Mutable Values
Announcements
Objects

(Demo)
Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; **classes** are first-class values in Python
- Object-oriented programming:
  - A metaphor for organizing large programs
  - Special syntax that can improve the composition of programs
- In Python, every value is an object
  - All **objects** have **attributes**
  - A lot of data manipulation happens through object **methods**
  - Functions do one thing; objects do many related things
Example: Strings

(Demo)
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0</td>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1</td>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
</tr>
<tr>
<td>0 1 0</td>
<td>2</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>’</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
</tr>
<tr>
<td>0 1 1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
</tr>
<tr>
<td>1 0 0</td>
<td>4</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>1 0 1</td>
<td>5</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
</tr>
<tr>
<td>1 1 0</td>
<td>6</td>
<td>\</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
</tr>
<tr>
<td>1 1 1</td>
<td>7</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

16 columns: 4 bits

- Layout was chosen to support sorting by character code
- Rows indexed 2-5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission

(Demo)
Representing Strings: the Unicode Standard

• 137,994 characters in Unicode 12.1
• 150 scripts (organized)
• Enumeration of character properties, such as case
• Supports bidirectional display order
• A canonical name for every character

LATIN CAPITAL LETTER A
DIE FACE-6
EIGHTH NOTE

http://ian-albert.com/unicode_chart/unichart-chinese.jpg
Mutation Operations
Some Objects Can Change

First example in the course of an object changing state

The same object can change in value throughout the course of computation

All names that refer to the same object are affected by a mutation

Only objects of *mutable* types can change: lists & dictionaries
Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```python
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2

```  

```python
def mystery(s):
    s.pop()
    s.pop()

def another_mystery():
    four.pop()
    four.pop()
```  

```python
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> another_mystery() # No arguments!
>>> len(four)
2
```
Tuples

(Demo)
Tuples are Immutable Sequences

Immutable values are protected from mutation

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

An immutable sequence may still change if it contains a mutable value as an element

```python
>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR
```

The value of an expression can change because of changes in names or objects

```python
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
>>> x + x
```

Name change:

```python
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```python
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
[1, 2, 3, 1, 2, 3]
```

Next lecture: ooze can change turtle’s binding

```python
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
[1, 2, 3]
>>> turtle
['Anything could be inside!']
```
Mutation
Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```python
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```

```python
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10]
>>> b
[10, 20]
>>> a == b
False
```
Identity Operators

Identity

<exp0> is <exp1>

evaluates to True if both <exp0> and <exp1> evaluate to the same object

Equality

<exp0> == <exp1>

evaluates to True if both <exp0> and <exp1> evaluate to equal values

Identical objects are always equal values

(Demo)
Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call.

```python
>>> def f(s=[]):
...     s.append(3)
...     return len(s)
... >>> f()
1
>>> f()
2
>>> f()
3
```

Each time the function is called, `s` is bound to the same value!
Lists
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>( s.append(t) ) ( t = 0 )</td>
<td>( s \rightarrow [2, 3, [5, 6]] ) ( t \rightarrow 0 )</td>
</tr>
<tr>
<td>extend</td>
<td>( s.extend(t) ) ( t[1] = 0 )</td>
<td>( s \rightarrow [2, 3, 5, 6] ) ( t \rightarrow [5, 0] )</td>
</tr>
<tr>
<td>addition &amp; slicing</td>
<td>( a = s + [t] ) ( b = a[1:] ) ( a[1] = 9 ) ( b[1][1] = 0 )</td>
<td>( s \rightarrow [2, 3] ) ( t \rightarrow [5, 0] ) ( a \rightarrow [2, 9, [5, 0]] ) ( b \rightarrow [3, [5, 0]] )</td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append adds one element to a list</td>
<td>[ s.append(t) ] [ t = 0 ]</td>
<td>( s \rightarrow [2, 3, [5, 6]] ) ( t \rightarrow 0 )</td>
</tr>
<tr>
<td>extend adds all elements in one list to another list</td>
<td>[ s.extend(t) ] [ t[1] = 0 ]</td>
<td>( s \rightarrow [2, 3, 5, 6] ) ( t \rightarrow [5, 0] )</td>
</tr>
</tbody>
</table>
| addition & slicing create new lists containing existing elements | \[ a = s + [t] \]
  \[ b = a[1:] \]
  \[ a[1] = 9 \]
  \[ b[1][1] = 0 \] | \( s \rightarrow [2, 3] \)
  \( t \rightarrow [5, 0] \)
  \( a \rightarrow [2, 9, [5, 0]] \)
  \( b \rightarrow [3, [5, 0]] \) |
| The list function also creates a new list containing existing elements | \[ t = list(s) \] \[ s[1] = 0 \] | \( s \rightarrow [2, 0] \)
  \( t \rightarrow [2, 3] \) |
Lists in Environment Diagrams

Assume that before each example below we execute:

\( s = [2, 3] \)
\( t = [5, 6] \)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one element to a list</td>
<td>( s.append(t) ) ( t = 0 )</td>
<td>( s \rightarrow [2, 3, [5, 6]] ) ( t \rightarrow 0 )</td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td>( s.extend(t) ) ( t[1] = 0 )</td>
<td>( s \rightarrow [2, 3, 5, 6] ) ( t \rightarrow [5, 0] )</td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td>( a = s + [t] ) ( b = a[1:] ) ( a[1] = 9 ) ( b[1][1] = 0 )</td>
<td>( s \rightarrow [2, 3] ) ( t \rightarrow [5, 0] ) ( a \rightarrow [2, 9, [5, 0]] ) ( b \rightarrow [3, [5, 0]] )</td>
</tr>
<tr>
<td>The <strong>list</strong> function also creates a new list containing existing elements</td>
<td>( t = \text{list}(s) ) ( s[1] = 0 )</td>
<td>( s \rightarrow [2, 0] ) ( t \rightarrow [2, 3] )</td>
</tr>
<tr>
<td><strong>slice assignment</strong> replaces a slice with new values</td>
<td>( s[0::] = t ) ( s[3:] = t ) ( t[1] = 0 )</td>
<td></td>
</tr>
</tbody>
</table>

![Environment Diagram](image)
Lists in Environment Diagrams

Assume that before each example below we execute:
s = [2, 3]
t = [5, 6]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one element to a list</td>
<td>s.append(t) t = 0</td>
<td>s → [2, 3, [5, 6]] t → 0</td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td>s.extend(t) t[1] = 0</td>
<td>s → [2, 3, 5, 6] t → [5, 0]</td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td>a = s + [t] b = a[1:] a[1] = 9 b[1][1] = 0</td>
<td>s → [2, 3] t → [5, 0] a → [2, 9, [5, 0]] b → [3, [5, 0]]</td>
</tr>
<tr>
<td>The <strong>list</strong> function also creates a new list containing existing elements</td>
<td>t = list(s) s[1] = 0</td>
<td>s → [2, 0] t → [2, 3]</td>
</tr>
<tr>
<td><strong>slice assignment</strong> replaces a slice with new values</td>
<td>s[0:0] = t s[3:] = t t[1] = 0</td>
<td>s → [5, 6, 2, 5, 6] t → [5, 0]</td>
</tr>
</tbody>
</table>
## Lists in Environment Diagrams

Assume that before each example below we execute:

```python
s = [2, 3]
t = [5, 6]
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| **pop** removes & returns the last element | `t = s.pop()` | `s → [2]`
|                         |                       | `t → 3`         |
| **remove** removes the first element equal to the argument | `t.extend(t)`
|                         | `t.remove(5)`        | `s → [2, 3]`
|                         |                       | `t → [6, 5, 6]` |
| **slice assignment** can remove elements from a list by assigning `[]` to a slice. | `s[:1] = []`
|                         | `t[0:2] = []`        | `s → [3]`
|                         |                       | `t → []`        |
Lists in Lists in Lists in Environment Diagrams

\[ t = [1, 2, 3] \]
\[ t[1:3] = [t] \]
\[ t.extend(t) \]

[t] evaluates to:
\[ [1, [...], 1, [...]] \]

\[ t = [[1, 2], [3, 4]] \]
\[ t[0].append(t[1:2]) \]

[[1, 2, [[3, 4]]], [3, 4]]