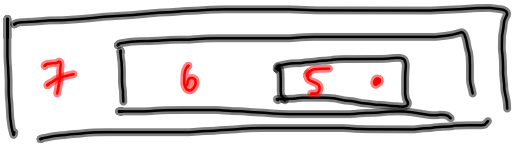
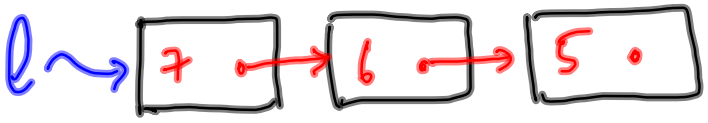
Classes & user defined datatypes

Recursive datatypes

```
data List a = Nil | Node a List a

Node 7 (Node 6 (Node 5 Nil))
```

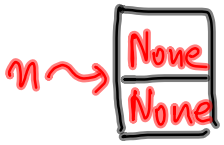




← More natural for Python classes

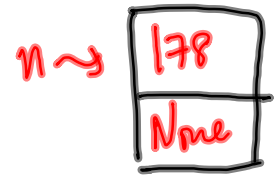
```
class Node:
    def __init__(self, initval = None):
        self.value = initval
        self.next = None

    def append(self,x):
        =

    def isempty(self):
        return (self.value == None)
```

```
n = Node()
```



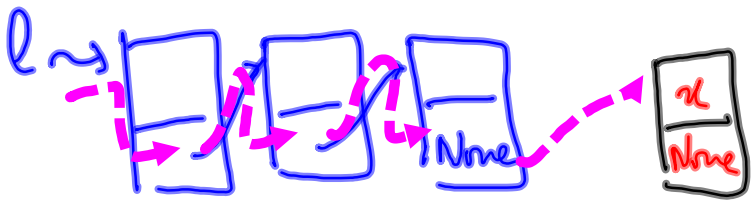
[]

```
n = Node(178)
```



[178]

l.append(x)



Node(x)

append l x

append Nil x = Node x Nil

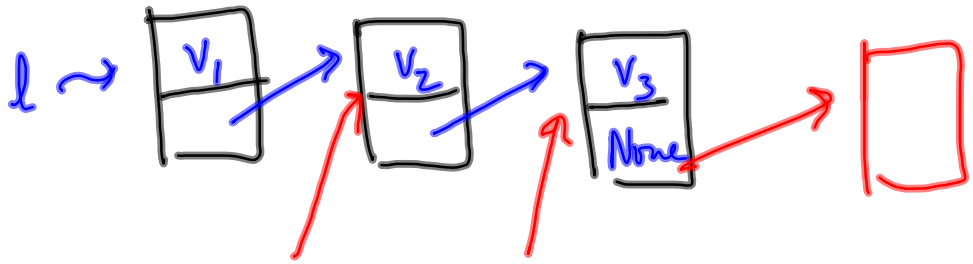
append (Node y l) x = Node y (append l x)

```
def append (self, x):
    if self.isempty():
        self.value = x
        return
    # self.next.append(x)
    if self.next == None:
        newnode = Node(x)
        self.next = newnode
        return
    self.next.append(x)
```

append Nil x = Node x Nil

Won't work if self.next == None

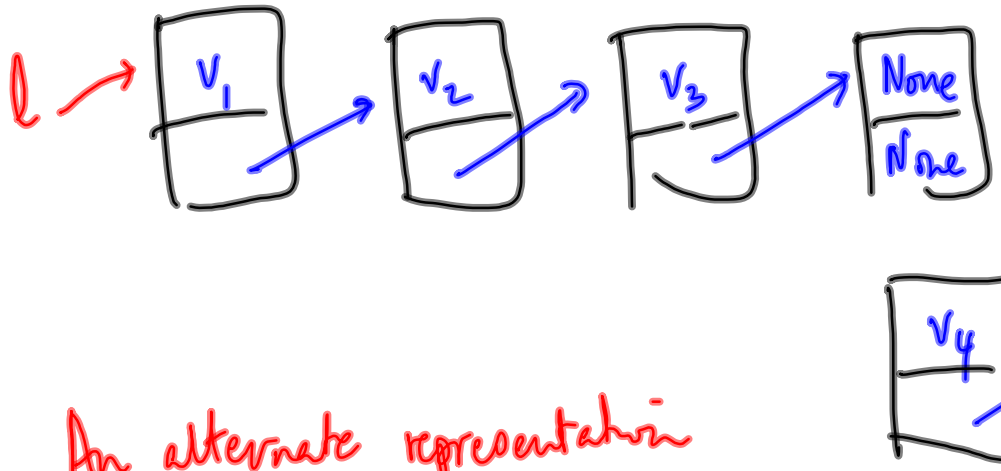
self.next = Node(x)



Alternatively, walk to end of list in a loop

```
# self.value != None
blah = self
while blah.next != None:
    blah = blah.next
newnode = Node(x)
blah.next = newnode
```

← blah is the last node in list

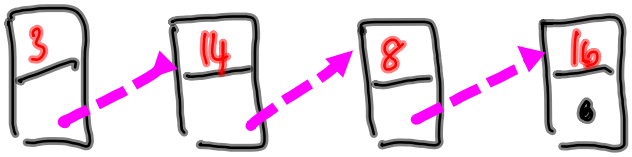


An alternate representation for lists

Need only one base case

```
def delete(self,x):      # remove first x in l, if any
```

delete 8



to delete head, copy 14 and delete second node

current.next.value is 8

bypass next node

current.next = current.next.next

Work with alternative representation

```
class Node:
    def __init__(self, initval = None):
        self.value = initval
        if initval != None:      # initval could be 0
            self.next = Node()
        else:
            self.next = None
```

```
def append(self, x):
    if self.value == None:
        self.value = x
        self.next = Node()
    else:
        self.next.append(x)
```

```
def delete(self, x):
    if self.value == None:
        return
    if self.value != x:
        self.next.delete(x)
        return
    # Copy second node
    self.value = self.next.value
    self.next = self.next.next
    return
```

Some more features of Python classes

Special functions

__str__ implicitly called when str( ) is invoked on object

Should return string

Similar functions implicitly invoked for +, *

Can use one function to define another

```
class Queue:
    .
    .
    .
    def rotateq(self):
        val = self.removeq()
        self.insertq(val)
```