Q3) linked list = [1 → 2 → 3 → 4 → 5 → None].

inserting a value 66 at the third(3) index.

Pseudocode:-

```
def insert_at_index(index, value):
    current_node = head
    current_node_index = 0
    while current_node:
        if (current_node_index+1 == index): #checking index value
            new_node = LinkedNode(value) #creating a new node using LinkedNode class
            next_node = current_node.next
            current_node.next = new_node
            new_node.next = next_node
            break      # breaking out of the loop once inserting is done.
        else:
            current_node = current_node.next   #iterating to next node
            current_node_index += 1            #incrementing the index
```

Variables:-

① current_node :- The variable that stores head and will be used to traverse

② current_node_index :- The variable that keeps track of the index of the current_node

③ new_node :- Creating a new node and inserting it into the linked list.

④ next_node :- The variable that used to store the node that comes after the current_node.

Linked list :- 1 → 2 → 3 → 4 → 5 → None.
(1 is head)

steps :- current_node = head = 1

current_node_index = 0

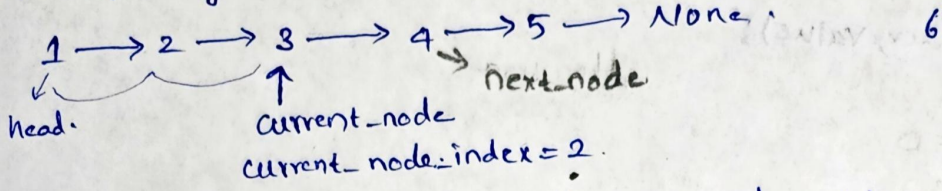
Entering to the while loop and checking for the condition which is not true (1 ≠ 3). traversing to the next node untill the condition satisfies and by incrementing current_node_index.

When current_node_index = 2 then the condition satisfies.

new_node will be created by calling the Linked Node Class (66).

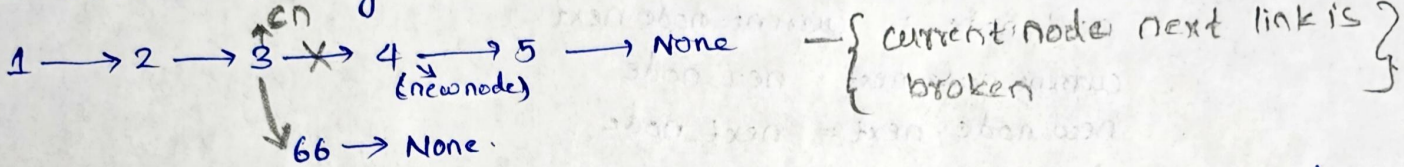
→ new_node — 66 → None.

next_node is used to store the value of current_node.next bcz we are updating the current_node.next to a new value.



So next_node = 4 (we saved next node value for further use)

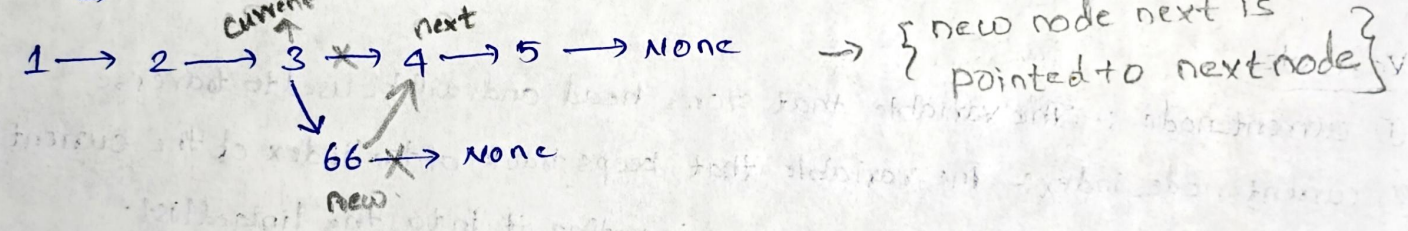
Now we will be pointing current_node_next to the new_node.



→ by breaking the orginal link we will point Current_node.next to new_node.

after this we will be pointing. new node next to old current.next node.

⇒ new_node.next = next_node.



final linkelist

1 → 2 → 3 → 66 → 4 → 5 → None.

Q4) linked list = 1 → 2 → 3 → 4 → 5 → None.

head

Reversing the linked list:-

Pseudocode:-

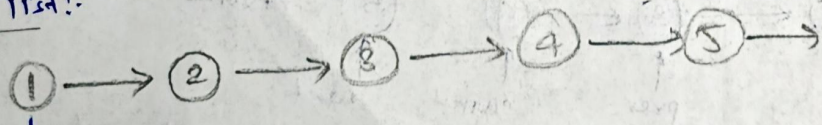
```
def reverseList(head):
    current_node = head
    previous_node = None
    next_node = None
    while current_node:
        next_node = current_node.next   # storing the next_node
        current_node.next = previous_node.  # updating the current node next
        previous_node = current_node   # moving the prev to current
        current_node = next_node.   # moving the current to next
    head = previous_node.      # updating the head value.
    return head.
```

Variables:-

1) current_node:- stores head and will be used to traverse
2) previous_node:- initally set to none keep track of previous node of current node.
3) nextnode:- initally set to none. keep track of next_node of current node.
4) head :- head is the first element of the linked list.

Algorithm:-

Linked list:-



head
current_node.

prev = None
next = None.

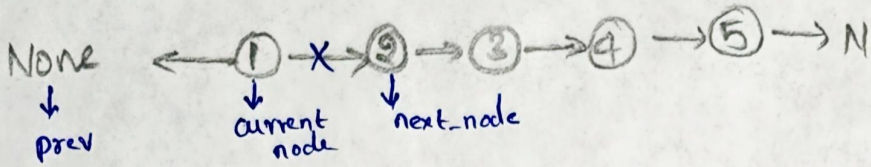
Step 1:-

Entering to the while loop

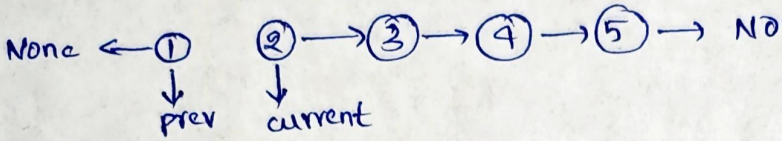
storing the next node of current node because as we are updating the current node next to none while breaking the linkage.

now next_node = 2

→ now current_node next will be point to previous_node (None)



→ now updating previous and current node to their next nodes respectively.

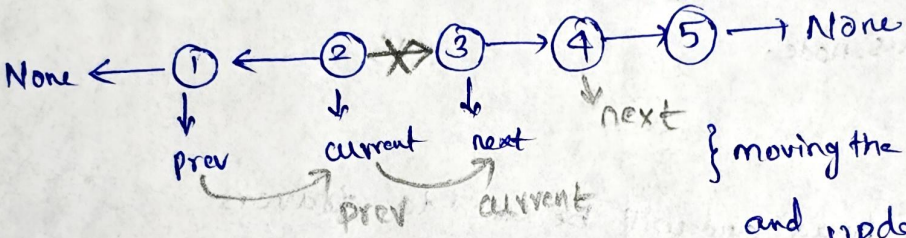


as we broke the link the head is pointing to prev (1).

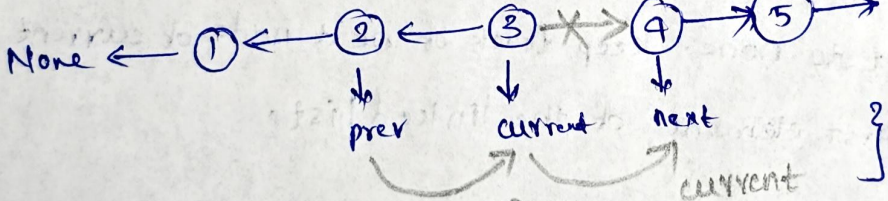
As going to the next iteration.

next_node = 3

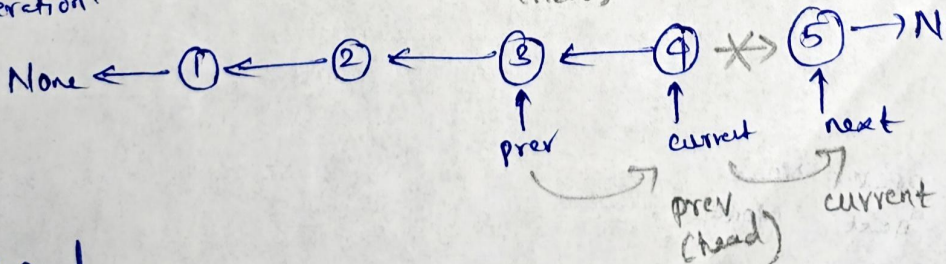
we break the link b/w 2 and 3. and will point 2 to 1 (prev).



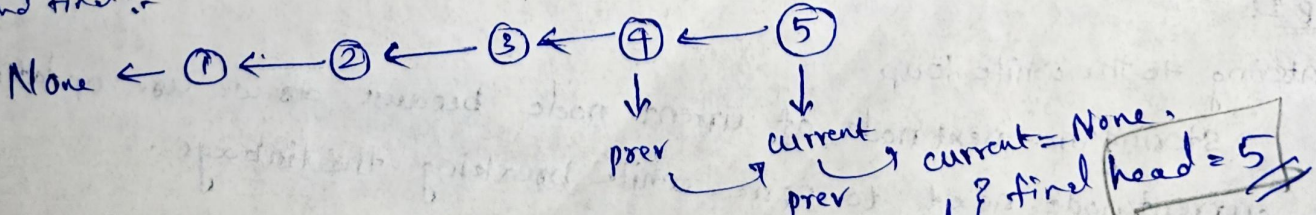
next iteration.



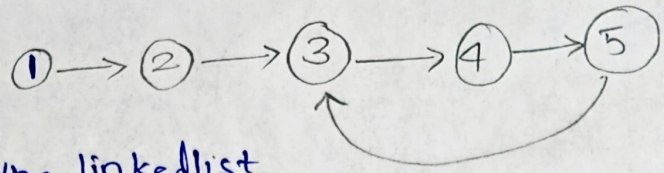
next iteration.



next and final:-



as current is None comes out of the loop. linked list is reversed.

Q5:- linked list 

detect cycle in the linkedlist

Pseudo code:-

```
def detect cycle (hedd):
    slow = head     # initalising slow as head
    fast = head     # initalising fast as head
    while slow and fast.next and fast:
        slow = slow.next      # traversing by one node
        fast = fast.next.next # traversing by two nodes.
        if (slow == fast):    # checking condition for cycle
            return True
    return False.
```

Explanation:-

→ Variables:-

① slow :- initalising and storing it as head, traversing by one node

② fast :- initalising and storing it as fast, traversing by two nodes

Condition:→ slow and fast and fast.next will checle for the node. is none and next node points to is none. prevents accessing the next node of none.

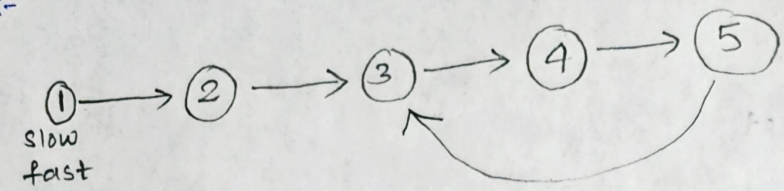
Algorithm:-

Slow and fast pointers are intialised as head.

Entering to the while loop as slow pointer increments by one node and fast pointer increments by two nodes.

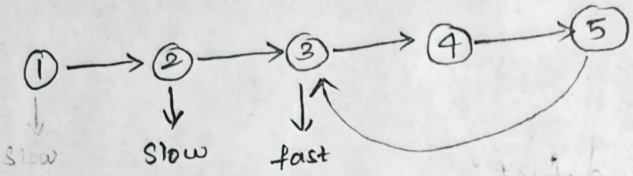
After this it checles for the condition if slow and fast pointer ~~has~~ is in the same node the function will return True else False

Step 1:-



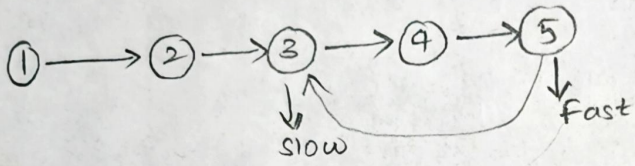
{ slow move by one node
fast move by two nodes }

Enters the loop and traverse to next nodes.



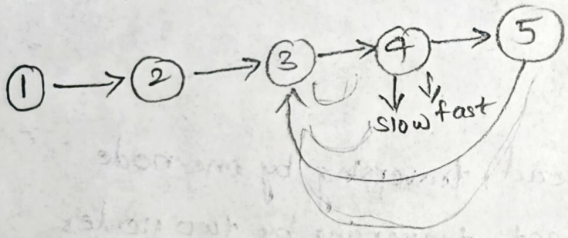
(2)

slow and fast are not same hence traverse to next nodes.



(3)

traverse to next.



slow and fast pointers are have same value hence the condition return True.

If there is no cycle in the linked list the loop will ~~re~~end as slow and fast and fast.next points to none hence the function will return false.

→ If there is a cycle definiently fast and slow pointers will have same value because fast pointer moves fast (by 2 nodes) and slow pointer moves slow (by one node) eventully fast and slow will overlap and returns True

Example lets assume length is 10 fast will move 2 and slow will move by 1

2-1 = 1 ⇒ 10-1 = 9 (everytime the value decreases and eventully ~~are~~ becomes zero ~~and~~ means overlap).

Q6) linked list 1→2→3→4→5→None

delete the value (3)

Pseudocode:-

```
def delete_a_value(head,value):
    current_node = head  # intialising current_node to head
    previous_node = None
    while current_node:
        if current_node.value == value  # checking the value is equal to given value
            previous_node.next = current_node.next

        else:
            previous_node = current_node  # traverse to the next node by storing the prev node
            current_node = current_node.next

    return -1
```

Variables:-

① current_node :- stores head and will be used for traverse

② previous_node :- stores None and will be used to track previous node.

Algo:-

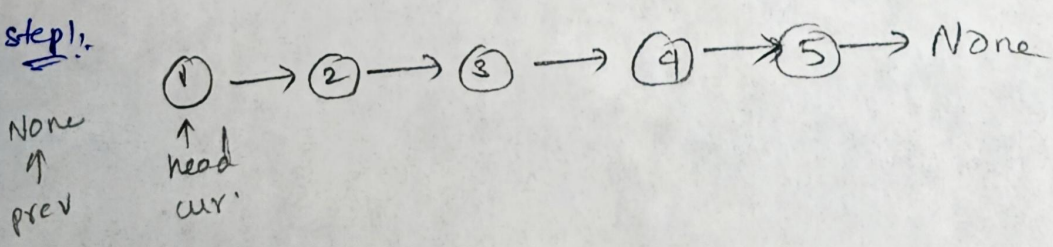
current_node and previous_node is initalised and previous_node is used to track previous of current node for updating its next to current next.

Enters the while loop:-

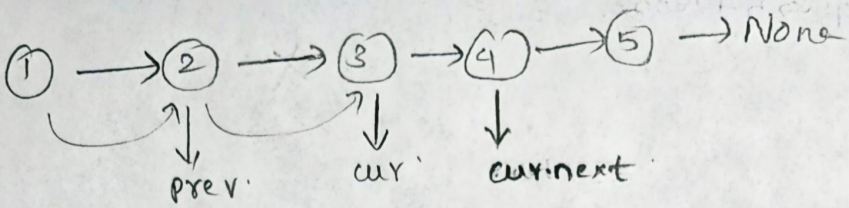
linked list will traverse untill the match is found by storing previous node and current node.

Once prev reached.

step1:-



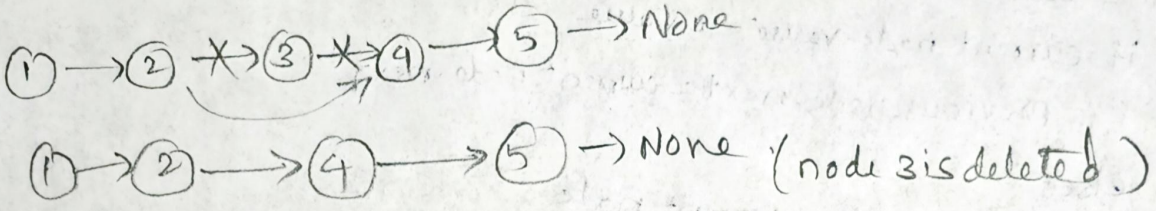
Step2



once reached

update prev.next = current_node.next

Here we deleted (3) node because we are pointing 2 → 4



It breaks the linkage b/w 2 and 3. And 2 directly points to 4

final linked list is