BONIFACE MWANGI WARIGI

REG NO:SCT212-0726/2022

UNIT :ICS 2105 DSA

<u>QUIZ 2 -LINKED LIST</u>

QUESTION ONE (1):

To solve this problem, we can 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 meet the slow pointer again. If there is no cycle, the fast pointer will reach the end of the list.

Here is the Python code for this problem:

```python
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, we can use a similar approach to the one used in the previous problem. When the slow and fast pointers meet, we reset the slow pointer to the head of the list and move both pointers one step at a time. The point where they meet again is the start of the cycle.

Here is the Python code for this problem:

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

    slow = head
    fast = head.next

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

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

    slow = head
    while slow != fast.next:
        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 pointer and traverse the list, moving the current node's next pointer to the previous node. We then update the previous and current nodes for the next iteration.**

**Here is the Python code for this problem:**

```python
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
```