Lecture 10:
Lists and Sequences
(Sections 10.0-10.2, 10.4-10.6, 10.8-10.13)
CS 1110
Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]
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]

Sequence is a name we give to both
Lists Have Methods Similar to String

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

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

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

But to get the length of a list you use a function, not a class method:

\[ \text{len}(x) \]
\[ x\text{.len()} \]
# Things that Work for All Sequences

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

<table>
<thead>
<tr>
<th>Methods</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s.\text{index}('s') \rightarrow 0)</td>
<td>(x.\text{index}(5) \rightarrow 0)</td>
</tr>
<tr>
<td>(s.\text{count}('t') \rightarrow 1)</td>
<td>(x.\text{count}(6) \rightarrow 2)</td>
</tr>
<tr>
<td>(\text{len}(s) \rightarrow 6)</td>
<td>(\text{len}(x) \rightarrow 6)</td>
</tr>
<tr>
<td>(s[4] \rightarrow \text{“h”})</td>
<td>(x[4] \rightarrow 15)</td>
</tr>
<tr>
<td>(s[1:3] \rightarrow \text{“li”})</td>
<td>(x[1:3] \rightarrow [6, 9])</td>
</tr>
<tr>
<td>(s[3:] \rightarrow \text{“thy”})</td>
<td>(x[3:] \rightarrow [6, 15, 5])</td>
</tr>
<tr>
<td>(s[-2] \rightarrow \text{“h”})</td>
<td>(x[-2] \rightarrow 15)</td>
</tr>
<tr>
<td>(s + \text{‘toves’} \rightarrow \text{“slithy toves”})</td>
<td>(x + [1, 2] \rightarrow [5, 6, 9, 6, 15, 5, 1, 2])</td>
</tr>
<tr>
<td>(s * 2 \rightarrow \text{“slithyslithy”})</td>
<td>(x * 2 \rightarrow [5, 6, 9, 6, 15, 5, 5, 6, 9, 6, 15, 5])</td>
</tr>
<tr>
<td>(\text{‘t’ in s} \rightarrow \text{True})</td>
<td>(15 \text{ in } x \rightarrow \text{True})</td>
</tr>
</tbody>
</table>
Representing Lists

Wrong:

Global Space

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

Correct:

Global Space

\[ x = \text{id1} \]

Heap Space

Indices

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>-2</td>
</tr>
</tbody>
</table>
Lists vs. Class Objects

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

Objects
- Attributes are named
  - Example: p.x

Global Space
- x

Heap Space
- list
  - 0: 5
  - 1: 7
  - 2: 4
  - 3: -2

Global Space
- p
- id3

Heap Space
- id3
  - Point3
  - x: 1
  - y: 2
  - z: 3
Lists Can Hold Any Type

**Global Space**

<table>
<thead>
<tr>
<th>id1</th>
<th>list_of_integers = [5, 7, 4, -2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>id2</td>
<td>list_of_strings = ['h', 'i', '', 'there!']</td>
</tr>
</tbody>
</table>

**Heap Space**

<table>
<thead>
<tr>
<th>id1</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>

<table>
<thead>
<tr>
<th>id2</th>
<th>list</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>'h'</td>
</tr>
<tr>
<td>1</td>
<td>'i'</td>
</tr>
<tr>
<td>2</td>
<td>''</td>
</tr>
<tr>
<td>3</td>
<td>'there!'</td>
</tr>
</tbody>
</table>
No Really, Lists Can Hold Any Type!

list_of_points = [Point3(81,2,3), Point3(6,2,3)…]

Global Space

list_of_points  id1

list_of_various_types  id9

Heap Space

id1

0  id2
1  id3
2  id6
3  id7

id9

0  5
1  3.1416
2  'happy'
3  id5

id2

Point3

x 81  y 2  z 3

id3

Point3

x 6  y 2  z 3

id5

Point3

x 10  y 20  z 13

id6

Point3

x 4  y 4  z 3

id7

Point3

x 1  y 2  z 2
Lists of Objects

- List elements are variables
  - Can store base types and ids
  - Cannot store folders

Global Space

<table>
<thead>
<tr>
<th>ID</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>p1</td>
</tr>
<tr>
<td>id2</td>
<td>p2</td>
</tr>
<tr>
<td>id3</td>
<td>p3</td>
</tr>
</tbody>
</table>

x = [p1, p2, p3]

Heap Space

<table>
<thead>
<tr>
<th>ID</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>Point3</td>
</tr>
<tr>
<td></td>
<td>x: 1</td>
</tr>
<tr>
<td></td>
<td>y: 2</td>
</tr>
<tr>
<td></td>
<td>z: 3</td>
</tr>
<tr>
<td>id2</td>
<td>Point3</td>
</tr>
<tr>
<td></td>
<td>x: 4</td>
</tr>
<tr>
<td></td>
<td>y: 5</td>
</tr>
<tr>
<td></td>
<td>z: 6</td>
</tr>
<tr>
<td>id3</td>
<td>Point3</td>
</tr>
<tr>
<td></td>
<td>x: 7</td>
</tr>
<tr>
<td></td>
<td>y: 8</td>
</tr>
<tr>
<td></td>
<td>z: 9</td>
</tr>
</tbody>
</table>

How do I get this y?

x[1].y

p1 = Point3(1, 2, 3)
p2 = Point3(4, 5, 6)
p3 = Point3(7, 8, 9)
List Assignment

• **Format:**
  
  \[
  \text{<var>}[\text{<index>}] = \text{<value>}
  \]

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

• Strings cannot do this
  
  - Strings are **immutable**

```python
x = [5, 7, 4, -2]
x[1] = 8
s = “Hello!”
s[0] = ‘J’
```

```
TypeError: 'str' object does not support item assignment
```
List Methods Can Alter the List

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

### Example

- **<list>.append(<value>)**
  - Adds a new value to the end of list
  - \(x.append(-1)\) changes the list to [5, 6, 5, 9, -1]

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

- **<list>.sort()**
  - What do you think this does?
1st Clicker Question

• 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
1st Clicker Answer

- 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  CORRECT
  D: ERROR
  E: I don’t know

(Original elements 1-4 are shifted down to be elements 2-5)
Recall: identifier assignment → no swap

```python
import shapes
def swap(p, q):
    tmp = p
    p = q
    q = tmp

p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
swap(p, q)
```

At the end of `swap`: parameters `p` and `q` are swapped
global `p` and `q` are unchanged
Recall: Attribute Assignment $\rightarrow$ swap!

import shapes

def swap(p, q):
    tmp = p.x
    p.x = q.x
    q.x = tmp

p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)

swap(p, q)

At the end of swap: parameters $p$ and $q$ are unchanged, global $p$ and $q$ are unchanged, attributes $x$ are swapped.
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

x = [5, 4, 7, 6, 5]
swap(x, 3, 4)
print x[3]

What gets printed?
A: 5
B: 6
C: Something else
D: I don’t know
2\textsuperscript{nd} Clicker Answer

```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
```

\[ x = [5,4,7,6,5] \]

swap(x, 3, 4)

\[ \text{print } x[3] \]

What gets printed?

A: 5 \text{ CORRECT}  
B: 6  
C: Something else  
D: I don’t know
2nd Clicker Explanation (1)

```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

x = [5, 4, 7, 6, 5]
swap(x, 3, 4)
print x[3]
```

Global Space

```
x id4
```

Call Frame

```
swap

b id4 h 1

k 4
```

Heap Space

```
0 5
1 4
2 7
3 6
4 5
```
```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

x = [5, 4, 7, 6, 5]
swap(x, 3, 4)
print(x[3])
```

```
1
2
3
```
```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

x = [5,4,7,6,5]
swap(x, 3, 4)
print x[3]
```
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

x = [5,4,7,6,5]
swap(x, 3, 4)
print x[3]
List Slices Make Copies

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

**copy means new folder**
3\textsuperscript{rd} Clicker Question

- 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: \textbf{ERROR}  
E: I don’t know
3rd Clicker Answer

- 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  CORRECT
D: ERROR
E: I don’t know
4th Clicker Question

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

A: 7  
B: 5  
C: 6  
D: ERROR  
E: I don’t know
4th Clicker Answer

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

A: 7 \( \text{CORRECT} \)
B: 5
C: 6
D: ERROR
E: I don’t know
• 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
• Execute the following:
  >>> a = 5
  >>> b = 7
  >>> x = [a, b, a+b]

• What is x[2]?

A: 'a+b'
B: 12  CORRECT
C: 57
D: ERROR
E: I don’t know
Lists and Strings Go Hand in Hand

>>> text = 'A sentence is just a list of words'
>>> words = text.split()
>>> words
['A', 'sentence', 'is', 'just', 'a', 'list', 'of', 'words']

>>> lines = text.split('
')
>>> lines
['A sentence is just', ' a list of words']

>>> hyphenated = '-'.join(words)
'A-sentence-is-just-a-list-of-words'

>>> hyphenated2 = '-'.join(lines[0].split()+lines[1].split())
'A-sentence-is-just-a-list-of-words'

{text.split(<sep>): return a list of words in text (separated by <sep>, or whitespace by default)

<sep>.join(words): concatenate the items in the list of strings words, separated by <sep>.

Turns string into a list of words

Turns string into a list of lines

Combines elements with hyphens

Merges 2 lists, combines elements with hyphens