61A Lecture 18
Announcements
Sequences
The Sequence Abstraction
The Sequence Abstraction

red, orange, yellow, green, blue, indigo, violet.
The Sequence Abstraction

red, orange, yellow, green, blue, indigo, violet.

There isn't just one sequence class or data abstraction (in Python or in general).
The Sequence Abstraction

red, orange, yellow, green, blue, indigo, violet.

There isn't just one sequence class or data abstraction (in Python or in general).

The sequence abstraction is a collection of behaviors:
The Sequence Abstraction

red, orange, yellow, green, blue, indigo, violet.

There isn't just one sequence class or data abstraction (in Python or in general).

The sequence abstraction is a collection of behaviors:

- **Length.** A sequence has a finite length.
- **Element selection.** A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0.
The Sequence Abstraction

red, orange, yellow, green, blue, indigo, violet.

0, 1, 2, 3, 4, 5, 6.

There isn't just one sequence class or data abstraction (in Python or in general).

The sequence abstraction is a collection of behaviors:

**Length.** A sequence has a finite length.

**Element selection.** A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0.
The Sequence Abstraction

red, orange, yellow, green, blue, indigo, violet.

0, 1, 2, 3, 4, 5, 6.

There isn’t just one sequence class or data abstraction (in Python or in general).

The sequence abstraction is a collection of behaviors:

**Length.** A sequence has a finite length.

**Element selection.** A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0.

There is built-in syntax associated with this behavior, or we can use functions.
The Sequence Abstraction

red, orange, yellow, green, blue, indigo, violet.

0, 1, 2, 3, 4, 5, 6.

There isn't just one sequence class or data abstraction (in Python or in general).

The sequence abstraction is a collection of behaviors:

**Length.** A sequence has a finite length.

**Element selection.** A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0.

There is built-in syntax associated with this behavior, or we can use functions.

A list is a kind of built-in sequence
Linked Lists
Linked List Structure

A linked list is either empty or a first value and the rest of the linked list
Linked List Structure

A linked list is either empty or a first value and the rest of the linked list

3, 4, 5
Linked List Structure

A linked list is either empty or a first value and the rest of the linked list

3, 4, 5
A linked list is either empty or a first value and the rest of the linked list:

3, 4, 5
A linked list is either empty or a first value and the rest of the linked list.

3, 4, 5
Linked List Structure

A linked list is either empty or a first value and the rest of the linked list. Here's an example:

- **Link instance**
  - first: 3
  - rest: Link instance

- **Link instance**
  - first: 4
  - rest: Link instance

- **Link instance**
  - first: 5
  - rest: Link instance

- **Link.empty**

The list is: 3, 4, 5
A linked list is either empty or a first value and the rest of the linked list.

A linked list is a pair.

\[ 3, 4, 5 \]
A linked list is either empty or a first value and the rest of the linked list

A linked list is a pair

The first (zeroth) element is an attribute value

3, 4, 5
A linked list is either empty or a first value and the rest of the linked list.

Linked List Structure

The first (zeroth) element is an attribute value.

The rest of the elements are stored in a linked list.
A linked list is either empty or a first value and the rest of the linked list.

A linked list is a pair, The first (zeroth) element is an attribute value.

3, 4, 5

A class attribute represents an empty linked list.

The rest of the elements are stored in a linked list.
A linked list is either empty or a first value and the rest of the linked list.

A linked list is a pair:
- The first (zeroth) element is an attribute value.
- The rest of the elements are stored in a linked list.

A class attribute represents an empty linked list.

Link instance:
- first: 3
- rest:

Link instance:
- first: 4
- rest:

Link instance:
- first: 5
- rest:

Link instance:
- Link.empty

Link(3, Link(4, Link(5, Link.empty))))
A linked list is either empty or a first value and the rest of the linked list.

$$\text{Link instance}$$

<table>
<thead>
<tr>
<th>first</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

$$\text{Link instance}$$

<table>
<thead>
<tr>
<th>first</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

$$\text{Link instance}$$

<table>
<thead>
<tr>
<th>first</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

$$\text{Link instance}$$

<table>
<thead>
<tr>
<th>first</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$$\text{Link.empty}$$

$$\text{Link}(3, \text{Link}(4, \text{Link}(5, \text{Link.empty})))$$
A linked list is either empty or a first value and the rest of the linked list.

\[ \text{Link(3, Link(4, Link(5, Link.empty)))} \]
A linked list is either empty or a first value and the rest of the linked list:

```
Link(3, Link(4, Link(5, Link.empty)))
```
A linked list is either empty or a first value and the rest of the linked list.

\[
\text{3, 4, 5}
\]

```
Link instance
first: 3
rest:  Link instance
first: 4
rest:  Link instance
first: 5
rest:  Link instance
first: Link.empty
```

\[
\text{Link(3, Link(4, Link(5, Link.empty)))}
\]
A linked list is either empty or a first value and the rest of the linked list

\[
\text{Link(3, Link(4, Link(5, Link.empty)))}
\]
Linked List Structure

A linked list is either empty or a first value and the rest of the linked list.

\[3, 4, 5\]

```
Link instance
first: 3
rest:  

Link instance
first: 4
rest:  

Link instance
first: 5
rest:  
```

\[
\text{Link}(3, \text{Link}(4, \text{Link}(5)))
\]
Linked List Class

\[ \text{Link}(3, \text{Link}(4, \text{Link}(5))) \]
Linked List Class

Linked list class: attributes are passed to `__init__`

```
Link(3, Link(4, Link(5 ))
```
Linked List Class

Linked list class: attributes are passed to __init__

```python
class Link:

    Link(3, Link(4, Link(5)))
```

Linked List Class

Linked list class: attributes are passed to \texttt{\_\_init\_}

```python
class Link:

def \_\_init\_(self, first, rest=empty):
    Link(3, Link(4, Link(5 )))
```
Linked List Class

Linked list class: attributes are passed to __init__

class Link:

    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)

    Link(3, Link(4, Link(5)))
Linked List Class

Linked list class: attributes are passed to __init__

class Link:

def __init__(self, first, rest=empty):
    assert rest is Link.empty or isinstance(rest, Link)
    self.first = first
    self.rest = rest

Link(3, Link(4, Link(5))))
Linked List Class

Linked list class: attributes are passed to __init__

class Link:

def __init__(self, first, rest=empty):
    assert rest is Link.empty or isinstance(rest, Link)
    self.first = first
    self.rest = rest

Link(3, Link(4, Link(5)))
Linked List Class

Linked list class: attributes are passed to \texttt{\_\_init\_}

class \texttt{Link}:

\begin{verbatim}
def \_\_init\_(self, first, rest=empty):
    assert rest is \texttt{Link.empty} or \texttt{isinstance(rest, Link)}
    self.first = first
    self.rest = rest
\end{verbatim}

\texttt{help(isinstance)}: Return whether an object is an instance of a class or of a subclass thereof.

\texttt{Link(3, Link(4, Link(5))})
Linked List Class

Linked list class: attributes are passed to __init__

class Link:

    empty = ()

    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest

help(isinstance): Return whether an object is an instance of a class or of a subclass thereof.

    Link(3, Link(4, Link(5))))
Linked List Class

Linked list class: attributes are passed to \_\_init\_

```python
class Link:
    empty = ()

    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest

Link(3, Link(4, Link(5)))
```

help(isinstance): Return whether an object is an instance of a class or of a subclass thereof.
Linked List Class

Linked list class: attributes are passed to `__init__`

```python
class Link:
    empty = ()

def __init__(self, first, rest=empty):
    assert rest is Link.empty or isinstance(rest, Link)
    self.first = first
    self.rest = rest
```

(Demo)

```python
Link(3, Link(4, Link(5)))
```

`help(isinstance)`: Return whether an object is an instance of a class or of a subclass thereof.
Sequence Operations
Linked List Class

More special method names:

__getitem__  Element selection []
__len__      Built-in len function
Linked List Class

Linked lists are sequences

More special method names:

`__getitem__`  Element selection []

`__len__`  Built-in len function
Linked List Class

Linked lists are sequences

class Link:
    empty = ()

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

More special method names:
    __getitem__    Element selection []
    __len__       Built-in len function
Linked List Class

Linked lists are sequences

class Link:

    empty = ()

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

def __getitem__(self, i):
    if i == 0:
        return self.first
    else:
        return self.rest[i-1]

More special method names:

    __getitem__    Element selection []
    __len__        Built-in len function
Linked List Class

Linked lists are sequences

```python
class Link:
    empty = ()

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

def __getitem__(self, i):
    if i == 0:
        return self.first
    else:
        return self.rest[i-1]
```

More special method names:

- `__getitem__`  Element selection `[]`
- `__len__`     Built-in len function

This element selection syntax
Linked List Class

Linked lists are sequences

class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        assert ... 
        self.first = first 
        self.rest = rest 
    def __getitem__(self, i):
        if i == 0:
            return self.first 
        else:
            return self.rest[i-1] 

More special method names:
__getitem__  Element selection []
__len__  Built-in len function
Linked List Class

Linked lists are sequences

class Link:
    empty = ()

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

    def __getitem__(self, i):
        if i == 0:
            return self.first
        else:
            return self.rest[i-1]

    def __len__(self):
        return 1 + len(self.rest)

More special method names:
    __getitem__  Element selection []
    __len__     Built-in len function
Linked List Class

Linked lists are sequences

class Link:
    empty = ()

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

    def __getitem__(self, i):
        if i == 0:
            return self.first
        else:
            return self.rest[i-1]

    def __len__(self):
        return 1 + len(self.rest)

More special method names:
__getitem__ Element selection []
__len__    Built-in len function
Linked List Class

Linked lists are sequences

class Link:
    empty = ()

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

def __getitem__(self, i):
    if i == 0:
        return self.first
    else:
        return self.rest[i-1]

def __len__(self):
    return 1 + len(self.rest)

More special method names:

__getitem__ Element selection []
__len__ Built-in len function

Methods can be recursive too!

(Demo)
Property Methods
Often, we want the value of instance attributes to stay in sync.

For example, what if we wanted a Ratio to keep its proportion when its numerator changes?
Property Methods

Often, we want the value of instance attributes to stay in sync.

For example, what if we wanted a Ratio to keep its proportion when its numerator changes?

```python
>>> s = Link(3, Link(4, Link(5)))
```
Property Methods

Often, we want the value of instance attributes to stay in sync.

For example, what if we wanted a Ratio to keep its proportion when its numerator changes?

```python
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
```
Often, we want the value of instance attributes to stay in sync.

For example, what if we wanted a Ratio to keep its proportion when its numerator changes?

```python
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
>>> s.second = 6
```
## Property Methods

Often, we want the value of instance attributes to stay in sync.

For example, what if we wanted a Ratio to keep its proportion when its numerator changes:

```python
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
>>> s.second = 6
>>> s.second
6
```
**Property Methods**

Often, we want the value of instance attributes to stay in sync.

For example, what if we wanted a Ratio to keep its proportion when its numerator changes?

```python
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
>>> s.second = 6
>>> s.second
6
>>> s
Link(3, Link(6, Link(5)))
```
Property Methods

Often, we want the value of instance attributes to stay in sync

For example, what if we wanted a Ratio to keep its proportion when its numerator changes

```python
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
>>> s.second = 6
>>> s.second
6
>>> s
Link(3, Link(6, Link(5)))
```

No method calls!
Property Methods

Often, we want the value of instance attributes to stay in sync.

For example, what if we wanted a Ratio to keep its proportion when its numerator changes.

```python
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
>>> s.second = 6
>>> s.second
6
>>> s
Link(3, Link(6, Link(5)))
```

The `@property` decorator on a method designates that it will be called whenever it is looked up on an instance.
Often, we want the value of instance attributes to stay in sync.

For example, what if we wanted a Ratio to keep its proportion when its numerator changes.

```python
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
>>> s.second = 6
>>> s.second
6
>>> s
Link(3, Link(6, Link(5)))
```

The `@property` decorator on a method designates that it will be called whenever it is looked up on an instance.

A `@<attribute>.setter` decorator on a method designates that it will be called whenever that attribute is assigned. `<attribute>` must be an existing property method.
Property Methods

Often, we want the value of instance attributes to stay in sync.

For example, what if we wanted a Ratio to keep its proportion when its numerator changes.

```python
>>> s = Link(3, Link(4, Link(5)))
>>> s.second
4
>>> s.second = 6
>>> s.second
6
>>> s
Link(3, Link(6, Link(5)))
```

The `@property` decorator on a method designates that it will be called whenever it is looked up on an instance.

A `@<attribute> .setter` decorator on a method designates that it will be called whenever that attribute is assigned. `<attribute>` must be an existing property method.

(Demo)
Linked List Processing

[<map exp> for <name> in <iter exp> if <filter exp>]

(Demo)