Lecture 5

Lists (& Sequences)
# Announcements For This Lecture

## Readings
- Sec 10.0-10.2, 10.4-10.6
- Chapter 4 for Friday

## Lab
- Shorter lab since A1 due
  - Similar to Lab 1
  - Evaluate and Guess

## Assignment 1
- Due this Fri, Sep. 26
  - First attempt at solution
  - Will resubmit until good
  - **Goal**: 24 hour turnaround
- Will post A2 on Saturday
  - Do while revising A1
  - Due date may be extended
### Sequences: Lists of Values

#### String

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td></td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

- **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

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>15</td>
<td>23</td>
<td></td>
</tr>
</tbody>
</table>

- **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 \( \backslash ' \) 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 \('c d'\)

### 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...
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]

  x
  
  id2
  
  id2
  list
  
  x[0]  5
  x[1]  7
  x[2]  4
  x[3] -2

RGB

- Attributes are named
  - Example: c.red

  c
  
  id3
  
  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?
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)
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.
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

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

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

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 |

9/22/14
Lists & Sequences
15
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]?

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]?
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]?

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:
  ```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]?
  ```

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
  >>> p = Point(1,0,0)
  >>> q = Point(0,1,0)
  >>> r = Point(0,0,1)
  >>> s = [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
  >>> p = Point(1,0,0)
  >>> q = Point(0,1,0)
  >>> r = Point(0,0,1)
  >>> s = [r,b,g]
  ```
Nested Lists

- Lists can hold any objects
- Lists are objects
- Therefore lists can hold other lists!

\[ \begin{align*}
\mathbf{a} &= [2, 1] \\
\mathbf{b} &= [3, 1] \\
\mathbf{c} &= [1, 4, \mathbf{b}] \\
\mathbf{x} &= [1, \mathbf{a}, \mathbf{c}, 5]
\end{align*} \]

\[ \mathbf{x} = [1, [2, 1], [1, 4, [3, 1]], 5] \]
Two Dimensional Lists

Table of Data

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>8</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

Images

Each row, col has a value

Each row, col has an RGB value

Store them as lists of lists (row-major order)

d = 
[[5, 4, 7, 3], [4, 8, 9, 7], [5, 1, 2, 3], [4, 1, 2, 9], [6, 7, 8, 0]]
Overview of Two-Dimensional Lists

- Access value at row 3, col 2:
  \[d[3][2]\]
- Assign value at row 3, col 2:
  \[d[3][2] = 8\]
- An odd symmetry
  - Number of rows of \(d\): \(\text{len}(d)\)
  - Number of cols in row \(r\) of \(d\): \(\text{len}(d[r])\)
How Multidimensional Lists are Stored

- \( b = [[9, 6, 4], [5, 7, 7]] \)

- \( b \) holds name of a one-dimensional list
  - Has `len(b)` elements
  - Its elements are (the names of) 1D lists

- \( b[i] \) holds the name of a one-dimensional list (of ints)
  - Has `len(b[i])` elements
Image Data: 2D Lists of Pixels

b[0][0] is a white pixel

0 1 2 3 4 5 6 7 8 9 10 11 12

b
id1

id1
list

id2
id3
...

id2
list

id23
id24
...

RGB
red 255
green 255
blue 255
Ragged Lists: Rows w/ Different Length

- \( b = [[17,13,19],[28,95]] \)

- Will see applications of this later
Slices and Multidimensional Lists

- Only “top-level” list is copied.
- Contents of the list are not altered
- \( b = \[[9, 6], [4, 5], [7, 7]\] \)

\[ x = b[:2] \]
Slices and Multidimensional Lists

- Create a nested list
  ```python
  >>> b = [[9,6],[4,5],[7,7]]
  ```

- Get a slice
  ```python
  >>> x = b[:2]
  ```

- Append to a row of x
  ```python
  >>> x[1].append(10)
  ```

- x now has nested list
  ```python
  [[9, 6], [4, 5, 10]]
  ```

- What are the contents of the list (with name) in \( b \)?

  A: [[9,6],[4,5],[7,7]]
  B: [[9,6],[4,5,10]]
  C: [[9,6],[4,5,10],[7,7]]
  D: [[9,6],[4,10],[7,7]]
  E: I don’t know
Python is Slow with Large Matrices

- Python is *interpreted*, not *compiled*
  - Slower than other languages
  - Very evident when using lists
- Big problem for scientific work
- **Solution**: Use another language?
- Python can be a *bridge language*
  - Write complex function in C/C++
  - Write a one-line Python function
  - Python function calls C/C++ one

This is A LOT of folders
Python is Slow with Large Matrices

- Python is *interpreted*, not *compiled*.
  - Slower than other languages.
  - Very evident when using lists.
- Big problem for scientific work.
- Solution: Use another language?
- Python can be a *bridge language*.
  - Write complex function in C/C++
  - Write a one-line Python function
  - Python function calls C/C++ one

Many modules have done the C/C++ part for you already.

This is a lot of folders.
Module For Large Lists: numpy

- numpy has C++ code for
  - Storing large lists
  - Turning lists in matrices
  - Transposing matrices
  - Multiplying matrices

- But must use its list type
  - Called arrays in numpy
  - Can copy from Python list
  - Very efficient with folder creation in calculations

Python Memory

```
y = numpy.array(x)
```

x

<table>
<thead>
<tr>
<th>id1</th>
<th>x[0]</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x[1]</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>x[2]</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>x[3]</td>
<td>-2</td>
</tr>
</tbody>
</table>

numpy Memory

```
y = numpy.array(x)
```

y

```
y = numpy.array(x)
```

id2
Module For Large Lists: **numpy**

- numpy has C++ code for
  - Storing large lists
  - Turning lists in matrices
  - Transposing matrices
  - Multiplying matrices
- But must use its list type
  - Called arrays in numpy
  - Can copy from Python list
  - Very efficient with folder creation in calculations

```python
>>> import numpy
>>> plist = [2,1,0,0,3,-2,1,1,1]
>>> # Convert to numpy array
>>> nlist = numpy.array(plist)
>>> # Reshape into a matrix
>>> matrix = nlist.reshape([3,3])
>>> # Get the top-left submatrix
>>> matrix[:2,:2]
array([[2, 1],
       [0, 3]])
```

`npdemo.py` has more examples