

■ Delete Node in a Linked List

A sneaky trick — you don't need the previous node!

● Linked List

■ EASY

■ Intuition First

The usual deletion needs the previous node: `prev.next = node.next`. But this problem gives you the node to delete directly, not its previous. The trick: copy the NEXT node's value into current node, then skip the next node. You're erasing the current by overwriting it with its successor!

List: $4 \rightarrow 5 \rightarrow 1 \rightarrow 9$ Delete node with value 5

Normal approach (need prev):

$4 \rightarrow [5] \rightarrow 1 \rightarrow 9$ `prev.next = node.next`
 ↑ we don't have prev here!

Trick approach (only have node):

Step 1: `node.val = node.next.val` → node becomes 1
 $4 \rightarrow 1^* \rightarrow 1 \rightarrow 9$

Step 2: `node.next = node.next.next` → skip old '1'
 $4 \rightarrow 1 \rightarrow 9$ ✓

Effectively: we 'became' the next node and deleted it

■ Java Code

```
1  class Solution {
2      public void deleteNode(ListNode node) {
3          // Copy next node's value into current node
4          node.val = node.next.val;
5          // Skip over the next node
6          node.next = node.next.next;
7      }
8  }
```

■ Complexity

Type	Value	Meaning
Time	$O(1)$	Just two assignments — constant time
Space	$O(1)$	No extra space used

■ ■ Constraint

This trick ONLY works because the problem guarantees the given node is NOT the tail. If it were the tail, `node.next` would be null and this would crash.

■ Quick Revision

✓	Copy next node's value into current node
✓	Skip next node: <code>node.next = node.next.next</code>
✓	Works because node is guaranteed not to be tail
✓	$O(1)$ time, $O(1)$ space — elegantly simple