

■ Detect Cycle in a Linked List

Floyd's Cycle Detection — fast and slow pointers meet inside a cycle

● Linked List

■ MEDIUM

■ Intuition First

Imagine two runners on a looped track. If there IS a loop, the faster runner will eventually lap the slower one — they'll meet. If there's NO loop, the fast runner just falls off the end. Same logic: if fast and slow ever point to the same node, there's a cycle.

List with cycle: $3 \rightarrow 2 \rightarrow 0 \rightarrow -4 \rightarrow (\text{back to } 2)$

slow: $3 \rightarrow 2 \rightarrow 0 \rightarrow -4 \rightarrow 2 \dots$

fast: $3 \rightarrow 0 \rightarrow 2 \rightarrow 0 \rightarrow 2 \dots$

^ they meet here!

List without cycle: $1 \rightarrow 2 \rightarrow 3 \rightarrow \text{null}$

fast reaches null $\rightarrow$ no cycle

■ Java Code

```
1 public class Solution {
2     public boolean hasCycle(ListNode head) {
3         ListNode slow = head;
4         ListNode fast = head;
5
6         while (fast != null && fast.next != null) {
7             slow = slow.next;
8             fast = fast.next.next;
9
10            if (slow == fast) return true; // Cycle detected!
11        }
12        return false; // fast reached null  $\rightarrow$  no cycle
13    }
14 }
```

■■ Complexity

Type	Value	Meaning
Time	O(n)	Fast pointer meets slow within one full loop traversal
Space	O(1)	Only two pointers used

■ Bonus — Find Cycle Start

To find WHERE the cycle begins: when `slow==fast`, reset one pointer to head. Move both one step at a time. They meet at the cycle start! (Floyd's full algorithm)

■ Quick Revision

✓	Use slow and fast pointers (Floyd's algorithm)
✓	If <code>slow == fast</code> at any point → cycle exists
✓	If fast reaches null → no cycle
✓	To find cycle start: reset one ptr to head, step both by 1
✓	Time $O(n)$, Space $O(1)$