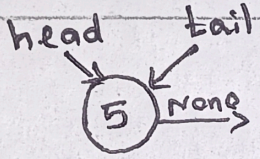
```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

class LinkedList:
    def __init__(self, value):
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 1

LL = LinkedList(5)
print(LL.head.value)
print(LL.tail.value)
```

5
5

here we didnt pass the next parameter to Node because its default to None this is the better way to create Node but technically we can also

separate node class to create new nodes.

Creating a new node and seting the head and tail to the node. its look like.

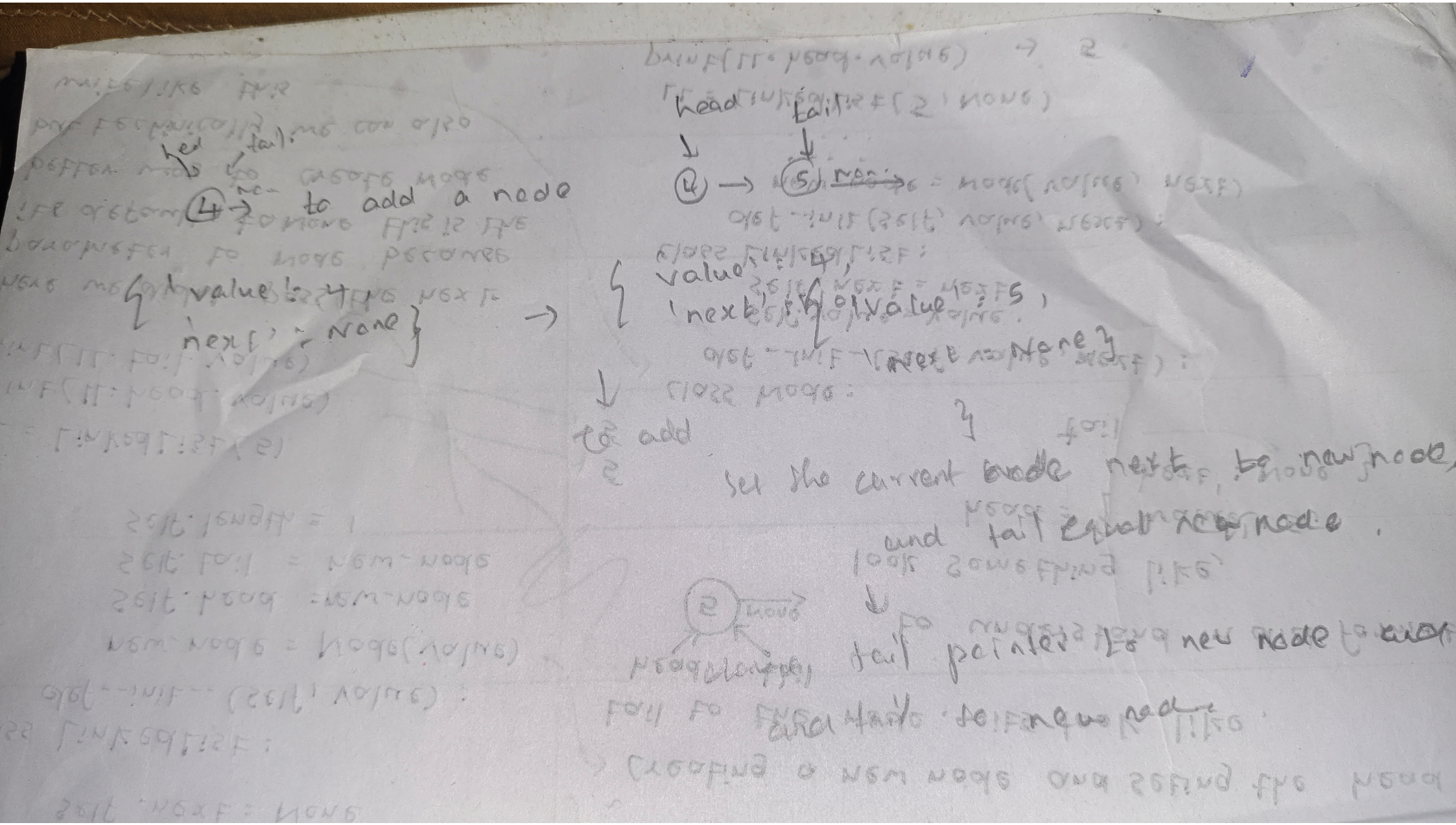


to understand in dict format it look something like,

head = { 'value': 5,
'next': None }

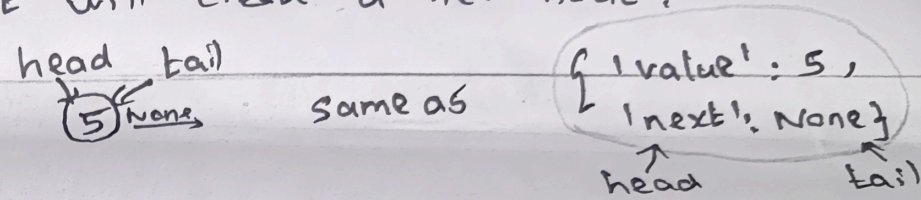
tail →

```
class Node:
    def __init__(self, value, next):
        self.value = value
        self.next = next
class LinkedList:
    def __init(self, value, next):
        new_node = Node(value, next)
        ....
        ....
LinkedList(5, None)
```

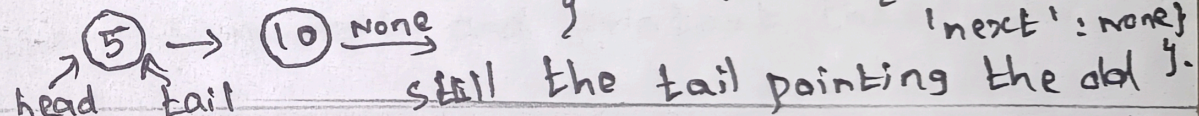

head
tail
4
5
value: 4
next: None
value: 5
next: None
to add a node
to add
set the current node next to new node
and tail equal new node.
2
next
tail points to new node

```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

class LinkedList:
    def __init__(self, value):
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node

    def append(self, value):
        new_node = Node(value)
        self.tail.next = new_node
        self.tail = new_node
        self.length += 1

ll = LinkedList(5)
ll.append(10)
```

it will create a new node.

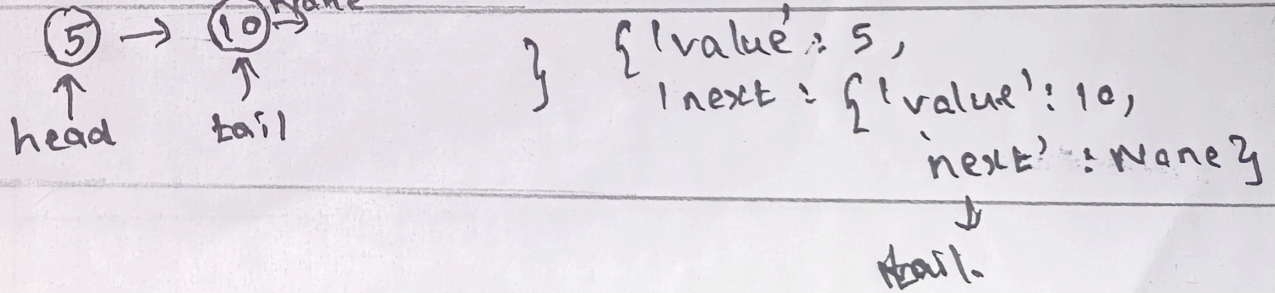


(5) Node have our node pointing none so when adding new node the last node tail is 5 pointing None get this pointing to new node like tail.next = new one



still the tail pointing the old.

and and also swith the tail point to last that the added one like self.tail = new one,



print:

```
ll = LinkedList(5)
ll.append(10)
ll.append(15)
ll.append(20)

def print_list(self):
    current = self.head
    while current is not None:
        print(current.value)
        current = current.next

ll.print_list()
```

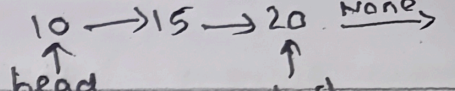
LinkedList class method.

when calling this function, to print our list, to start from, first set the head as for current and using while loop we need to stop when the current is None, until we move the pointer using .next and printout that value we get like.

5
10
15
20

how pop works:

↑ represents




```
Linked_list_dict = { "head" : { "value" : 10,
                                "next" : { "value" : 15,
                                           "next" : { "value" : 20,
                                                      "next" : None
                                                    }
                                         }
                              }
                   }
```

(this basis not work for all, like node have one element or none element need to add extra condition for that)

```
start = Linked_list_dict['head']
second = start['next']
third = second['next']
print(start)
print(second)
print(third)
```

→ {'value':10, 'next':{'value':15, 'next':{'value':20, 'next':None}}}

→ {'value':15, 'next':{'value':20, 'next':None}}

→ {'value':20, 'next':None}

conditions

```
if third['next'] is None:
    second['next'] = None
```

(1st) when we get the node next is None we can understand thats a end of the node to remove that first need to set the last before value next as None. that how we set the 15 node as None. now we removed the pointer. now its

head 10 → 15 None 20 (tail.)

but tail still pointing 20 for that we

```
int(start)
```

```
def pop(self):
    current = self.head
    current_before = self.head
    while current.next is not None:
        current_before = current
        current = current.next
    current_before.next = None
    self.tail = current_before
```

here we using 2 pointers moving current pointer and keep tracking with current_before when get current is None, make the before node as last node, change the pointer to None and add the tail.

(2). we need add edge case condition to work with i) list with one node and ii) Empty list.

pop function:

```
def pop(self):
    if self.length == 0:
        return None
    if self.length == 1:
        self.head = None
        self.tail = None
        self.length -= 1
        return
    current = self.head
    current_before = self.head
    while current.next is not None:
        current_before = current
        current = current.next
    current_before.next = None
    self.tail = current_before
    self.length -= 1
return current.value
```

if no values return None.

if only one node just make head and tail to None. decrease one size.

else do it like previously said. like traversal method & moving through linked list node by node until reach certain condition).

how prepend works :

```
Linked_list_dict = {"head" : {"value" : 10,
                              "next" : {"value" : 15,
                                        "next" : {"value" : 20,
                                                  "next" : None }
                                        } } }
```

new ~~node~~ :
{'value': 5, 'next' : None}

```
LL = Linked_list_dict
print (LL)
```

→ {'head' : {'value' : 10, 'next' : {'value' : 15, 'next' : {'value' : 20, 'next' : None }}}}

```
new = {'value' : 5, 'next' : LL['head']}
LL['head'] = new
print (LL).
```

→ {'head' : {'value' : 5, 'next' : {'value' : 10, 'next' : {'value' : 15, 'next' : {'value' : 20, 'next' : ~~None~~Node }}}}}

here to add a new node in front of the LL. just add the whole head to the new node next and get the head to the new node.

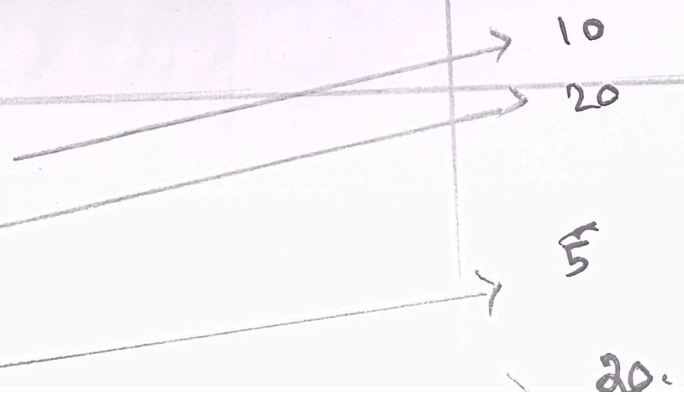
also need to write edge case Senerio, if LL is empty

just add the head and tail to new node.

prepend method -

```
def prepend(self, value):
    new_nod = Node(value)
    if self.length == 0:
        self.head = new_node
        self.tail = new_node
        return
    new_node.next = self.head
    self.head = new_node
    self.length += 1
    return new_node.value.
```

```
ll = LinkedList(10)
ll.append(15)
ll.append(20)
print(ll.head.value)
print(ll.tail.value)
ll.prepend(5)
```



pop first method

```
Linked_list_dict = {
    "head" : {"value" : 10, 'next': {'value' : 13, 'next' :{'value' :20,
                                                            'next': None}
                                     }}}
```

```
ll = linked_list_dict
print (ll)
temp = ll ['head'] ['next']
ll ['head'] = temp
print (ll)
```

here to remove the first value we gonna set the head.next to a new variable and assigened that as a linkedlis head

#. also need to handle edge case for list is empty (or) have one.

pop first.

```
def pop_first(self):
    if self.length == 0:
        return None
    if self.length == 1:
        self.head = None
        self.tail = None
        self.length -= 1
        return
```

handling edge cases.

```
to_remove = self.head.value
temp = self.head.next
self.head = temp
self.temp:
self.length -= 1
return to_remove.
```

→ just stored to return the poped value.

→ Storing all the other node without the head.

→ assigninging the non head added temp to head and decreased one length

→ and retured that poped value.

get and set method.

```
def get(self, value):
    pointer1 = self.head
    index = 0
    while pointer1 is not None:
        if value == index:
            return pointer1.value
        pointer1 = pointer1.next
        index += 1
    return None.
```

To get the value from LL using index using pointer we start from head through loop this until condition, and increasing index if the index matches return the current pointer node value. else none in end the index not there.

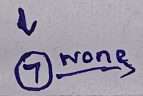
```
def Set(self, set_index, value):
    pointer = self.head
    index = 0
    while pointer is not None:
        if index == set_index:
            pointer.value = value
            return True
        index += 1
```

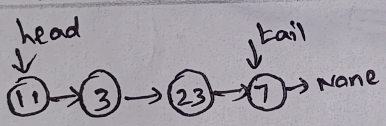
same as here to set value from index and value. if the current index matches the set_index we passing, change the pointer value to what we pass

Doubly LinkedlisL.

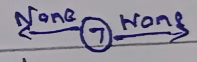
Constructor.

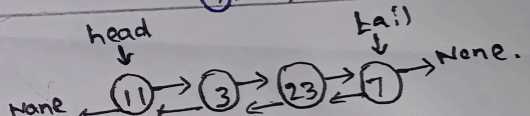
Single LL ⇒ {'value' : 7
'nexcE : None}





Doubly LL ⇒ {'value' : 7 ,
'next' : None,
'prev' : None }





```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None
        self.prev = None

class DoublyLinkedList:
    def __init__(self, value):
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 1

def print_list(self):
    temp = self.head
    while temp is not None:
        print(temp.value)
        temp = temp.next
```

7 same as {'value': 7, 'next': None, 'prev': None}