Week-5 Practical

CP2410: Algorithm and Data Structure

Nguyen Quoc Minh Quan

13740328

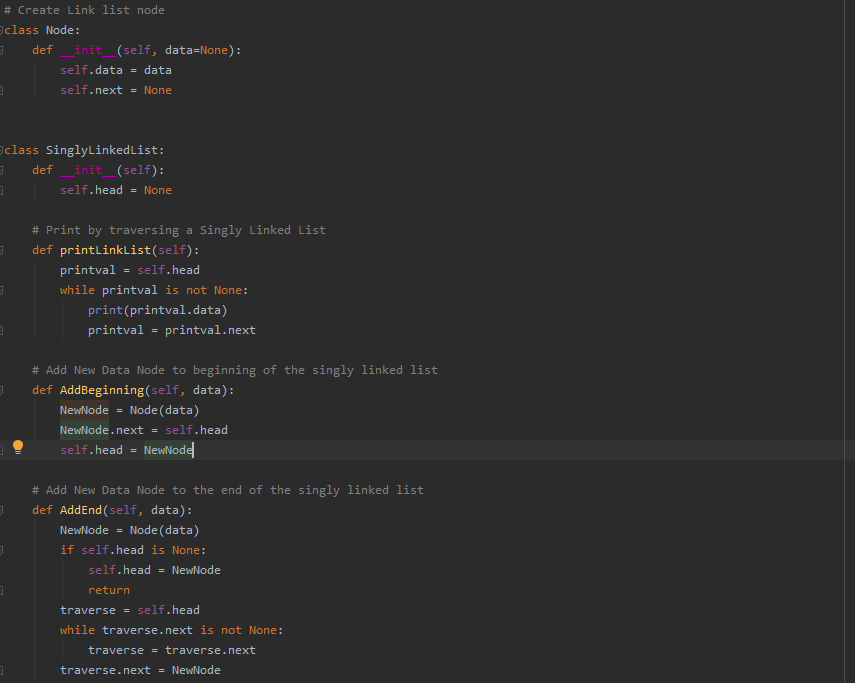
GitHub: <https://github.com/minhquan0902/CP2410--Practicals.git>

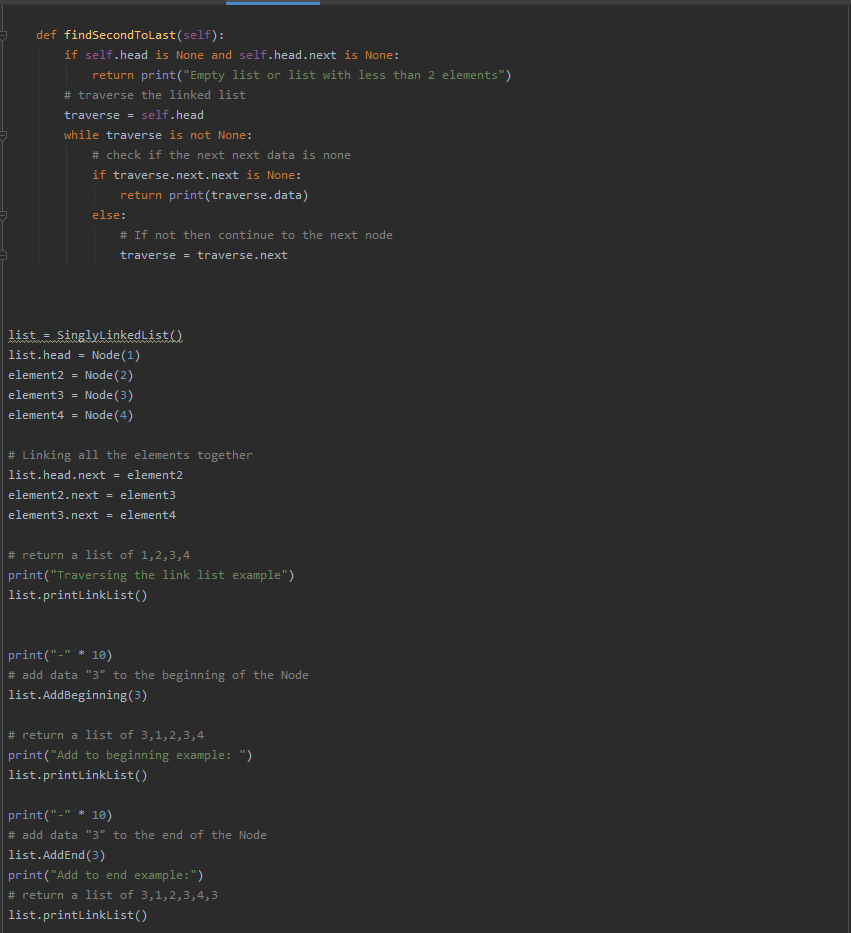
Task 1: (R-7.1) Give an algorithm for finding the second-to-last node in a singly linked list in which the last node is indicated by a next reference of None.

The idea is to traverse the linked list following the below approach:

1. If the list is empty or contains less than 2 elements, return false.
2. Otherwise check if the current node is the second last node of the linked list or not. That is, **if (current\_node->next-next == NULL)** then the current node is the second last node.
3. If the current node is the second last node, print the node otherwise move to the next node.
4. Repeat the above two steps until the second last node is reached.

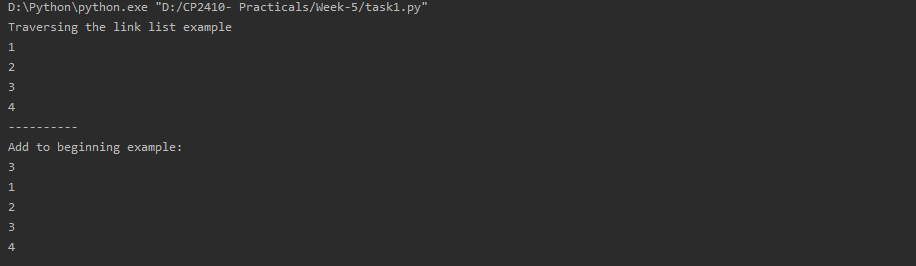
Code execution:

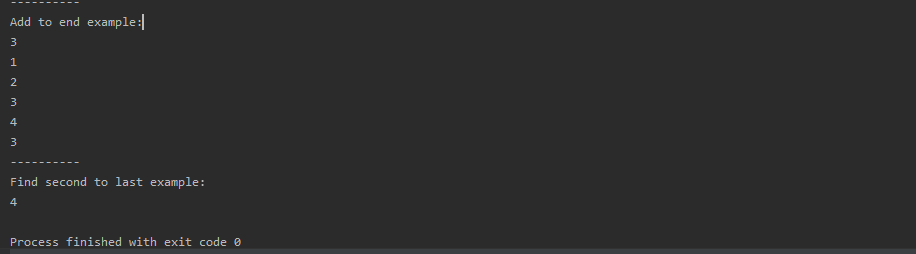






Result:



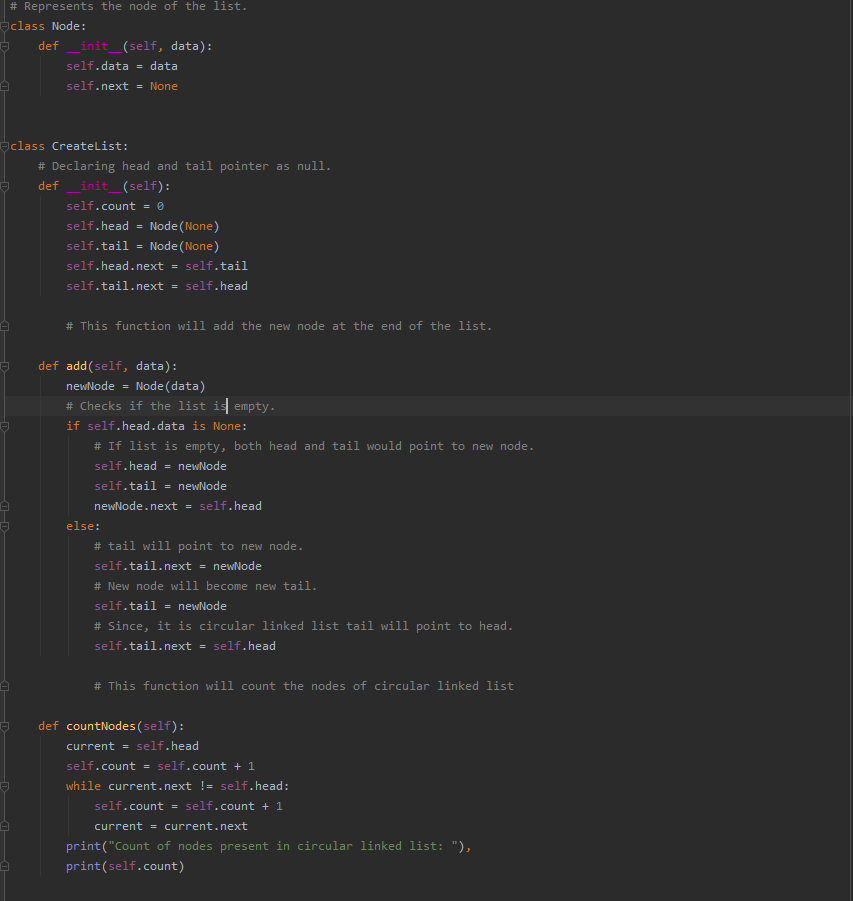


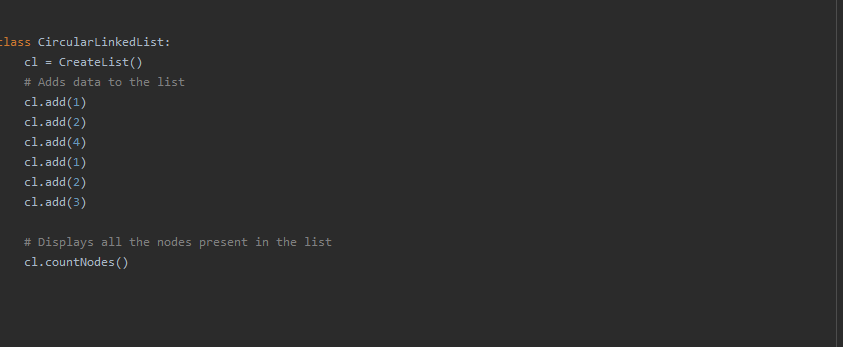
Task 2: (R-7.5) Implement a function that counts the number of nodes in a circularly linked list.

Algorithm

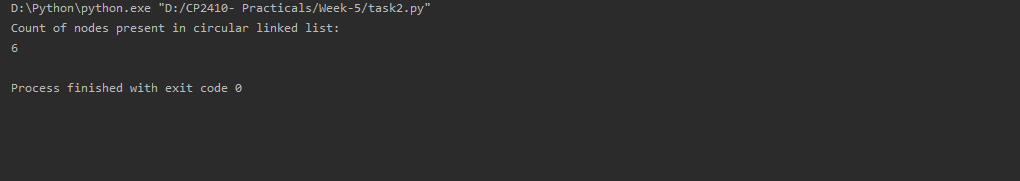
1. Define a Node class which represents a node in the list. It has two properties data and next which will point to the next node.
2. Define another class for creating the circular linked list and it has two nodes: head and tail. It has two methods: add () and display ().
3. add () will add the node to the list:
   1. It first checks whether size is null or head is null; then it will insert the node as the head.
   2. Both head and tail will point to a newly added node.
   3. If the head is not null, the new node will be the new tail, and new tail will point to the head as it is a circular linked list.
4. countNodes() will count the number of nodes present in the list.
   1. Define new node current which will point to the head node.
   2. Traverse through the list to count the nodes by making the current node to point to next node in the list till current points to head again.

Code Execution:



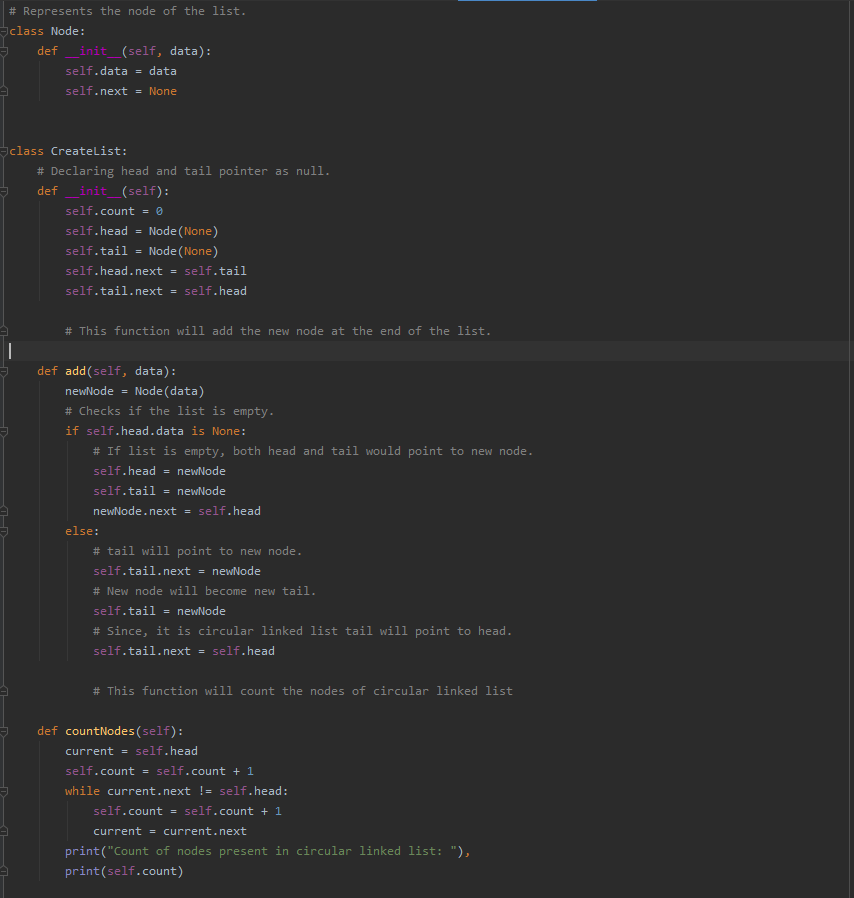


Result:



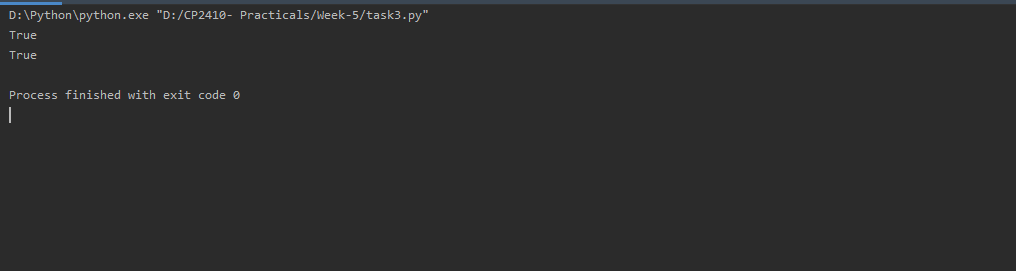
Task 3: (R-7.6) Suppose that x and y are references to nodes of circularly linked lists, although not necessarily the same list. Describe a fast algorithm for telling if x and y belong to the same list.

Code execution:





Result:



Task 5: (R-7.11) Implement a function, L.max(), that returns the maximum element from a PositionalList (ch07/positional\_list.py) instance L. Assume all values in L are numbers

# Redo the previously problem with max as a method of the PositionalList

# class, so that calling syntax L.max() is supported.

pp = PositionalList()

print(pp, "max:", pp.max()) # empty list

for \_ in range(10):

pp.add\_first(random.randint(0, 100))

print(pp, "max:", pp.max())

# recursion version

pp = PositionalList()

print(pp, "max:", pp.max\_recursion()) # empty list

for \_ in range(5):

pp.add\_first(random.randint(0, 100))

print(pp, "max:", pp.max\_recursion().element())