Question 1

Given a singly linked list, delete **middle** of the linked list. For example, if given linked list is 1->2->**3**->4->5 then linked list should be modified to 1->2->4->5.If there are **even** nodes, then there would be **two middle** nodes, we need to delete the second middle element. For example, if given linked list is 1->2->3->4->5->6 then it should be modified to 1->2->3->5->6.If the input linked list is NULL or has 1 node, then it should return NULL

Soln:

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def delete_middle_node(head):
    if not head or not head.next:
        return None

    dummy = ListNode(0)
    dummy.next = head

    slow = fast = dummy

    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next

    slow.next = slow.next.next

    return dummy.next
```

Question 2

Given a linked list of N nodes. The task is to check if the linked list has a loop. Linked list can contain self loop.

Soln:

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
```

```
        self.next = next

def has_cycle(head):
    slow = fast = head

    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next

        if slow == fast:
            return True

    return False
```

Question 3

Given a linked list consisting of L nodes and given a number N. The task is to find the Nth node from the end of the linked list.

Soln:

```
class ListNode:

    def __init__(self, val=0, next=None):

        self.val = val

        self.next = next


def nth_from_end(head, n):

    if not head:

        return None


    first = second = head


    # Move first pointer N nodes ahead
```

```
    for _ in range(n):
        if not first:
            return None
        first = first.next

    # Move both pointers together until first reaches the end
    while first:
        first = first.next
        second = second.next

    return second
```

Question 4

Given a singly linked list of characters, write a function that returns true if the given list is a palindrome, else false.

Examples:

Input: R->A->D->A->R->NULL

Output: Yes

Input: C->O->D->E->NULL

Output: No

Soln:

```
class Node:
        def __init__(self, data):
```

```
        self.data = data
        self.ptr = None



def ispalindrome(head):
    # Temp pointer
    slow = head
    # Declare a stack
    stack = []

    ispalin = True

    # Push all elements of the list
    # to the stack
    while slow != None:
        stack.append(slow.data)

        # Move ahead
        slow = slow.ptr

    # Iterate in the list again and
    # check by popping from the stack
    while head != None:
```

```
            # Get the top most element
            i = stack.pop()

            # Check if data is not
            # same as popped element
            if head.data == i:
                ispalin = True
            else:
                ispalin = False
                break

            # Move ahead
            head = head.ptr

    return ispalin

# Driver Code
# Addition of linked list
one = Node(1)
two = Node(2)
three = Node(3)
four = Node(4)
five = Node(3)
```

```
six = Node(2)

seven = Node(1)


# Initialize the next pointer

# of every current pointer

one.ptr = two

two.ptr = three

three.ptr = four

four.ptr = five

five.ptr = six

six.ptr = seven

seven.ptr = None


# Call function to check palindrome or not

result = ispalindrome(one)

print("isPalindrome:", result)
```

Question 5

Given a linked list of N nodes such that it may contain a loop.

A loop here means that the last node of the link list is connected to the node at position X(1-based index). If the link list does not have any loop, X=0.

Remove the loop from the linked list, if it is present, i.e. unlink the last node which is forming the loop.

Soln:

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def detect_and_remove_loop(head):
    if not head or not head.next:
        return

    slow = fast = head

    # Detect the loop using Floyd's Cycle Detection algorithm
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next

        if slow == fast:
            break

    if slow == fast:
        # Loop exists, find the meeting point of tortoise and hare
        slow = head
        while slow.next != fast.next:
```

```
        slow = slow.next
        fast = fast.next

    # Break the loop
    fast.next = None
```

Question 6

Given a linked list and two integers M and N. Traverse the linked list such that you retain M nodes then delete next N nodes, continue the same till end of the linked list.

Difficulty Level: Rookie

Soln:

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def retain_delete(head, M, N):
    if not head:
        return None

    current = head
    previous = None

    while current:
```

```
    # Retain M nodes
    for _ in range(M):
        if not current:
            return head
        previous = current
        current = current.next

    # Delete N nodes
    for _ in range(N):
        if not current:
            break
        current = current.next

    # Update the previous node's next pointer
    previous.next = current

return head
```

Question 7

Given two linked lists, insert nodes of second list into first list at alternate positions of first list. For example, if first list is 5->7->17->13->11 and second is 12->10->2->4->6, the first list should become 5->12->7->10->17->2->13->4->11->6 and second list should become empty. The nodes of second list should only be inserted when there are positions available. For example, if the first list is 1->2->3 and second list is 4->5->6->7->8, then first list should become 1->4->2->5->3->6 and second list to 7->8.

Use of extra space is not allowed (Not allowed to create additional nodes), i.e., insertion must be done in-place. Expected time complexity is O(n) where n is number of nodes in first list.

Soln:

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def insert_at_alternate_positions(first, second):
    if not second:
        return first

    current_first = first
    current_second = second

    while current_first and current_second:
        next_first = current_first.next
        next_second = current_second.next

        current_first.next = current_second
        current_second.next = next_first

        current_first = next_first
```

```
        current_second = next_second


    return first
```

Question 8

Given a singly linked list, find if the linked list is circular or not.

> **A linked list is called circular if it is not NULL-terminated and all nodes are connected in the form of a cycle. Below is an example of a circular linked list.**

Soln:

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def is_circular(head):
    if not head or not head.next:
        return False


    slow = head
    fast = head.next


    while fast and fast.next:
        if slow == fast:
            return True
```

```
        slow = slow.next
        fast = fast.next.next

    return False
```