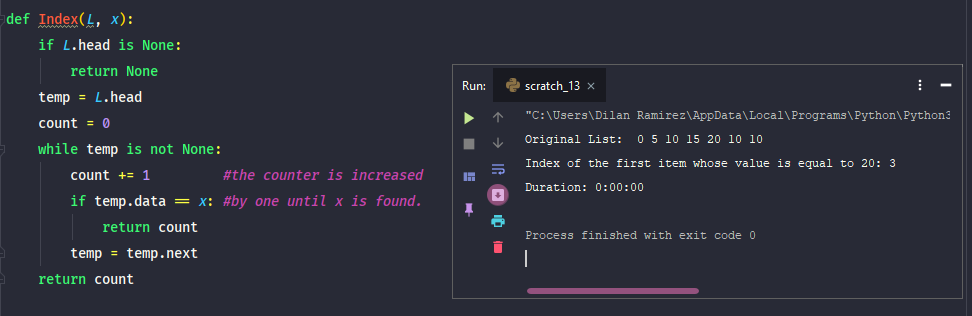
**c.**



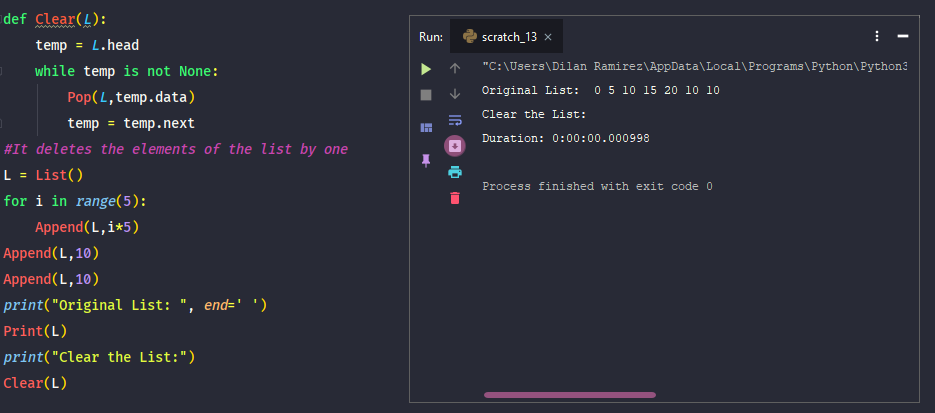
**d.**



**e.**

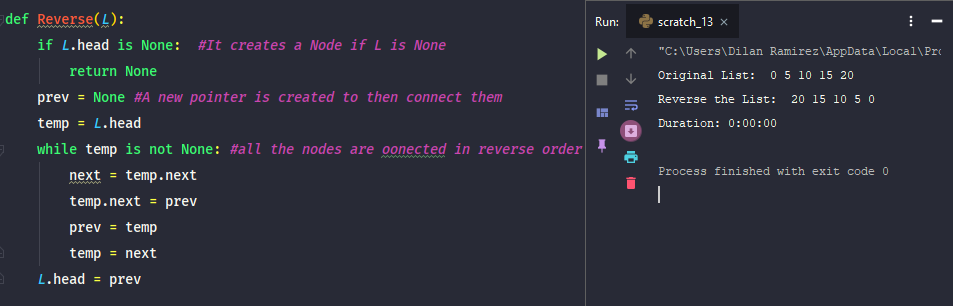


**f.**



**g.**

**h.**



# Conclusion

By doing this lab, I learned how linked-lists work, and what the problems are by working with them. Some of the problems are, connecting the nodes after manipulating them, and keep track of the head, tail and next of every node.

# Appendix

#Exercise A

def Copy(L):

L2 = List()

temp = L.head

print("Copy: ",end='')

while temp is not None: #the elements are Appended in a new list while temp is not None

Append(L2, temp.data)

print(temp.data, end=' ')

temp = temp.next

print()

#Exercise B

def ItemAt(L,i):

temp = L.head

if temp == None:

return None

if i == 0: #if i is 0, the first element is returned

return temp.data

count = 0

while temp is not None: #if the element is not the first, the counter is increasing by one.

count += 1

if i == count: #if the counter is equal to i, the element is returned

return temp.data

temp = temp.next

return False

#Exercise C

def Pop(L,i):

if L.head == None:

return

if L.head.data == i:

if L.head == L.tail:

L.head = None

L.tail = None

else:

L.head = L.head.next

else:

temp = L.head

while temp.next is not None and temp.next.data != i:

temp = temp.next

if temp.next is not None:

if temp.next == L.tail:

L.tail = temp

L.tail.next = None

else:

temp.next = temp.next.next

#Exercise D

def Count(L, x):

if L.head is None:

return None

times = 0

temp = L.head

while temp is not None:

if temp.data == x: #if the element of thelist is equal to x, the counter is increased.

times += 1

temp = temp.next

return times

#Exercise E

def Index(L, x):

if L.head is None:

return None

temp = L.head

count = 0

while temp is not None:

count += 1 #the counter is increased by one until x is found.

if temp.data == x:

return count

temp = temp.next

return count

#Exercise F

def Clear(L):

temp = L.head

while temp is not None:

Pop(L,temp.data) #It deletes the elements of the list by one

temp = temp.next

# Exercise G

def Sublist(L, start, end):

if L.head is None:

return None

t = L.head

count = 0

stop = GetLength(L)

while t is not None:

if start == 1 and end == stop:

Append(M,t.data)

t = t.next

return M

#Exercise H

def Reverse(L):

if L.head is None: #It creates a Node if L is None

return None

prev = None #A new pointer is created to then connect them

temp = L.head

while temp is not None: #all the nodes are oonected in reverse order

next = temp.next

temp.next = prev

prev = temp

temp = next

L.head = prev

# Table 4.1.1

def Append(L, x):

if L.head is None: #It creates a Node if L is None

L.head = Node(x)

L.tail = L.head

else: # If L is not None, it add a new Node after the first one starting by the tail

L.tail.next = Node(x)

L.tail = L.tail.next

def Prepend(L,x):

if L.head is None: #It creates a Node if L is None

L.head = Node(x)

L.head = L.tail

else: # If L is not None, it add a new Node after the first one starting by the head

node = Node(x)

node.next = L.head

L.head = node

print("Prepends item: ", end=' ')

Print(L)

def insertAfter(L,w,x):

if L.head is None: #It creates a Node if L is None

L.head = Node(x)

L.tail = L.head

temp = L.head

if x == 0: #if the position is 0, the Node is Prepended

Prepend(L,x)

node = Node(x)

while temp is not None: #if not, the value is searched until it is found, so a new node is added after it.

if temp.data == w:

node.next = temp.next

temp.next = node

temp = temp.next

Print(L)

def Remove(L,x):

if L.head == None: #It creates a Node if L is None

return None

if L.head.data == x: #if the first element is x, so it is deleted

if L.head == L.tail:

L.head = None

L.tail = None

else:

L.head = L.head.next

else: #if not, the elements is searched hutil it is found, and then deleted

temp = L.head

while temp.next != None and temp.next.data != x:

temp = temp.next

if temp.next != None:

if temp.next == L.tail:

L.tail = temp

L.tail.next = None

else:

temp.next = temp.next.next

Print(L)

def Search(L,x):

if L.head is None: #It creates a Node if L is None

return False

temp = L.head

while temp is not None: #the element is searched until it is found

if temp.data == x:

return True #if it is found, a True statement is returned

temp = temp.next

return False #if it is not found, a False statement is returned

def Print(L):

t = L.head

while t is not None:

print(t.data, end=' ')

t = t.next

print()

def IsEmpty(L):

return L.head == None

def PrintReverse(L):

if L.head is None: #It creates a Node if L is None

return None

prev = None #A new pointer is created to then connect them

temp = L.head

while temp is not None: #all the nodes are connected in reverse order

next = temp.next

temp.next = prev

prev = temp

temp = next

L.head = prev

Print(L)

def Sort(L):

state = True

while state:

state = False

temp = L.head

while temp.next is not None:

if temp.data > temp.next.data: #the elementes are switched until the condition is not true

currentnode = temp.data

temp.data = temp.next.data

temp.next.data = currentnode

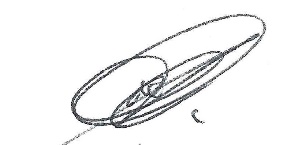
state = True

temp = temp.next

Print(L)

# Honesty Certification

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provide inappropriate assistance to any student in the class.

 07/ 01 / 2019

Dilan Ramirez Date