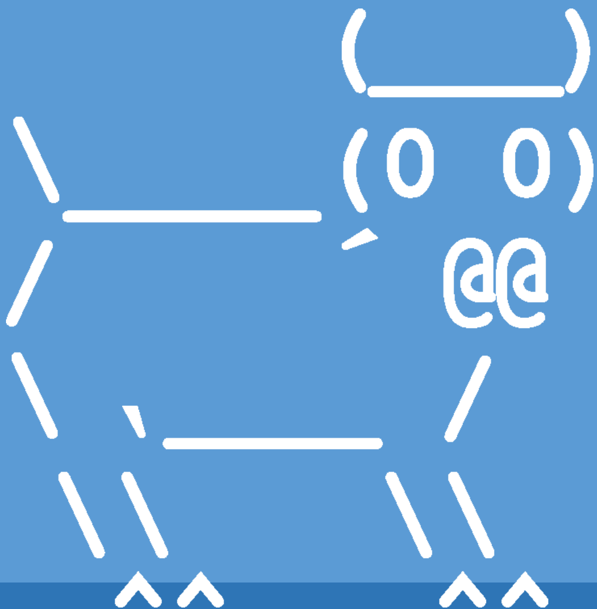
LinkedList Elements Methods



Introduction to Computer Science

Iris Howley

TODAY'S LESSON

A private class holding values in a list

(Building more methods for the Element class)

Steps for Recursion

1. Know when to stop.
2. Decide how to take one, repeated step.
3. Break the journey down into that step plus a smaller journey.

Recursive Approach

- **REDUCE** the problem to smaller subproblem(s) (smaller version(s) of itself)
- **DELEGATE** the smaller problems to the recursion fairy *(formally known as induction hypothesis)* and assume they're solved correctly
- **COMBINE** the solution(s) of the smaller subproblems to reach/return the solution

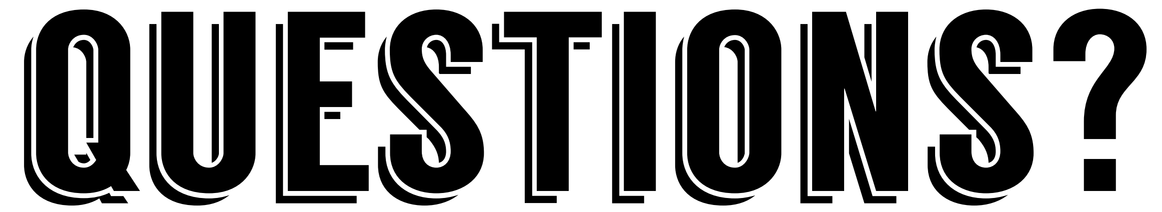
Linked Lists – Element Class

- See example code on the course website!

LinkedList.py

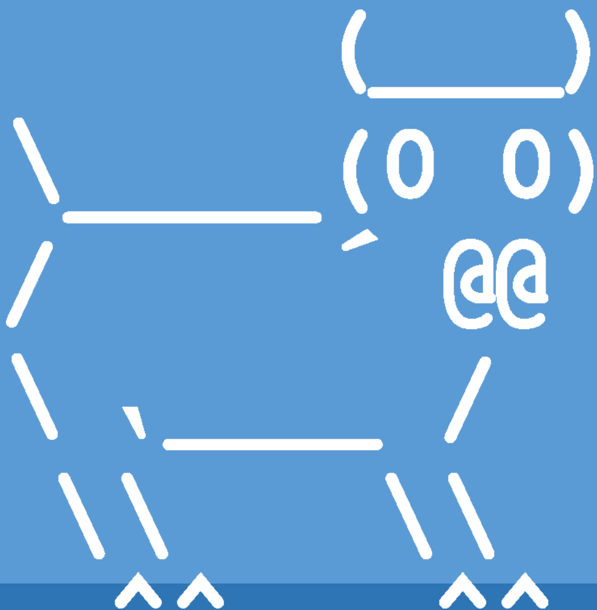
Testing Element getitem & setitem in interactive Python

```
>>> from LinkedList import Element
>>> e1 = Element(1234)
>>> e2 = Element(5, e1)
>>> str(e2)
'(value=5,next=(value=1234,next=None))'
>>> e2[1] = 'abcd'
>>> str(e2)
'(value=5,next=(value=abcd,next=None))'
```



Please contact me!

Linked Lists



Introduction to Computer Science

Iris Howley

TODAY'S LESSON

`Linked List Class`

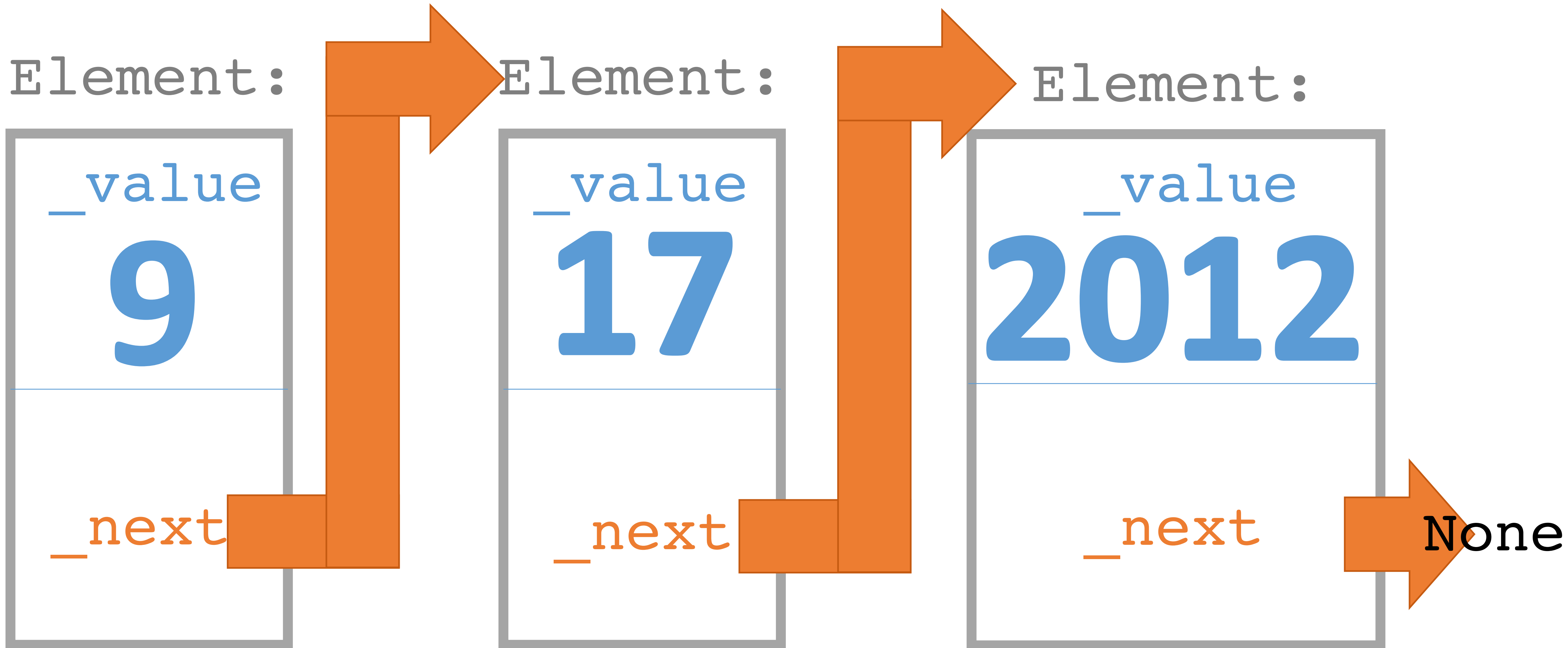
(A public face for linked Elements for better encapsulation)

HOW TO CREATE AN EMPTY LIST?

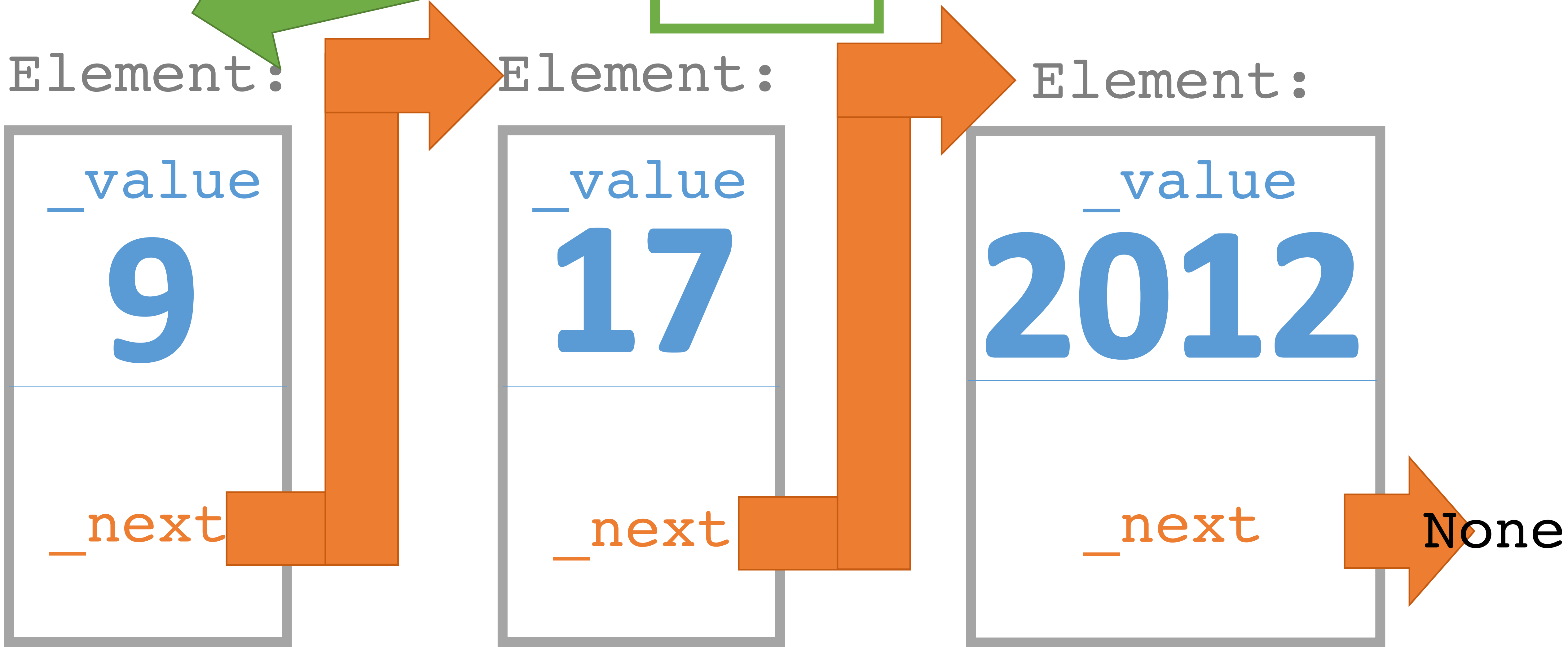
- `elemList = Element(None)`

- ...This isn't actually an empty list!
- `len(elemList)` would return 1!
- It's an Element list with one item with value **`None`**

What is a list?



class LinkedList: _head



Linked List

- Class LinkedList is a “wrapper class” for our “container class”, Element

- This implementation, LinkedList mostly:
  1. Points to the first element of the list
  2. Handles the empty case
  3. Passes the heavy-lifting (i.e., all other cases) to Element

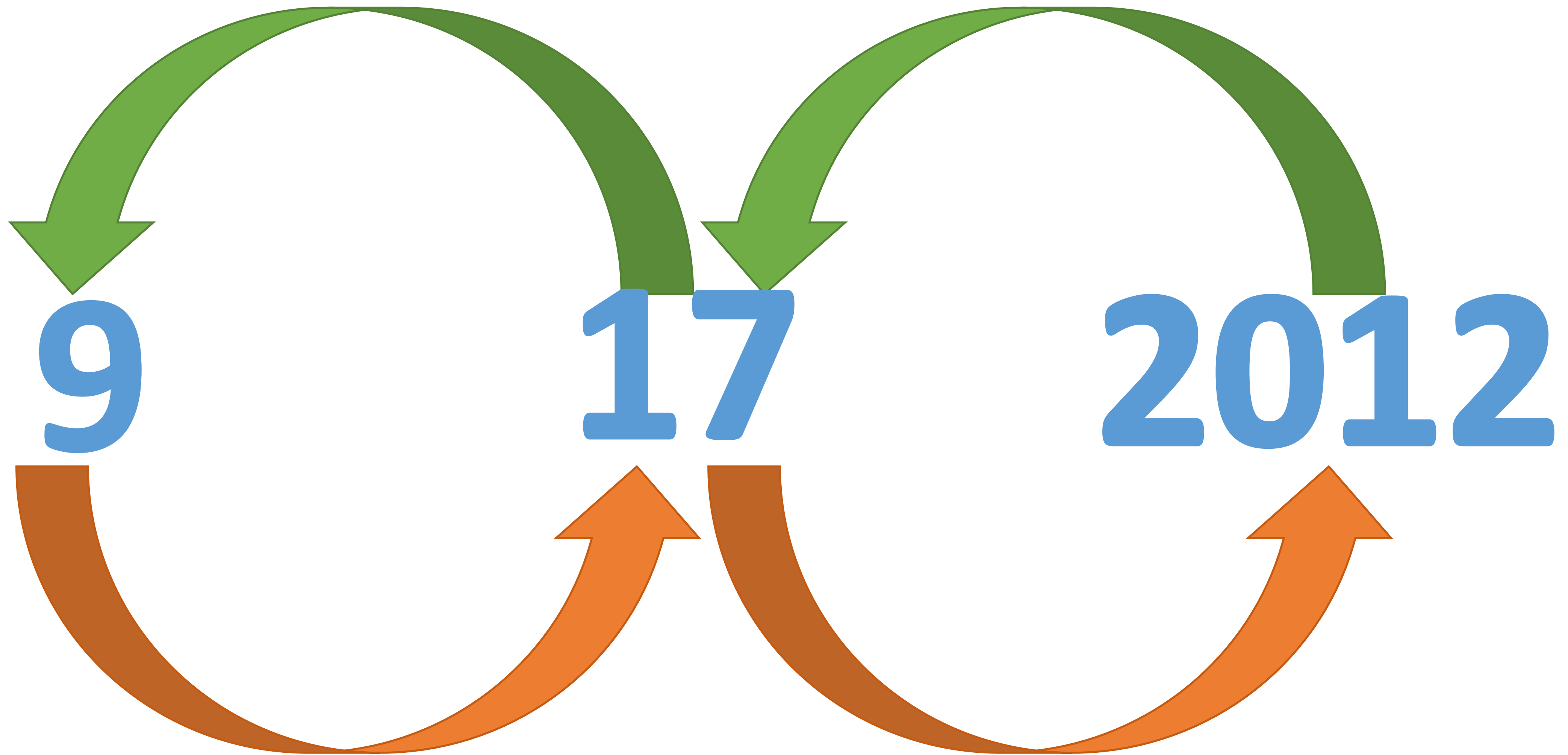
Linked Lists – LinkedList Class

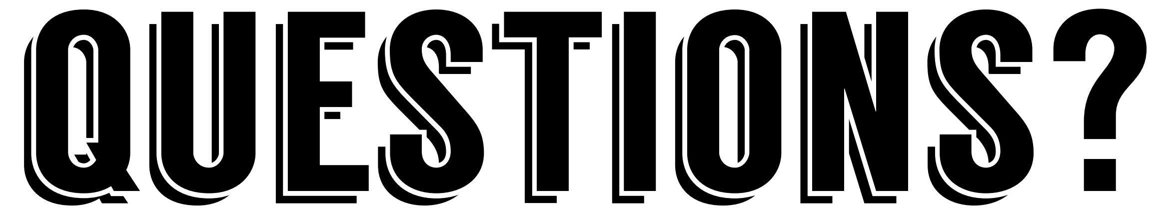
- See example code on the course website!

LinkedList.py

Thought question:

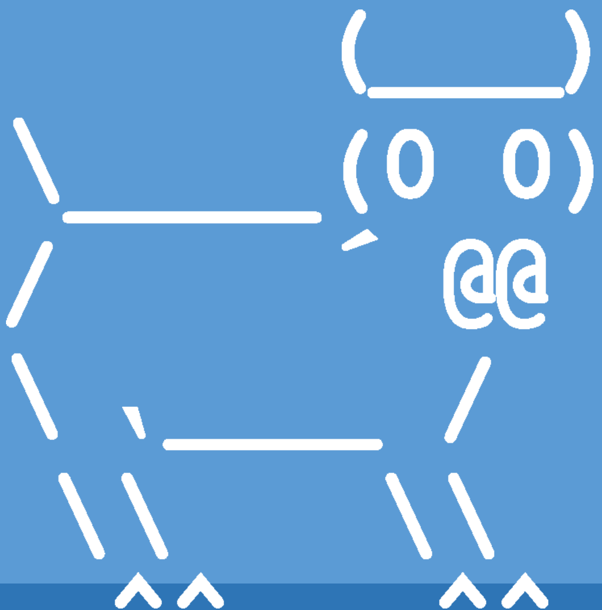
How would you build a doubly-linked list?





Please contact me!

Sorting Linked Lists



Introduction to Computer Science

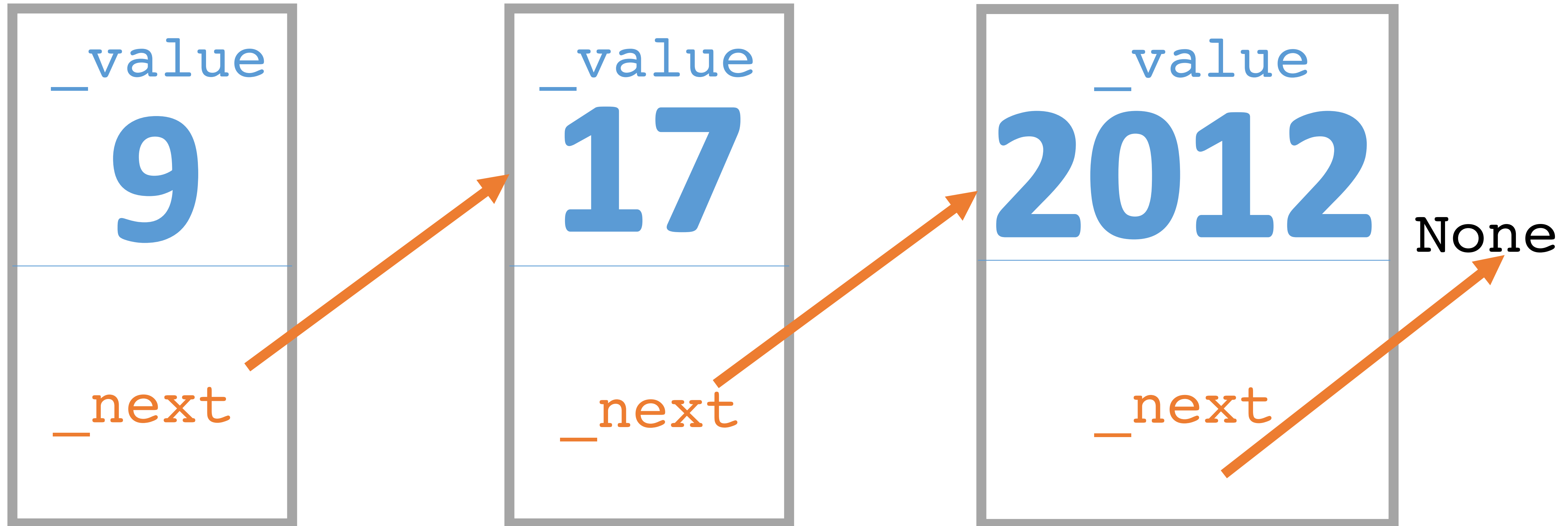
Iris Howley

TODAY'S LESSON

`Rearranging our LinkedList`

(Arranging data is useful, let's start with reversing it)

Element: Reverse

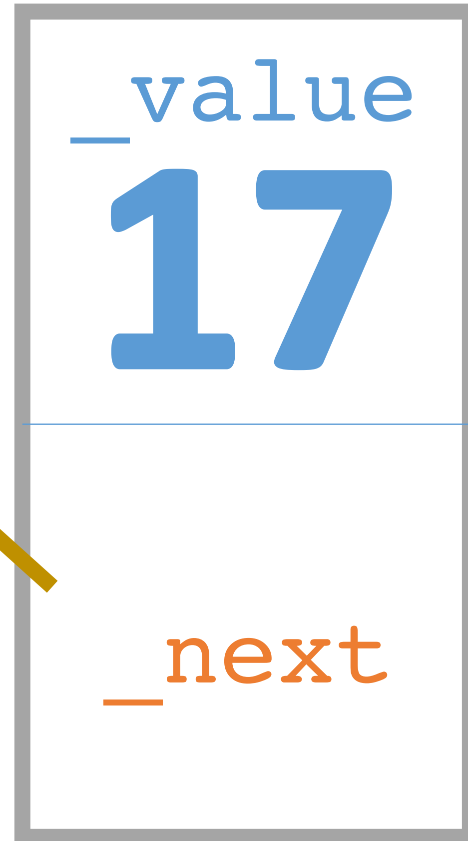


Element: Reverse

None

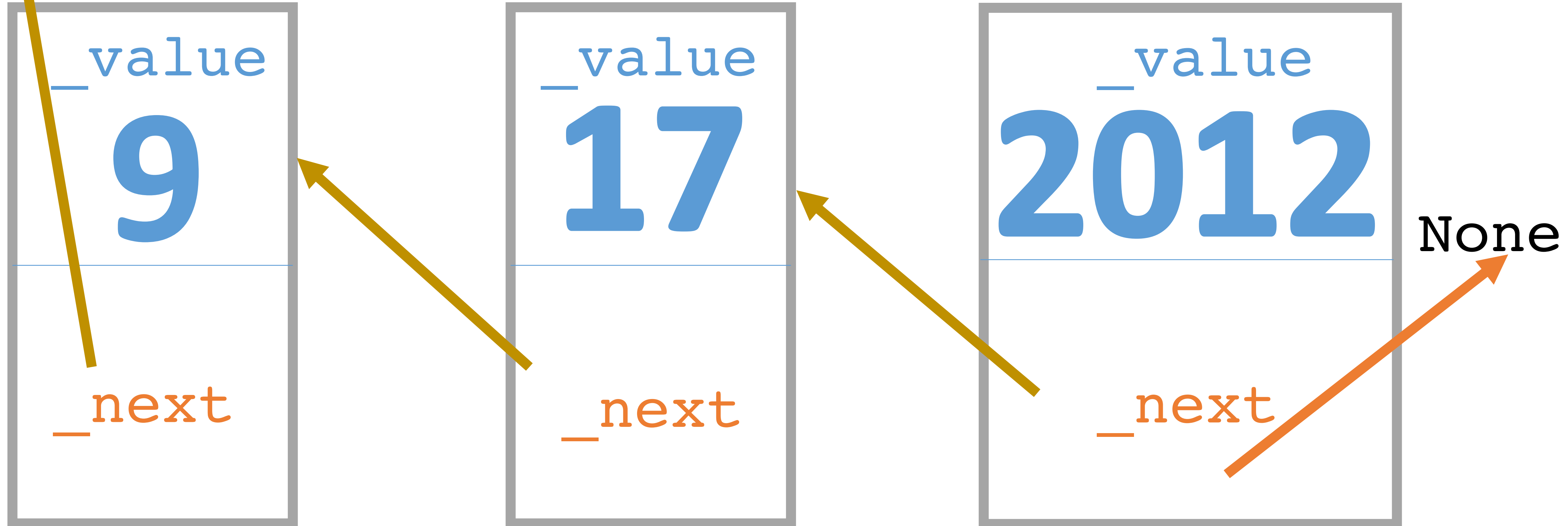
Element: Reverse

None

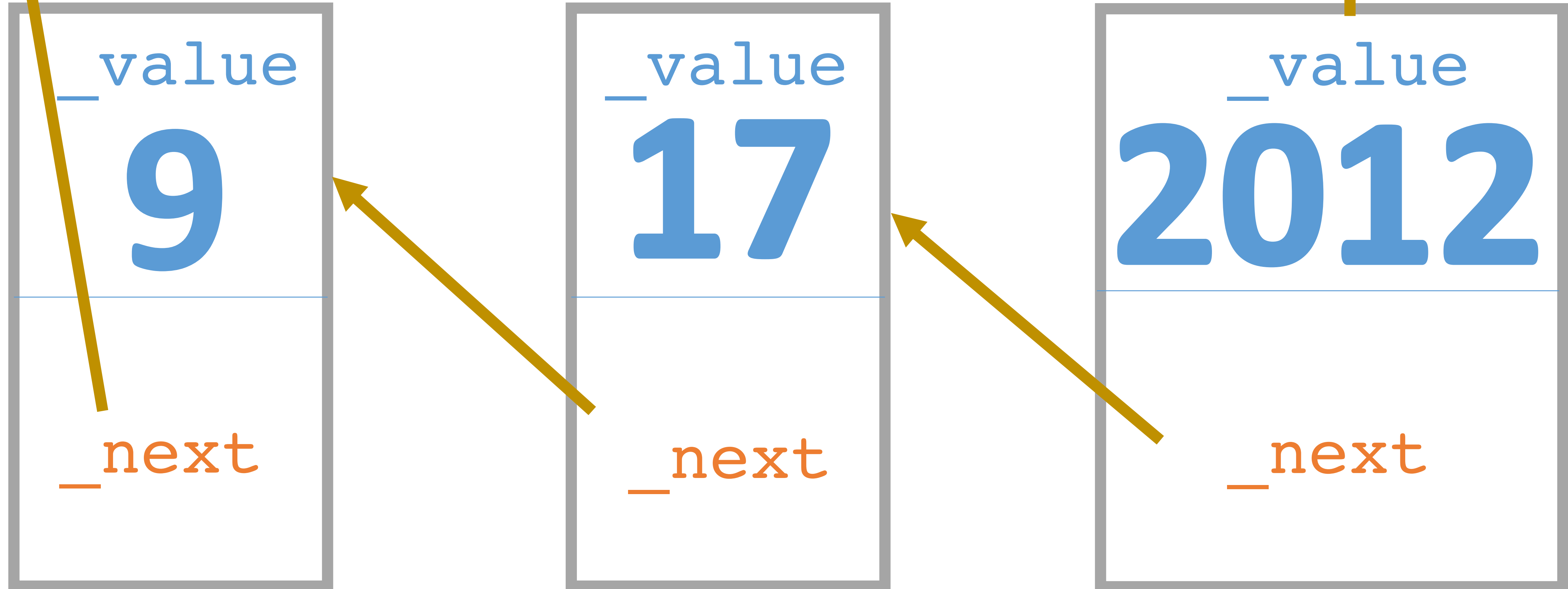


None

Element: Reverse



Element: Reverse



Element: Reverse

- Reverse the order of the elements in this list
- *Repeated step:* Make this Element's `.next` be the previous Element
- *Stopping condition:* If we're at the end of the list, `self` is the new front of the list
- *Breaking journey into smaller pieces*: Otherwise, call `reverse` on the next Element, passing `self` as an argument representing the new `.next` of the next Element

Element: Reverse

```
class Element:
  def reverse(self, newNext=None):
    oldNext = self.next
    self.next = newNext
    if oldNext is None:
      return self
    else:
      return oldNext.reverse(self)
```

LinkedList: Reverse

- Reverse the order of the elements in this list
- If this isn't an empty list, then use Element's `reverse` method on the `_head` of this list

```
def reverse(self):
    if self._head is not None:
        self._head = self._head.reverse()
```

TODAY'S LESSON

Implementing a sort method

(Sorting lists is very useful, how is it done?)

Linked Lists – LinkedList Class

- See example code on the course website!

LinkedList.py

Element: Ordered Insert

- Given a value, inserts that value in the correct (ordered) location of the list
- *Stopping conditions*:

1. If our value is less than the current value, then insert it before & return it
2. Otherwise, check to see if we're at the end of the list, and if so, then this value becomes the last item in our list (return previous Element)

- *Breaking journey into smaller pieces*: Otherwise, keep comparing to rest of list items (return previous Element)

Sorting LinkedLists

```
class Element:
 def orderedInsert(self,v):
   if v <= self.value:
     return Element(v,self)
   elif not self.next:
     self.next = Element(v)
     return self
   else:
     self.next =
     self.next.orderedInsert(v)
     return self
```

Testing orderedInsert in Interactive Python

```
>>> from LinkedList import Element
>>> ele = Element(3)
>>> print(ele.orderedInsert(4))
(value=3,next=(value=4,next=None))
>>> print(ele.orderedInsert(1))
(value=1,next=(value=3,next=(value=4,next=None)))
```

LinkedList: Sort

- Create a new, empty list that will be the sorted version of this list
- Look at each item in existing list, and insert each item into the new list in a sorted sequence
  - If our new list is empty, add the first item so we have something to compare future items to
- Set the head of the list to be this new, sorted list

Sorting LinkedLists

```
class Element:
 def orderedInsert(self,v):
   if v <= self.value:
     return Element(v,self)
   elif not self.next:
     self.next = Element(v)
     return self
   else:
     self.next =
     self.next.orderedInsert(v)
     return self
```

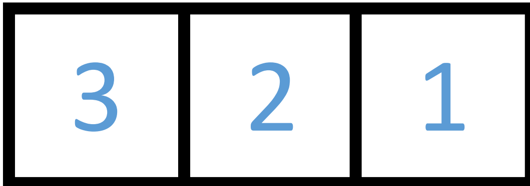
```
class LinkedList:
 def sort(self):
  newList = None
  for item in self:
     if newList is None:
        newList = Element(item)
     else:
        newList =
        newList.orderedInsert(item)
  self._head = newList
```

Testing Sort in Interactive Python

```
>>> from LinkedList import *
>>> ll = LinkedList()
>>> ll.extend([5,1,6,9,9,2,3,1,7,8])
>>> print(ll)
[5, 1, 6, 9, 9, 2, 3, 1, 7, 8]
>>> ll.sort()
>>> print(ll)
[1, 1, 2, 3, 5, 6, 7, 8, 9, 9]
```

Ordered Insert Sort

ll =



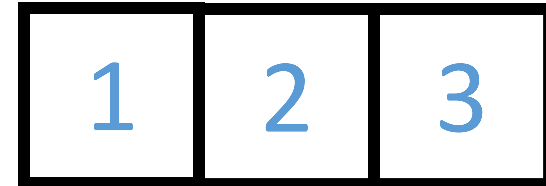
v

```
(LinkedList)
if newList is None:
   newList = Element(item)
```

```
(Element)
if v <= self.value:
    return Element(v,self)
```

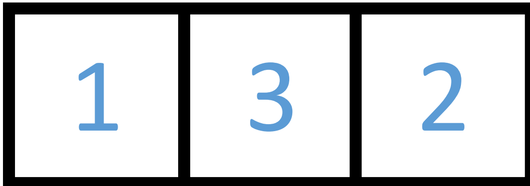
```
if v <= self.value:
   return Element(v,self)
```

newList



Ordered Insert Sort

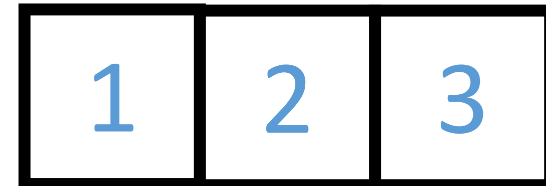
ll =



v

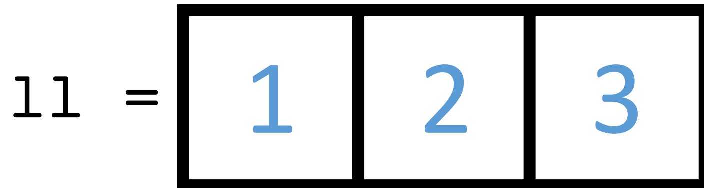
```
(LinkedList)
if newList is None:
  newList = Element(item)

elif not self.next:
    self.next = Element(v)
    return self
else:
    self.next =
    self.next.orderedInsert(v)
    return self
---
if v <= self.value:
    return Element(v,self)
```

newList



Ordered Insert Sort

ll =



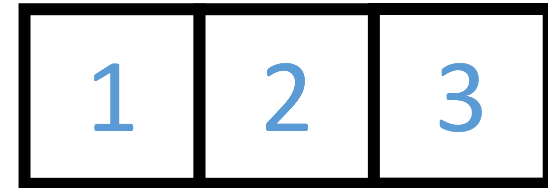
v

```
if newList is None:
  newList = Element(item)
```

```
elif not self.next:
    self.next = Element(v)
    return self
```

```
elif not self.next:
    self.next = Element(v)
    return self
```

newList



Ordered Insert Sort

- How many comparisons are we making to do this sort?

- `ll = LinkedList()`
- `ll.extend([1,2,3])`
- `ll.sort()`

n = len(ll)

For each element of ll, we have n-1 comparisons in the worst case

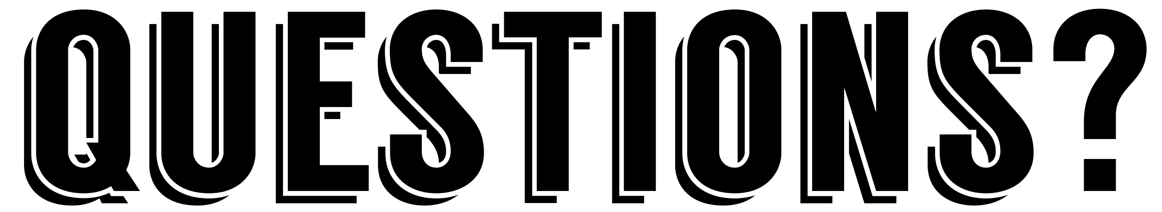
Computer Science drops the -1

Ordered Insert Sort

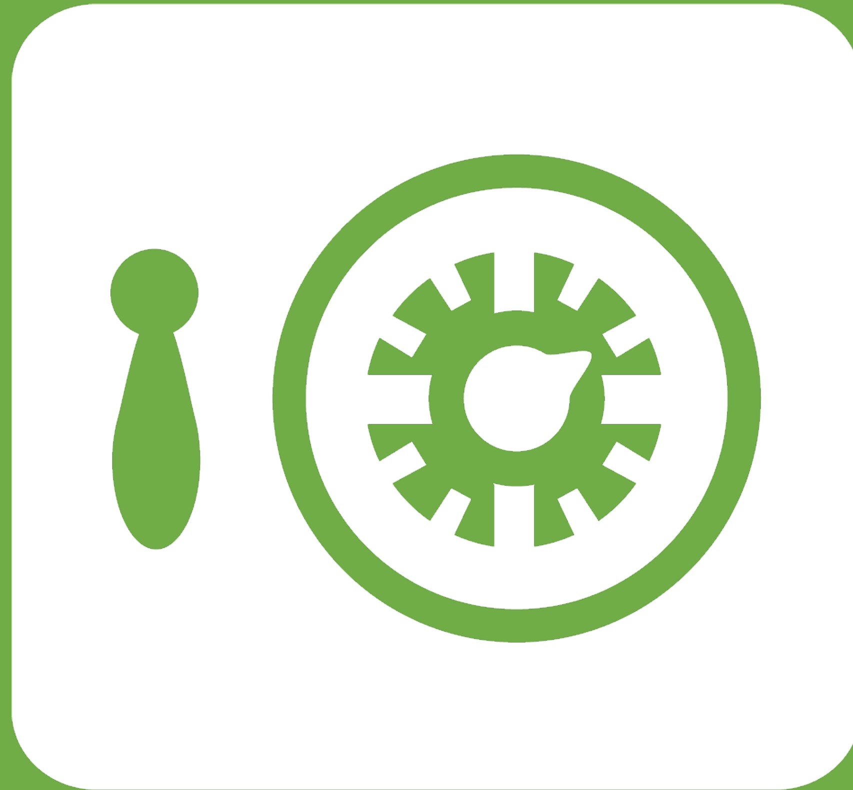
- How many comparisons are we making to do this sort?

- `ll = LinkedList()`
- `ll.extend([1,2,3])`
- `ll.sort()`

For each element of ll → n
We have ~n comparisons in the worst case → *n
$O(n^2)$ comparisons



Please contact me!

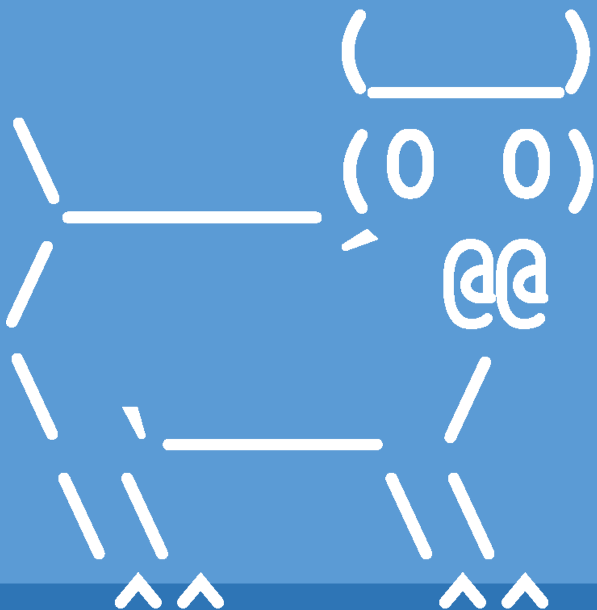


Leftover Slides

Steps for Recursion

- Know when to stop.
- Decide how to take one step.
- Break the journey down into that step plus a smaller journey.

Introducing the Linked List Wrapper



Introduction to Computer Science

Iris Howley

TODAY'S LESSON

`Wrapper classes allow for encapsulation`

(Simpler public interfaces that use hidden private classes)

From Interactive Python

- >>> from LinkedList import *
- >>> ll = LinkedList()
- >>> ll.append(11)
- >>> ll[0]
- 11
- >>> ll[0] = 92
- >>> ll[0]
- 92
- >>> mylist = [0,1,2]
- >>> mylist[2]
- 2
- >>> e = Element(42)
- >>> e.next = Element(99, Element(1000))
- >>> e.value
- 42
- >>> e
- <LinkedList.Element object at 0x101d8e1d0>
- >>> str(e)
- 'Element(42), next=<LinkedList.Element object at 0x101d8e518>'
- >>> e.value
- 42
- >>> e.next.value
- 99
- >>> e.next.next.value
- 1000
- >>> bool(None)
- False
- >>> e
- <LinkedList.Element object at 0x101d8e1d0>
- >>> bool(e)
- True
- >>> from LinkedList import *
- >>> ll = LinkedList()
- >>> ll.append(5)
- >>> str(ll)
- '[5]'
- >>> ll.append(700)
- >>> str(ll)
- '[5,700]'
- >>> ll.append("hello")
- >>> str(ll)
- '[5,700,hello]'
- >>> l = []
- >>> l.extend([5,4,3])
- >>> l
- [5, 4, 3]
- >>> 3 in l
- True
- >>> l.append([66,77,88])
- >>> l
- [5, 4, 3, [66, 77, 88]]
- >>> l.extend([1000,100000,10000000])
- >>> l
- [5, 4, 3, [66, 77, 88], 1000, 100000, 10000000]
- >>> l.extend("hello")
- >>> l
- [5, 4, 3, [66, 77, 88], 1000, 100000, 10000000, 'h', 'e', 'l', 'l', 'o']
- >>> l.extend(100)
- Traceback (most recent call last):
-   File "<stdin>", line 1, in <module>
- TypeError: 'int' object is not iterable
- >>> 3 in l
- True