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

Orange text indicates updates made after lecture

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

- No labs Tues 2/25 (Feb Break); Wedn labs → office hrs
- *Only if* you need to request a makeup exam, do the CMS “assignment” called “Prelim 1 Conflict.” Legitimate reasons needed to request a makeup.
- Students with SDS accommodation letter: must email us if you don’t hear from us by Noon on Feb 26, Wedn.
- A1 revision process: A1 closed on CMS for grading. Set your CMS notifications to “receive email when ...” When feedback is released, read resubmission instructions
- A2 released, due Fri 2/28
- Read § 4.2, 10.3 before next lecture
**Review: Storage in Python**

- **Global Space**
  - What you “start with”
  - Stores global variables, modules & functions
  - Lasts until you quit Python

- **Heap Space**
  - Where “folders” are stored
  - Have to access indirectly

- **Call Frame Stack**
  - Parameters
  - Other variables local to function
  - Lasts until function returns
Don’t draw module folder, function folder

End of last lecture we saw

- Module folder is created upon `import`, for example,
  ```python
  import math
  ```
- Function folder is created with `def` (the function header), for example,
  ```python
  def get_feet(height_in_inches):
  ```

Don’t draw those folders and the variables that store their ids; we only explained those folders to explain what you see in Python Tutor. *Do not draw them.*
Sequences: Lists of Values

String

- $s = 'abc \ d'$
- 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

- $x = [5, 6, 5, 9, 15, 23]$
- Separate by commas
- Access \textbf{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]
- $\text{len}(x) \rightarrow 6$, length of list

\textbf{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:

```
len(x)
```

```
x.len()
```
Representing Lists

Wrong:

Global Space

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

Correct:

Global Space

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

Heap Space

Indices
Lists vs. Class Objects

List

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

Objects

- Attributes are named
  - Example: \(p.x\)
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
0 5
1 7
2 4
3 -2

id2
0 'h'
1 'i'
2 '
3 'there!'
No Really, Lists Can Hold Any Type!

list_of_points = [Point3(81,2,3), Point3(6,2,3), Point3(4,4,3), Point3(1,2,2)]

global_space

local_space

heap_space

id1

id2

id3

id6

id7

id9

id5

list_of_points

list_of_various_types
No Really, Lists Can Hold Any Type!

```python
list_of_points = [Point3(81,2,3),
                 Point3(6,2,3),
                 Point3(4,4,3),
                 Point3(1,2,2)]

list_of_various_types= [5,
                        3.14, ‘happy’,
                        Point3(10,0,13)]
```

Global Space

<table>
<thead>
<tr>
<th>id1</th>
<th>list</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>id2</td>
</tr>
<tr>
<td>1</td>
<td>id3</td>
</tr>
<tr>
<td>2</td>
<td>id6</td>
</tr>
<tr>
<td>3</td>
<td>id7</td>
</tr>
</tbody>
</table>

Heap Space

<table>
<thead>
<tr>
<th>id2</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x 81 y 2 z 3</td>
</tr>
<tr>
<td>1</td>
<td>id2</td>
</tr>
<tr>
<td>2</td>
<td>id6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id3</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x 6 y 2 z 3</td>
</tr>
<tr>
<td>1</td>
<td>id2</td>
</tr>
<tr>
<td>2</td>
<td>id6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id6</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x 4 y 4 z 3</td>
</tr>
<tr>
<td>1</td>
<td>id3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id7</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x 1 y 2 z 2</td>
</tr>
<tr>
<td>1</td>
<td>id6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id9</th>
<th>list</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3.14</td>
</tr>
<tr>
<td>2</td>
<td>‘happy’</td>
</tr>
<tr>
<td>3</td>
<td>id5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id5</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x 10 y 0 z 13</td>
</tr>
<tr>
<td>1</td>
<td>id9</td>
</tr>
</tbody>
</table>
```
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

- 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

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
- id4
  - list
    - 0
      - id1
    - 1
      - id2
    - 2
      - id3

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 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*
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?
Q1: Insert into list

- 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
A1: Insert into list

• 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

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 \texttt{swap}: parameters \(p\) and \(q\) are unchanged, global \(p\) and \(q\) are unchanged, attributes \(x\) are swapped.
Q2: Swap List Values?

```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
A2: Swap List Values?

```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
```

```python
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
Q2: Swap List Values - 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
```

Heap Space

```
  id4
  0  5
  1  4
  2  7
  3  6
  4  5
```

Call Frame

```
   swap  
   b     h  3
   k     4
```
**Q2: Swap List Values - Explanation (2)**

```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
```

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

< Diagram of list manipulation and corresponding heap space usage >
Q2: Swap List Values - Explanation (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])
```

Global Space

- `x`:
  - `id4`

Heap Space

- `id4`:
  - `0`:
    - `5`
  - `1`:
    - `4`
  - `2`:
    - `7`
  - `3`:
    - `6`
  - `5`
  - `4`:
    - `5`
```
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: a slice of a list is a new list

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

copy means new folder

Global Space

Heap Space

id5

\[ \text{id6} \]

list

0 5
1 6
2 5
3 9

0 6
1 5
Q3: List Slicing

- 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
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:
  
  ```python
  >>> x = [5, 6, 5, 9, 10]
  >>> y = x
  >>> y[1] = 7
  ```

• What is \( x[1] \)?

<table>
<thead>
<tr>
<th>A: 7</th>
<th>B: 5</th>
<th>C: 6</th>
<th>D: ERROR</th>
<th>E: I don’t know</th>
</tr>
</thead>
</table>


• 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 = ‘slithy’

- 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 = [5, 6, 9, 6, 15, 5]

- 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
**Lists and Strings Go Hand in Hand**

- **`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>`.

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

- 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.