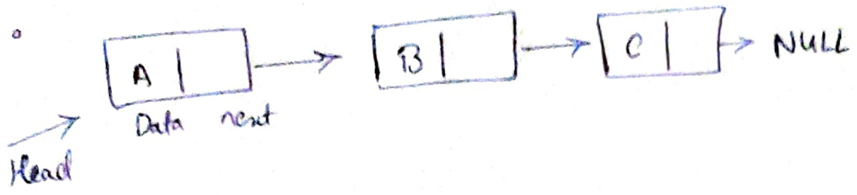
Linked List

- Linear data structure
- 

Python template

```
class Node :
    def __init__ (self, data):
        self.data = data
        self.next = None

class LinkedList :
    def __init__ (self):
        self.head = None

    def append (self, new_data):
        new_node = Node(new_data)

        if self.head is None:
            self.head = new_node
            return

        last = self.head
        while (last. next):
            last = last.next

        last.next = new_node

    def printList (self):
        temp = self.head
        while (temp):
            print (temp.data)
            temp = tem.next
```

```
# drivers code
ll = LinkedList ()
ll. append (6)
ll. append(4)
    :
ll. printList ()
```

Length

```
def length (self):
    temp = self.head
    count = 0
    while (temp):
        Count += 1
        temp = temp.next
    print ( count )
```

✈ double Linked List

```
class Node:
    def __init__(self, data):
        self.data = data
        Self.next = self.prev = None


def append (self, new_data):
    new_node = Node (new_data)
    if self.head is None:
        new_node.prev = None
        Self.head = new_node

    x = Self.head
    while (x.next):
        x = x.next
    x.next = new_node
```

```
# delete at position x
class Solution:
    def deleteNode (self, head, x):
        y = head
        count = 0
        while (y):
            count += 1
            y = y.next
        if (x == 1):
            head = head.next
            head.prev = None

        else:
            y = head
            i = 0
            while (y):
                i += 1
                if (i==x & x==count):
                    y.prev.next = None
                elif (i==x):
                    y.next.prev = y.prev
                    y.prev.next = y.next
                    break
                y = y.next
```

Q1 Insert Node at middle

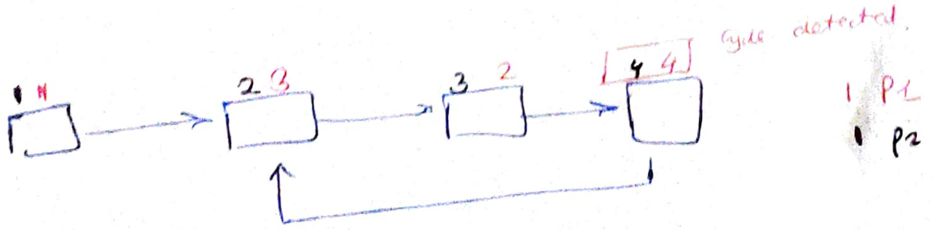
# concept slow & fast pointer

```
def insertAtMid (self , x):
    if (self.head == None):
        self.head = Node (x)

    else :
        newNode = Node(x)
        slow = self.head
        fast = self.head.next
        while ( fast! = None and fast.next != None ):

            slow = slow.next
            fast = fast.next.next
        newNode . next = slow. next
        slow.next = newNode
```

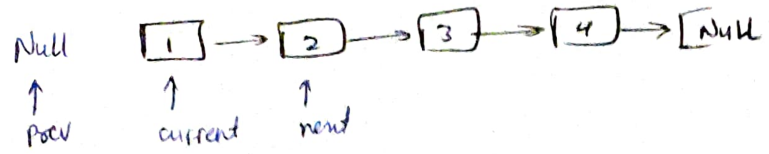
Q2 Detect loop in Linked List

```
def hasCycle (self, head : ListNode ) -> bool:
    p1 = head
    p2 = head
    while p1 and p1.next :
        p1 = p1 , next .next
        p2 = p2 . next
        if p1 == p2:
            return True          (has cycle.
    return false.
```

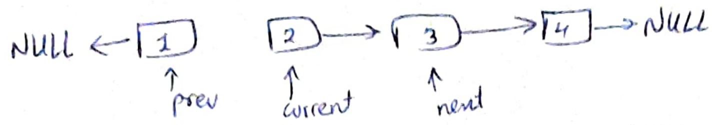
eg



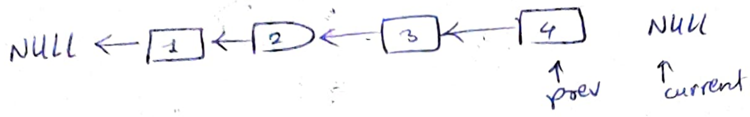
Q3) Reverse a Linked List



1) reverse link betn prev & current & shift all by 1 step



2) continue this till reach current at NULL.



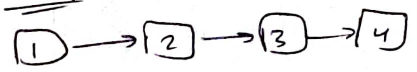
iterative

```
def reverseList (self, head):
    prev = None
    curr = head
    while curr != None :
        nextTemp = curr.next
        curr.next = prev
        prev = curr
        curr = nextTemp
    return prev
```

recursion

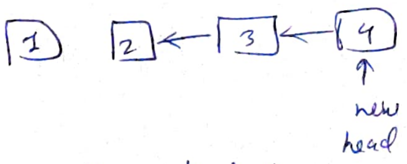
```
class Solution (object):
    def reverseList (self, head):
        if head == None :
            return None
        if head.next == None :
            return head
        p = self.reverseList (head.next)
        head.next.next = head
        head.next = None
        return p.
```

Concept



keep 1 & reverse 2 → 3 → 4 with 4 as head

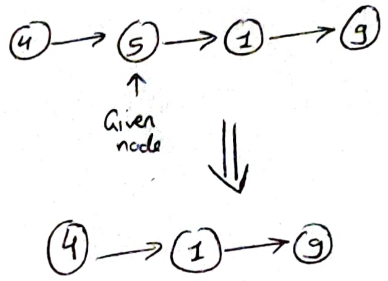
So



now 2 points to 1
point 1 to None
return new head.

Q4) $\overset{LC}{}$ Delete a node in a linked list

Write a function to delete a node in singly - linked list.
You will not be given access to head of the list, instead you will be given access to the node to be deleted directly.
It is guaranteed that node to be deleted in not a tail node.



concept
$$\begin{tikzcd} & \arrow[d] & & \\ 4 \arrow[r] & \textcircled{5} \arrow[r] & 1 \arrow[r] & 9 \end{tikzcd}$$

replace 5value with next node value

$$\begin{tikzcd} 4 \arrow[r] & \textcircled{1} \arrow[r] & 1 \arrow[r] & 9 \end{tikzcd}$$ =)

& now delete next node

$$\begin{tikzcd} 4 \arrow[r] & \textcircled{1} \arrow[rr, bend left] & & 9 \end{tikzcd}$$

Soln

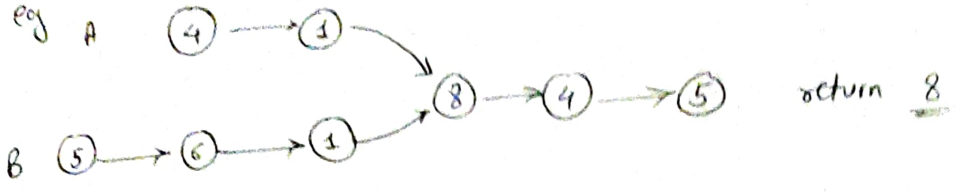
```
class Solution :
    def deleteNode (self, node):
        node.val = node.next.val
        node.next = node.next.next.
```

Q5) Merge two sorted list

```
def merge ( l1 : ListNode , l2 : ListNode) -> ListNode :
    a = List Node ()
    t = a
    while (l1 and l2):
        if ( l1.val <= l2.val ):
            t.next = ListNode (l1.val)
            l1 = l1.next
        else :
            t.next = List Node ( l2.val)
            l2 = l2.next
        t = t.next
```

```
if l2 :
    t.next = l2

if l1 :
    t.next = l1

return a.next.
```

Q6) Find intersection of two linked list.
if no intersection return null.

eg 

Concept First find length of both A & B.

eg lenA = 5
lenB = 6

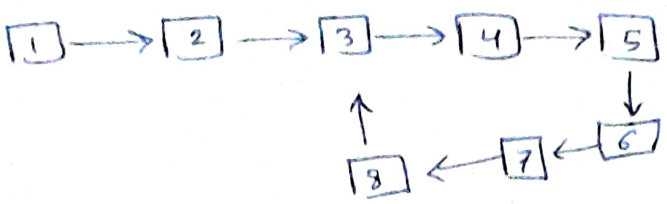
So now move pointer for B by (lenB - lenA) value
So we get pointer B pointing to 6.

now traverse both pointer A & B & return when pointer A = pointer B.

```
def getIntersection (headA ,headB) → ListNode :
    t1 = headA
    t2 = headB
    l1 = l2 =0
    while (t1):
        t1 = t1 . next
        l1 += 1
    while (t2):
        t2 = t2. next
        l2 += 1

    if l1 > l2 :
        long = headA
        Short = headB
    else :
        long = headB
        Short = headA

    for _ in range (abs (l1 - l2)):
        long = long . next

    while Short != long:
        long = long. next
        Short = Short . next
    return short
```

Q7) Delete a loop in linked list.

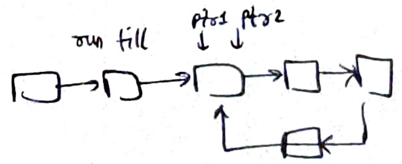
Concept

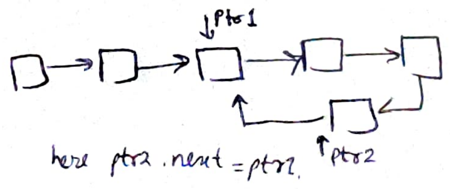


1) to delete loop just make $\boxed{8}$ point to NULL.
2) we take 2 pointers ptr1 & ptr2, same like last question of 'intersection of linked list', we move 1 pointer n (no. of nodes in loop) times & traverse both ptr1 & ptr2, at loop start point (3) both ptr1 & ptr2 next will point. so here make ptr2. next = NULL to remove loop.

Soln

```
def detectLoop(head):                    # to delete loop
    slow = fast = head                     first detect it
    while (fast and fast.next):
        slow = slow.next
        fast = fast.next.next

        if slow == fast:
            removeLoop(slow, head)
            return 1
    return 0.

def removeLoop(slow, head):
    ptr1 = ptr2 = slow.
    count = 1
    while (ptr1.next != ptr2):           # count nodes in loop
        ptr1 = ptr1.next
        count += 1

    ptr1 = head
    ptr2 = head                           # make ptr2 point count nodes ahead
    for i in range(count):
        ptr2 = ptr2.next
    while (ptr2 != ptr1):
        ptr1 = ptr1.next
        ptr2 = ptr2.next
    while (ptr2.next != ptr1):
        ptr2 = ptr2.next.
    ptr2.next = None
```


run till
ptr1 ptr2


ptr1
here ptr2.next = ptr1.
ptr2