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

http://www.cs.cornell.edu/courses/cs1110/2018sp

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]
Lecture 10 Announcements

- Last call for a one-on-one!
  - CMS: OPTIONAL: one-on-ones

- Prelim 1 is March 13. LAST DAY to register a conflict or a need for accommodation. See website:

- CMS: Prelim 1 conflicts

- A1 Revisions: open from Mar 1st thru Mar 7, 11:59pm. 1 just means “not done”. many people got a 1 due to test cases.
Lecture 10 Announcements

A2 is out and about!
Handout provides links to worked examples of diagramming call frames, objects, variables, etc.
Preparatory to starting A2:
(a) *immediately* try those worked examples
(b) go to office/consulting hours if you have any difficulty

*This will make A2 go much smoother!*
# 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\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:

\[ \text{len}(x) \]
\[ x\.len() \]
A Word about Testing

Suppose you hear a rumor that the `count` method is not implemented correctly.

```python
>>> lab_scores = [5, 6, 5, 9, 0, 10, 8, 0, 7]
>>> lab_scores.count(0)
2
```

Looks good to me!

Are we done?
What should I be testing?

Common Cases: typical usage (see previous slide)

Edge Cases: live at the boundaries

- Target location in list: first, middle, last elements
- Input size: 0, 1, 2, many (length of lists, strings, etc.)
- Input Orders: max(big, small), max(small, big)…
- Element values: negative/positive, zero, odd/even
- Element types: int, float, str, etc.
- Expected results: negative, 0, 1, 2, many

Not all categories/cases apply to all functions.
Use your judgement!
Things that Work for All Sequences

$s = 'slorthy'$

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

- $s$.index('s') → 0
- $s$.count('t') → 1
- len(s) → 6
- s[4] → "h"
- s[1:3] → "li"
- s[3:] → "thy"
- s[-2] → "h"
- s + ' toves' → "slithy toves"
- s * 2 → "slithyslithy"
- 't' in s → True

- x.index(5) → 0
- x.count(6) → 2
- len(x) → 6
- x[4] → 15
- x[1:3] → [6, 9]
- x[3:] → [6, 15, 5]
- x[-2] → 15
- x + [1, 2] → [5, 6, 9, 6, 15, 5, 1, 2]
- x * 2 → [5, 6, 9, 6, 15, 5, 5, 6, 9, 6, 15, 5]
- 15 in x → True
Representing Lists

Wrong:

Global Space

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

Correct:

Global Space

x = id1

Heap Space

Indices

id1

list

0

5

1

7

2

4

3

-2

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

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

**Objects**
- Attributes are named
  - Example: p.x

---

**Global Space**

<table>
<thead>
<tr>
<th>Heap Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>id2</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Heap Space**

<table>
<thead>
<tr>
<th>Global Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
</tr>
<tr>
<td>id3</td>
</tr>
</tbody>
</table>

**Global Space**

<table>
<thead>
<tr>
<th>Heap Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>id3</td>
</tr>
<tr>
<td>Point3</td>
</tr>
</tbody>
</table>

**Heap Space**

<table>
<thead>
<tr>
<th>id3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>z</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Global Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>id3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heap Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Lists Can Hold Any Type

```
list_of_integers = [5, 7, 4, -2]
list_of_strings = ['h', 'i', '', 'there!']
```

**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>
```

**Global Space**

```
list_of_integers  id1
list_of_strings   id2
```
No Really, Lists Can Hold Any Type!

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

Heap Space

```
list
  0: id2
  1: id3
  2: id6
  3: id7
```

Global Space

```
list_of_points  id1
list_of_various_types  id9
```

```
id9
  0: 5
  1: 3.1416
  2: 'happy'
  3: id5
```

```
id2
  Point3(x=81, y=2, z=3)
```

```
id3
  Point3(x=4, y=4, 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
- p1  id1
- p2  id2
- p3  id3
- x   id4

Heap Space
- id1  Point3
  - x  1  y  2  z  3
- id2  Point3
  - x  4  y  5  z  6
- id3  Point3
  - x  7  y  8  z  9

How do I get this y?
- x[1].y

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

- **Format:**
  - `<var>[<index>] = <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'
```

Type Error: 'str' object does not support item assignment

```
x = [5, 7, 4, -2]
x[1] = 8
s = "Hello!"
s[0] = 'J'
```

Global Space

Heap Space
List Methods Can Alter the List

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

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

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

- `\texttt{<list>.sort()}`
  - What do you think this does?

See Python API for more
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:
  ```python
  >>> 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 → swap!

```python
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
**2nd Clicker Question**

```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])
```

What gets printed?

A: 5  
B: 6  
C: Something else  
D: I don’t know
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  CORRECT
B: 6
C: Something else
D: I don’t know
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]

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]
```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])
```
List Slices Make Copies

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

Copy means new folder.
3rd 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: ERROR
  E: I don’t know
• 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
4\textsuperscript{th} 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:
  ```python
  >>> x = [5, 6, 5, 9, 10]
  >>> y = x
  >>> y[1] = 7
  ```
- What is x[1]?

A: 7  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
Lists and Expressions / 5th Clicker A

- 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**

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