Lecture 10:
Lists and Sequences

(Sections 10.0-10.2, 10.4-10.6, 10.8-10.13, 12.1, 12.2)

CS 1110
Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]
Today: A1 feedback out, revisions enabled

• Set your CMS notifications to get email when "one of your grades is changed"

• Watch for instruction announcement, but: the expected first "grade" is -99999
  = "there's something we'd like you to fix"

• Revising will change -99999 to -9999 to ... until 10/10!
Magical traditions: names have power

- **Function:** a genie in a bottle you can call on.
  - You put input into the bottle, the genie assigns them "private nicknames" (the **parameter** names)
  - Does "hidden magic"/"scratch work" inside its **call frame** -- the "bottle".
  - Can *delegate* by calling other genies.

- **Call stack:** list of *pending* delegated function calls (to-do list).

- **Object:** can be affected by a function that knows its "secret name", or **id**
  - Created by a special function call "let there be a new Point": **Point(...)**; returns the secret name of new object so you can access it --- *if* you store it somewhere safe (a variable).
Analogies to ('foreign') languages

• The "dot" (.) is like an apostrophe: x.y is like "x's y", or the "y that belongs to x"

• methods: functions :: irregular verbs : verbs
  ▪ Different calling syntax: some_string.
Depicted: when line 6 of swap_x() has been executed, but before function ends (returns None). The objects were affected.

```python
import shapes

def swap_x(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_x(p, q)
```

Global Space

- `p`: id6
- `q`: id7

Heap Space

- id6: Point3
  - x: 1
  - y: 2
  - z: 3

- id7: Point3
  - x: 3
  - y: 4
  - z: 5

Call Stack

```
swap_x
    p
    tmp
    1
    q
```

Depicted: when line 6 of `bad_swap()` has been executed, but before function ends (returns None). The objects *weren't* affected.

```python
import shapes

def bad_swap(p, q):
    tmp = p
    p = q
    q = tmp

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

    bad_swap(p, q)
```
```python
def replace_first(s, target, rep):
    """Returns: copy of s with the FIRST instance of target in s replaced by rep...""
    pos = s.index(target)
    before = s[:pos]
    after = s[pos + len(target):]
    return before + rep + after

def replace2(s, target, rep):
    """Replace first two occurrences of target in s...""
    before = s[:s.index(target) + len(target)]
    after = s[s.index(target) + len(target):]
    if s.count(target) > 1:
        after = replace_first(after, target, rep)
    else:
        WARNING = <does something>
    return replace_first(before, target, rep) + after

x = "Millississippi"
```

`replace2(x, 'L', 's')` vs.
`replace2(x, 'LL', 's')`
Sequences: Lists of Values

**String**
- $s = 'abc d'$
  - 0 1 2 3 4
  - a b c 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'
- $\text{len}(s) \rightarrow 5$, length of string

**List**
- $x = [5, 6, 5, 9, 15, 23]$
  - 0 1 2 3 4 5
  - 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]
- $\text{len}(x) \rightarrow 6$, length of list

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
Representing Lists

Wrong:

- Global Space

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

Correct:

- Global Space

\[ x \quad \text{id1} \]

- Heap Space

Indices

\[
\begin{array}{c}
\text{id1} \\
0 \quad 5 \\
1 \quad 7 \\
2 \quad 4 \\
3 \quad -2 \\
\end{array}
\]
Lists: objects with special syntax (like nouns with "weird" plurals)

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

Objects
- Attributes are named
  - Example: p.x

Global Space
- x
- id2

Heap Space
- list
- id2
- 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

Expression evaluates to value; value goes in list

list_of_integers = [5, 7, 3+1, -2]
list_of_strings = ['h', 'i', '', 'there!']

Global Space
list_of_integers id1

list_of_strings id2

Heap Space

id1 list
0 5
1 7
2 4
3 -2

id2 list
0 'h'
1 'i'
2 ''
3 'there!'
Lists of Objects

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

Global Space
- p1 with id1
- p2 with id2
- p3 with id3
- x with id4

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

How do I get this y?
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

How do I get this y?

p2.y or x[1].y

Heap Space

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

```
p1 = Point3(1, 2, 3)
p2 = Point3(4, 5, 6)
p3 = Point3(7, 8, 9)
x = [p1, p2, p3]
```
List is *mutable*; strings are not

- **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'

TypeError: 'str' object does not support item assignment
```

<table>
<thead>
<tr>
<th>Global Space</th>
<th>Heap Space</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x</code></td>
<td><img src="image" alt="Heap Space Diagram" /></td>
</tr>
<tr>
<td><code>s</code></td>
<td><img src="image" alt="Heap Space Diagram" /></td>
</tr>
</tbody>
</table>
List Methods Can Alter the List

- **<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
  - `y.insert(2,-1)` changes the list to `[15, 16, -1, 15, 19]`

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

\[ x = [5, 6, 5, 9] \]
\[ y = [15, 16, 15, 19] \]
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 = [5,4,7,6,8]
swap(x, 3, 4)
print(x[3])

What gets printed?

A: 8
B: 6
C: Something else
D: I don’t know
List Slices Make Copies:
a slice of a list is a new list

\[ x = [5, 6, 5, 9] \]
\[ y = x[1:3] \]
Q3: List Slicing

• 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
A3: List Slicing

• 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
Q4

- Execute the following:
  ```
  >>> 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
A4

- Execute the following:
  >>> 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
Things that Work for All Sequences

\[ s = \text{‘slithy’} \quad x = [5, 6, 9, 6, 15, 5] \]

- \( s.\text{index(‘s’)} \rightarrow 0 \)
- \( s.\text{count(‘t’)} \rightarrow 1 \)
- \( \text{len}(s) \rightarrow 6 \)
- \( s[4] \rightarrow \text{“h”} \)
- \( s[1:3] \rightarrow \text{“li”} \)
- \( s[3:] \rightarrow \text{“thy”} \)
- \( s[-2] \rightarrow \text{“h”} \)
- \( s + \text{‘ toves’} \rightarrow \text{“slithy toves”} \)
- \( s * 2 \rightarrow \text{“slithyslithy”} \)
- \( \text{‘t’ in s} \rightarrow \text{True} \)

- \( x.\text{index(5)} \rightarrow 0 \)
- \( x.\text{count(6)} \rightarrow 2 \)
- \( \text{len}(x) \rightarrow 6 \)
- \( x[4] \rightarrow 15 \)
- \( x[1:3] \rightarrow [6, 9] \)
- \( x[3:] \rightarrow [6, 15, 5] \)
- \( x[-2] \rightarrow 15 \)
- \( x + [1, 2] \rightarrow [5, 6, 9, 6, 15, 5, 1, 2] \)
- \( x * 2 \rightarrow [5, 6, 9, 6, 15, 5, 5, 6, 9, 6, 15, 5] \)
- \( 15 \text{ in } x \rightarrow \text{True} \)
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)
>>> hyphenated
'A-sentence-is-just-a-list-of-words'
>>> hyphenated2 = '-'.join(lines[0].split()+lines[1].split())
>>> hyphenated2
'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
Returning multiple values

• Can use lists/tuples to **return** multiple values

```
INCHES_PER FOOT = 12

def to_feet_and_inches(height_in_inches):
    feet = height_in_inches // INCHES_PER FOOT
    inches = height_in_inches % INCHES_PER FOOT
    return [feet, inches]

all_inches = 68  # Prof. Bracy wrote this
data= to_feet_and_inches(all_inches)
print(You are “+str(data[0])+” feet, “+str(data[1])+” inches.”)
```