Linked Lists

Accouncements

- Last day to turn in midterm regrades is tomorrow
- Project Party 3:00 - 5:30 pm in Woz, Soda 430-438
- Lab 08 and HW 04 due today
- Ants has been released!
  - Checkpoint 1 due 7/21
  - Checkpoint 2 due 7/25
  - Project due 7/28
- Last python lecture!!

Why Linked Lists?

Python lists are implemented as a "dynamic array", which isn't optimal for all use cases.

😭 Inserting an element is slow, especially near front of list:

"A"	"B"	"C"	"D"	"E"	"F"
0	1	2	3	4	5
3300	3301	3302	3303	3304	3305

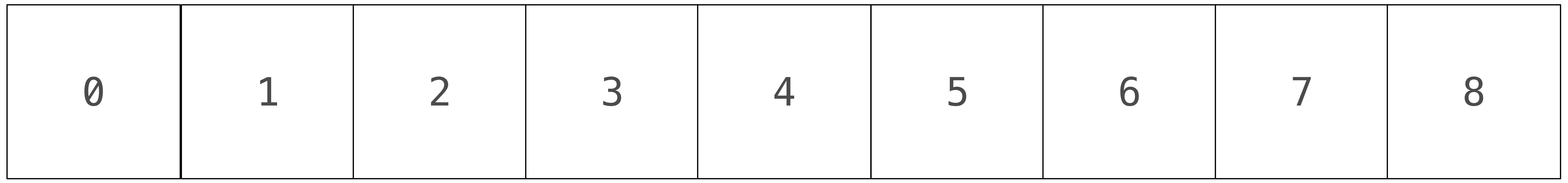
L.insert(“AA”, 1)

⬇

"A"	"AA"	"B"	"C"	"D"	"E"	"F"
0	1	2	3	4	5	6
3300	3301	3302	3303	3304	3305	3306

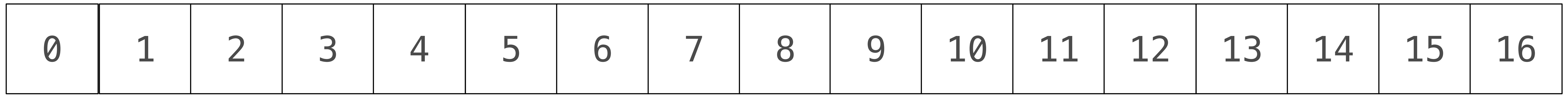
List Operations

Insert

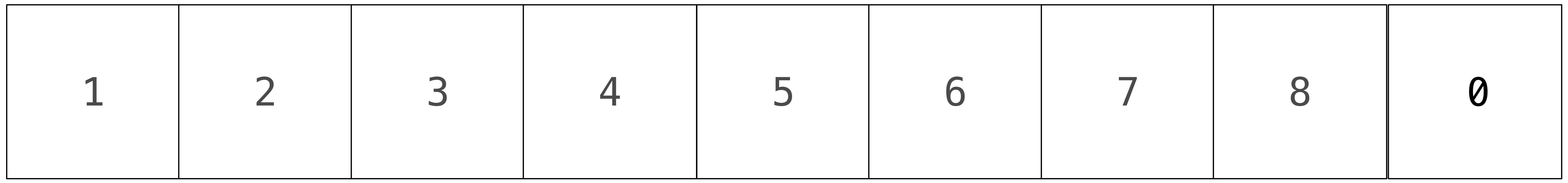


```
L.insert(0, 0)
```

Linear



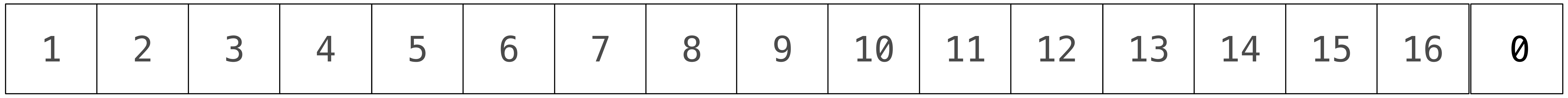
Append



`L.append(0)`

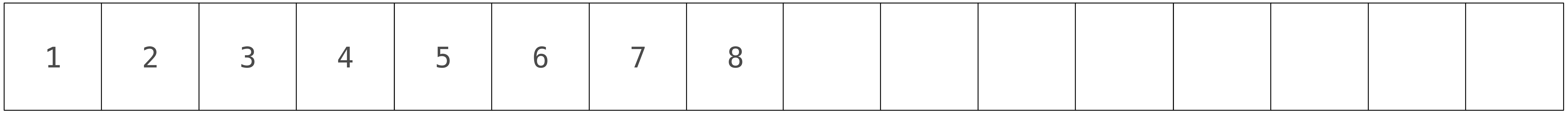
Linear

Find new memory and copy over old elements



Append

Allocate twice as much memory as requested

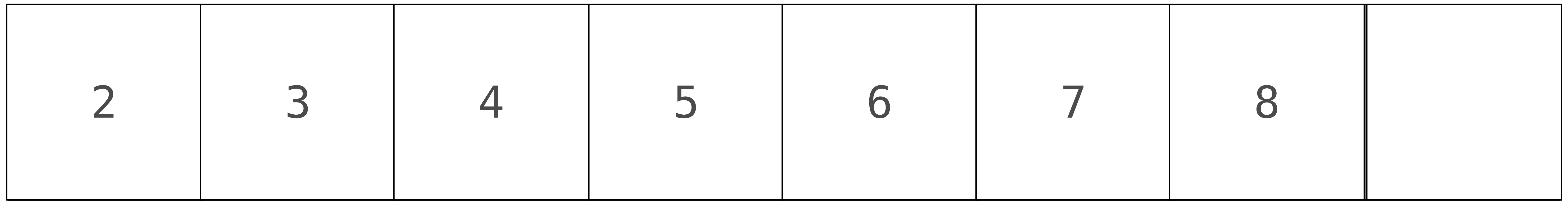


L.append(0)

Constant, but sometimes, linear

Inserting too many elements can require re-creating the entire list in memory, if it exceeds the pre-allocated memory.

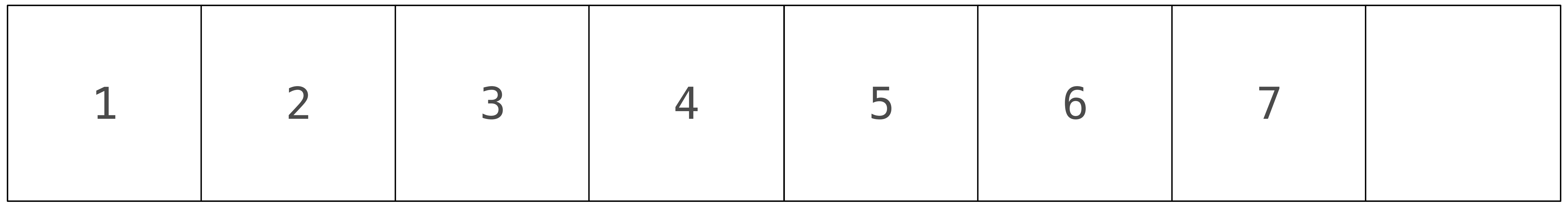
Delete



```
del L[0]
```

Linear

Delete



```
del L[7]
```

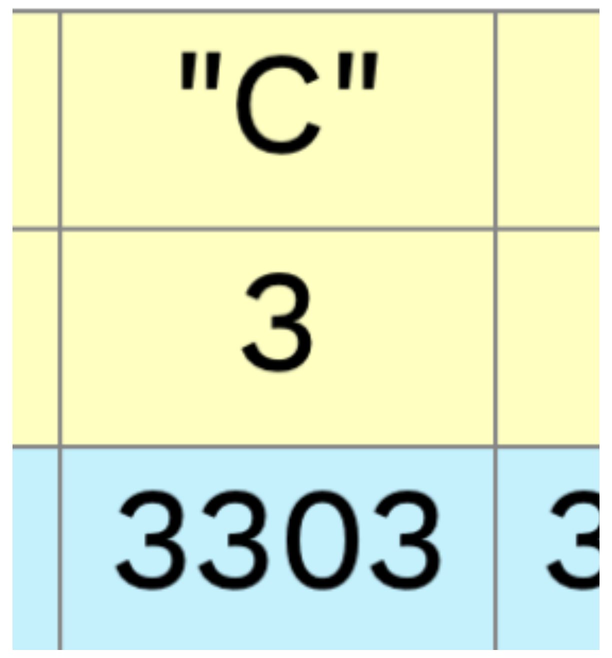
Constant

Access

`Python lists`

"A"	"B"	"C"	"D"	"E"	"F"
0	1	2	3	4	5
3300	3301	3302	3303	3304	3305

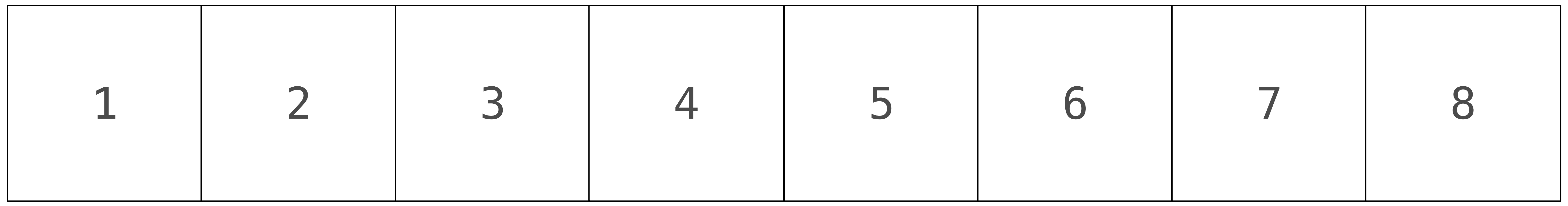
`l[2]`



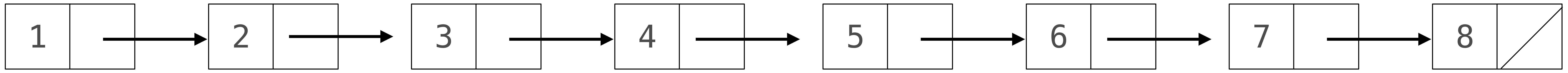
Linked Lists

Lists vs. Linked Lists

A list is like a bus

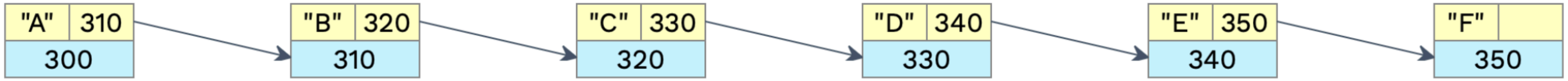


A linked list is like a train



Linked Lists

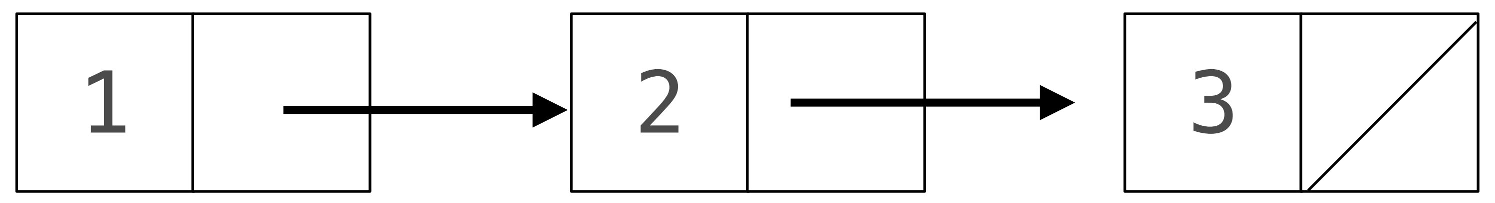
A linked list is a chain of objects where each object holds a **value** and a **reference to the next link.** The list ends when the final reference is empty.



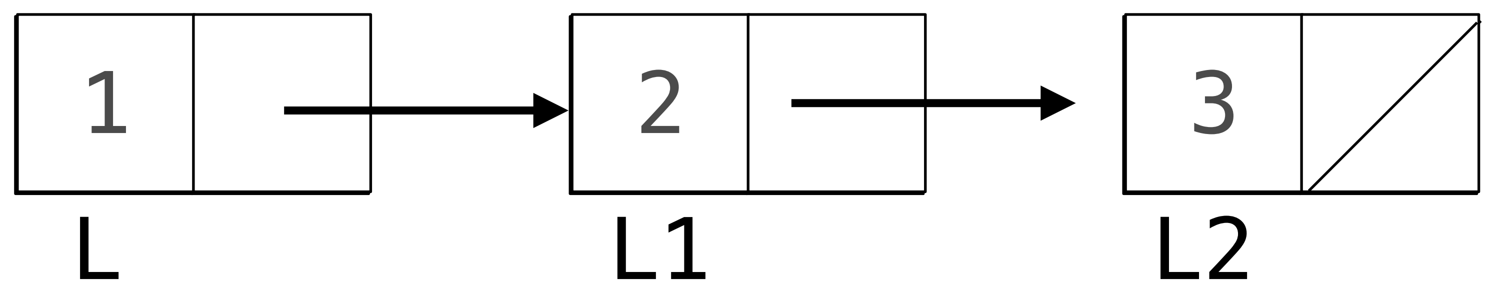
Linked List Class

Linked Lists Class

```
class Link:
    empty = ()

    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest
```



```
L = Link(1)
L2 = Link(2)
L3 = Link(3)
```



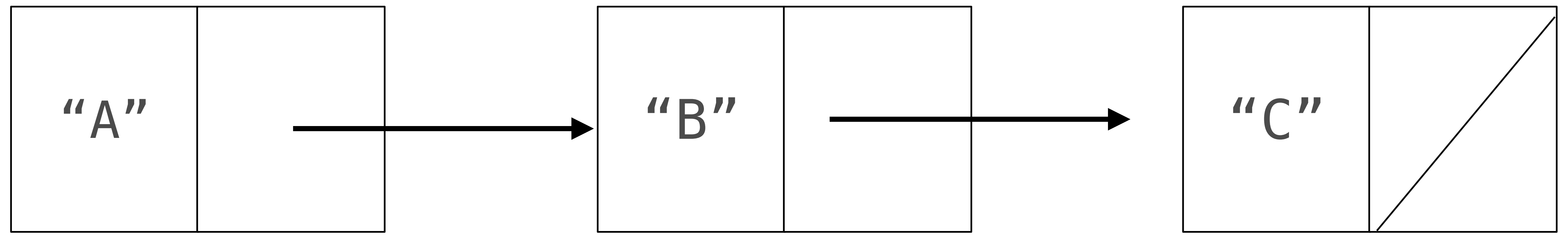
L.rest = L2
L1.rest = L3

```
L = Link(1, Link(2), Link(3))
```

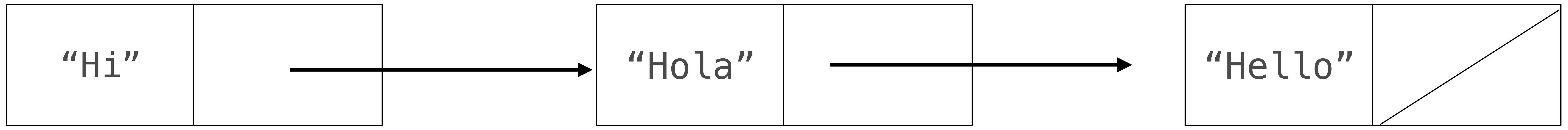
Mutating Linked Lists

Attribute assignments can change `first` and `rest` attributes of a `Link`

```
s = Link("A", Link("B", Link(“C")))
```



```
s.first = "Hi"
s.rest.first = "Hola"
s.rest.rest.first = “Hello"
```



Beware Infinite Lists

The rest of a linked list can contain the linked list as a sub-list.

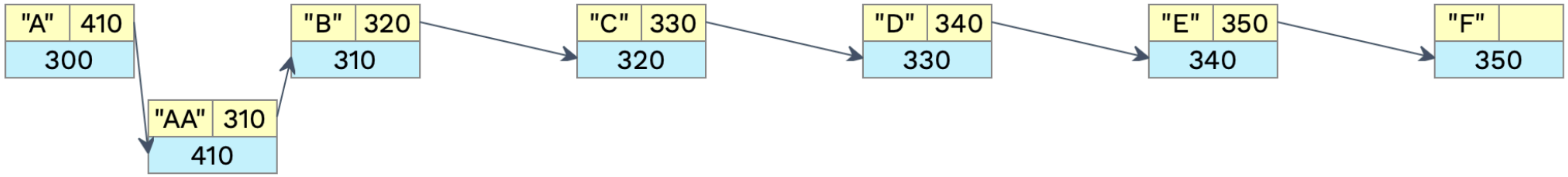
Demo

Iterative Print Linked List

Linked List Operations

Insert

Linked lists require more space but provide faster insertion.



```
Inserting “AA” after the first node, s = Link(“AA”)
temp = L.rest
L.rest = s
s.rest = temp
```

Constant

Linked lists require more space but provide faster insertion

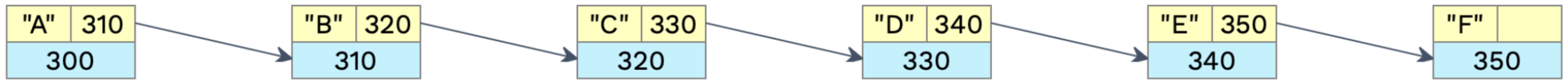
insertAfter method

```
class Link:
     empty = ()

     def __init__(self, first, rest=empty):
         self.first = first
         self.rest = rest

    #insert a node, l after a node
    def insertAfter(self, l):
            temp = self.rest
            self.rest = l
            l.rest = temp
```

Delete

No matter which node you want to delete, it takes one step:

Point the rest of the node *before* the one to delete to the one *after*



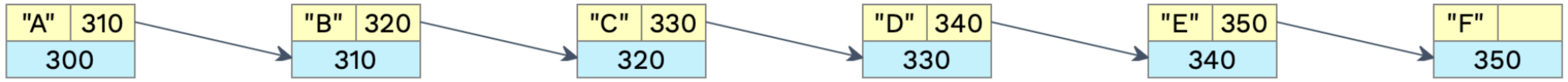
Deleting "AA"

```
L.rest = L.rest.rest
```

Constant

Access

I need to iterate through all the previous nodes using .rest



Accessing “D”

```
node_b = L.rest
node_c = node_b.rest
node_d = node_c.rest
```

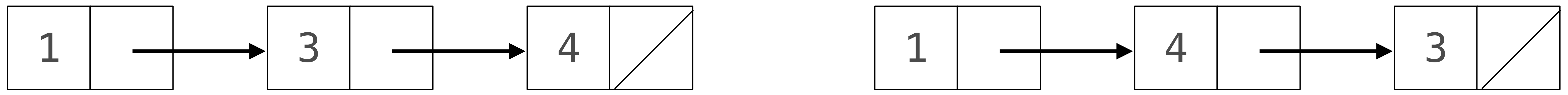
Linear

Comparing Operations

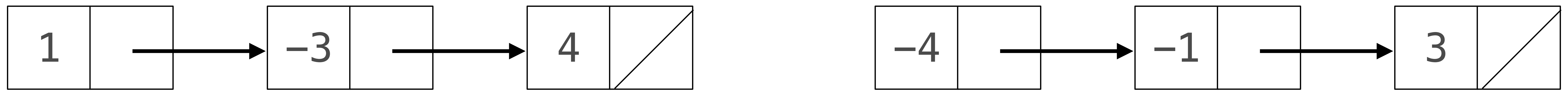
```
Lists

     insert:   linear
     append:   constant, sometimes linear
     delete:   linear
     find:     linear
     access:   constant

Linked Lists

     insert:   constant
    delete:   constant
     find:     linear
```

Linked List Exercises

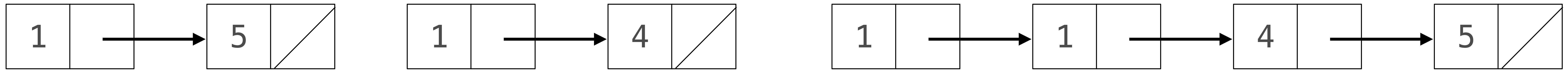
Is a linked list s ordered from least to greatest?



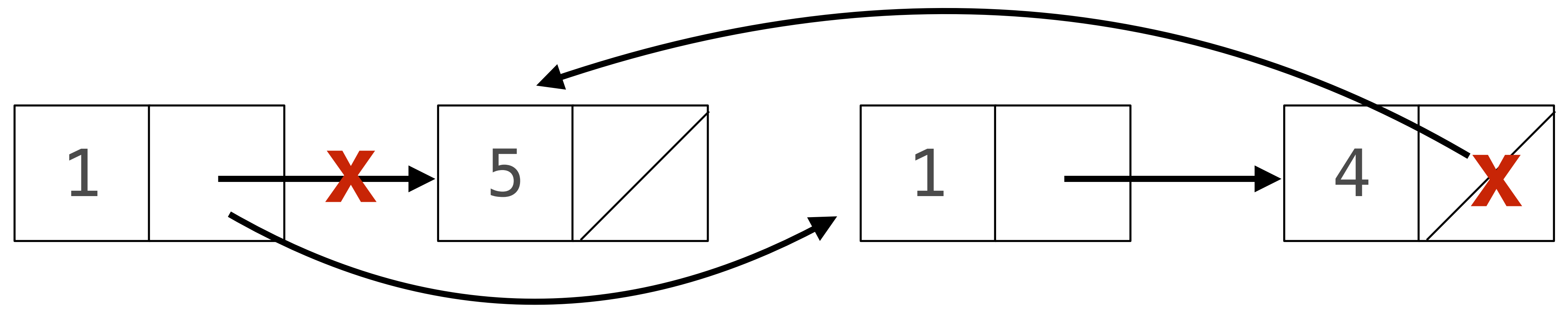
Is a linked list s ordered from least to greatest by absolute value (or a key function)?



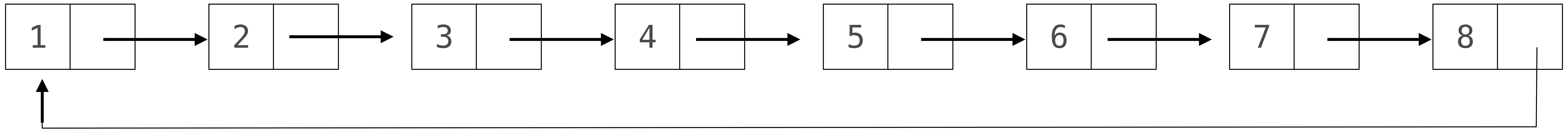
Create a sorted Link containing all the elements of both sorted Links s & t.

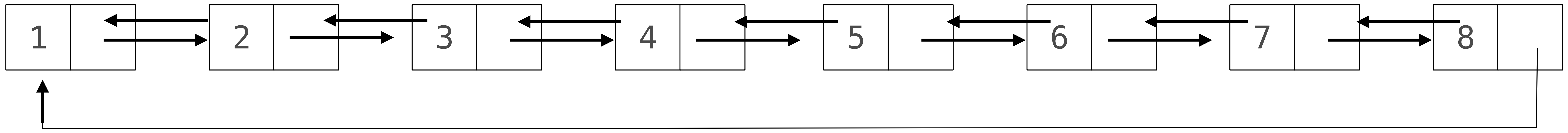


Do the same thing, but never call Link.



Circular, Doubly Linked Lists





Doubly Linked List

```
class Dlink:
    def __init__(self, data):
        self.data = data
        self.next = self
        self.prev = self
```

```
dl = Dlink(9)
```