Given head, the head of 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. Internally, "pos" is used to denote the index of the node that tail's "next" pointer is connected to.
Note: pos is not passed as a parameter

Return True if there's a cycle in the linked list. Otherwise, return False.

ex 1) 

input: head = [3,2,0,-4] output: True → There's a cycle in the linked list that connects the tail to the 2nd node.
pos: 1

Time Complexity: O(n)
Space Complexity: O(1)

```
def hasCycle(head):

    slow = fast = head

    while fast and fast.next: // we make sure we don't go out of bounds

        slow = slow.next
        fast = fast.next.next

        if slow == fast:
            return True

    return False
```

Floyd's Cycle Finding Algorithm

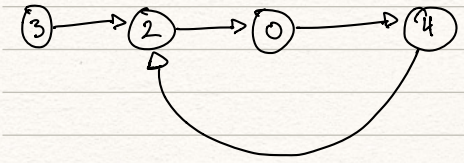
The slow pointer moves one step at a time
The fast pointer moves two steps at a time ⟶ If there is no cycle, the fast pointer will eventually reach the end and we can return False

Now consider a cyclic list and image the slow and fast pointers are two runners racing around a circle track

The fast runner will eventually meet the slow runner...

Why? → The fast runner is just one step behind the slow runner
In the next iteration, they both increment by one and two steps respectively and meet each other

Walk-through



1st iteration

slow = fast = head
slow = fast = 3

] we initialize our pointers

while fast and fast.next: → while (3) and (3.next = 2) # we make sure there are nodes left and we don't go out of bounds

slow = slow.next = 2
fast = fast.next.next = 2.next = 0

] we set the speeds of our pointers

if there's no cycle the loop breaks
if there is we'll reach the if-statement

if slow == fast: (2 == 0 #F)
    return True

] if the pointers are pointing to the same node.val we have a cyle ∴ return true

2nd iteration:

while fast and fast.next: # fast = 0 and fast.next = 4

slow = slow.next = 0
fast = fast.next.next = 4.next = 2

if slow == fast: # 0 == 2 → false
    return True

] This is an example of how the fast runner is just one step behind the slow runner and how they'll meet in the next iteration

3rd iteration:

while fast and fast.next: # fast = 2 and fast.next = 4

slow = slow.next = 4
fast = fast.next.next = 2.next = 4

if slow == fast: # 4 == 4 → this linked list has a cyle
    return True