# Linked List Questions

# Reverse a Linked List

- **Problem Statement:** Given the *head* of a singly linked list, write a program to reverse the linked list, and return *the head pointer to the reversed list*.

**Input Format:** `head = [3,6,8,10]`
`This means the given linked list is 3->6->8->10 with head pointer at node 3.`

**Result:** `Output = [10,6,8,3]`

# Find middle element in a Linked List

- **Problem Statement:** Given the **head** of a singly linked list, return *the middle node of the linked list*. If there are two middle nodes, return the second middle node.

**Input Format:** ( Pointer / Access to the **head** of a Linked list ) head = [1,2,3,4,5]
**Result:** [3,4,5]

**Input Format:** Input: head = [1,2,3,4,5,6]
**Result:** [4,5,6]

# Add two numbers represented as Linked Lists

- **Problem Statement**: Given the **heads** of two non-empty linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the **sum** as a linked list.

**Input Format:** (Pointer/Access to the head of the two linked lists)
num1 = 342, num2 = 564
l1 = [2,4,3] l2 = [5,6,4]

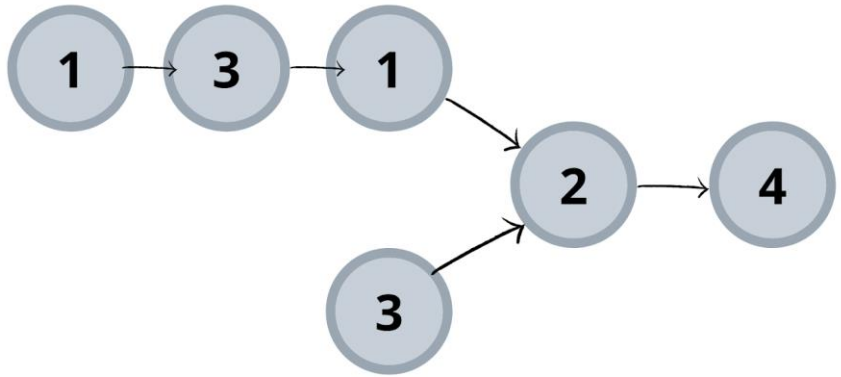
**Result:** sum = 807; L = [7,0,8]

**Input Format:** (Pointer/Access to the head of the two linked lists)
l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]
**Result:** [8,9,9,9,0,0,0,1]

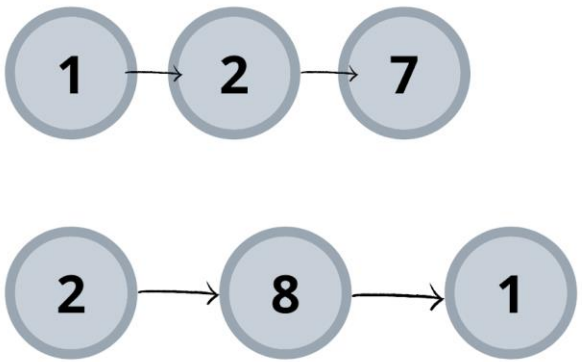
# Find intersection of Two Linked Lists

- **Problem Statement:** Given the heads of two singly linked-lists **headA** and **headB**, return **the node at which the two lists intersect**. If the two linked lists have no intersection at all, return **null**.

```
Example 1: Input: List 1 = [1,3,1,2,4], List
2 = [3,2,4] Output: 2
```



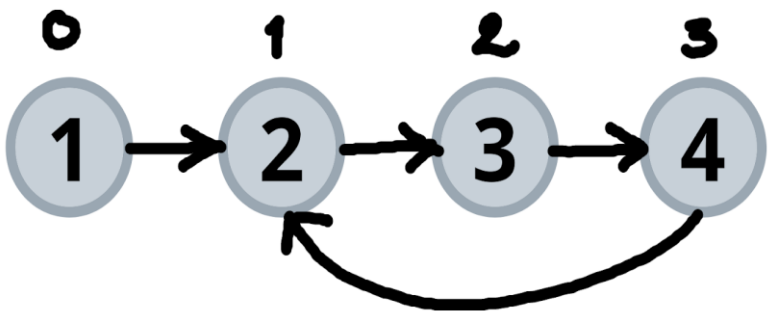
**Example 2: Input:** List1 = [1,2,7], List 2 = [2,8,1] **Output:** Null



# Detect a Cycle in a Linked List

- **Problem Statement:** Given *head*, the head of a linked list, determine if the linked list has a cycle in it. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer.
- Return *true* if there is a cycle in the linked list. Otherwise, return *false*.

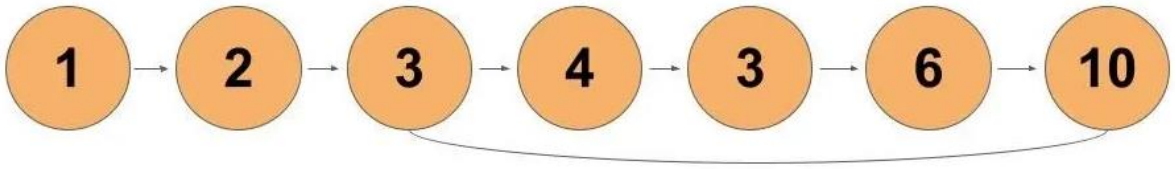
**Example 1: Input:** Head = [1,2,3,4]
**Output:** true **Explanation:** Here, we can see that we can reach node at position 1 again by following the next pointer. Thus, we return true for this case.



# Starting point of loop in a Linked List

- **Problem Statement:** Given the head of a linked list, return *the node where the cycle begins. If there is no cycle, return* null.

**Example 1: Input:** `head = [1,2,3,4,3,6,10]` **Output:** `tail connects to node index 2`



# Rotate a Linked List

- **Problem Statement:** Given the head of a linked list, rotate the list to the right by k places.

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