NAME: GIKURU JOSEPH NDERITU
REGNO: SCT212-0574/2022
ASSIGNMENT 2: DATASTRUCTURES AND ALGORITHM
COURSE: BSc COMPUTER TECHNOLOGY
YEAR: 2.2

QUESTION ONE (1):
To solve this problem, we use Floyd's cycle-finding algorithm, also known as the "tortoise and the hare" algorithm. This algorithm uses two pointers, one that moves two steps at a time and another that moves one step at a time. If there is a cycle in the list, the fast pointer will eventually catch up to the slow pointer. If there is no cycle, the fast pointer will reach the end of the list.

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

def hasCycle(head):
    if head is None:
        return False

    slow = head
    fast = head.next

    while slow != fast:
        if fast is None or fast.next is None:
            return False

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

    return True
```

QUESTION TWO (2):
To find the node where the cycle begins. After finding that a cycle exists (using the "tortoise and the hare" algorithm), we can reset one of the pointers to the head of the list and move both pointers one step at a time. The point where they meet will be the start of the cycle.

```
def detectCycle(head):
    if head is None:
        return None

    slow = head
```

```
    fast = head

    while True:
        if fast is None or fast.next is None:
            return None

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

        if slow == fast:
            break

    slow = head
    while slow != fast:
        slow = slow.next
        fast = fast.next

    return slow
```

QUESTION THREE (3):
To reverse a linked list, we can use a simple iterative approach. We start with a null previous node and the current node as the head. Then, for each node, we save the next node, update the next of the current node to the previous node, and move the previous and current nodes one step forward.

```
def reverseList(head):
    prev = None
    curr = head

    while curr is not None:
        next_node = curr.next
        curr.next = prev
        prev = curr
        curr = next_node

    return prev
```