DAY - 28

① Detect loop in linked list :-

(i) ⇒ [Without hashmap]

Time Complexity : O(n).
Auxiliary Loop : O(1).

Approach :-

→ Have a visited flag with each node.
→ Traverse the linked list and keep marking visited nodes.
→ If you see a visited node again, then there is a loop.
(This solution works in O(n) but requires additional information with each node.)
→ A variation of this solution that doesn't require modification to basic data structure - can be implemented using a hash. Just store the addresses of visited nodes in a hash and if see an address that already exists in hash then there is a loop.

Implementation:-

```
class Node:
    def __init__(self):
        self.data = 0
        self.next = None
        self.flag = 0
def push(head_ref, new_data):
    new_node = Node()
    new_node.data = new_data
    new_node.flag = 0
    new_node.next = (head_ref)

    head_ref = new_node
    return head_ref

def detect_loop(h):
    while (h != None):
        if (h.flag == 1)
            return true
        h.flag = 1
        h = h.next
    return False
```

(ii) => Floyd's Cycle-Finding Algorithm

Approach:-

-> Traverse linked list using two pointers.

-> Move one pointer by one, another pointer by two.

-> If these pointers meet at the same node then there is a loop. If pointers do not meet, then linked list doesn't have loop.

Detection part Implementation:-

```
def detectLoop(self):
    slow_p = self.head # First pointer
    fast_p = self.head # another pointer
    while(slow_p and fast_p and fast_p.next):
        slow_p = slow_p.next
        fast_p = fast_p.next.next

        if slow_p == fast_p:
            return True.
```