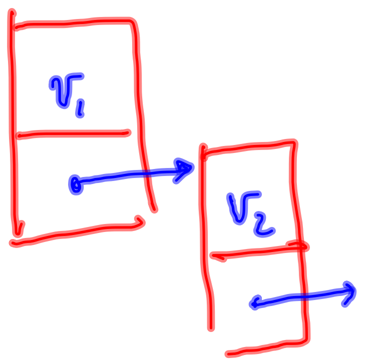
class List:

```
    def __init__(self, x=None):
        self.value = x
        self.next = None
        return
```



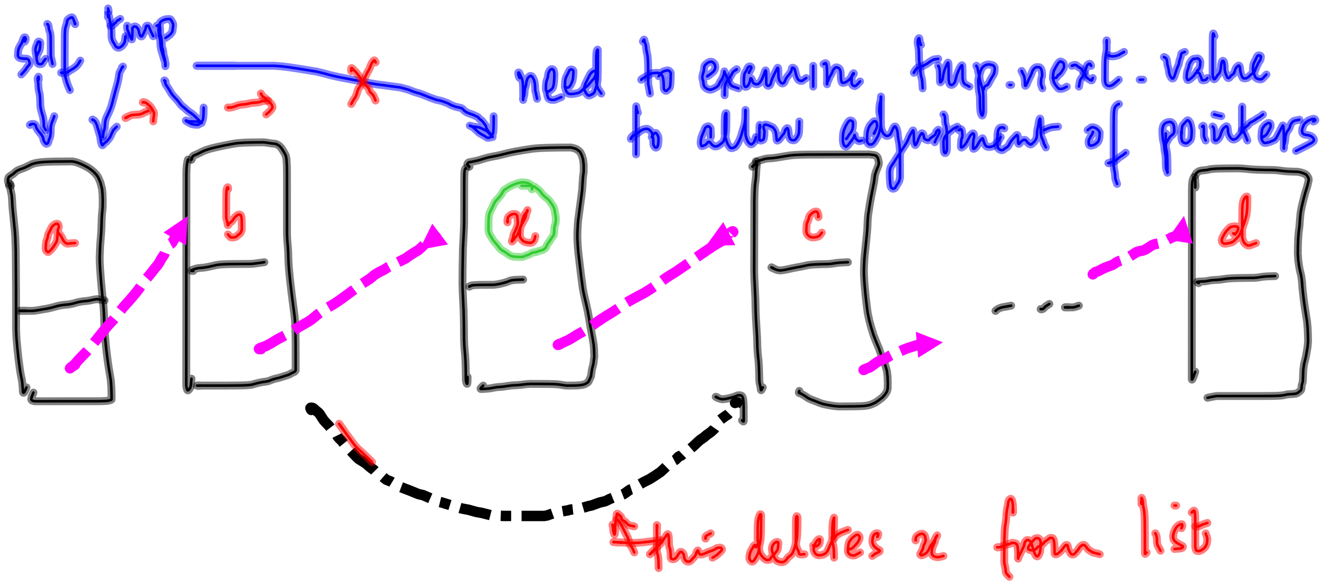
append < append recursive / append iteratively

insert  may need to copy values and adjust pointer delete:

def delete(self, x):

delete first occurrence of x in the list, if any

need to examine tmp.next.value to allow adjustment of pointers



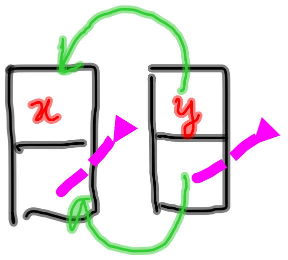
Boundary condition

Last node is deleted (>1 node) ✓

First value is x

next is None — set value to None ✓

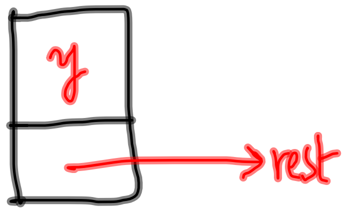
next != None — copy value & next from next node



```
def deletei(self, x):
    if self.isempty():                    || empty list
        return
    if self.value == x:
single  || if self.next == None:
value      self.value = None
  x        return
        else:
first x, ||  self.value = self.next.value
≥2 vals      self.next = self.next.next
             return
    # ~~2 or more nodes~~,
    # first value != x
    tmp = self
    while tmp.next != None:
        if tmp.next == x :
            tmp.next =
                tmp.next.next
            return
        else:
            tmp = tmp.next
    return
```

## Delete recursively?

Difficulty: Must delete x from current node

delete x 

```
if x==y
    delete this value y
else
    delete x from rest
```

delete value $\longrightarrow$ set value to None

Cannot have value None inside a list

## Recursive step

```
if self.value == x :
    self.value = None
else
    if self.next != None:
        self.next.deleter(x)
        if self.next.value == None:
            self.next = self.next.next
```

What if first value is x?

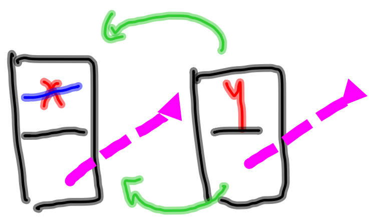
next is None

self.value = None

$[x] \longrightarrow [\,]$

next != None

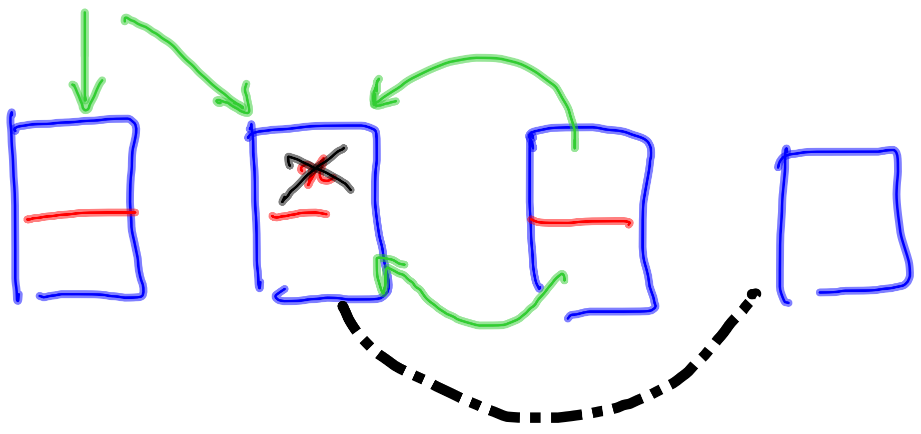
Copy second node & bypass it, like delete i

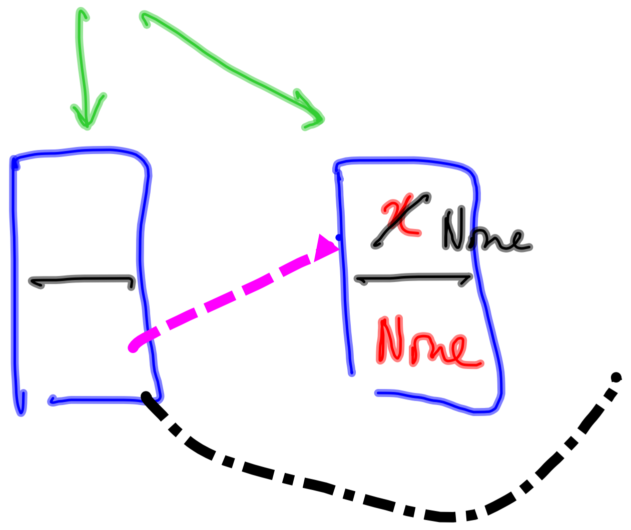


Code:

See sample code

Good case

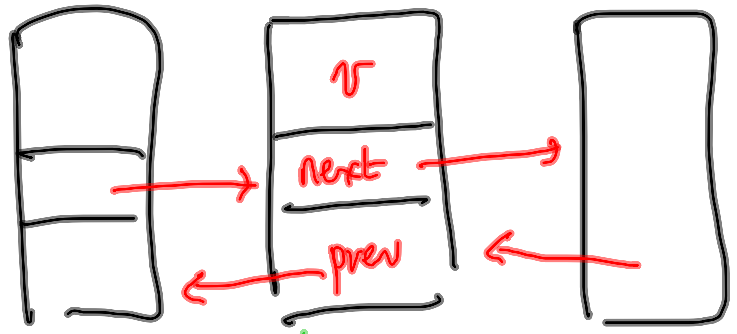



~~x~~ None
None

We are free to define our structures as we want

Lists – going backwards is tedious

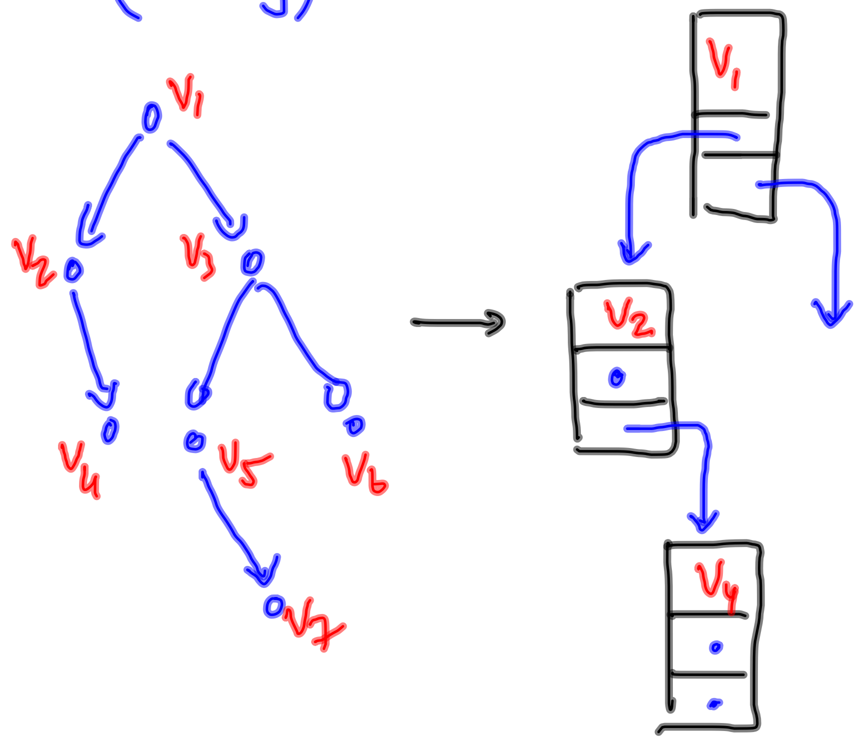
We could redesign our nodes



Doubly Linked lists

Need to update both pointers on insert, delete

Lists $\rightarrow$ Trees (binary)



v1
v2
v3
v4
v5
v6
v7
v1
v2
v4