**CS 2302 – Linked Lists**

You are given the following two classes.

**class** Node:  
 **def** \_\_init\_\_(self, item=**None**, next=**None**):  
 self.item = item  
 self.next = next

**class** SinglyLinkedList:  
 **def** \_\_init\_\_(self, head=**None**):  
 self.head = head

1. Implement the following non-static methods inside of you SinglyLinkedList class

* addLast(item) <- Method that receives an item as input, creates a node that stores this value, and appends it to the end of this list
* addFirst(item) <- Method that receives an item as input, creates a node that stores this value, and appends it to the beginning of the list
* add(index, item) <- Method that receives an item as input, creates a node that stores this value, and adds it to the list at the specified location (index)
* clear() <- Method that removes all elements in the list (Hint: you only need one line of code to do this)
* contains(item) <- Method that receives an item as input and returns true if the value is stored in the linked list; false, otherwise.
* getIndexOf(item) <- Method that receives an item as input and returns index of the node that stores *item.* If item is not in the list, you should return -1
* get(index) <- Method that returns the item associated with the node at position *index* in the list. Return None if there is not item at that specific index.
* getFirst() <- Method that returns the item associated with the first node in the list
* getLast() <- Method that returns the item associated with the last node in the list
* remove(index) <- Method that removes the node at position *index* in the list if it exists. You should return *true* if the index was valid and the node was removed; false, if index was invalid.
* removeFirst() <- Method that removes the first node in the list if it exists. You should return *true* if the node was removed; false, if the list was empty.
* removeLast() <- Method that removes the last node in the list if it exists. You should return *true* if the node was removed; false, if the list was empty.
* size() <- Method that returns the number of nodes in the list
* isEmpty() <- Method that true if the list is empty; false, otherwise.
* printList() <- Method that prints all items in the list (starting from the first node)

**Hint:**

* Your methods should call other methods in the class. For example, to implement addFirst(item), you can just call add(0, item). This makes it easier to implement methods with similar logic.

1. Solve the following problems

**2.1 Reverse Linked List**

Write a method called *reverse\_list.* As the name suggests, this method reverses the items stored inside of a singly linked list. This method is to be added to your *SinglyLinkedList* class. Because of this, it does not receive any parameters. You can use self.head to access the head of the list

**Example:**

**Before calling the method:** 1->2->3->4->5->None

**After calling the method:** 5->4->3->2->1->None

**2.2 Remove Duplicates from Sorted List**

Write a method called *delete\_duplicates*. This method is also meant to be added to your SinglelinkedList class. It deletes all duplicates such that each element in the list appears only once.

**Example 1:**

**Before calling the method**: 1->1->2

**After calling the method**: 1->2

**Example 2:**

**Before calling the method**: 1->1->2->3->3

**After calling the method**: 1->2->3

**2.3 Extra Credit**

Write a method called *has\_cycle.* This method is meant to be added to you SinglyLinkedList class. This method returns true if there is a cycle in the list; false, otherwise. Can you solve this using O(1) - constant memory?

**Example 1:**

**Return True**

A drawing of a person

Description automatically generated

**Example 2:**

**Return True**

A close up of a clock

Description automatically generated

**Example 3:**

**Return False**

A picture containing object

Description automatically generated