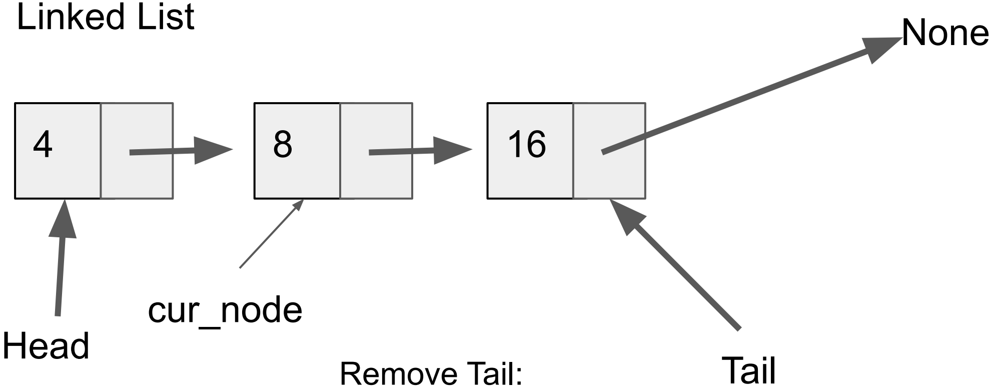
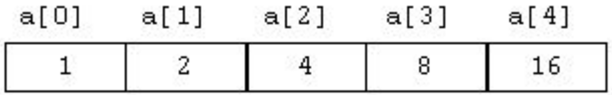
Linked List



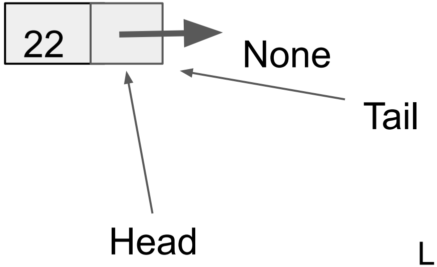
Array



a[0]	a[1]	a[2]	a[3]	a[4]
1	2	4	8	16

Add to tail

```
1)  Check if there's a tail
2)  If there is a tail (general case):
    a)  Create a new node with the value we
        want to add, next = None
    b)  Set current tail.next to the new node
    c)  Set self.tail to the new node
3)  If there is no tail (empty list)
    a)  Create a new node
    b)  Set self.head and self.tail to the new
        node
```



Remove Tail:

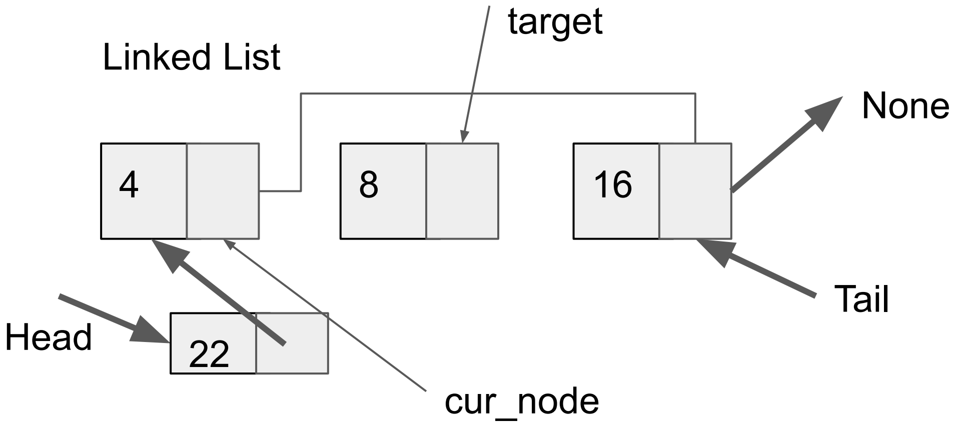
```
Check if it's there
General case:
  1)  Start at head and iterate to the next-to-last node
      a)  Keep moving current_node forward
      b)  Stop when current_node.next == self.tail
  2)  Save the current_tail value
  3)  Set self.tail to current_node
  4)  Set current_node.next to None
```

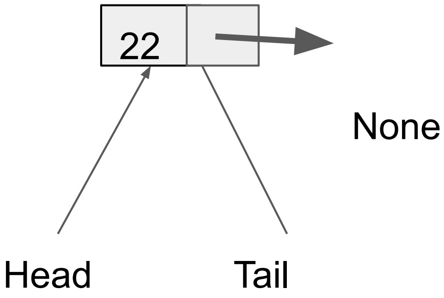
List of 1 element:

```
Save the current_tail.value
Set self.tail to None
Set self.head to None
```

Remove head:

```
Check if head
    If head (General case):
        Set self.head to current_head.next
        Return current_head.value
    If not head (empty list)
        Return None
    List with one element:
        Set self.head to current_head.next / None
        Set self.tail to None
    Decrement length by 1
```





Add to head:

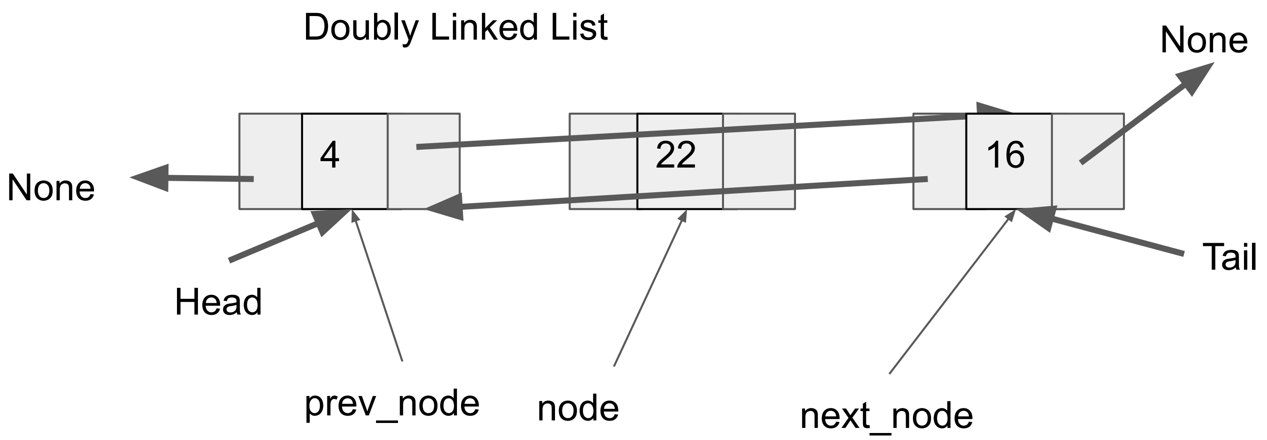
1) Is there a head?
2) If no head / empty list:
   a) Create the new node with next = None
   b) Set self.head = new node
   c) Set self.tail = new node
3) If head:
   a) Create the new node
   b) New_node.next = self.head
   c) Set self.head = new_node

Remove at index i:

0) Check that length > i. If not, return None

-- we need a pointer to previous node

1) Iterate through the loop i-1 times:
   a) Current = self.head
   b) For i-1 times…
      i) Current = current.next
2) To_remove = cur_node.next
3) Cur_node.next = to_remove.next
4) To_remove.next = None



```
def delete(self, node):
    -   Check for empty pointers
    -   Get previous node = node.prev
    -   Set prev_node.next to node.next
    -   Next_node = node.next
    -   Set next_node.previous = previous_node
    -   Decrement length
    -   Set node.prev = None
    -   Set node.next = None
    -   Return node.value
```

```
Get max: return the maximum value in the list
    -   If length == 0 return None
    -   If length == 1 return self.head.value
    -   Current_max starts out as self.head.value
    -   Iterate through the list
        -   Stop when current_node is None
    -   Compare current_max to each value and
        update current_max if value >
        current_max
    -   Move current_node forward
    -   Return current_max
```