1. Given a singly linked list, find middle of the linked list. For example, if given linked list is 1->2->3->4->5 then output should be 3. DONE

If there are even nodes, then there would be two middle nodes, we need to print second middle element. For example, if given linked list is 1->2->3->4->5->6 then output should be 4.

class Solution(object):

def middleNode(self, head):

slow = fast = head

while fast and fast.next:

slow = slow.next

fast = fast.next.next

return slow

1. Given a singly linked list, rotate the linked list counter-clockwise by k nodes. Where k is a given positive integer smaller than or equal to length of the linked list. For example, if the given linked list is

10->20->30->40->50->60 and k is 4, the list should be modified to 50->60->10->20->30->40. DONE

rotate the linked list, we need to change next of kth node to NULL, next of last node to previous head node, and finally change head to (k+1)th node. So we need to get hold of three nodes: kth node, (k+1)th node and last node.

Traverse the list from beginning and stop at kth node. Store pointer to kth node. We can get (k+1)th node using kthNode->next. Keep traversing till end and store pointer to last node also. Finally, change pointers as stated above.

# This function rotates a linked list counter-clockwise and updates the head. The function assumes that k is smaller than size of #linked list.

    def rotate(self, k):

        if k == 0:

            return

        # Let us understand the below code for example k = 4

        # and list = 10->20->30->40->50->60

        current = self.head

        # current will either point to kth or NULL after

        # this loop

        # current will point to node 40 in the above example

        count = 1

        while(count <k and current is not None):

            current = current.next

            count += 1

        # If current is None, k is greater than or equal

        # to count of nodes in linked list. Don't change

        # the list in this case

        if current is None:

            return

        # current points to kth node. Store it in a variable

        # kth node points to node 40 in the above example

        kthNode = current

        # current will point to lsat node after this loop

        # current will point to node 60 in above example

        while(current.next is not None):

            current = current.next

        # Change next of last node to previous head

# Change next of last node to previous head

        # Next of 60 is now changed to node 10

        current.next = self.head

        # Change head to (k+1)th node

        # head is not changed to node 50

        self.head = kthNode.next

        # change next of kth node to NULL  next of 40 is not NULL

        kthNode.next = None

1. Given a linked list, write a function to reverse every k nodes (where k is an input to the function).If a linked list is given as 1->2->3->4->5->6->7->8->NULL and k = 3 then output will be 3->2->1->6->5->4->8->7->NULL. DONE

Algorithm: *reverse(head, k)*

1) Reverse the first sub-list of size k. While reversing keep track of the next node and previous node. Let the pointer to the next node be *next* and pointer to the previous node be *prev*.

2) *head->next = reverse(next, k)* /\* Recursively call for rest of the list and link the two sub-lists \*/

3) return *prev* /\* *prev* becomes the new head of the list (see the diagrams of iterative method of [this post)](https://www.geeksforgeeks.org/reverse-a-linked-list/) \*/

class Node:

    # Constructor to initialize the node object

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

class LinkedList:

    # Function to initialize head

    def \_\_init\_\_(self):

        self.head = None

    def reverse(self, head, k):

        current = head

        next  = None

        prev = None

        count = 0

        # Reverse first k nodes of the linked list

        while(current is not None and count < k):

            next = current.next

            current.next = prev

            prev = current

            current = next

            count += 1

1. Given a linked list, check if the the linked list has a loop. Linked list can contain self loop. DONE

class Node:

    # Constructor to initialize the node

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

class LinkedList:

    # Function to initialize head

    def \_\_init\_\_(self):

        self.head = None

    # Function to insert a new node at the beginning

    def push(self, new\_data):

        new\_node = Node(new\_data)

        new\_node.next = self.head

        self.head = new\_node

    def printList(self):

        temp = self.head

        while(temp):

            print (temp.data,end=" ")

            temp = temp.next

  def detectLoop(self):

         s = set()

         temp=self.head

         while (temp):

            if (temp in s):

                return True

            # If we are seeing the node for the first time, insert it in hash

            s.add(temp)

            temp = temp.next

             return False

1. Given a linked list, the task is to find the n'th node from the end. DONE

class Node:

    def \_\_init\_\_(self, new\_data):

        self.data = new\_data

        self.next = None

class LinkedList:

    def \_\_init\_\_(self):

        self.head = None

    # createNode and and make linked list

    def push(self, new\_data):

        new\_node = Node(new\_data)

        new\_node.next = self.head

        self.head = new\_node

    # Function to get the nth node from

    # the last of a linked list

    def printNthFromLast(self, n):

        temp = self.head # used temp variable

        length = 0

        while temp is not None:

            temp = temp.next

            length += 1

        # print count

        if n > length: # if entered location is greater

                       # than length of linked list

            print('Location is greater than the' +

                         ' length of LinkedList')

            return

        temp = self.head

        for i in range(0, length - n):

            temp = temp.next

        print(temp.data)

1. Given two linked lists sorted in ascending order. Merge them in-place and return head of the merged list.   For example, if first list is 10->20->30 and second list is 15->17, then the result list should be 10->15->17->20->30. DONE

It is strongly recommended to do merging in-place using O(1) extra space.

Using Recursion - stack space

# Node class

class Node:

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

# Constructor to initialize the node object

class LinkedList:

    # Function to initialize head

    def \_\_init\_\_(self):

        self.head = None

    # Method to print linked list

    def printList(self):

        temp = self.head

        while temp :

            print(temp.data, end="->")

            temp = temp.next

    # Function to add of node at the end.

    def append(self, new\_data):

        new\_node = Node(new\_data)

        if self.head is None:

            self.head = new\_node

            return

        last = self.head

        while last.next:

            last = last.next

 last.next = new\_node

# Function to merge two sorted linked list.

def mergeLists(head1, head2):

    temp = None

    # List1 is empty then return List2

    if head1 is None:

        return head2

    # if List2 is empty then return List1

    if head2 is None:

        return head1

    # If List1's data is smaller or equal to List2's data

    if head1.data <= head2.data:

        temp = head1

        # Again check List1's data is smaller or equal List2's  data and call mergeLists function.

        temp.next = mergeLists(head1.next, head2)

    else:

             temp = head2

        temp.next = mergeLists(head1, head2.next)

    return temp

1. Given a **L**inked **L**ist where every node represents a linked list and contains two pointers of its type: DONE  
   (i) a **next**pointer to the next node  
   (ii) a**bottom** pointer to a linked list where this node is head.  
     
   You have to**flatten** the linked list to a **single linked list** which is

**For Ex:**Shown below is a given linked list

           5 -> 10 -> 19 -> 28       |    |     |     |       V    V     V     V       7    20    22    35       |          |     |       V          V     V       8          50    40       |                |       V                V       30               45

and after flattening it, the sorted linked list looks like:

 5->7->8->10->19->20->22->28->30->35->40->45->50

The  node structure has **3** fields as mentioned. A **next pointer**, a **bottom** pointerand a **data** part.

class Node:

def \_\_init\_\_(self, node\_data):

self.data = node\_data

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

self.tail = None

def insert\_node(self, node\_data):

node = Node(node\_data)

if not self.head:

self.head = node

else:

self.tail.next = node

self.tail = node

# Complete the printLinkedList function below.

def printList(self):

while self.head is not None:

print(self.head.data)

self.head = self.head.next

1. Given a singly linked list, write a function to swap elements pairwise. For example, if the linked list is 1->2->3->4->5 then the function should change it to 2->1->4->3->5, and if the linked list is 1->2->3->4->5->6 then the function should change it to 2->1->4->3->6->5. DONE

Iterative - Start from the head node and traverse the list. While traversing swap data of each node with its next node’s data.

# Node class

class Node:

    # Constructor to initialize the node object

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

class LinkedList:

    # Function to initialize head

    def \_\_init\_\_(self):

        self.head = None

    # Function to pairwise swap elements of a linked list

    def pairwiseSwap(self):

        temp = self.head

        # There are no nodes in ilnked list

        if temp is None:

            return

        # Traverse furthur only if there are at least two

        #  left

        while(temp is not None and temp.next is not None):

            # Swap data of node with its next node's data

            temp.data, temp.next.data = temp.next.data, temp.data

            # Move temo by 2 fro the next pair

            temp = temp.next.next

    # Function to insert a new node at the beginning

    def push(self, new\_data):

        new\_node = Node(new\_data)

        new\_node.next = self.head

        self.head = new\_node

    # Utility function to prit the linked LinkedList

    def printList(self):

        temp = self.head

        while(temp):

            print temp.data,

            temp = temp.next

1. Given two numbers represented by two lists, write a function that returns sum list. The sum list is list representation of addition of two input numbers. DONE

Suppose you have two linked list 5->4 ( 4 5 )and 5->4->3( 3 4 5) you have to return  a resultant linked list 0->9->3 (3 9 0).

Traverse both lists. One by one pick nodes of both lists and add the values. If sum is more than 10 then make carry as 1 and reduce sum.

# Node class

class Node:

    # Constructor to initialize the node object

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

class LinkedList:

    def \_\_init\_\_(self):

        self.head = None

    # Function to insert a new node at the beginning

    def push(self, new\_data):

        new\_node = Node(new\_data)

        new\_node.next = self.head

        self.head = new\_node

    # Add contents of two linked lists and return the head node of resultant list

    def addTwoLists(self, first, second):

        prev = None

        temp = None

        carry = 0

        while(first is not None or second is not None):

   # Calculate the value of next digit in

            fdata = 0 if first is None else first.data

            sdata = 0 if second is None else second.data

            Sum = carry + fdata + sdata

            carry = 1 if Sum >= 10 else 0

            # update sum if it is greater than 10

            Sum = Sum if Sum < 10 else Sum % 10

            # Create a new node with sum as data

            temp = Node(Sum)

            # if this is the first node then set it as head

            if self.head is None:

                self.head = temp

            else :

                prev.next = temp

            # Set prev for next insertion

            prev = temp

            # Move first and second pointers to next nodes

            if first is not None:

                first = first.next

            if second is not None:

                second = second.next

        if carry > 0:

            temp.next = Node(carry)

1. Given a singly linked list of integers, Your task is to complete the function isPalindrome that returns true if the given list is palindrome, else returns false. DONE

Using recursion:

Use two pointers left and right. Move right and left using recursion and check for following in each recursive call.

1) Sub-list is palindrome.

2) Value at current left and right are matching.

If both above conditions are true then return true.

def previous\_node(self, node):

current = self.head

while current != node:

prev = current

current = current.next

return prev

def utility\_pal(self, left, right):

while left != right:

if left.data == right.data:

right\_prev = self.previous\_node(right)

return self.utility\_pal(left.next, right\_prev)

else:

return("Not palindrome")

return("Palindrome")

def palindrome\_rec(self):

if self.head is None:

return

right = self.head

while right.next:

right = right.next