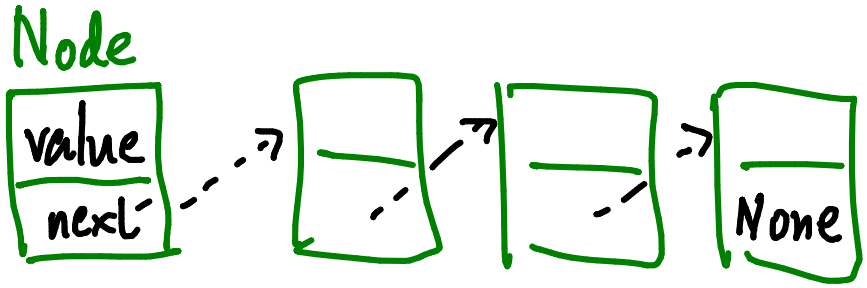
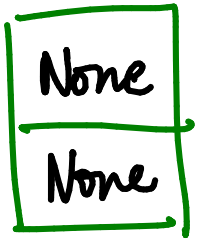
Classes & Objects

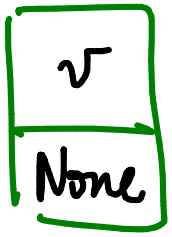
"Linked" list



Empty list



Singleton



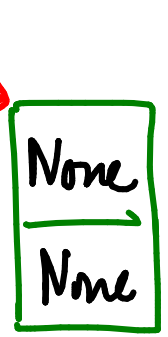
To Do: insert(v)

append(v)

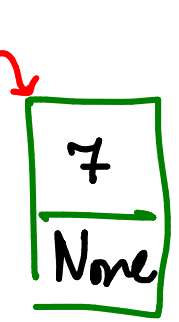
delete(v)

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

n = Node()

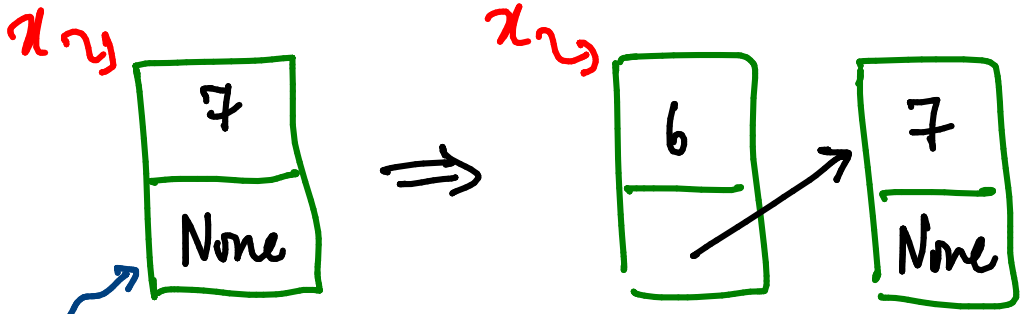


x = Node(7)



Insert

x.insert(6)



self — cannot change what self points to

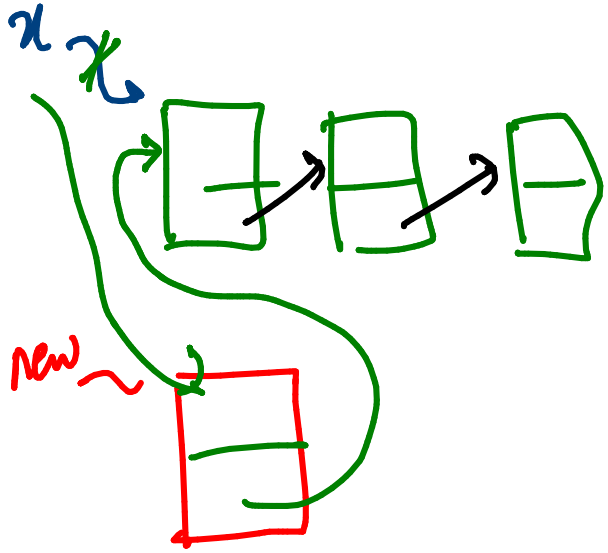
First attempt

```
def insert(self, v):
```

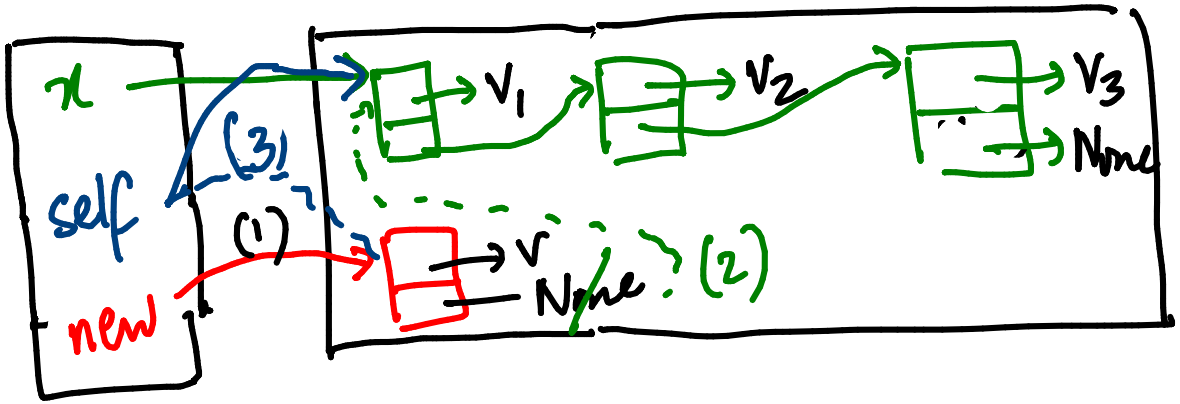
(1) `new = Node(v)`

(2) `new.next = self`

(3) `self = new` !!

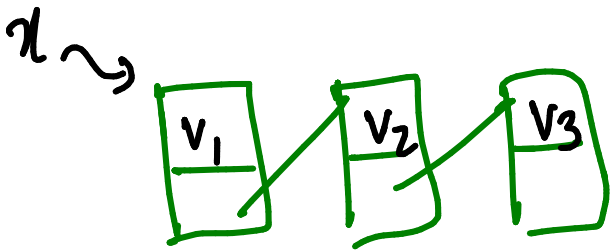


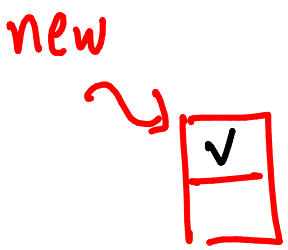
↳ x doesn't change!



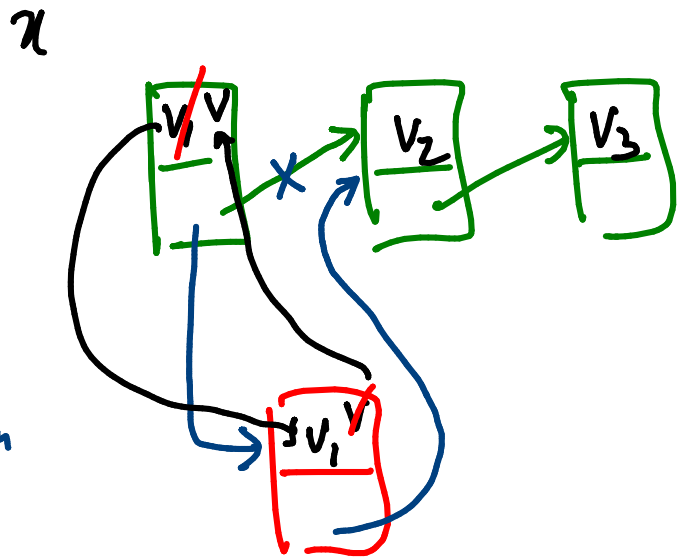
Solution

Plumbing





"Insert" new between v_1 & v_2



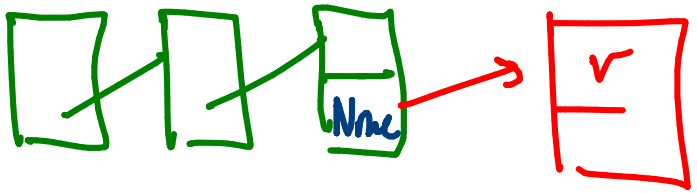
Exchange values v & v_1

```
def insert(self, v):
    if self.isempty():    # self.value == None
        self.value = v
        return
    new = Node(v)
    new.value = self.value
    new.next = self.next
    self.value = v
    self.next = new
    return
```

new = Node(self.value)

append(v)

```
def append(self,v):
    if self.isempty():
        self.value = v
        return
    if self.next == None:
        new = Node(v)
        self.next = new
    else:
        self.next.append(v)
    return
```



Recursive implementation

Explicitly traverse links till you reach None

```
def append_iterative(self,v):
    if self.isempty():
        self.value = v
        return
    appendpos = self
    while appendpos.next != None:
        appendpos = appendpos.next
    new = Node(v)
    appendpos.next = new
    return
```

Same effect, but not good style

```
while self.next
        !=None:
    self = self.next
```

Typically

function calls are expensive

recursion costs more than iteration

Tail recursion

Recursive call is last step

Return value is not required to complete current call

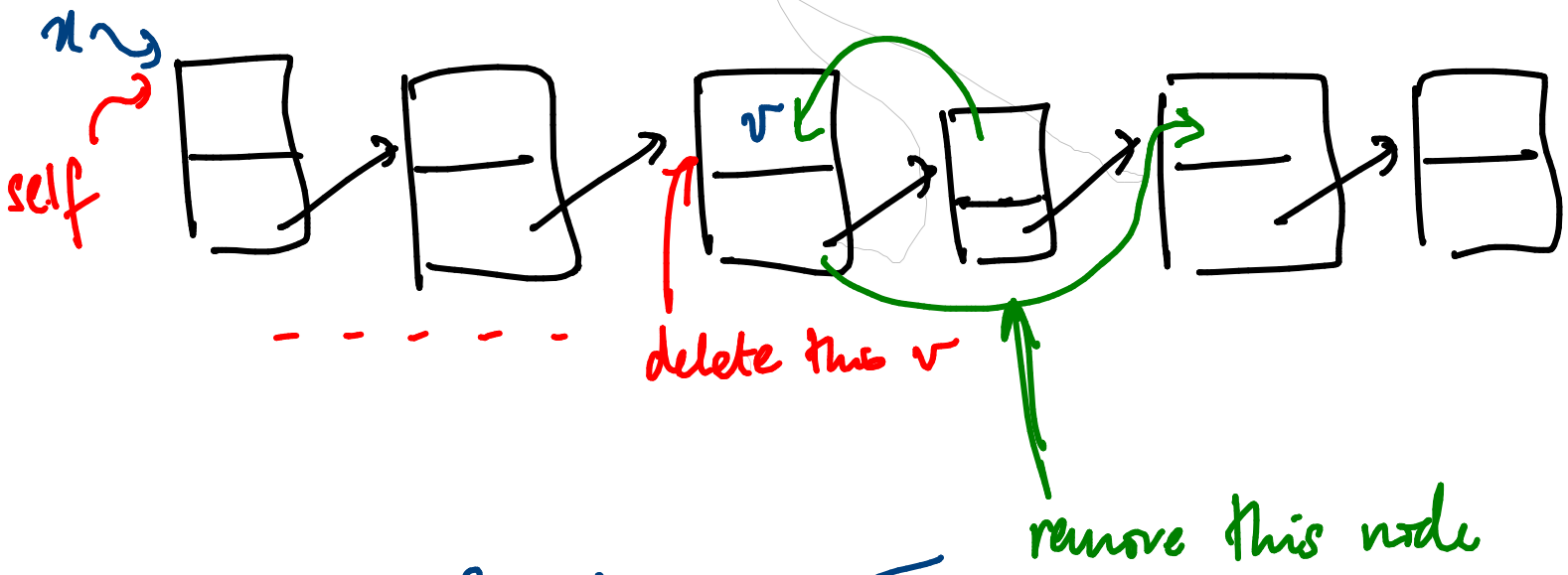
Compilers can optimize tail recursion as loop

```
factorial(n)
    ⋮
return (n * factorial(n-1)) vs return (factorial(n-1) * n)
```

delete(v) — define meaning first

-no v — do nothing

-multiple v- delete first v



What if v is last value in list?

look ahead one step

if self.next.value == v, bypass next node

```
def delete(self,v):
    if self.isempty():
        return
    if self.value==v and
       self.next==None:
        self.value=None
        return
```

Fix special cases and clean up

```
if self.value==v and
   self.next!=None:
    self.value=self.next.value
    self.next=self.next.next
if self.next.value==v:
    Bypass next node
else
    self.next.delete(v)
```