Lecture 7

Objects
### Announcements For This Lecture

**Lab**
- Work on Assignment 1
  - Credit = submit on time
  - Nothing else to do

**Readings**
- Thursday: Read 5.1-5.4
- Tuesday: Read Chap 4

**Assignment 1**
- Due Thursday at Mid.
  - Due *before* midnight
  - Submit something…
  - Can resubmit to Sep. 28
- Grades posted Saturday
- Posted a Survey in CMS
  - Fill it out when done

9/15/15
Type: Set of values and the operations on them

- **Type `int`:**
  - **Values:** integers
  - **Ops:** +, −, *, /, %, **

- **Type `float`:**
  - **Values:** real numbers
  - **Ops:** +, −, *, /, **

- **Type `bool`:**
  - **Values:** True and False
  - **Ops:** not, and, or

- **Type `str`:**
  - **Values:** string literals
    - Double quotes: "abc"
    - Single quotes: 'abc'
  - **Ops:** + (concatenation)

Are the the only types that exist?
Type: Set of values and the operations on them

• Want a point in 3D space
  ▪ We need three variables
  ▪ $x, y, z$ coordinates

• What if have a lot of points?
  ▪ Vars $x_0, y_0, z_0$ for first point
  ▪ Vars $x_1, y_1, z_1$ for next point
  ▪ …
  ▪ This can get really messy

• How about a single variable that represents a point?
Type: Set of values and the operations on them

- Want a point in 3D space
  - We need three variables
  - $x$, $y$, $z$ coordinates
- What if have a lot of points?
  - Vars $x_0$, $y_0$, $z_0$ for first point
  - Vars $x_1$, $y_1$, $z_1$ for next point
  - ...
  - This can get really messy
- How about a single variable that represents a point?
- Can we stick them together in a “folder”? 
- Motivation for objects
Objects: Organizing Data in Folders

- An object is like a **manila folder**
- It contains other variables
  - Variables are called **attributes**
  - These values can change
- It has an **ID** that identifies it
  - Unique number assigned by Python (just like a NetID for a Cornellian)
  - Cannot ever change
  - Has no meaning; only identifies

<table>
<thead>
<tr>
<th>id1</th>
<th>x</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>y</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td>z</td>
<td>5.0</td>
</tr>
</tbody>
</table>
Classes: Types for Objects

• Values must have a type
  ▪ An object is a **value**
  ▪ Object type is a **class**

• **Modules** provide classes
  ▪ Will show how later

• **Example**: geom
  ▪ Part of CornellExtensions
  ▪ Just need to import it
  ▪ Classes: Point2, Point3
Constructor: Function to make Objects

- How do we create objects?
  - Other types have **literals**
  - **Example:** 1, 'abc', true
  - No such thing for objects
- **Constructor Function:**
  - Same name as the class
  - **Example:** Point3(0,0,0)
  - Makes an object (manila folder)
  - Returns folder ID as value
- **Example:** p = Point3(0, 0, 0)
  - Creates a Point object
  - Stores object’s ID in p
Constructors and Modules

```python
>>> import geom

Need to import module that has Point class.

>>> p = geom.Point3(0,0,0)

Constructor is function. Prefix w/ module name.

>>> id(p)

Shows the ID of p.

Actually a big number
```

```python
id2
```

```
Point3
```

```
x 0.0
y 0.0
z 0.0
```
Object Variables

• Variable stores object name
  ▪ **Reference** to the object
  ▪ Reason for folder analogy

• Assignment uses object name
  ▪ **Example**: q = p
    ▪ Takes name from p
    ▪ Puts the name in q
    ▪ Does not make new folder!

• **This is the cause of many mistakes in this course**
Objects and Attributes

- Attributes are variables that live inside of objects
  - Can **use** in expressions
  - Can **assign** values to them

- **Access:** `<variable>..<attr>`
  - **Example:** `p.x`
  - Look like module variables

- Putting it all together
  - `p = geom.Point3(1,2,3)`
  - `p.x = p.y + p.z`

<table>
<thead>
<tr>
<th>id3</th>
<th>x</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>y</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>z</td>
<td>3.0</td>
</tr>
</tbody>
</table>
Objects and Attributes

- Attributes are variables that live inside of objects
  - Can **use** in expressions
  - Can **assign** values to them

- **Access**: `<variable>..<attr>`
  - **Example**: `p.x`
  - Look like module variables

- Putting it all together
  - `p = geom.Point3(1,2,3)`
  - `p.x = p.y + p.z`
Exercise: Attribute Assignment

• Recall, q gets name in p
  >>> p = geom.Point3(0,0,0)
  >>> q = p

• Execute the assignments:
  >>> p.x = 5.6
  >>> q.x = 7.4

• What is value of p.x?
  A: 5.6
  B: 7.4
  C: id4
  D: I don’t know
Exercise: Attribute Assignment

- Recall, q gets name in p
  >>> p = geom.Point3(0,0,0)
  >>> q = p

- Execute the assignments:
  >>> p.x = 5.6
  >>> q.x = 7.4

- What is value of p.x?

A: 5.6  
B: 7.4 CORRECT  
C: id4
D: I don’t know
Exercise: Attribute Assignment

• Recall, q gets name in p

```python
>>> p = geom.Point3(0,0,0)
>>> q = p
```

• Execute the assignments:

```python
>>> p.x = 5.6
>>> q.x = 7.4
```

• What is value of p.x?

A: 5.6
B: 7.4  CORRECT
C: id4
D: I don’t know
Call Frames and Objects

- Mutable objects can be altered in a function call
  - Object vars hold names!
  - Folder accessed by both global var & parameter

- Example:

```python
def incr_x(q):
    q.x = q.x + 1

>>> p = geom.Point3()

>>> incr_x(p)
```

Mutable objects can be altered in a function call
- Object vars hold names!
- Folder accessed by both global var & parameter

**Example:**

```python
def incr_x(q):
    q.x = q.x + 1

>>> p = geom.Point3()
>>> incr_x(p)
```

9/15/15
Call Frames and Objects

• Mutable objects can be altered in a function call
  ▪ Object vars hold names!
  ▪ Folder accessed by both global var & parameter

• Example:

```python
def incr_x(q):
    q.x = q.x + 1

>>> p = geom.Point3()

>>> incr_x(p)
```

Global STUFF

```
Point3
  x 0.0 1.0
...
```

Call Frame

```
def incr_x(q):
    q.x = q.x + 1

>>> p = geom.Point3()

>>> incr_x(p)
```
Methods: Functions Tied to Objects

- **Method**: function tied to object
  - Method call looks like a function call preceded by a variable name:
    
    \[(\text{variable}) \cdot (\text{method})(\langle\text{arguments}\rangle)\]

  - **Example**: \(p\.distanceTo(q)\)
  - **Example**: \(p\.abs() \# \text{makes } x,y,z \geq 0\)

- Just like we saw for strings
  - \(s = 'abracadabra'\)
  - \(s\.index('a')\)

- Are strings objects?
Surprise: All Values are in Objects!

- Including basic values
  - int, float, bool, str
- Example:
  >>> x = 2.5
  >>> id(x)
- But they are immutable
  - Contents cannot change
  - Distinction between value and identity is immaterial
  - So we can ignore the folder

2.5

id5

float

id5

x

2.5

x
Surprise: All Values are in Objects!

• Including basic values
  - int, float, bool, str

• Example:
  >>> x = 'foo'
  >>> id(x)

• But they are immutable
  - No string method can alter the contents of a string
  - x.replace('o','y') evaluates to 'fyy' but x is still 'foo'
  - So we can ignore the folder
Class Objects

- Use name **class object** to distinguish from other values
  - Not int, float, bool, str
- Class objects are **mutable**
  - You can change them
  - Methods can have effects besides their return value
- **Example:**
  - `p = Point(3, -3, 0)`
  - `p.clamp(-1, 1)`

**Example: Files**

- `f = open('jabber.txt')`
- `s = f.read()`
- `f.close()`

Opens a file on your disk; returns a **file object** you can read
# Base Types vs. Classes

<table>
<thead>
<tr>
<th>Base Types</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Built-into Python</td>
<td>• Provided by modules</td>
</tr>
<tr>
<td>• Refer to instances as <em>values</em></td>
<td>• Refer to instances as <em>objects</em></td>
</tr>
<tr>
<td>• Instantiate with <em>literals</em></td>
<td>• Instantiate w/ <em>constructors</em></td>
</tr>
<tr>
<td>• Are all immutable</td>
<td>• Can alter attributes</td>
</tr>
<tr>
<td>• Can ignore the folders</td>
<td>• Must represent with folders</td>
</tr>
</tbody>
</table>
Aside: Name Resolution

- \( \langle \text{object} \rangle.\langle \text{name} \rangle \) means
  - Go the folder for \textit{object}
  - Look for \textit{attr/method name}
  - If missing, check \textit{class folder}
- Class folder is a \textit{shared folder}
  - Only one for the whole class
  - Shared by all objects of class
  - Stores common features
  - Typically where methods are
- Do not worry about this yet

\[ \begin{array}{c|c}
\text{id3} & \text{id4} \\
\hline
\text{Point3} & \text{Point3} \\
\hline
x & 5.0 & 7.4 \\
y & 2.0 & 0.0 \\
z & 3.0 & 0.0 \\
\end{array} \]

\[
\begin{align*}
\text{p} & \quad \text{id3} \\
\text{id3} & \quad \text{Point3} \\
\text{x} & \quad 5.0 \\
y & \quad 2.0 \\
z & \quad 3.0 \\
\text{q} & \quad \text{id4} \\
\text{id4} & \quad \text{Point3} \\
x & \quad 7.4 \\
y & \quad 0.0 \\
z & \quad 0.0 \\
\end{align*}
\]

\[
\text{Point} \quad \text{Point}
\]

\[
\begin{align*}
\text{Point} & \quad \text{Point} \\
\_\text{init\_}(x, y, z) & \quad \text{distanceTo}(\text{other}) \\
\text{abs}() & \quad \text{abs}() \\
\end{align*}
\]
Where To From Here?

• Right now, just try to understand objects
  ▪ All Python programs use objects
  ▪ Most small programs use objects of classes that are part of the Python Library

• OO Programming is about creating classes
  ▪ Eventually you will make your own classes
  ▪ Classes are the primary tool for organizing more complex Python programs
  ▪ But we need to learn other basics first
A1: The Module `urllib2`

- Module `urllib2` is used to read web pages
  - Function `urlopen` creates a `url` object
  - `u = urllib2.urlopen('http://www.cornell.edu')`

- `url` has a method called `read()`
  - Returns contents of web page
  - **Usage**: `s = u.read()`  # `s` is a string
• Module urllib2 is used to read web pages
  ▪ Function urlopen creates a url object
  ▪ `u = urllib2.urlopen('http://www.cornell.edu')`

• url has a method called read()
  ▪ Returns contents of web page
  ▪ **Usage**: `s = u.read()` # s is a string