LeetCode 206. Reverse Linked List

1. Problem Title & Link

- **206. Reverse Linked List**
- https://leetcode.com/problems/reverse-linked-list/

2. Problem Statement (Short Summary)

We are given the head of a singly linked list.
We must **reverse the list** and return the new head.

3. Examples (Input → Output)

Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]

Input: head = [1,2]
Output: [2,1]

Input: head = []
Output: []

4. Constraints

- The number of nodes in the list is in the range [0, 5000].
- -5000 <= Node.val <= 5000

5. Thought Process (Step by Step)

There are **two ways**:

1. **Iterative approach** (most common in interviews):
   - Use three pointers: prev, curr, next.
   - Reverse links one by one.
2. **Recursive approach**:
   - Base case: empty list or single node.
   - Reverse rest of the list and attach current node at the end.

6. Pseudocode (Iterative)

```
prev = null
curr = head
while curr is not null:
    next = curr.next
    curr.next = prev
```

```
    prev = curr
    curr = next
return prev
```

7. Code Implementation

✅ **Python (Iterative)**

```
class Solution:
          def  reverseList(self,  head:  Optional[ListNode])  ->
Optional[ListNode]:
        prev, curr = None, head
        while curr:
            nxt = curr.next
            curr.next = prev
            prev = curr
            curr = nxt
        return prev
```

✅ **Java (Iterative)**

```
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode prev = null, curr = head;
        while (curr != null) {
            ListNode next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
}
```

✅ **Python (Recursive)**

```
class Solution:
          def  reverseList(self,  head:  Optional[ListNode])  ->
Optional[ListNode]:
        if not head or not head.next:
            return head
```

```
        new_head = self.reverseList(head.next)
        head.next.next = head
        head.next = None
        return new_head
```

8. Time & Space Complexity Analysis

- **Iterative**:
  - Time: O(n)
  - Space: O(1)
- **Recursive**:
  - Time: O(n)
  - Space: O(n) (stack frames)

9. Common Mistakes / Edge Cases

- Forgetting to set curr.next = prev → infinite loop.
- Not handling empty list (head = None).
- In recursion, forgetting head.next = None → cycle created.

10. Variations / Follow-Ups

- Reverse a **portion** of linked list (LeetCode 92).
- Reverse in **k-groups** (LeetCode 25).
- Reverse **doubly linked list** (slightly easier).

11. Dry Run (Iterative Approach)

👉 Input: head = [1,2,3]

Initialize: prev = None, curr = 1

- Step 1:
  - nxt = 2
  - 1.next = None
  - prev = 1, curr = 2

    → List: 1 <- None, Remaining: [2,3]
- Step 2:
  - nxt = 3
  - 2.next = 1
  - prev = 2, curr = 3

    → List: 2 -> 1, Remaining: [3]
- Step 3:
  - nxt = None

- 3.next = 2
- prev = 3, curr = None
  → List: 3 -> 2 -> 1

✅ Final Output: [3,2,1]