Lecture 12

Lists (& Sequences)
# Announcements for Today

## Reading

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

### Prelim, Oct 12th 7:30-9:30
- Material up to October 3rd
- Study guide next week

### Conflict with Prelim time?
- Submit to Prelim 1 Conflict assignment on CMS
- Must be in by next Tuesday!

## Assignments

- A2 is almost finished
  - **Tomorrow** in Gates 216
  - Graded out of 50 points
  - **Mean**: 44.5, **Median**: 46
  - **A**: 46 (56%), **B**: 37 (35%)

- A3 due next week
  - Due on Thurs, Oct. 5
  - Will grade over break
Sequences: Lists of Values

### String
- \( s = 'abc d' \)
- Put characters in quotes
  - Use \( \backslash \) 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'

<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>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 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]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>15</td>
<td>23</td>
</tr>
</tbody>
</table>
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 [ ]
  - Use commas to separate

- 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]
Lists Have Methods Similar to String

- `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

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

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

Wrong

\[ x = \{5, 6, 7, -2\} \]

Box is “too small” to hold the list

Correct

\[ x = \text{id1} \]

Variable holds id

Put list in a “folder”

\[ \text{id1} \]

Unique tab identifier

\[
\begin{align*}
0 & : 5 \\
1 & : 7 \\
2 & : 4 \\
3 & : -2 \\
\end{align*}
\]

\[ x = [5, 7, 4, -2] \]
Lists vs. Class Objects

List

- Attributes are indexed
  - Example: \( x[2] \)

<table>
<thead>
<tr>
<th>id2</th>
<th>list</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>-2</td>
</tr>
</tbody>
</table>

RGB

- Attributes are named
  - Example: \( c.red \)

<table>
<thead>
<tr>
<th>id3</th>
<th>RGB</th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>128</td>
</tr>
<tr>
<td>green</td>
<td>64</td>
</tr>
<tr>
<td>blue</td>
<td>255</td>
</tr>
</tbody>
</table>
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>Point3</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 are Mutable

- **List assignment:**
  
  `<var>[<index>] = <value>`
  
  - Reassign at index
  - Affects folder contents
  - Variable is unchanged

- Strings cannot do this
  
  - `s = 'Hello World!'`
  - `s[0] = 'J'` **ERROR**
  - String are **immutable**

- **x = [5, 7, 4, -2]**

- **x[1] = 8**
Lists are Mutable

- **List assignment:**
  
  `<var>[<index>] = <value>`

  - Reassign at index
  - Affects folder contents
  - Variable is unchanged

- **Strings cannot do this**
  
  - `s = 'Hello World!'`
  - `s[0] = 'J'` **ERROR**
  - String are **immutable**

- **x = [5, 7, 4, -2]**
  
  - `x[1] = 8`
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?
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)
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.

swap(x, 3, 4)
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)
Lists and Functions: Swap

```
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)
```
List Slices Make Copies

\[ x = [5, 6, 5, 9] \]
\[ y = x[1:3] \]

\[
\begin{array}{c|c}
0 & 5 \\
1 & 6 \\
2 & 5 \\
3 & 9 \\
\end{array}
\]

\[
\begin{array}{c|c}
0 & 6 \\
1 & 5 \\
\end{array}
\]

copy = new folder
Exercise Time

- Execute the following:
  ```python
  >>> 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]?

-1

• Execute the following:
  >>> x = [5, 6, 5, 9, 10]
  >>> y = x[1:]
  >>> y[0] = 7

• What is x[1]?

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

- Execute the following:
  ```python
  >>> 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 = cornell.RED
>>> b = cornell.BLUE
>>> g = cornell.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:
  >>> r = cornell.RED
  >>> b = cornell.BLUE
  >>> g = cornell.GREEN
  >>> x = [r, b, g]