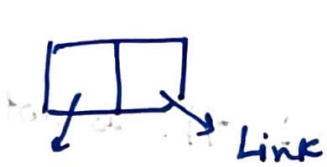
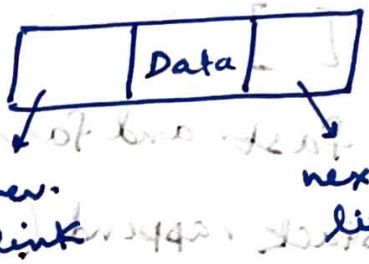
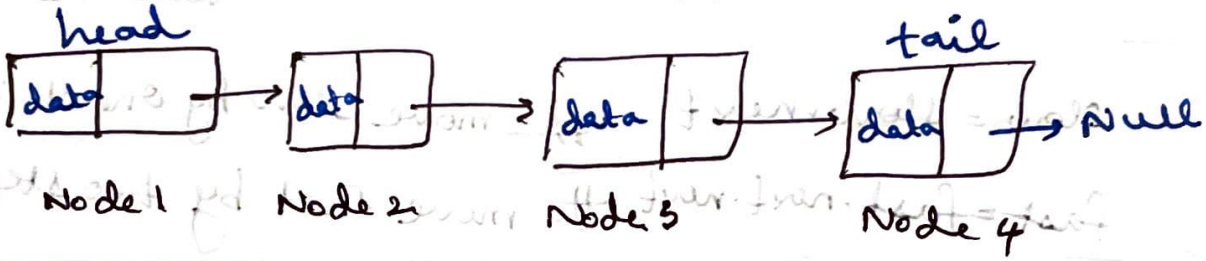
27/Dec/2022

Linked List : chain of nodes (Dynamic data structure)



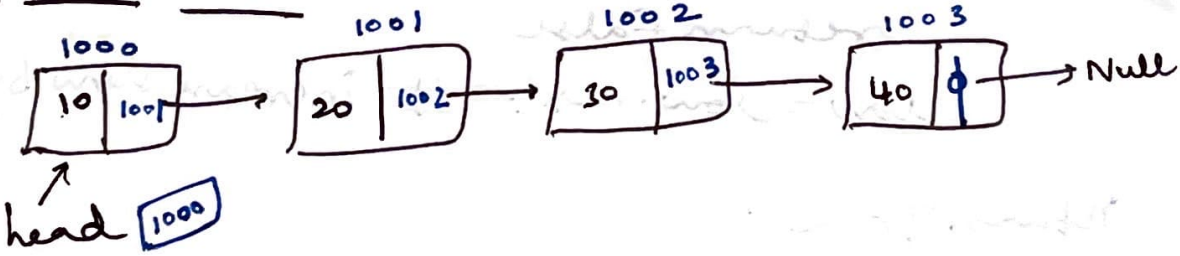




---

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List

Singly Linked List :

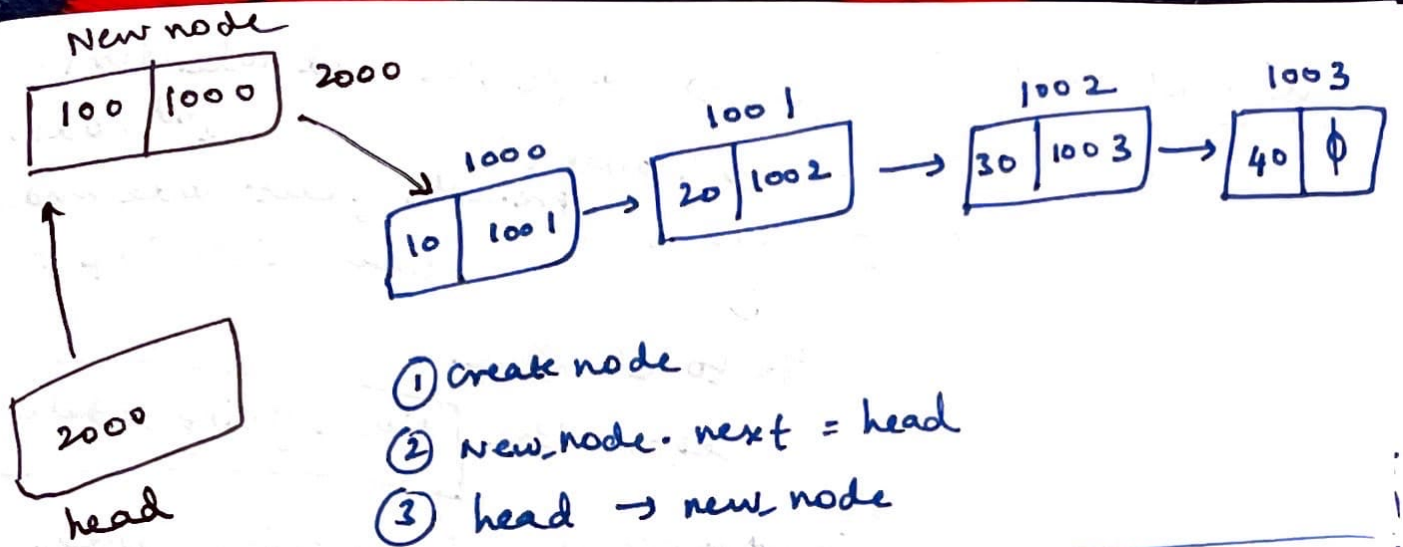


Insertion : 1. begin
2. end
3. in-between

At Beginning : 1. Create Node

2. Change new_node next = head

3. Point head → new_node



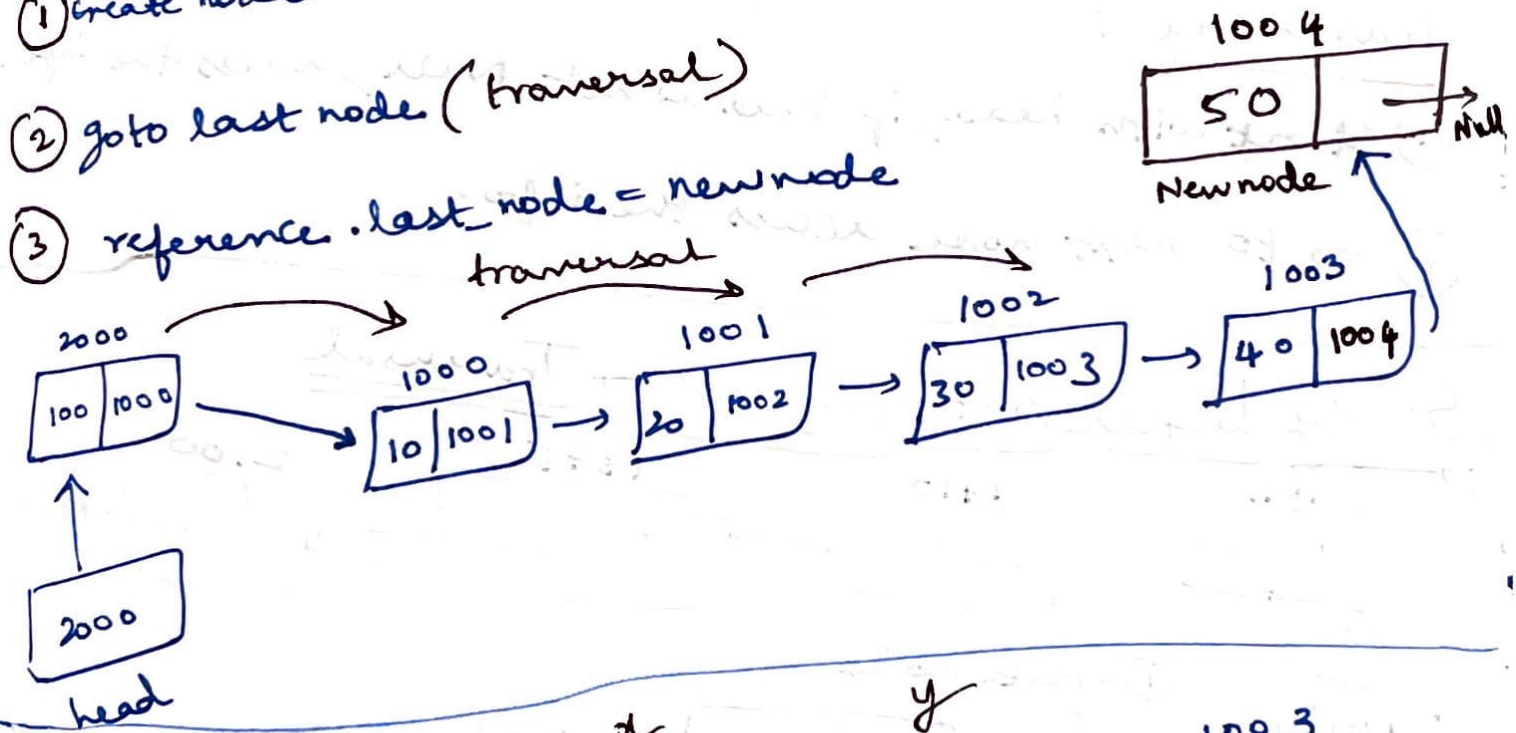
① Create node
② New_node . next = head
③ head → new_node

At ending:

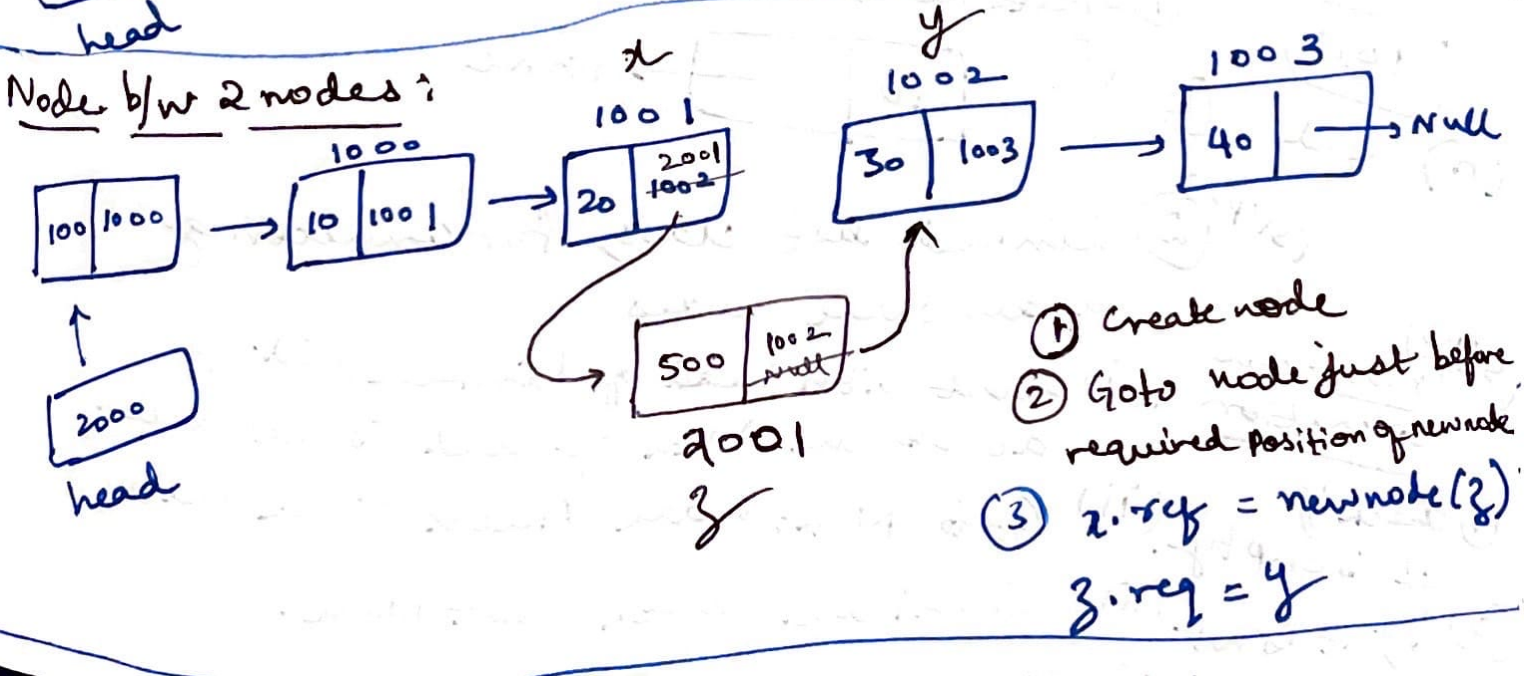
① Create node

② goto last node (traversal)

③ reference . last_node = new node



Node b/w 2 nodes:



① Create node
② Goto node just before required position of new node
③ z.ref = new node (z)
z.ref = y

Deletion :

(1) Begin → change the head node link to 2nd node

(2) End → Travel to pre. node of last node and change its link to Null

(3) In-between

(1) go to element before the Node going to deleted

(2) change the link with node next the deleted Node

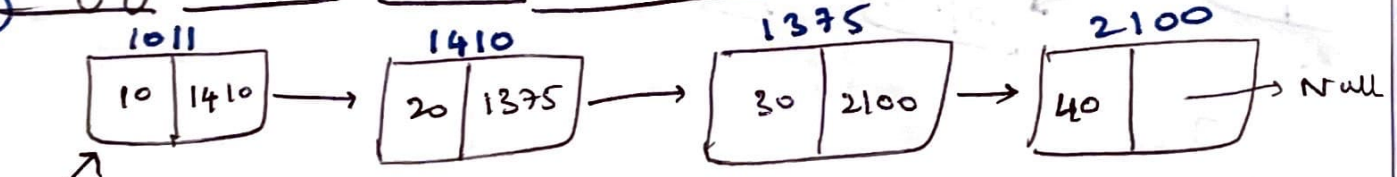
Traversal :

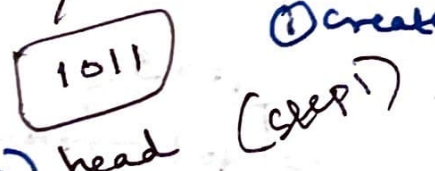
(1) start with head, if head is not Null, access the info.

(2) Go to next node access the info.

---

(1) Singly Linked list Program → Traversal





(step1) (1) create node

(step2) (2) Linked list class, for connecting individual nodes

(step3) Traversal operation : (i) Start with head of Linked list, access the data if head is not null.

(1) Check if LL is empty.

(2) if it is not empty. (ii) Go to Next node Access Node data

(iii) Continue until last Node.

class Node : → class constructor | initialization method

(step1) Create node

```
def __init__(self, data):
    self.data = data
    self.ref = None → Keyword for empty value
```

```
node1 = Node(10)
print(node1) #<Main.Node object at 0x00000178FC3502E0>
```

(step2) connecting nodes

```
class LinkedList:
    def __init__(self):
        self.head = None  # empty Linked List
```

(step3) traversal of nodes

```
def print_LL(self):
    if self.head is None:
        Print("Linked list is empty!")
    else:
        n = self.head
        while n is not None:
            Print(n.data)
            n = n.ref
```

n = 1011
n.data → 10
n.ref → 1410
n = self.head
For going to next node put n = n.ref (n = 1410)

```
LL1 = LinkedList()  # LL1 is obj from LinkedList class
LL1.Print_LL()  # Linked list is empty!
```

Loops :

(1) For loop → when we know
How many time we need to
execute Loop Body.

while loop → used, when we know
(2) stopping condition

(Here we need, stopping condition, i.e n → null)

Singly Linked List → Inserting / Adding element at
(2) beginning of Linked List

add_begin ⟶ Data (100)

(i) New_node = Node (Data) → 100 # create node
(ii) New_node.ref = head # change new_node next to head
(iii) head = new_node # Point head to new_node.ref

(New_node is the object, from the class Node).

```
def add_begin ( self, data ):
    New_node = Node(data)        #  | data | None |
    New_node.ref = self.head
    self.head = New_node
```

Lln add_begin(10) # o/p : 10

Inserting element at ending of Linked list :

add_end → Data (200)

(i) New_node = Node (Data) → 200

(ii) we need to check if L.L is empty (or) Not

(iii) n.ref = newnode

(If condition) If L.L → empty → Go to first node

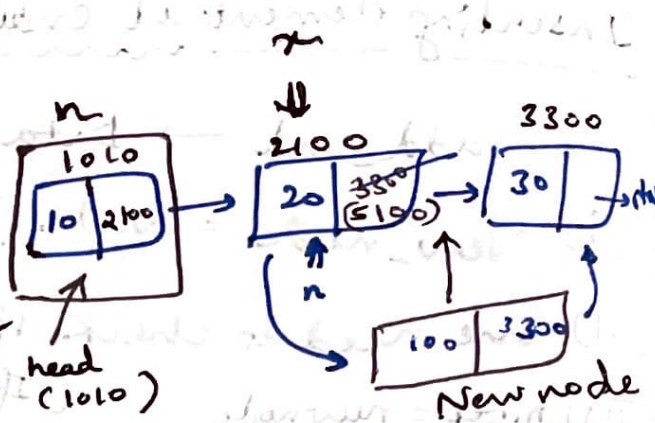
(else condition) If L.L → Non empty → Go to last node

```
def add_end (self, data):
    new_node = Node (data)
    if self.head is None:          # If Linked List is empty
        Self.head = New_node
    else:                          # If Linked List is Not empty
        n = self.head
        while n.ref is Not None:
            n = n.ref              # This loop goes to last node
        n.ref = new_node

LL1.add_end (100)
```

Add / Insertion

Inbetween → After Node
Inbetween → Before Node



after_node (self, data, x)

x → given node after "x node" we need to insert new Node!

→ Find x

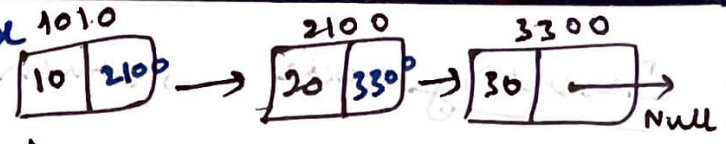
① → n = self.heade (n is First node)
② n.data == x
③ n = n.ref (to move n to next node)

we put this inside a loop while n is not None

break the loop.

```
def add_after(self, data, x):
    n = self.head
    while n is not None:          # n is not empty L.L.
        if x == n.data:           # "x node" is found so break.
            break
        else:
            n = n.ref
    if n is None:
        Print("node is not present in LL").
    else:
        new_node = Node(data)     # Creating new node after "x node".
        new_node.ref = n.ref
        n.ref = new_node
```

Before_node (self



add_before (self, data, x)

head (self.head.data = 10)

[1010]

data of the node before which we want to add new node

① Before First node
↳ create node, (adding element at beginning of linked list)

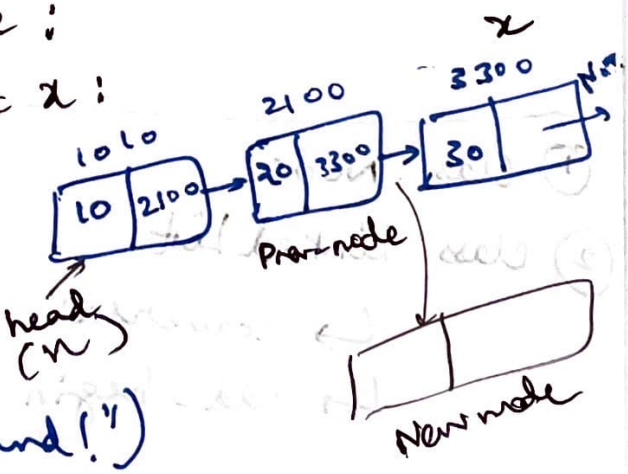
② rest positions
↳ ① find Prev node
↳ ② New_node after Prev node

```
def add_before (self, data, x):
    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 is not None:
        If n.ref.data == x:
            break
        n = n.ref
    if n.ref is None:
        Print ("Node is not found!")
    else:
        new_node = Node(data)
        new_node.ref = n.ref
        n.ref = new_node
```

If self.head is None: Print ("L.L is empty") return.

(adding element at beginning of the linked list).

we find prev. node



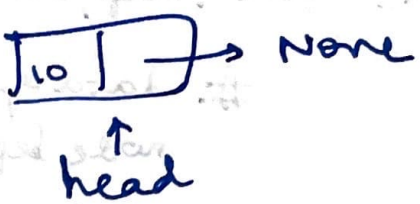
adding new node after prev-node (same code as adding_after x node)

Inserting node : → when Linked list is Empty.

Check If L.L → is empty? [For that head → None]

① create node

② head → new_node



Check If L.L → Not empty

↳ Then print( ) → msg.

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

---

① Class Node

② Class Linked List

↳ traversal

↳ add_begin

↳ add_end

↳ add_after

↳ add_before

↳ insert_empty