```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


class LinkedList:
    def __init__(self):
        self.head = None

    def print_list(self):
        current = self.head
        while current:
            print(current.data)
            current = current.next

    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)

    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 in the list")
            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 = 0
        print("BHi")
        while cur_node and count != pos:
            print("Hi")
            prev = cur_node
            cur_node = cur_node.next
            count = count + 1

        if cur_node is None:
            return

        # print(cur_node.data)
        prev.next = cur_node.next
        cur_node = None

    def swap_nodes(self, key_1, key_2):
        if key_1 == key_2:
            return

        prev_1 = None
        curr_1 = self.head

        while curr_1 and curr_1.data != key_1:
            prev_1 = curr_1
            curr_1 = curr_1.next

        prev_2 = None
        curr_2 = self.head
        while curr_2 and curr_2.data != key_2:
            prev_2 = curr_2
            curr_2 = curr_2.next

        if not curr_1 or not curr_2:
            return

        if prev_1:
            prev_1.next = curr_2
        else:
            self.head = curr_2

        if prev_2:
            prev_2.next = curr_1
        else:
            self.head = curr_1

        curr_1.next, curr_2.next = curr_2.next, curr_1.next

    def reversed_iterative(self):
        prev = None
        cur = self.head
        while cur:
            nxt = cur.next
            cur.next = prev
            prev = cur
            cur = nxt

        self.head = prev

    def reverse_recursive(self):

        def _reverse_recursive(cur, prev):
            if not cur:
                return prev

            nxt = cur.next
            cur.next = prev
            prev = cur
            cur = nxt
            return _reverse_recursive(cur, prev)

        self.head = _reverse_recursive(cur=self.head, prev=None)

    def merge_sorted(self, llist):
        p = self.head
        q = llist.head
        s = None

        if not p:
            return q
        if not q:
            return p

        if p and q:
            if p.data <= q.data:
                s = p
                p = s.next
            else:
                s = q
                q = s.next
            new_head = s
        while p and q:
            if p.data <= q.data:
                s.next = p
                s = p
                p = s.next
            else:
                s.next = q
                s = q
                q = s.next
        if not p:
            s.next = q
        if not q:
            s.next = p
        return new_head

    def remove_duplicates(self):

        cur = self.head
        prev = None

        dup_values = dict()

        while cur:
            if cur.data in dup_values:
                # Remove Node
                prev.next = cur.next
                cur = None
            else:
                # Have not encountered element before
                dup_values[cur.data] = 1
                prev = cur
            cur = prev.next

    def find_node(self, pos):

        cur = self.head
        count = 0
        if count == pos:
            return cur.data
        while cur and count != pos:
            count += 1
            cur = cur.next
            if count == pos:
                return cur.data
        return cur

    def print_nth_node_last(self, n):

        # Method 1
        # total_len = self.len_iterative()
        # cur = self.head
        # while cur:
        #     if total_len == n:
        #         print(cur.data)
        #         return cur
        #     total_len -= 1
        #     cur = cur.next
        # if cur is None:
        #     return

        # Method 2

        p = self.head
        q = self.head

        count = 0
        while q and count < n:
            q = q.next
            count += 1

        if not q:
            print(str(n) + " is greater than the number of nodes in list. ")
            return

        while p and q:
            p = p.next
            q = q.next
        return p.data

    def count_occurences_iterative(self, data):
        count = 0
        cur = self.head

        while cur:
            if cur.data == data:
                count += 1
            cur = cur.next
        return count

    def count_occurences_recursive(self, node, data):
        if not node:
            return 0

        if node.data == data:
            return 1 + self.count_occurences_recursive(node.next, data)
        else:
            return self.count_occurences_recursive(node.next, data)

    def rotate_list(self, k):
        p = self.head
        q = self.head

        prev = None
        count = 0

        while p and count < k:
            prev = p
            p = p.next
            q = q.next
            count += 1
        p = prev

        while q:
            prev = q
            q = q.next
        q = prev
        q.next = self.head
        self.head = p.next
        p.next = None

    def is_palindrome(self):
        cur = self.head
        str1 = ""
        while cur:
            str1 += cur.data
            cur = cur.next
        return str1 == str1[::-1]

    def move_tail_to_head(self):
        last_node = self.head
        second_to_last_node = None

        while last_node.next:
            second_to_last_node = last_node
            last_node = last_node.next
        last_node.next = self.head
        second_to_last_node.next = None
        self.head = last_node

    def sum_two_lists(self, llist):
        p = self.head
        q = llist.head

        sum_list = LinkedList()
        carry = 0
        while p or q:
            if not p:
                i = 0
            else:
                i = p.data
            if not q:
                j = 0
            else:
                j = q.data
            s = i + j + carry
            if s >= 10:
                carry = 1
                remainder = s % 10
                sum_list.append(remainder)
            else:
                carry = 0
                sum_list.append(s)
            if p:
                p = p.next
            if q:
                q = q.next
        sum_list.print_list()
```