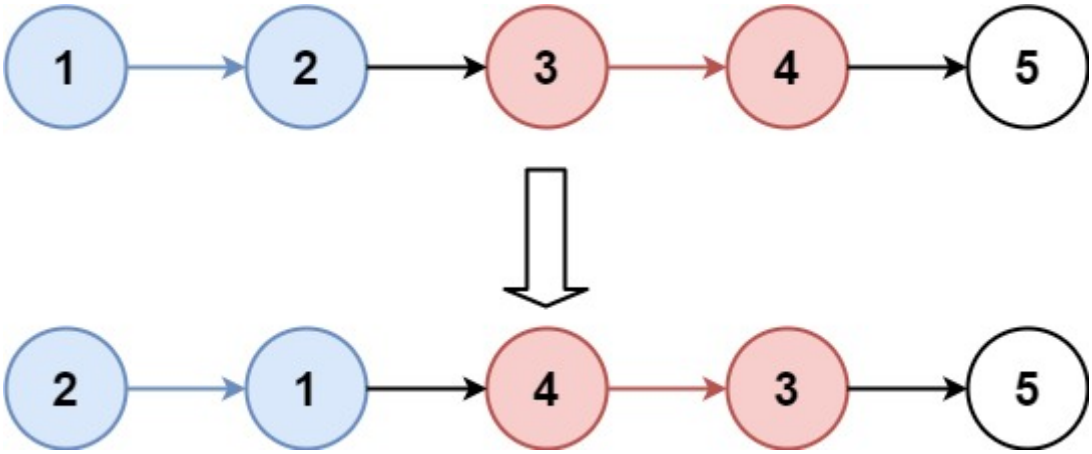
25. Reverse Nodes in k-Group

Given a linked list, reverse the nodes of a linked list *k* at a time and return its modified list.

k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of *k* then left-out nodes, in the end, should remain as it is.

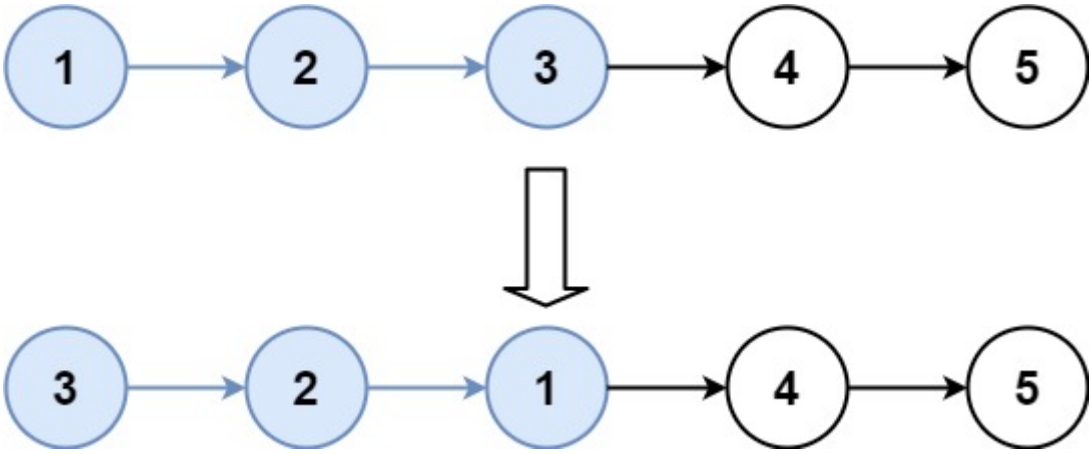
You may not alter the values in the list's nodes, only nodes themselves may be changed.

Example 1:



Input: head = [1,2,3,4,5], k = 2
Output: [2,1,4,3,5]

Example 2:



Input: head = [1,2,3,4,5], k = 3
Output: [3,2,1,4,5]

Example 3:

Input: head = [1,2,3,4,5], k = 1
Output: [1,2,3,4,5]

Example 4:

Input: head = [1], k = 1
Output: [1]

Constraints:

1. The number of nodes in the list is in the range `sz`.
2. `1 <= sz <= 5000`
3. `0 <= Node.val <= 1000`
4. `1 <= k <= sz`

```
def reverseKGroup(self, head: ListNode, k: int) -> ListNode:
        if head is None or head.next is None or k<=1:
            return head
        dummyHead = ListNode(0)
        dummyTail = ListNode(0)
        length = self.helper(head)
        curr = head
        while length>=k:
            tempHead = ListNode(0)
            tempTail = ListNode(0)
            temp = k
            while temp>0:
                nextt = curr.next
                curr.next = None
                self.addFirst(curr,tempHead,tempTail)
                curr = nextt
                temp = temp-1

            if dummyHead.next is None:
                dummyHead.next = tempHead.next
                dummyTail.next = tempTail.next
            else:
                dummyTail.next.next = tempHead.next
                dummyTail.next = tempTail.next
            length = length-k
        tempTail.next.next = curr
        return dummyHead.next
```

```
def addFirst(self,node,tempHead,tempTail):

    if tempHead.next is None:
        tempHead.next = node
        tempTail.next = node
    else:
        node.next = tempHead.next
        tempHead.next = node
def helper(self,head):
    count = 0
    curr = head
    while curr!=None:
        count = count+1
        curr = curr.next
    return count
```