Lecture 11

Lists (& Sequences)
Announcements for Today

Reading

- Read 10.0-10.2, 10.4-10.6
- Read all of Chapter 8 for Tue

Assignments

- Assignment 1 now complete
  - But still some grading to do
- Assignment 2 not graded
  - Done by weekend
  - Solutions in CMS soon
- Assignment 3 due next week
  - Before you leave for break
  - Same “length” as A1
  - Get help now if you need it

Prelim, Oct 17th 7:30-9:30
- Material up to October 8th
- Study guide next week

Conflict with Prelim time?
- Submit to Prelim 1 Conflict assignment on CMS
- Must be in by next Tuesday!
Using Color Objects in A3

- New classes in `colormodel`
  - RGB, CMYK, and HSV
- Each has its own attributes
  - **RGB**: red, blue, green
  - **CMYK**: cyan, magenta, yellow, black
  - **HSV**: hue, saturation, value
- Attributes have *invariants*
  - Limits the attribute values
  - Example: red is int in 0..255
  - Get an error if you violate

```python
>>> import colormodel
>>> c = colormodel.RGB(128,0,0)
>>> r = c.red
>>> c.red = 500 # out of range
AssertionError: 500 outside [0,255]
```
Using Color Objects in A3

- New classes in colormodel
  - RGB, CMYK, and HSV
- Each has its own attributes
  - **RGB**: red, blue, green
  - **CMYK**: cyan, magenta, yellow, black
  - **HSV**: hue, saturation, value
- Attributes have *invariants*
  - Limits the attribute values
  - Example: red is int in 0..255
  - Get an error if you violate

```
>>> import colormodel
>>> c = colormodel.RGB(128,0,0)
>>> r = c.red
>>> c.red = 500 # out of range
AssertionError: 500 outside [0,255]
```
def rgb_to_cmyk(rgb):

    """Returns: color rgb in space CMYK
    Precondition: rgb is an RGB object"
    # DO NOT CONSTRUCT AN RGB OBJECT
    # Variable rgb already has RGB object
    # 1. Access attributes from rgb folder
    # 2. Plug into formula provided
    # 3. Compute the new cyan, magenta, etc. values
    # 4. Construct a new CMYK object
    # 5. Return the newly constructed object

    Only time you will ever call a constructor
## Sequences: Lists of Values

### String

- **s = 'abc d'**
  - Put characters in quotes
    - Use `\'` for quote character
  - Access characters with `[]`
    - `s[0]` is 'a'
    - `s[5]` causes an error
    - `s[0:2]` is 'ab' (excludes c)
    - `s[2:]` is 'c d'

### List

- **x = [5, 6, 5, 9, 15, 23]**
  - Put values inside `[ ]`
    - Separate by commas
  - Access **values** with `[]`
    - `x[0]` is 5
    - `x[6]` causes an error
    - `x[0:2]` is [5, 6] (excludes 2\textsuperscript{nd} 5)
    - `x[3:]` is [9, 15, 23]
Sequences: Lists of Values

---

**String**

- \( s = 'abc d' \)
- Put characters in quotes
  - Use \' \ for quote character
- Access characters with \[ \]
  - \( s[0] \) is \'a'\n  - \( s[5] \) causes an error
  - \( s[0:2] \) is \'ab'\n  - \( s[2:] \) is \'c d'\n
**List**

- \( x = [5, 6, 5, 9, 15, 23] \)
- Put values inside \[ \]
  - Separate by commas
  - Access values with \[ \]
  - \( x[0] \) is 5
  - \( x[6] \) causes an error
  - \( x[0:2] \) is \[5, 6\] (excludes 2\(^{nd}\) 5)
  - \( x[3:] \) is \[9, 15, 23\]

---

Sequence is a name we give to both

10/3/13

Lists & Sequences
Lists Have Methods Similar to String

x = [5, 6, 5, 9, 15, 23]

- **index(value)**
  - Return position of the value
  - **ERROR** if value is not there
  - `x.index(9)` evaluates to 3

- **count(value)**
  - Returns number of times value appears in list
  - `x.count(5)` evaluates to 2

But you get length of a list with a regular function, not method: `len(x)`
Lists are Mutable

• Can alter their contents
  ▪ Use an assignment:
    `<var>[<index>] = <value>`
  ▪ Index is position, not slice

• Does not work for strings
  ▪ `s = 'Hello World!'`
  ▪ `s[0] = 'J'`  ERROR

• Represent list as a folder
  ▪ Variable holds tab name
  ▪ Contents are attributes

- `x = [5, 7, 4, -2]`
- `x[1] = 8`
When Do We Need to Draw a Folder?

- When the value **contains** other values
  - This is essentially what we mean by ‘object’
- When the value is **mutable**

<table>
<thead>
<tr>
<th>Type</th>
<th>Container?</th>
<th>Mutable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>float</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>str</td>
<td>Yes*</td>
<td>No</td>
</tr>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RGB</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>list</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Lists vs. Class Objects

**List**

- Attributes are indexed
  - Example: `x[2]`

**RGB**

- Attributes are named
  - Example: `c.red`

```plaintext
id2

x

id2

list

x[0]  5
x[1]  7
x[2]  4
x[3] -2

id3

RGB

red  128
green  64
blue  255
```
List Methods Can Alter the List

\[ x = [5, 6, 5, 9] \]

- **append(value)**
  - A *procedure method*, not a fruitful method
  - Adds a new value to the end of list
  - \( x.append(-1) \) *changes* the list to \([5, 6, 5, 9, -1]\)

- **insert(index, value)**
  - Put the value into list at index; shift rest of list right
  - \( x.insert(2,-1) \) *changes* the list to \([5, 6, -1, 5, 9,]\)

- **sort()**  
  What do you think this does?

See Python API for more
Lists and Functions: Swap

```python
def swap(b, h, k):
    """Procedure swaps b[h] and b[k] in b
    Precondition: b is a mutable list, h and k are valid positions in the list"""
    temp = b[h]
    b[h] = b[k]
    b[k] = temp
```

```
swap(x, 3, 4)
```

Swaps b[h] and b[k], because parameter b contains name of list.
Lists and Functions: Swap

```python
def swap(b, h, k):
    """Procedure swaps b[h] and b[k] in b
    Precondition: b is a mutable list, h and k are valid positions in the list""
    temp = b[h]
    b[h] = b[k]
    b[k] = temp
swap(x, 3, 4)
```

Swaps b[h] and b[k], because parameter b contains name of list.

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>id4</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>swap</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td></td>
<td>h</td>
<td>id4</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>temp</td>
<td>k</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>
```
Lists and Functions: Swap

```python
def swap(b, h, k):
    """Procedure swaps b[h] and b[k] in b
    Precondition: b is a mutable list, h and k are valid positions in the list"""
    temp = b[h]
    b[h] = b[k]
    b[k] = temp
```

Swaps b[h] and b[k], because parameter b contains name of list.

1. temp = b[h]
2. b[h] = b[k]
3. b[k] = temp
Lists and Functions: Swap

```python
def swap(b, h, k):
    """Procedure swaps b[h] and b[k] in b
    Precondition: b is a mutable list, h and k are valid positions in the list"""
    temp = b[h]
    b[h] = b[k]
    b[k] = temp
```

Swaps b[h] and b[k], because parameter b contains name of list.

Example:
```
swap(x, 3, 4)
```

- `x` is not defined in the example.
List Slices Make Copies

\[ x = [5, 6, 5, 9] \]

\[ y = x[1:3] \]

\[ x \]

\[ y \]

\[ x[0] \]
\[ x[1] \]
\[ x[2] \]
\[ x[3] \]

\[ y[0] \]
\[ y[1] \]

\[ \text{copy = new folder} \]
Exercise Time

• Execute the following:
  >>> x = [5, 6, 5, 9, 10]
  >>> x[3] = -1
  >>> x.insert(1,2)

• What is x[4]?

  A: 10
  B: 9
  C: -1
  D: ERROR
  E: I don’t know
Exercise Time

- Execute the following:
  ```python
  >>> x = [5, 6, 5, 9, 10]
  >>> x[3] = -1
  >>> x.insert(1,2)
  ```
- What is `x[4]`?

- Execute the following:
  ```python
  >>> x = [5, 6, 5, 9, 10]
  >>> y = x[1:]
  >>> y[0] = 7
  ```
- What is `x[1]`?

A: 7
B: 5
C: 6
D: ERROR
E: I don’t know
Exercise Time

- Execute the following:
  ```
  >>> x = [5, 6, 5, 9, 10]
  >>> x[3] = -1
  >>> x.insert(1,2)
  ```
- What is x[4]?

- Execute the following:
  ```
  >>> x = [5, 6, 5, 9, 10]
  >>> y = x[1:]
  >>> y[0] = 7
  ```
- What is x[1]?
• List brackets [] can contain expressions

• This is a list **expression**
  - Python must evaluate it
  - Evaluates each expression
  - Puts the value in the list

• Example:
  ```
  >>> a = [1+2, 3+4, 5+6]
  >>> a
  [3, 7, 11]
  ```

• Execute the following:
  ```
  >>> a = 5
  >>> b = 7
  >>> x = [a, b, a+b]
  ```

• What is x[2]?

  A: 'a+b'
  B: 12
  C: 57
  D: ERROR
  E: I don’t know
Lists and Expressions

- List brackets [] can contain expressions
- This is a list **expression**
  - Python must evaluate it
  - Evaluates each expression
  - Puts the value in the list
- Example:
  ```
  >>> a = [1+2, 3+4, 5+6]
  >>> a
  [3, 7, 11]
  ```
- Execute the following:
  ```
  >>> a = 5
  >>> b = 7
  >>> x = [a, b, a+b]
  ```
- What is x[2]?
  $12$
Lists of Objects

- List positions are variables
  - Can store base types
  - But cannot store folders
  - Can store folder identifiers
- Folders linking to folders
  - Top folder for the list
  - Other folders for contents
- Example:
  ```python
  >>> r = colormodel.RED
  >>> b = colormodel.BLUE
  >>> g = colormodel.GREEN
  >>> x = [r, b, g]
  ```
Lists of Objects

- List positions are variables
  - Can store base types
  - But cannot store folders
  - Can store folder identifiers
- Folders linking to folders
  - Top folder for the list
  - Other folders for contents
- Example:
  ```python
  >>> r = colormodel.RED
  >>> b = colormodel.BLUE
  >>> g = colormodel.GREEN
  >>> x = [r, b, g]
  ```