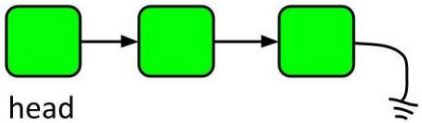
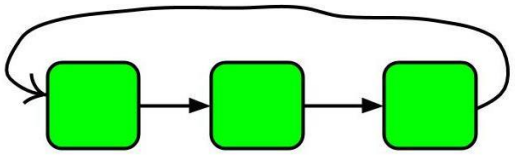
Linked Lists

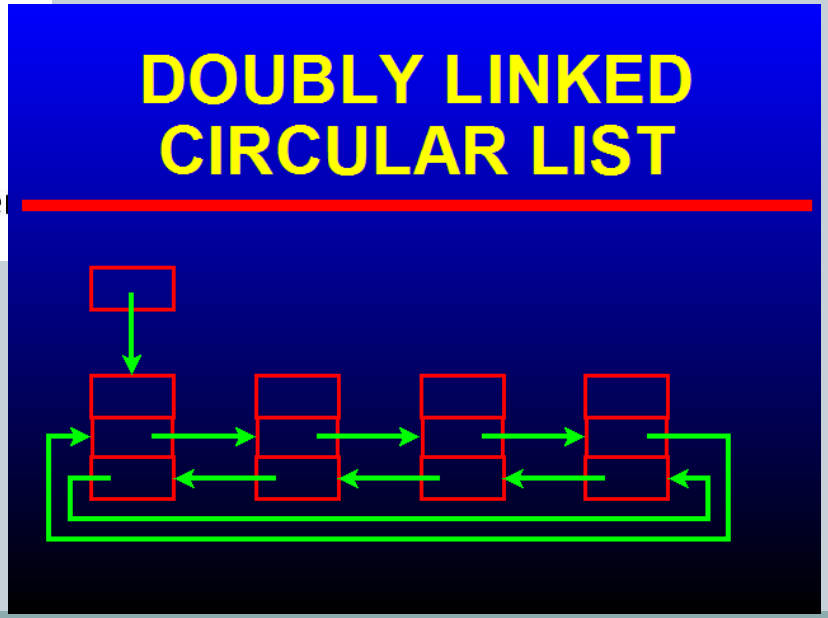
Singly Linked List

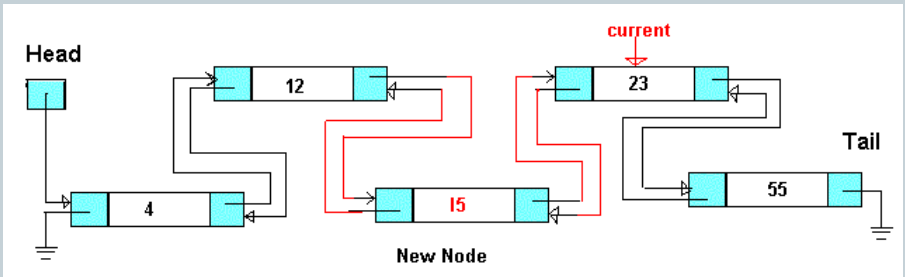


Circular Linked List



(no null pointe





Outline

- Sequential vs. Linked
- Linked List
- Building a Linked List
- Traversing a Linked List
- Implementation (Circular)

Sequential vs. Linked

- Sequential data structures
  - Put one object next to another
  - A block of consecutive memory in the computer
  - Python: list of objects
  - Arbitrary access, "get me the i^{th} object"
  - Fixed size, or dynamic but less efficient
- Linked data structures
  - Each object has link to another (or perhaps several)
  - Python: link is a reference to another object
  - Dynamic size
  - Flexible and widely-used way of organizing data
  - More challenging to code and debug

Sequential vs. Linked

Memory address	Value
C0	"The"
C1	"cat"
C2	"sat"
C3	-
C4	-
C5	-
C6	-
C7	-
C8	-
C9	-

Python list

Memory address	Value
C0	"cat"
C1	C8
C2	-
C3	-
C4	"The"
C5	C0
C6	-
C7	-
C8	"sat"
C9	null

linked list

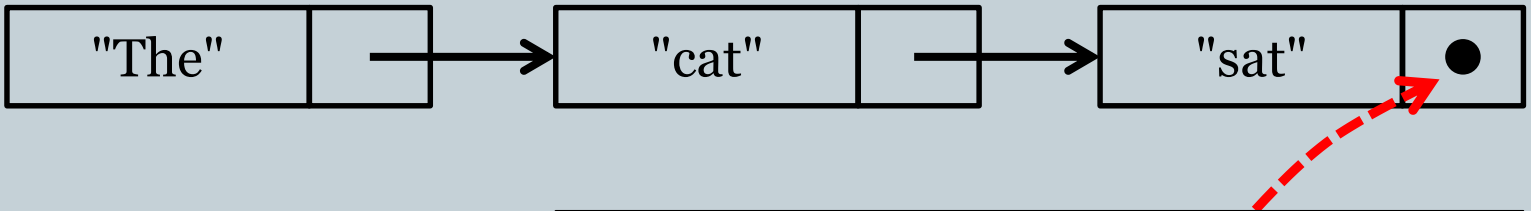
Linked List

- Linked list
  - Simplest linked data structure
  - Node is a recursive data structure
  - Each node contains:
    - An item (some data)
    - A pointer to next node in the list

```
class Node:

def __init__(self, s):
    self.item = s
    self.next = None
```

Three Node objects hooked together to form a linked list



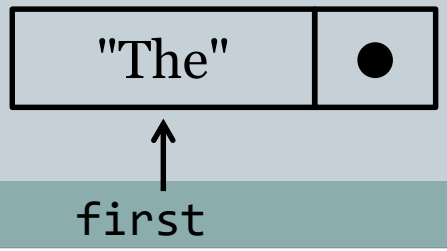
Special pointer value null (None in Python) terminates the list. We denote with a dot.

Building a linked list

```
first  = Node()
first.item  = "The"
```

first →

Memory address	Value
C0	-
C1	-
C2	-
C3	-
C4	"The"
C5	None/null
C6	-
C7	-
C8	-
C9	-

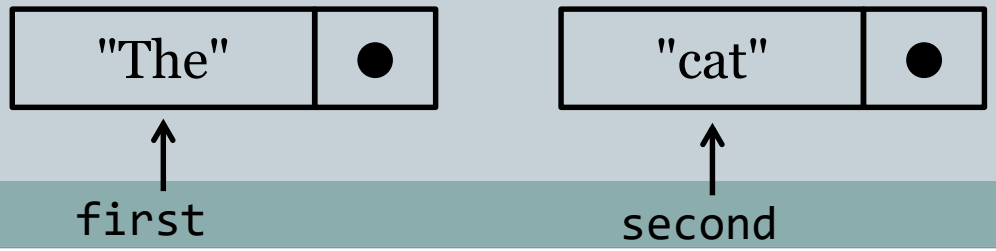


Building a linked list

```
first  = Node()
first.item  = "The"

second = Node()
second.item = "cat"
```

	Memory address	Value
second →	C0	"cat"
	C1	null
	C2	-
	C3	-
first →	C4	"The"
	C5	null
	C6	-
	C7	-
	C8	-
	C9	-



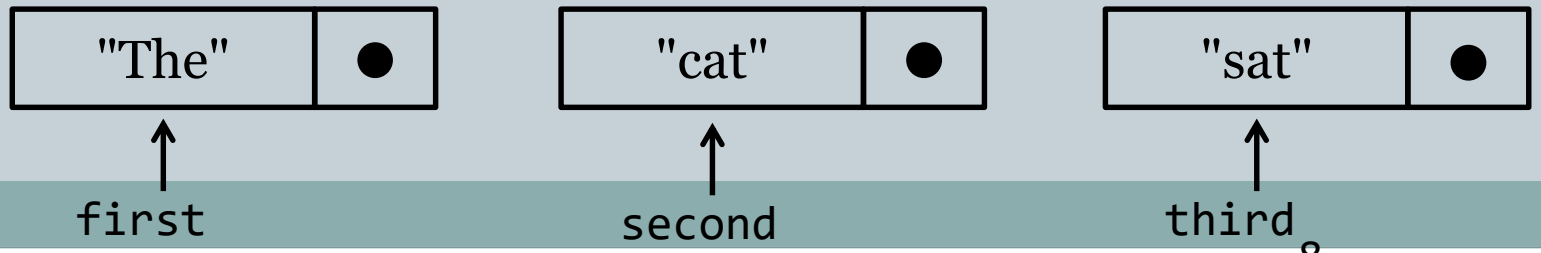
Building a linked list

```
first  = Node()
first.item  = "The"

second = Node()
second.item = "cat"

third  = Node()
third.item  = "sat"
```

	Memory address	Value
second →	C0	"cat"
	C1	null
	C2	-
	C3	-
first →	C4	"The"
	C5	null
	C6	-
	C7	-
third →	C8	"sat"
	C9	null



Building a linked list

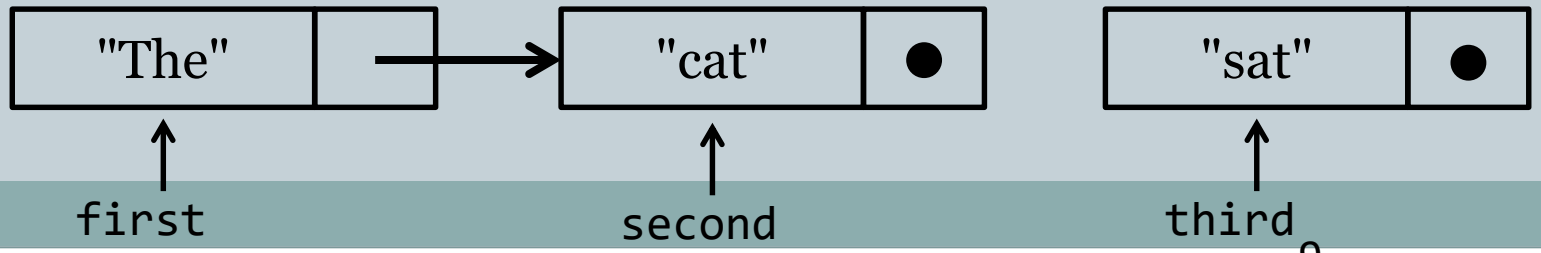
```
first  = Node()
first.item  = "The"

second = Node()
second.item = "cat"

third  = Node()
third.item  = "sat"

first.next  = second
```

	Memory address	Value
second →	C0	"cat"
	C1	null
	C2	-
	C3	-
first →	C4	"The"
	C5	C0
	C6	-
	C7	-
third →	C8	"sat"
	C9	null



Building a linked list

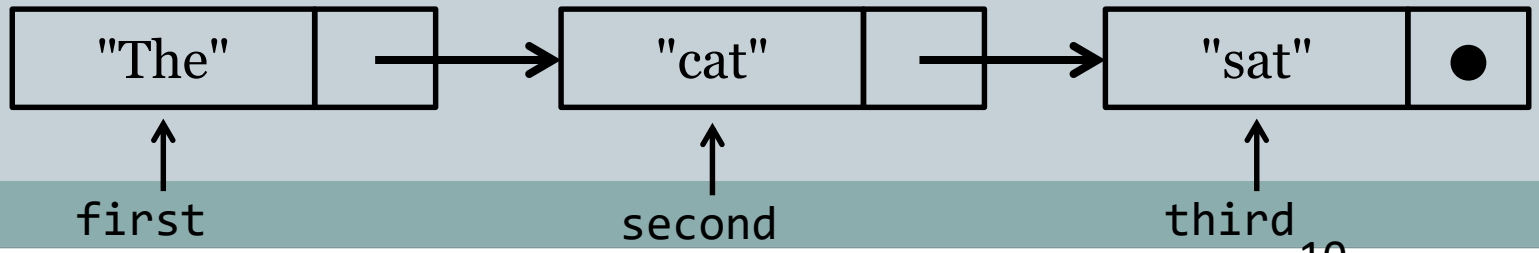
```
first  = Node()
first.item  = "The"

second = Node()
second.item = "cat"

third  = Node()
third.item  = "sat"

first.next  = second
second.next = third
```

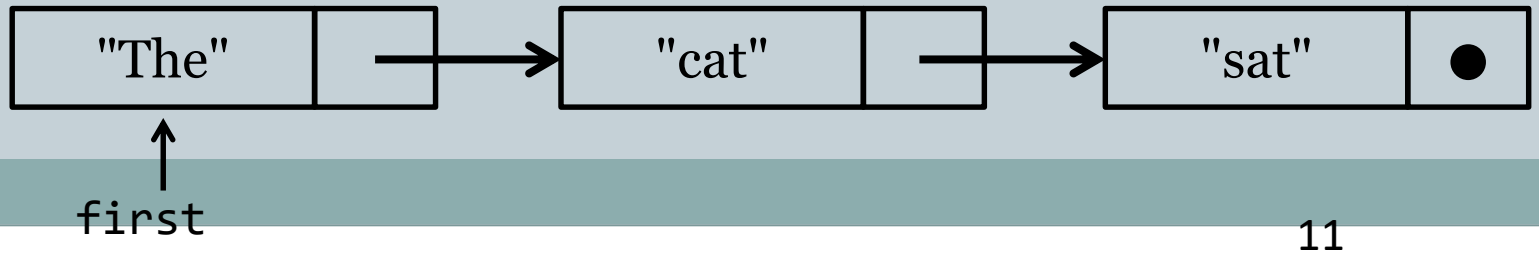
	Memory address	Value
second →	C0	"cat"
	C1	C8
	C2	-
	C3	-
first →	C4	"The"
	C5	C0
	C6	-
	C7	-
third →	C8	"sat"
	C9	null



Traversing a List

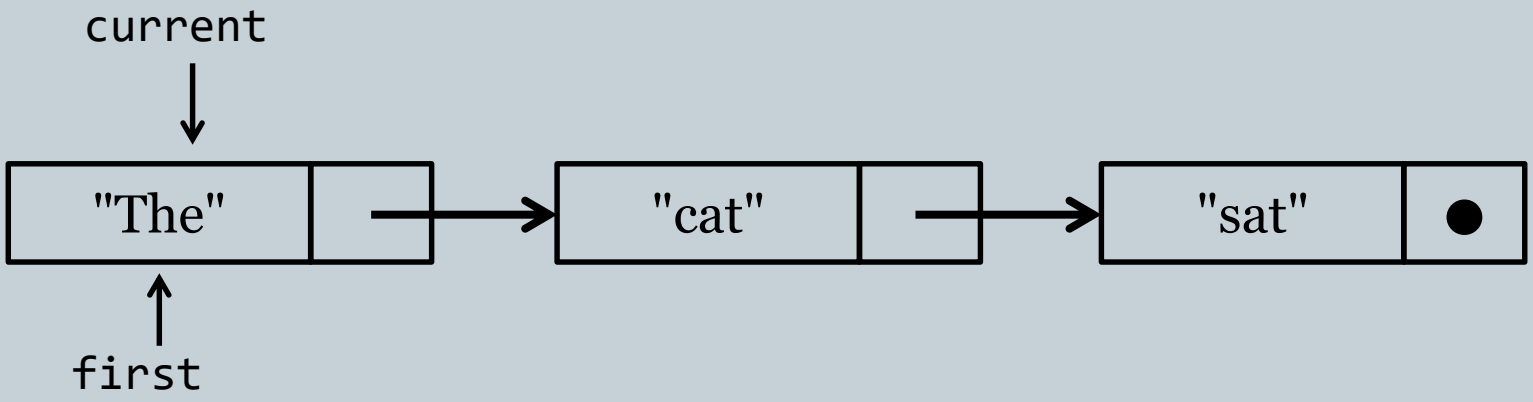
- Iterate over all elements in a linked list
  - Assume list is null terminated
  - Assume `first` instance variable points to start of list
  - Print all the strings in the list

```
current = first
while current != None:
    print(current.item)
    current = current.next
```



Traversing a list

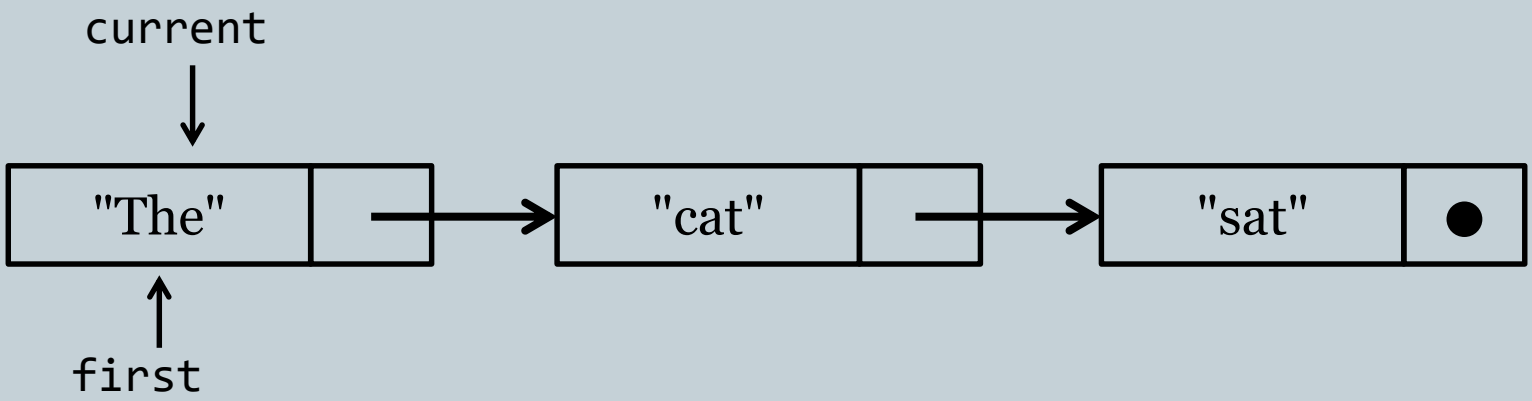
```
current = first
while current != None:
    print(current.item)
    current = current.next
```



Traversing a list

```
current = first
while current != None:
    print(current.item)
    current = current.next
```

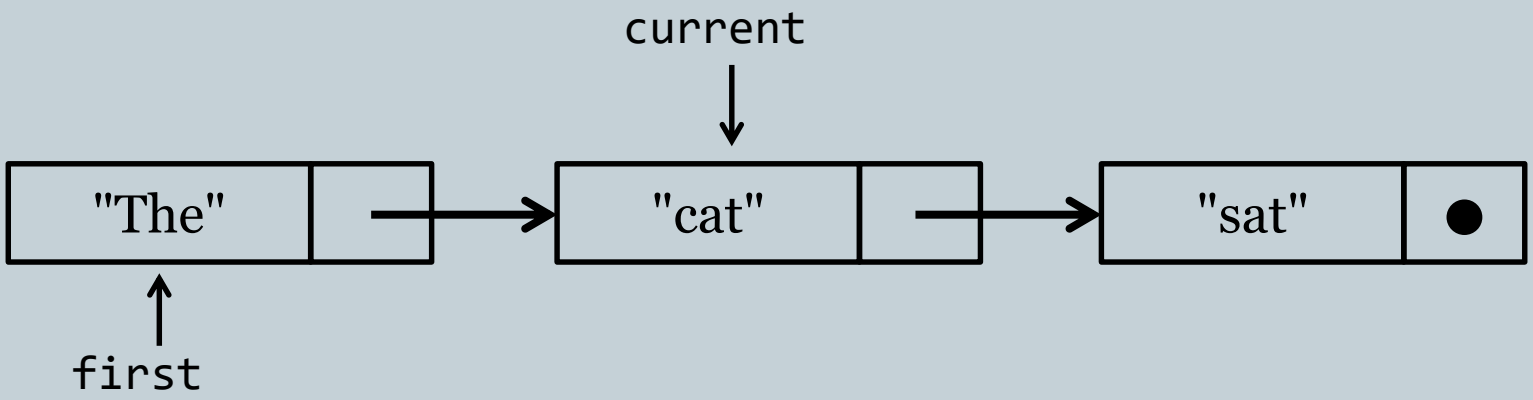
```
The
```



Traversing a list

```
current = first
while current != None:
    print(current.item)
    current = current.next
```

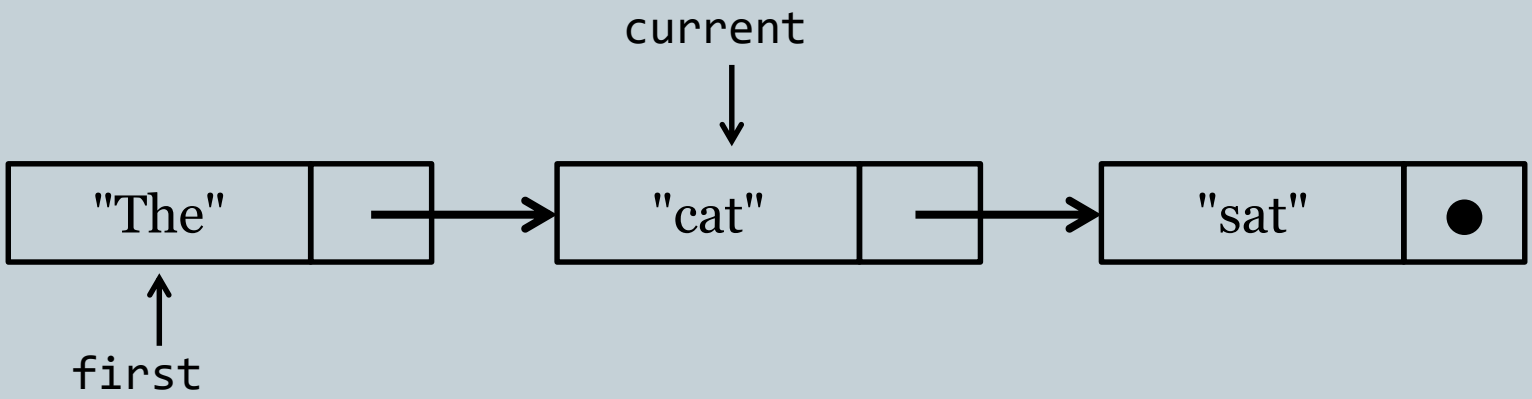
The



Traversing a list

```
current = first
while current != None:
    print(current.item)
    current = current.next
```

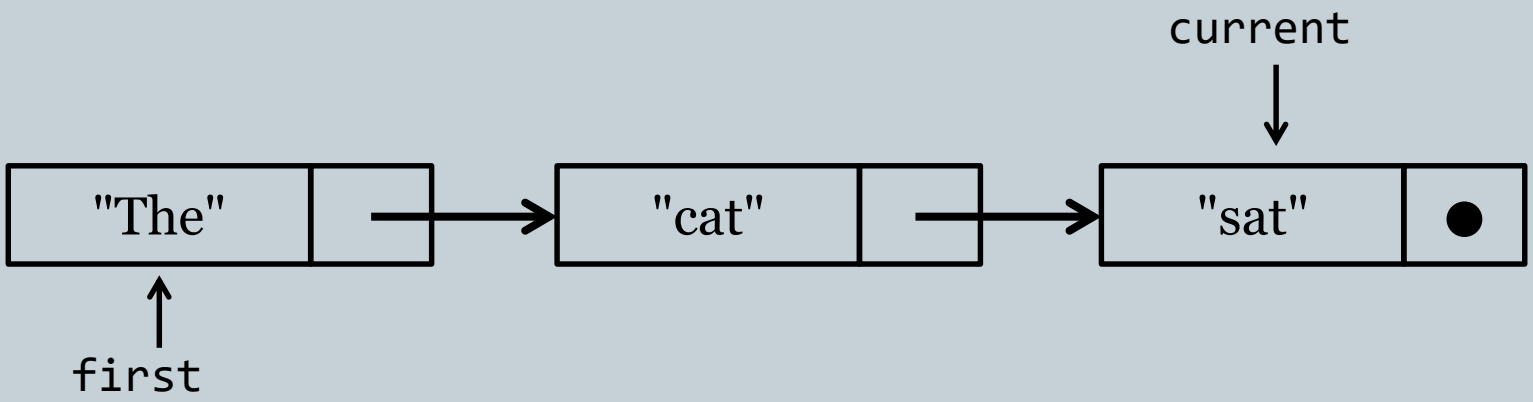
```
The
cat
```



Traversing a list

```
current = first
while current != None:
    print(current.item)
    current = current.next
```

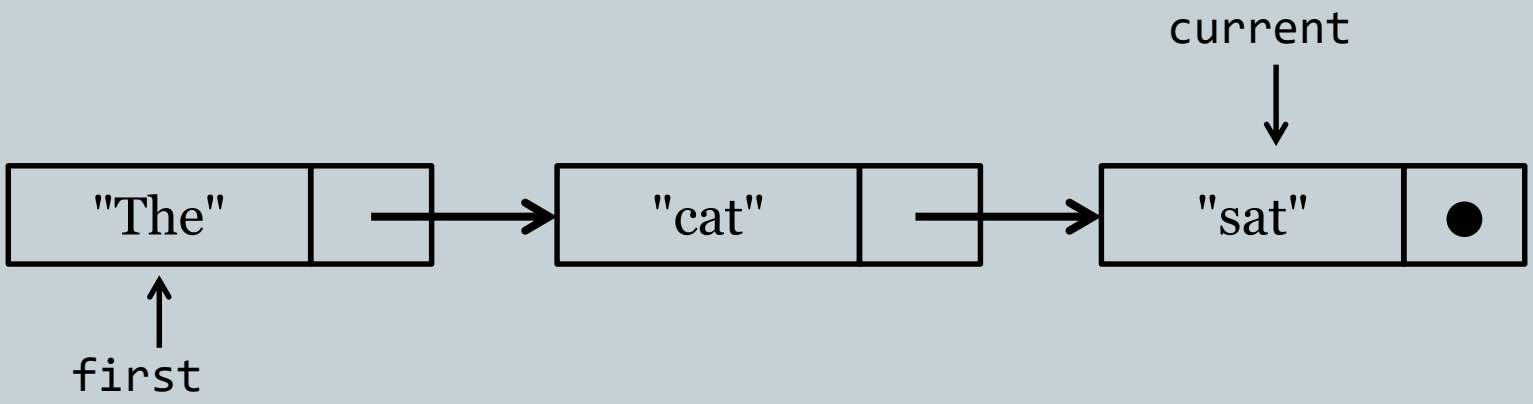
```
The
cat
```



Traversing a list

```
current = first
while current != None:
    print(current.item)
    current = current.next
```

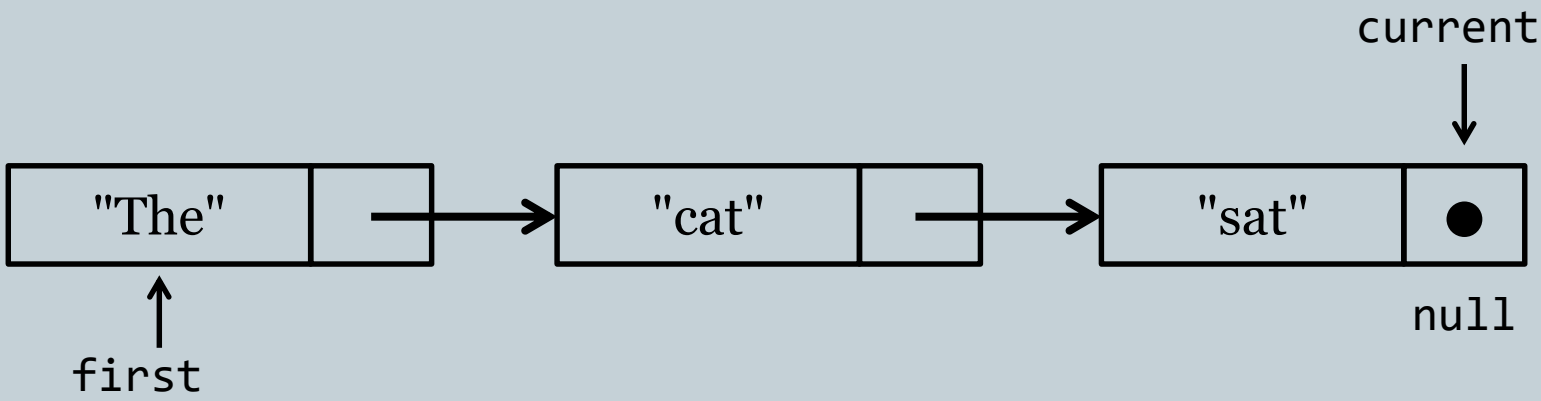
```
The
cat
sat
```



Traversing a list

```
current = first
while current != None:
    print(current.item)
    current = current.next
```

```
The
cat
sat
```



Playing with a Linked List

- What things might we want to do with a list?
  - Construct a node
  - Add a node to the end
  - Insert a node at a certain position
  - Remove a node from a position
  - Print out the list of nodes

Playing with a Linked List

- The definition of the class:

```
class LinkedList:

    #
    # Constructor for an empty linked list.
    #
    def __init__(self):
        self.length = 0
        self.start = None
```

Playing with a Linked List

- What things might we want to do with a list?
  - Construct a linked list

Playing with a Linked List

- What things might we want to do with a list?
  - Add a node to the end

Playing with a Linked List

- What things might we want to do with a list?
  - Insert a node at a certain position

Playing with a Linked List

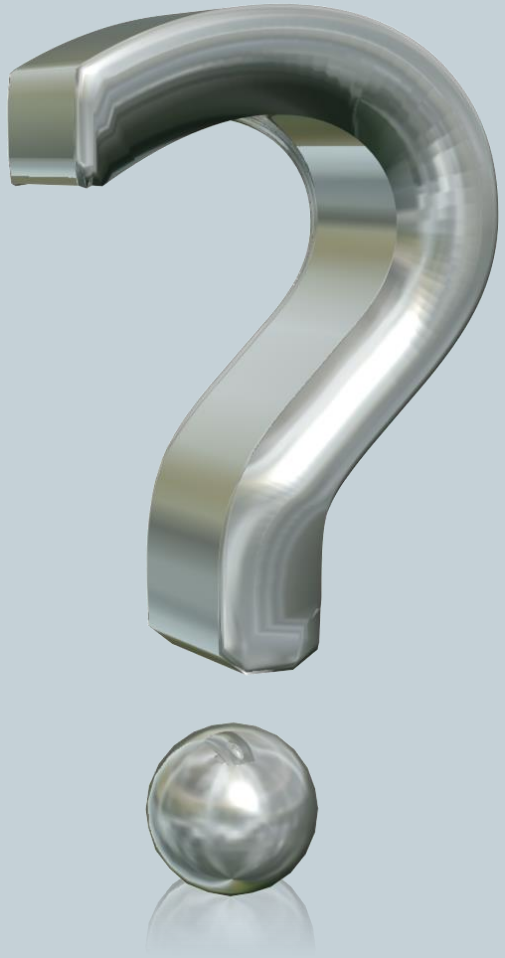
- What things might we want to do with a list?
  - Remove a node from a position

Playing with a Linked List

- What things might we want to do with a list?
  - Print out the list of nodes

Summary

- Sequential vs. Linked
- Linked List
- Building a Linked List
- Traversing a Linked List
- Implementation (Circular)



Your Turn

- Open Moodle, go to CSCI 136, Section 11
- Open the dropbox for today: Activity 1 – Linked Lists
- Drag and drop your program file to the Moodle dropbox
- You get: 1 point if you turn in something, 2 points if you turn in something that is correct.
  - On the class website is a file, Quote.py. There are three blank lines in the method insertWord that you need to fill in with real code to make it work. If you have done it correctly, when you run it, the output should be:

```
A rose is a rose.
5
rose
A rose is just a rose.
6
```