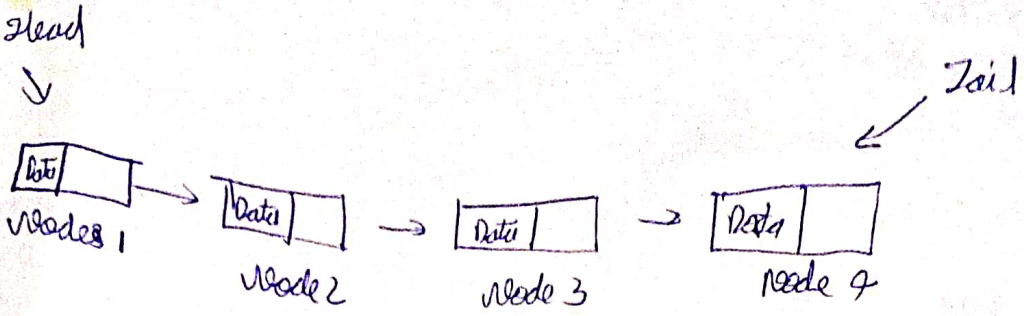


- Dynamic data structure (No need to mention space)

Advantage

- Dynamic data structure
- Insertion and deletion is easy
- Represent and manipulate polynomial

Disadvantage

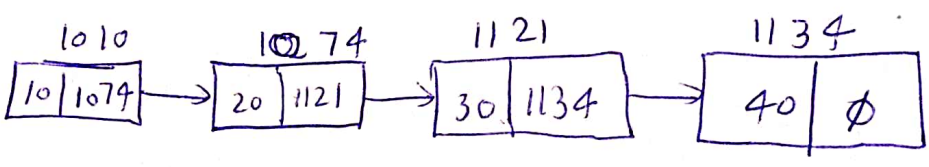
- Needs extra memory
- Random access not possible

- Singly linked list
- Doubly linked list
- Circular linked list

Singly linked list :- Contain only one link next link node

→ add
→ delete
→ traversal

- add / Insert :- • begin
  • end
  • inbetween



1] Create Node
2] Change new node next = head
3] head → new.node

- end

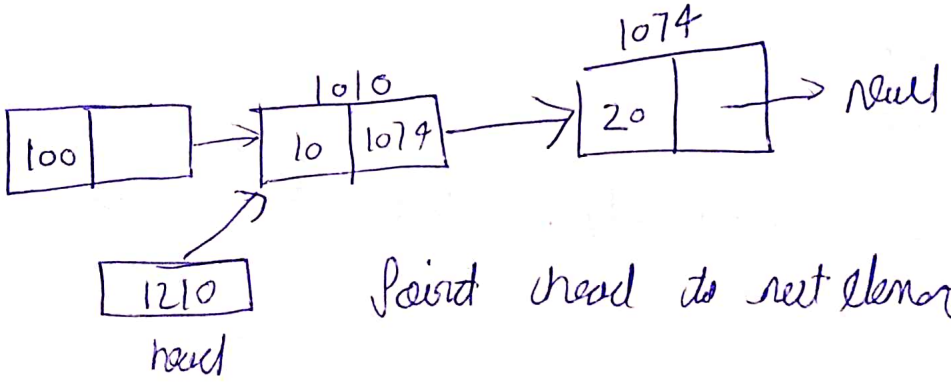
1) Locate a Node
2) goto last Node
3) ref last.next = newnode

- inbetween

1) Locate a Node
2) go to node just before required position of newnode
3) x.ref = newnode (2)
   z.ref = y

- Deletion

1) begin
2) end
3) inbetween



2) end

- go to the previous node of last node
- and make it Null reference

3) inbetween

- go to the previous node of node to be deleted
- change the reference

Create Node

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

} creating None

To link the nodes

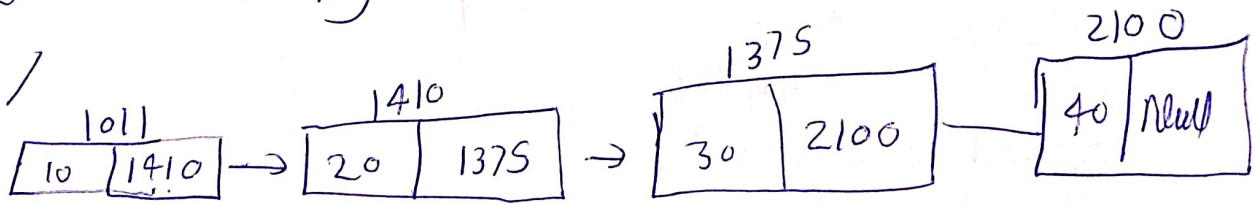
```
class LinkedList:
    def __init__(self):
        self.head = None

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

/*
Traversal

1] Linked list is empty
2] it is not empty

*/



1011
head

n = self.head
to print first node
~~n = self~~
print(n.data)
n = n.ref # move the next node

1] n = 1011
n.data = 10
n.ref = 1410

2] n = 1410
n.data = 20
n.ref = 1375

```
Class Link_list:
    def __init__(self):
        self.head = None
    def print_LL(self):
        if (self.head == none):
            print("Linked list is empty")
        else:
            n = self.head
            while (n != None):
                print(n.data)
                n = n.ref
```

/*

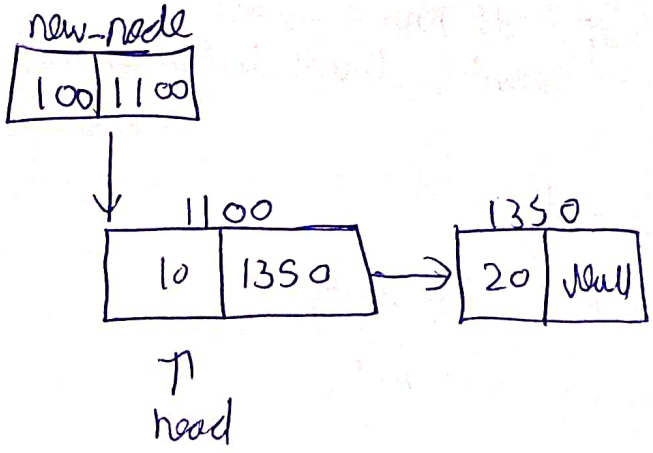
Adding Node

1) At head

i) Create Node

new_node = Node(Data)

ii) new node.ref = head




```
def add_begin(self, Data):
    obj1 = Node(self.Data)
    obj1.ref = self.head
    self.head = obj1
```

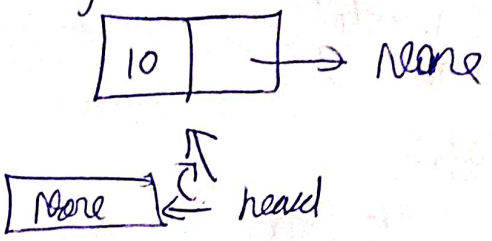
3) head ↪ new node

```
def add_begin(self, data):
    new_node = Node(data)  # Creating new node with ref = none
    new_node.ref = self.head  # Assigning ref of previous head to new head
    self.head = new_node  # Updating head
```

/*

- Adding Node
  1) At end of linked list
  1) Create Node
     obj = Node(10) # data = 10 ref = None
  - if Linked list is empty → it is first Node

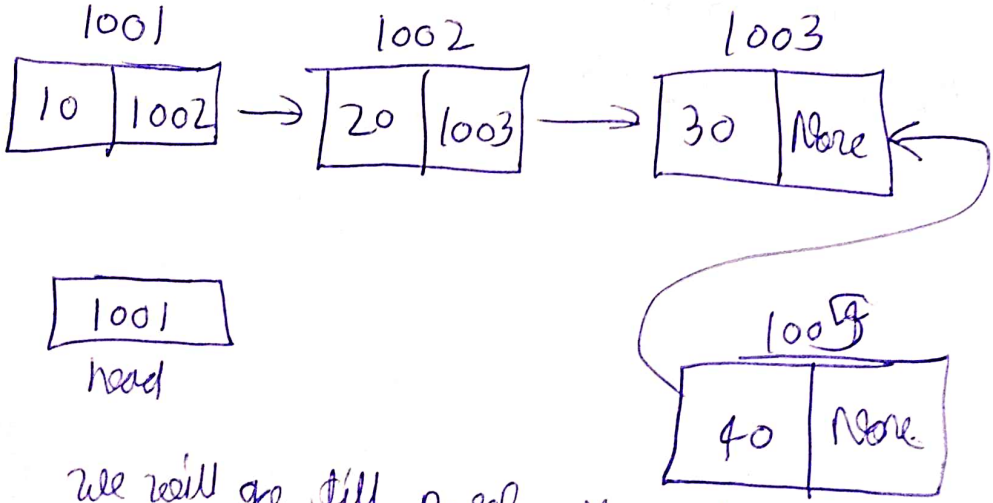
[self.head = obj



- if linked list is not empty
  traverse till last where node ref = None

~~while~~ (~~n = n.ref~~ we will keep on updating ref

```
n = n.head
while (n.ref != None):
    n = n.ref
n.ref = obj
```



We will go till n.ref = None till 30 and will assign the ref of new node 40

```
class Link_list(object):
    def __init__(self):
        self.head = None

    def print_LL(self):
        if (self.head == None):
            print("link list is empty")
        else:
            n = self.head
            while (n.ref != None):
                print(n.data)
                n = n.ref

    def add_begin(Data):
        new_node = Node(Data)
        new_node.ref = self.head
        self.head = new_node

    def add_end(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
```

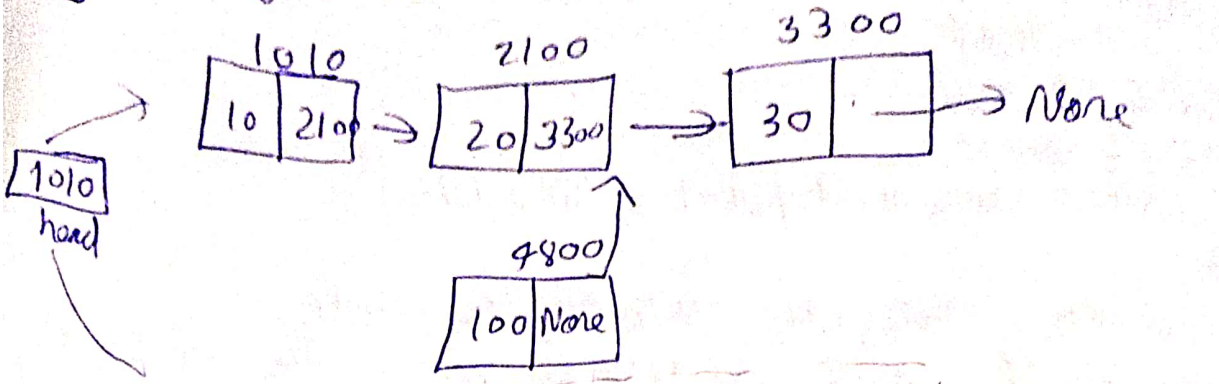
/*

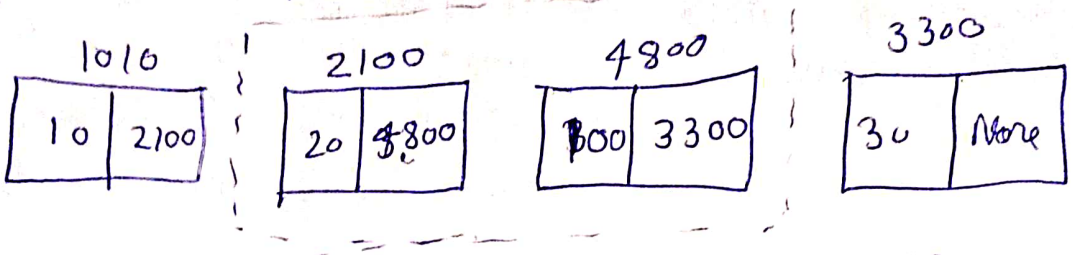
Adding Node

(7)

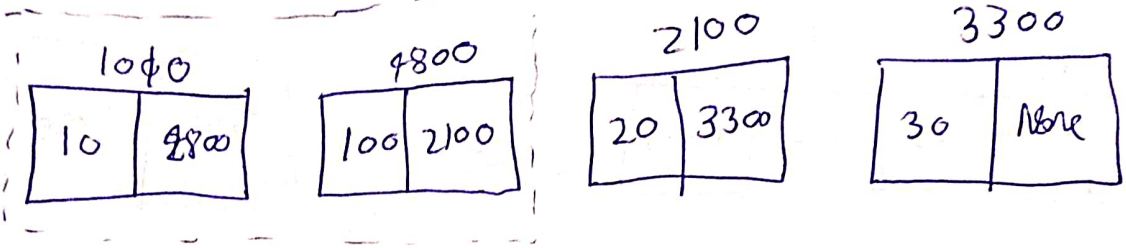
- Insertion
  1) After Node
  2) Before Node

1)





2)



1) Find x is the self-data after which node we have to insert Node

we will search in whole linked list

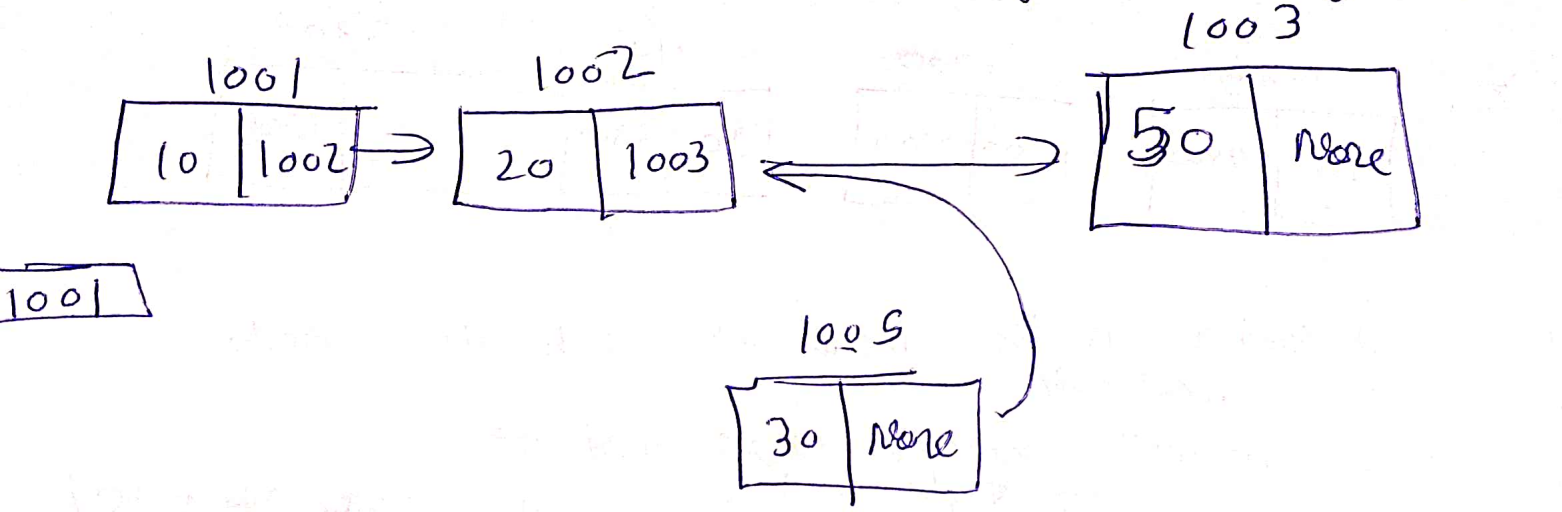
x = No of to search

n = self-head

obj = ~~Node~~ Node(10)

```
while (n != None):
    if (n.data == x):
        temp = n.ref        # we will store ref of current Node
        n.ref = obj         # we will update the targeted node ref to new node
        n = n.ref           # we will go in next link list new_node
        n.ref = temp        # and the remaining linklist is joined as
                            # next node ref is connected to new node
    else:
        n = n.ref           # if not found go to next node
```

```
n = self.head
while (n != None):
    if (n.data == x):
        break
    else:
        n = n.ref
if (n == none):
    print("Element is not present in linked list")
else:
    new_node = Node(data)   #creating new Node
    new_node.ref = n.ref    # As we are inserting after reference of
    n.ref = new_node        # current Node will assign to new node
                            # and in the x node's reference
                            # we will assign new_node reference
```



We need to add 30 after 20 we will traverse till 20 now reference of remaining linked list must be assign to newnode that means reference of 50 should assign to 30 and 20 should point to 30

2) Before Particular Node

```
class Linked_List:
    def __init__(self):
    def __init__(self, Data):
        self.data = Data
    so
```

```
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(n.ref != None):
                print(n.data)
                n = n.ref

    def add_begin(data):
        new_node = Node(data)
        new_node.ref = self.head
        self.hadd = new_node

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

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

/*

Adding Node

- Before Node
  1) Before First Node
  2) Rest Position

1) Before First Node

it will declare the head node

```
obj = NewNode(data)
self.head
obj.next = self.head
self.head = obj
```

2) Before Particular Node

1) Find New Node
2) New node after new node

Identity

New node ref.data == x

```
def add_before(self, data, x):
    if (self.head == None):                    } if there is no node
        print("Linklist is empty")
    elseif
    if (self.head.data == x):                  } if we want to add before head
        new_node = DataNode(Data)
        new_node.ref = self.head
        self.head = newnode
    n = n.self.head
    while(n.ref != None):          # we will search till the last node
        if (n.ref.data == x):      # next nodes data is matching
            new_node = DataNode(data)          # we want to add here
            new_n
            new_node.ref = n.ref
            n.ref = new_node
            break
        else:
            n = n.ref                          remaining cases
    if (n.ref == None):            # if we search all the linked list we don't find
        print("Node is not present in Linked list")
    else:
        new_node = Node(data)      # if we find the node
        new_node.ref = self.ref
        n.ref = new_node
```

```
class Linked_List :
    def __init__(self):
        self.head = None
    def print_LL(self):
        if (self.head == None):
            print("Linked List is empty")
        else:
            while
            n = self.head
            while (self.ref n.ref != None):
                print(n.data)
                n = n.ref
    def add_begin(self, data):
        new_node = Node(data)
        new_node.ref = self.head
        self.head = new_node
    def add_end(self, data):
        if (self.head == None):
            new_node = Node(Data)
            self.head = new_Node
        else: n = self.head
            while(n.ref != None):
                n.ref
            n.ref = new Node(data)
def add_after(self,data):
    n = self.head
    while (n.ref != None):
        if (n.data == x):
            break
        else:
            n = n.ref
    if (n.ref == None):
        print("Linked list is not present")
```

```
else:
    new_node = Node(data)
    Node
    new_node.ref = n.ref
    n.ref = new_node

def before_
def add_before(self, data, x):
    if (self.head == None):
        print("Linked list is empty")
        break
    if (self.head.data == x):
        new_node = Node(data)
        new_node.ref = self.head
        self.head = new_node
    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
```