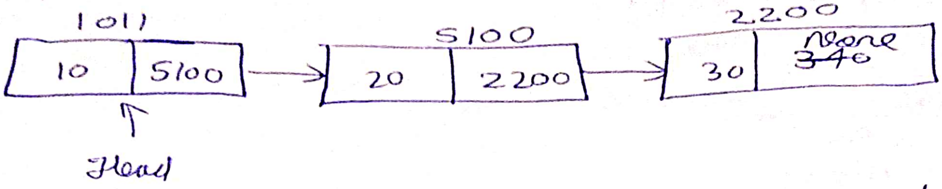
Removing Nodes

→ start
→ end
→ middle

i) Start



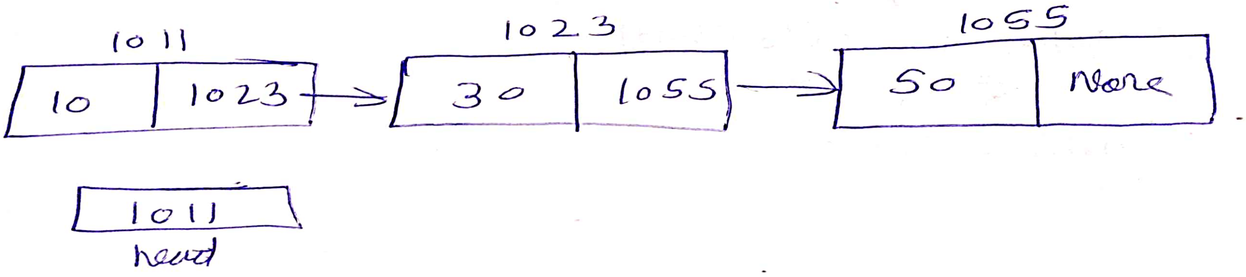
1] check weather linked list is empty or not
2] point head to second node

```
def Remove_begin (self):
    if (self.head == None):
        print (" Linked list is empty")
else:
    self.head = self.head.ref
```

} check weather linked list is empty or not

} first changing the head we can delete the Node

) End



1] check weather linked list is empty or not
∵ deletion from empty linked list not possible
2] First traverse till second last node
change second last reference to null

```
def remove_end(self):
    if (self.head == None):
        print("Linked list is empty")
    else:
        n = self.head
        ~~while (n != None):~~
            if (n.ref == None):
        while (n.ref.ref != None):
            n = n.ref
        ~~new_node = No~~
            ~~A = None~~
        n.ref = None
```

3] We also check ~~[illegible]~~ when only one element (node) is present

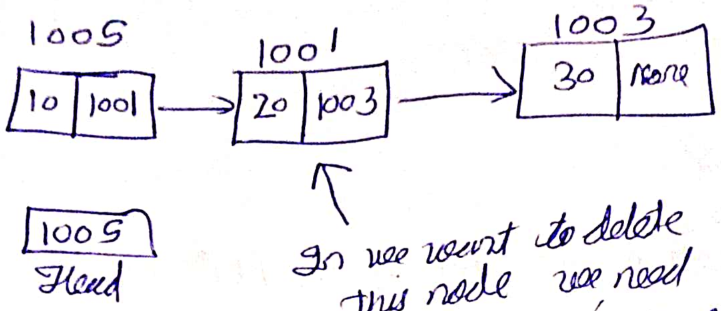
```
def remove_end(self):                                                    10
    if (self.head == None):
        print("Linked list is empty")          } if any node is not present
    elif (self.head.ref == None):
        self.head = None                       } if only one node is present
    else:                                        its ref fild is none
        ~~ref~~
        n = ~~A~~ self.head
        while (n.ref.ref != None):
            ~~A.re~~
            n = n.ref
        n.ref = None
```

3) Delete from any position of likad list

1) check any linked list is empty or not

2) If given is first Node or not then change the value of head

3)



So we want to delete this node we need to go to previous node

i) That is 10 assign the ~~xxx~~ reference of 20 to 10

```
def delete_value(self value):
    if (self.head == None):
        print("Linked List is empty")
    ~~elif (~~ return
        n = ~~A.~~ self.head
    ~~elif~~ (
    while(n.ref != None):
        if (n.ref.data == value):
            break
        else:
            n = n.ref
    if (n.ref == None):
        print("Linked list don't have that Node)
    else:
        n.ref = n.ref.ref
```

```
def delete_value(value):
    if(self.head == None):
        print("Linked list is empty")
        return
    if (self.head.data == value):
        self.head = self.head.ref
        return
    n = self.head
    while(n.ref != None):
        if(n.ref.data == value):
            break
        else:
            n = n.ref
    if(n.ref == None):
        print("Node is not present in the liked list")
    else:
        n.ref = n.ref.ref
```

check weather the linked list is emty or not

if first node is targe node then we need to

if any node in liked list other than first node we will traverse till its previous node and change the refernce

Linked list every operation

~~Traver~~

1) Traversal
2) Insertion
    i) Begin
    ii) End
    iii) Between
3) Deletion
    i) Begin
    ii) End
    iii) Between

```
class Node :
    def --init--(self, data):
        self.data = data
        self.ref = None
class Linked_List :
    def --init--(self):
        self.head = None
    def print_LL(self).
        if (self.head = = None):
            print("Linked List is empty")
        else:
            ~~while (self.head.~~
            n = self.head
            ~~while (self.~~
            while(n.ref ! = None):
                if (n.ref ! = None):
                    print(~~s~~n.data , end=" --> ")
                else:
                    print(n.data)

    def add_begin(self, data):
        new_node = Node(data)
        if (self.head = = None):
            self.head = new_node
        else:
            new_node.ref = self.head
            self.head = new_node

    def add_end(self, data):
        new_node = Node(data)
        if (self.head = = None):
            self.head = new_node
        else:
            n = self.head
            while (n.ref ! = None):
                n = n.ref
            n.ref = new_node
```

```
def add_before(self, data, x);
    if (self.head == None):
        print ("Node is not present in Linked list")
        return
  ~~else:~~
     n = n.
  if (self.head.data == x):
      new_node = Node(data)
      new_node.ref = self.head
      self.head = new_node
      return
  n = self.head
  while(n.ref != None):
     if(n.ref.data == x):
         break
     else:
         n = n.ref
  if (n.ref == None):
     print("Node is not present in Linked list")
  else:
     new_node = Node(data)
     new_node.ref = n.ref
     n.ref = new_node
def add_after(self, data, x):
  if (self.head == None):
     print ("Node is not present in Linked List")
  else:
     n = self.head
     while (n.ref != None):
        if (n.data == x):
            break
        else:
            n = n.ref
     if (n.ref == None):
        print("Node is not present in Linked list")
     else:
        new_node = Node(data)
        new_node.ref = n.ref
        n.ref = new_node
```

```
def remove_head(self):
    if (self.head == None):
        print("Linked List is empty")
    else:
        self.head = self.head.ref

def remove_end(self):
    if (self.head == None):
        print ("Linked List is empty")
    elif (self.head.ref == None):
        self.head == None
    else:
        n = self.head
        while (n.ref.ref != None):
            n = n.ref
        n.ref = None

def remove_value (~~se~~self, value):
    if (self.head == None):
        print("Noode is not present is LinkedList")
        return
    if (self.head.data == value):
        self.head = self.head.ref
        return
    n = self.head
    ~~while(n.ref != None):~~
        ~~if (n~~
    while(n.ref != None):
        if (n.ref.data == value):
            break
        else:
            n = n.ref
    if (n.ref == None):
        print("Node is not present in Linked list")
    else:
        n.ref = n.ref.ref
```