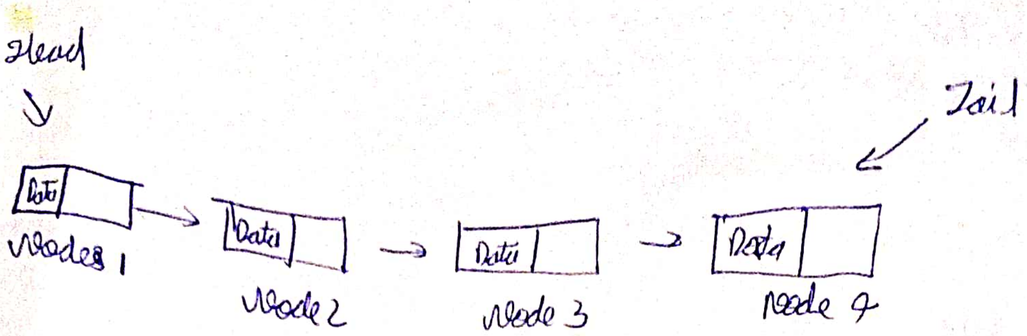


- Dynamic data structure (No need to mention space)

Advantage

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

Disadvantage

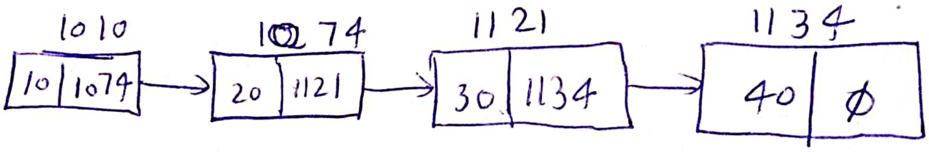
- Needs extra memory
- Random access not possible

- 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

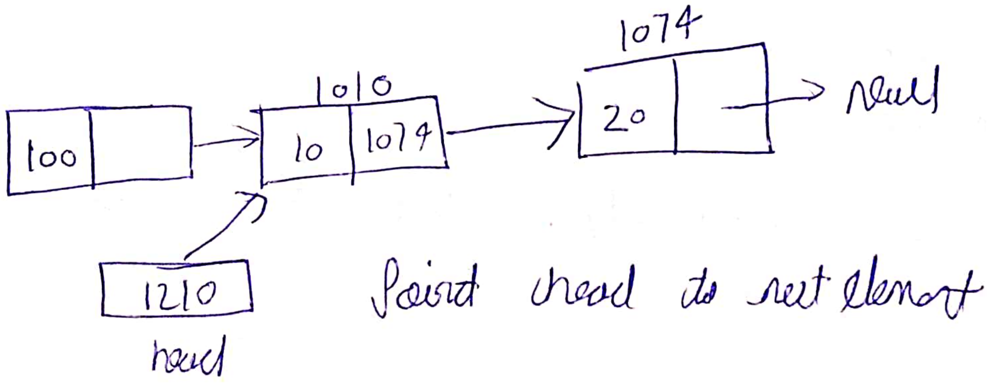
  1) Create a Node
  2) goto last Node
  3) ref lastnode = newnode

- inbetween

  1) Create a Node
  2) go to node just before required position of newnode
  3) x.ref = newnode (2)
     2.ref = 4

- Deletion

  1) begin
  2) end
  3) inbetween



Point head to next element

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

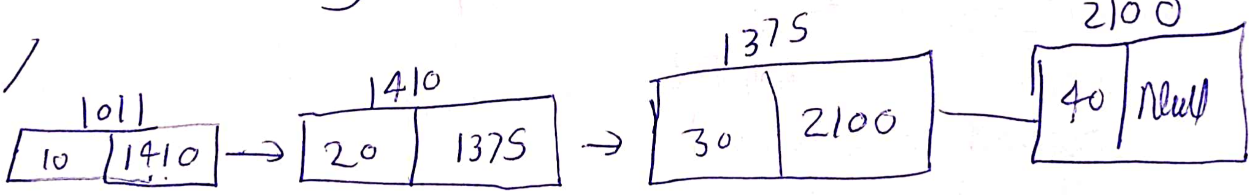
```
# Create Node
class Node :
    def  __init__(self, data):      } Creating None
        self.data = data
        self.ref = ~~self~~ None
# To link this nodes

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 to 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:
            ~~x~~ n = self.head
            while (n != None):
                print(n.data)
                n = n.ref
```

/*

Adding Node

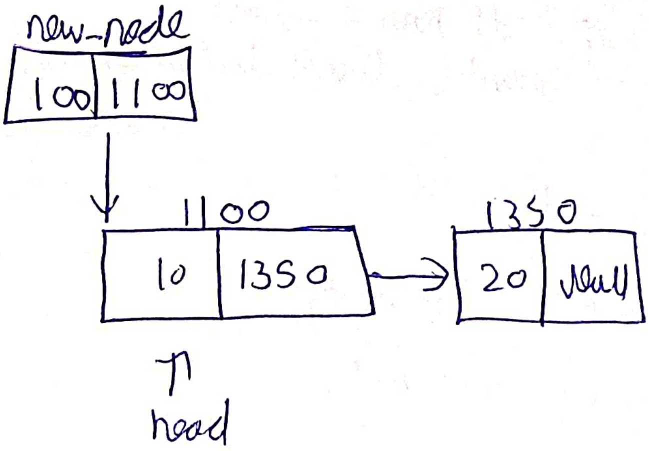
1) At head

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  # Assiging ref of previous head to new hea
        self.head = new_node  # Updating head
```

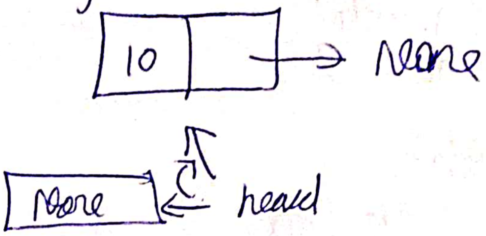
/*

- Adding Node
  i) At end of linked list
    i] Create Node

```
      obj = Node(10) # data = 10  ref = None
```

- if Linked list is empty → it is first Node

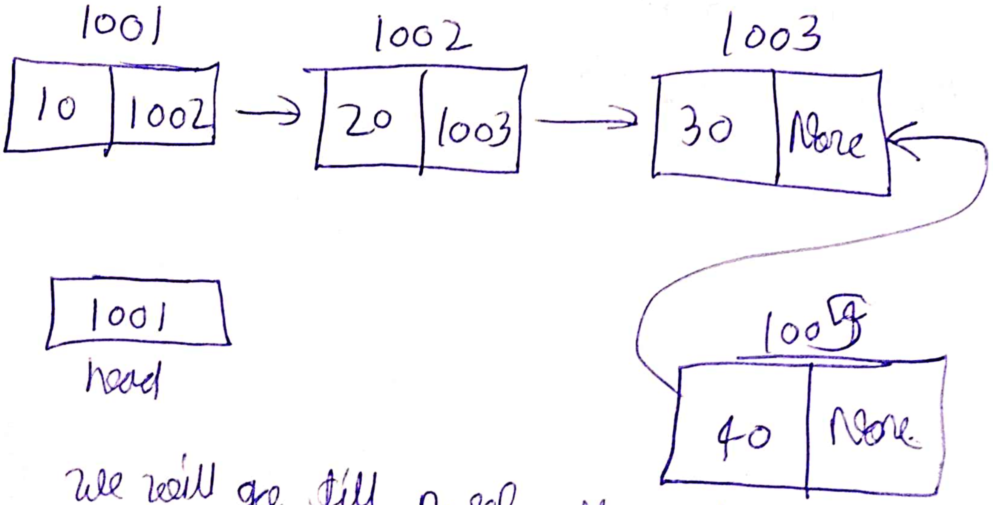
self.head = obj



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

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

```
      n = n.~~ref~~ 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

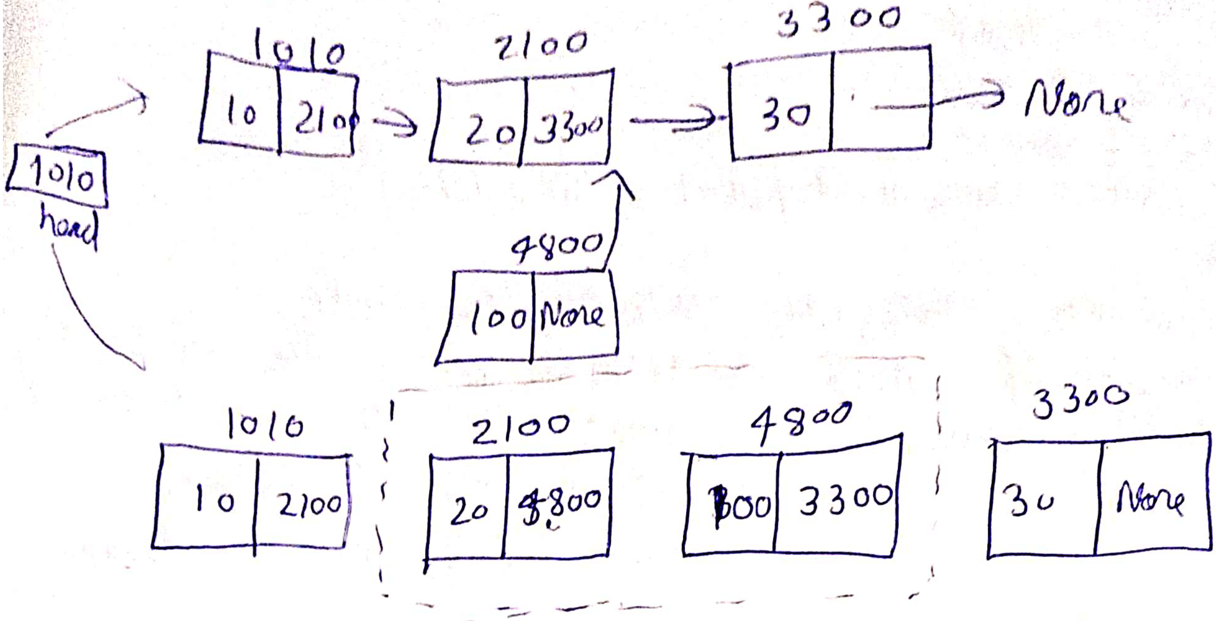
```
class Link_list(object):
    def __init__(self):
        self.head = None

    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
```

/*

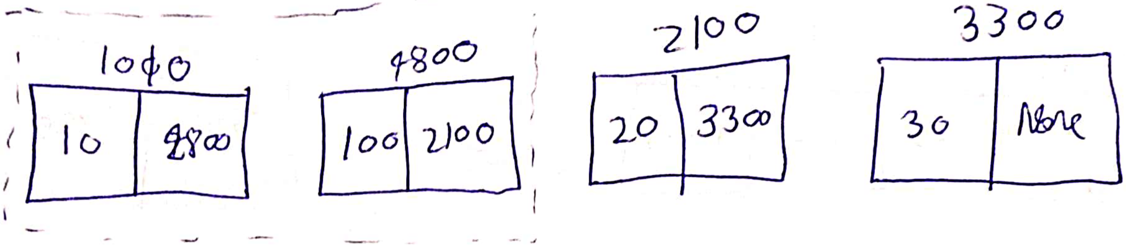
Adding Node

- Inbetween
  1) After Node
  2) Before Node

1)



2)

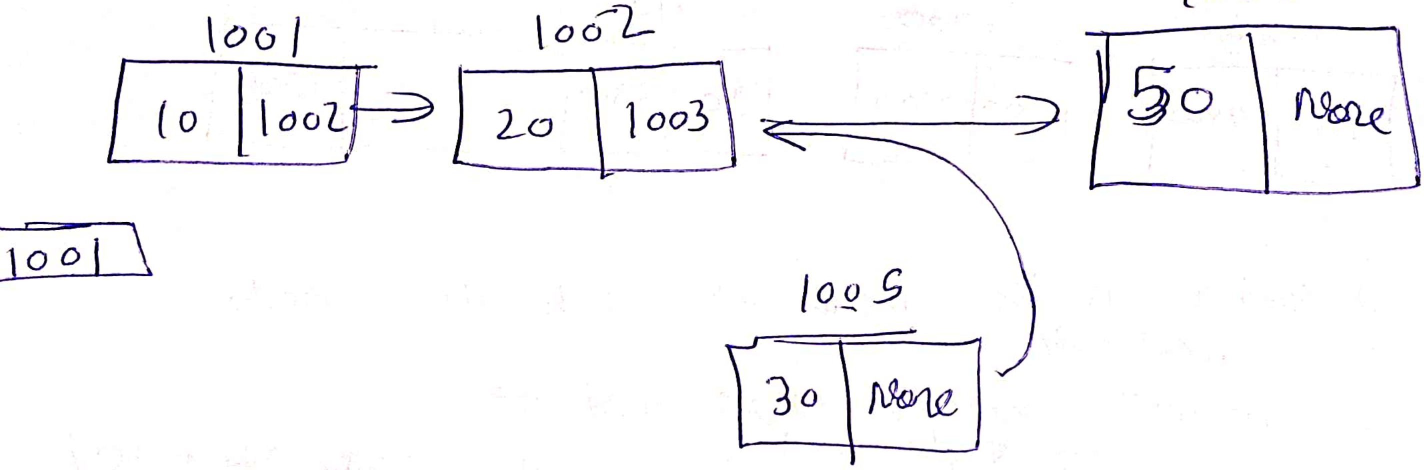


1) Find x is the self-data after which node we have to insert node

we will search in whole liked list

```
    x = No of to search
~~a~~ n = self-head                         obj = ~~[illegible]~~ Node(10)
while(n != None):
    if(n.data == x):
        temp1 = 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 = temp1      # and the remaining linklist is joined or
        break                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)~~ :
while (n.ref != None) :
    if (n.data == x) :
        break
    else :
        n = n.ref
~~if (n is not None)~~ :
if (n == none) :
    print ("Element is not present in liked 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 int the x ths reference
                           # we will assing 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 ~~so~~ 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.head = 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.ref != None):
        if (n.data = ~~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
```

Adding Node

- Before Node

  1) Before First Node
  2) Rest Position

1) Before First Node

   it will become the head node

```
obj = NewNode(data)
~~self.head~~
obj.ref = self.head
self.head = obj
```

2) Bfore Particular Node

1) Find Prev Node

2) New node after Prev node

Identify

~~New node~~ self.data == x

```
def add_before(self, data, x):
    if (self.head == None):                        } if there is
        print(" Linklist is empty ")               } no node
        return
    if (self.head.data == x):                      } if we want to add
        new_node = ~~Ds~~ Node(Data)               } before head
        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 = ~~Data~~ Node(data)    # we want to add here
            ~~new_n~~
            new_node.ref = n.ref
            n.ref = new_node
            break
        else:
            n = n.ref
    if (n.ref == None):
        print(" Node is not present in Linke list)    # if we search all
                                                      the linked list
                                                      we dont find
    else:
        new_node = Node(data)    # if we find
        new_node.ref = ~~Self~~ n.ref    the node
        n.ref = new_node
```

remaining cases

```
class Linked_List :
        def __init__(self):
            self.head = None
        def print_LL(self):
            if (self.head == None):
                print("Linked l-ist 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,x):
    n = self.head
    while (n.ref != None):
        ~~if (n.data ==~~
        if (n.data == x):
            break
    else:
        n = n.ref
if (n.ref == None):
    print("L~~inked~~ list is Not present")
    else:
        new_node = Node(data)
        ~~Add~~
        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
```