LeetCode 206 — Reverse Linked List

1. Problem Title & Link

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

2. Problem Statement (Short Summary)

Given the head of a singly linked list, reverse the list **in-place** and return the new head.

Example:

1 → 2 → 3 → 4 → 5

becomes

5 → 4 → 3 → 2 → 1

3. Examples (Input → Output)

Example 1

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

Output: [5,4,3,2,1]

Example 2

Input: head = [1,2]

Output: [2,1]

Example 3

Input: head = []

Output: []

4. Constraints

- 0 <= number of nodes <= 5000
- Node values: -5000 <= val <= 5000
- In-place reversal required
- O(1) extra space

5. Core Concept (Pattern / Topic)

Linked List Pointer Manipulation

You reverse a list by:

- Tracking three pointers:
  prev, **curr**, **next**
- Reversing the link at each step.

This is a **gold-standard pointer problem**.

6. Thought Process (Step-by-Step Explanation)

Start:

prev = None

curr = head

Repeat:

1. Save next: next = curr.next
2. Reverse link: curr.next = prev
3. Move prev forward: prev = curr
4. Move curr forward: curr = next

Loop until curr becomes null.

Return prev, which becomes the new head.

7. Visual / Intuition Diagram

Linked list:

1 → 2 → 3 → 4

Reversal steps:

Step 1:

prev = None

curr = 1

next = 2

1.next = None

List now: 1

Step 2:

prev = 1

curr = 2

next = 3

2.next = 1

List now: 2 → 1

Step 3:

prev = 2

curr = 3

next = 4

3.next = 2

List: 3 → 2 → 1

Step 4:

prev = 3

curr = 4

next = None

4.next = 3

List: 4 → 3 → 2 → 1

8. Pseudocode

```
prev = null
curr = head

while curr != null:
    next = curr.next
    curr.next = prev
    prev = curr
    curr = next

return prev
```

9. Code Implementation

✅ **Python**

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

        while curr:
            nxt = curr.next
            curr.next = prev
            prev = curr
            curr = nxt

        return prev
```

✅ **Java**

```
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;
    }
}
```

10. Time & Space Complexity

Metric	Complexity
Time	O(n) – visit each node once
Space	O(1) – constant space, in-place

11. Common Mistakes / Edge Cases

Mistakes:

❌ Forgetting to save curr.next before breaking the link

❌ Returning curr instead of prev

❌ Losing the rest of the list because of wrong pointer updates

Edge Cases:

- Empty list → return null
- Single node → return as is
- Already reversed list

12. Detailed Dry Run (Step-by-Step Table)

Input:

1 → 2 → 3

Step	prev	curr	next	Change	New List Head
Start	None	1	2	—	1
1	None ← 1	2	3	1.next = None	1
2	1 ← 2	3	None	2.next = 1	2
3	2 ← 3	None	—	3.next = 2	3

Return **3**.

Final:

3 → 2 → 1

13. Common Use Cases

- Reverse a list before processing
- Reverse segments (LC 92 Reverse Linked List II)
- Palindrome linked list check
- Stack-like behavior using linked list

14. Common Traps

- Not updating prev correctly
- Confusing node values with node references
- Infinite loop due to incorrect next pointer

15. Builds To (Related Problems)

- **LC 92** — Reverse Linked List II (reverse sublist)
- **LC 25** — Reverse Nodes in K-group
- **LC 234** — Palindrome Linked List
- **LC 21** — Merge Two Sorted Lists (pointer manipulation)

16. Alternate Approaches + Comparison

Approach	Time	Space	Notes
Iterative	O(n)	O(1)	✔ Best
Recursive	O(n)	O(n)	Clean but uses stack space
Stack-based	O(n)	O(n)	Not allowed (extra space)

17. Why This Solution Works (Short Intuition)

Reversal only requires flipping each node's .next pointer one by one
while maintaining the previous pointer.
The list is reversed fully in one linear scan.

18. Variations / Follow-Up Questions

- Reverse only the second half of the list (LC 234)
- Reverse in k-sized chunks (LC 25)
- Reverse alternate nodes
- Reverse recursively