Delete node in Linked List

Problem Description: Given a linked list and a position i, delete the node of ith position from Linked List iteratively. If position i is greater than length of LL, then you should return the same LL without any change. Indexing starts from 0. You don't need to print the elements, just delete the node and return the head of updated LL.

Sample Input:

```
3 -> 4 -> 5 -> 2 -> 6 -> 1 -> 9 -> null
3
```

Sample Output:

```
3 -> 4 -> 5 -> 6 -> 1 -> 9 -> null
```

How to approach?

To delete a node from a LL, all we need to do is to break the link that connects that node and its previous one, and then create a link between the previous node and the next node of the one that we wish to delete. For example, let's take a look at the sample input. We need to delete the node at index 3, which holds the value '2'. Let's call that node 'B' and let's call the node which is just behind it (at index 2) 'A'. Also, the node just ahead of 'B' can be called 'C'. So our linked list looks something like this.

```
3 -> 4 -> 5 -> 2 -> 6 -> 1 -> 9 -> null
          A    B    C
```

To recall, we need to delete node 'B'. So all we need to do is to go till node 'A' and assign node C to A's next. This will break the link between A and B, and create a new one between A and C.

Edge Cases: This way of thinking will almost always work. There are a couple of edge cases that we need to consider separately. These cases are:

1. The node to be deleted is the head node.
2. The node to be deleted is the tail node.

The first case is very easy to handle. If we need to delete the head node, we can simply return the next node of head.
For the second one, our “ABC” approach will work fine because in that case node ‘C’ will be ‘null’ and we assign that to node ‘A’ which will run perfectly fine.

Special Cases: There are some special cases to be considered as well but they are easy to handle. They are:

1. ‘head’ node is null: an empty linked list has been passed
2. The index is out of bounds of the linked list: The index is negative or it’s greater than or equal to the size of the linked list.

In both these cases, we can simply return head

The pseudo code for this approach is given on the next page

```
function deleteNode(headNode, position):

  if(headNode = null): //Special case 1
    return headNode

  if(position < 0): //Special case 2
    return headNode

  if(position = 0): //Edge case 1
    return headNode.next

  currentNode <- headNode
  currentPosition <- 0

  while(currentNode != null and currentPosition < position - 1):
    //We go till position - 1 because we want to land on the
    //node that comes just before the node to be deleted
    currentPosition <- currentPosition + 1
    currentNode <- currentNode.next

  if(currentNode = null or currentNode.next = null): //Special case 2
    return headNode

  currentNode.next <- currentNode.next.next

  return headNode
```