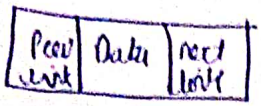
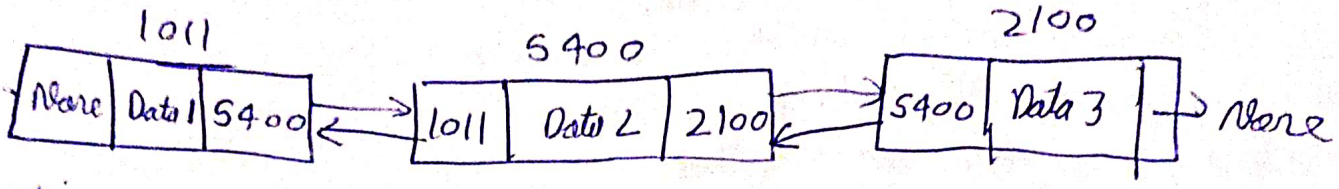
- Doubly Linked List
  - Every node have data link and next link

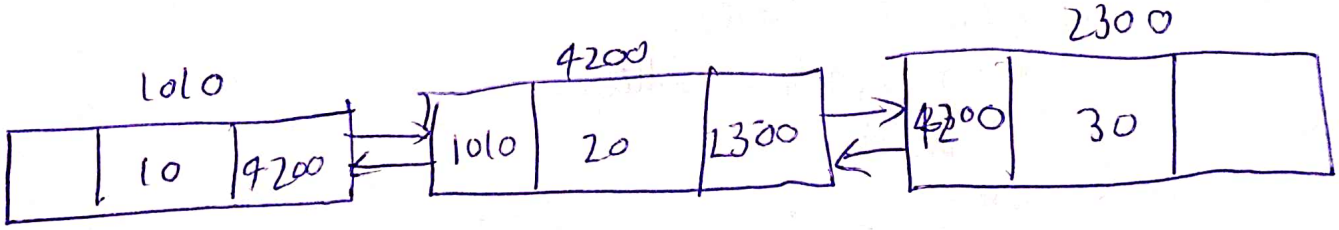




1] Insertion
  i] Begin
  ii] End
  iii] Between

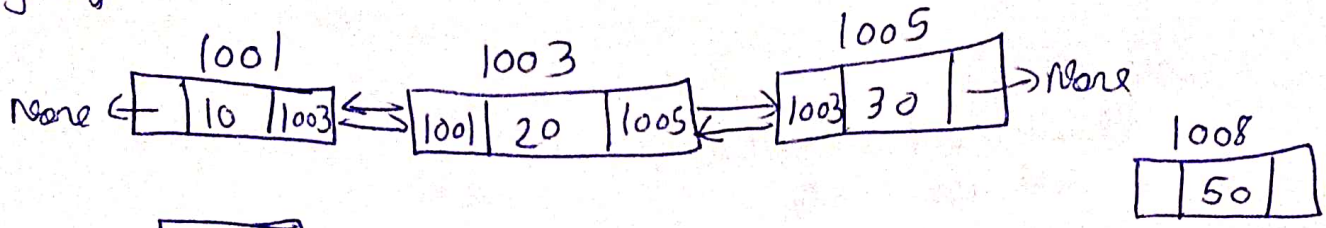
2] Deletion
  i] Begin (Head)
  ii] End
  iii] Between

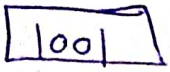
3] Traversal
  i] Forward
  ii] Backward
- Moving Forward and Backward is easier
- Need extra memory



1] Create Node
2] initially both value None (prev, next)
3] new_node.prev = head
4] 1st node prev = new_node
5] head = New node

ii] End





1] Create Node
2] Check Linked list empty or not
self.head == None
3] if empty
self.head = new_node
4] Traverse till last node
while (n.next != None):
n = n.next
new_node = Node(data)
new_node.prev = n
n.next = new_node

1) Create Node
2) go to last Node
3) last node next = new_node
4) new_node prev = lastnode

iii] Inbetween

1] Create Node
2] go to previous node after which you are adding new_node
3) n.next = new_node
4) new_node.prev = n
5] new_node.next = y
6] y.prev = new_node

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

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

    def printDoubleLL(self):
        if (self.head == None):
            print("Linked List is empty")
        else:
            n = self.head
            while(n != None):
                if(n.nref != None):
                    print(n.data, end=" --> ")
                else:
                    print(n.data)
                n = n.nref

    def printDoubleLL_reverse(self):
        if (self.head == None):
            print("Linked List is empty")
        else:
            n = self.head
            while(n.nref != None):
                n = n.nref
            while(n.pref != None):
                if (n.pref != None):
                    print(n.data, end=" --> ")
                else:
                    print(n.data)
                n = n.pref
```

Forward Traversal

Backward Traversal

- Insertion in doubly linked list
  i) Insert when empty
  ii) at begin
  iii) at end
  iv) after a node
  v) before a node

i) Insertion when empty

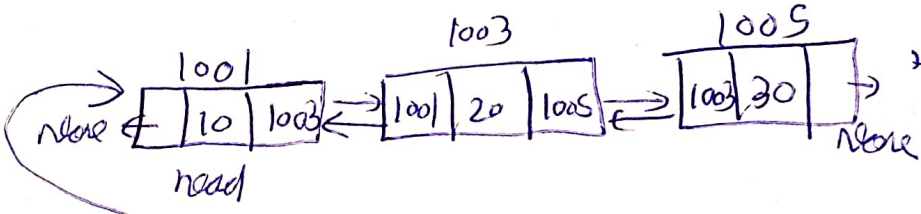
head → None

1] Create Node
2] self.head = new_node

```
def insert_empty(self, data):
    if (self.head == None):
        new_node = Node(data)
        self.head = new_node
    else:
        print ("Linked list is not empty")
```

ii) at begin

1] empty linked list or not
2] non empty



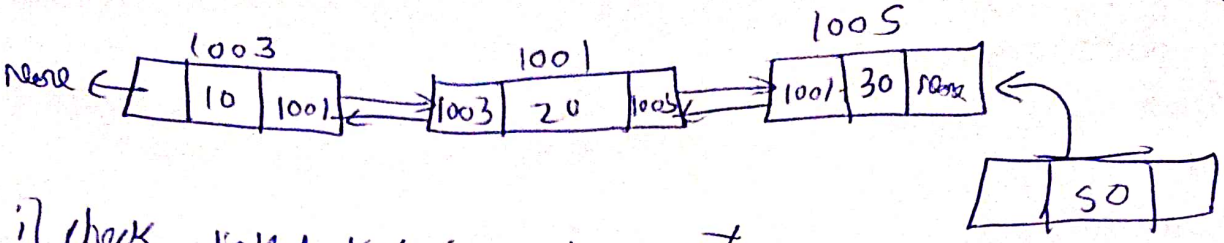

```
def insert_at_begin(self, data):
    if (self.head == None):
        new_node = Node(data)
        self.head = new_node
    else:
        new_node = Node(data)
        new_node.next = self.head
        self.head =
        self.head.prev = new_node
        self.head = new_node
```

check if linked list is empty or not

newnode next reference we assign head

In the prev head previous field store reference of new_node

Update the head



i) check linked list is empty or not
ii) Iterate till to last node
iii) Add the new_node

```
def add_at_end (self, data):
    if (self.head == None):
        new_node = Node(data)
        self.head = new_node
    else:
        new_node = Node(data)
        n = self.head
        while(n.nref != None):
            n = n.nref
        n.nref = new_node
        new_node.pref = n
```

add_after

1) Check if linked list is empty or not
   if empty we cannot add the element in doubly linked list
2) Traverse to given node
3) add_after that node

```
def add_after (self, data, x):
    if (self.head == None):
        print ("Linked list is empty")
    else:
        n = self.head
        while (n != None):
            if (n.data == x):
                break
            else:
                n = n.next
        if (n == None):
            print ("Node is not present in Linked list")
        else:
            new_node = Node(data)
            new_node.next = n.next
            new_node.prev = n
            n.next.prev = new_node
            n.next = new_node
            if (n.next != None):
                n.next.prev = new_node
            n.next = new_node
```

check linked list is empty or not

we will iterate till last node if node is present we will insert new node after targeted node

we will come out of loop by 2 cases either it is present in doubly linked list or it is not present

we want to change the previous reference of its next node

we should take care in case of last node

add_before

```
def add_before (self, data, x):
    if (self.head == None):
        print ("Doubly Linked list is empty")
        return
    if (self.head.data == x):
        new_node = Node(data)
        new_node.next = self.head
        self.head.prev = new_node
        self.head = new_node
        return
    else:
        n = self.head
        while (n != None):
            if (n.next.data == x):
                break
            else:
                n = n.next
        if (n == None):
            print("Node is not present in linked list")
        else:
            new_node = Node(data)      # New Node
            new_node.next = n.next     # New Node previous reference with next of previous node
            new_node.prev = n
            n.next.prev = new_node
            n.next = new_node
```

1) check if the linked list is empty or not

2) If only one node is present in linked list

3) For other test cases

add after

```
def add_after(self, data, x):
    if (self.head == None):
        print("Linked list is empty")
    else:
        n = self.head
        while (n != None):
            if (n.data == x)
                break
            else:
                n = n.next
        if (n == None):
            print("Node is not present in Doubly Linked list")
        elif( n.next == None):
            new_node = Node(data)
            new_node.prev = new_node
            n.next = new_node
        else:
            new_node = Node(data)
            new_node.next = n.next
            new_node.prev = n
            n.next.prev = new_node
            n.next = new_node
```