Lecture 7

Objects
Announcements For This Lecture

This Week

• Lab is OPTIONAL
  ▪ Time to work on A1
  ▪ Extra testing exercises
  ▪ Credit if you turn in A1

• A1 due Sunday at mid.
  ▪ Start early to avoid rush

• One-on-Ones this week
  ▪ Lots of spaces available

Readings

• Thursday: Read 5.1-5.4

• Tuesday: SKIM Chap 4
  ▪ Don’t use Swampy

AI Quiz

• Sent out e-mails Sunday

• Will start dropping today

9/13/16
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

9/13/16

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
Classes: Types for Objects

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

![Diagram showing classes and their instances]

- Classes are how we add new types to Python

```python
id2

<table>
<thead>
<tr>
<th>x</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>3.0</td>
</tr>
<tr>
<td>z</td>
<td>5.0</td>
</tr>
</tbody>
</table>
```

**Types**

- int
- float
- bool
- str

**Classes**

- Point3
- Point2
- Window
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

>>> 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
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`
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
  ```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
  C: id4
  D: I don’t know

id4

id4

p

q

id4

Point3

x 0.0
y 0.0
z 0.0
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

9/13/16  Objects  15
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
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)
```

```
9/13/16 Objects 17
```
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**

Call Frame
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

Call Frame

```
Point3

id5

p

id5

x

0.0 1.0

...```

```
9/13/16

Objects

19
```
Methods: Functions Tied to Objects

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

  - **Example**: p.distanceTo(q)
  - **Example**: p.abs() # makes x, y, z ≥ 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

9/13/16 Objects

21
Surprise: All Values are in Objects!

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

- Example:
  ```python
  >>> x = 'foo'
  >>> id(x)
  >>> 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**

```python
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>

9/13/16
Aside: Name Resolution

- \( \text{object} . \text{name} \) means
  - Go the folder for \textit{object}
  - Look for attr/method \textit{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{itemize}
  \item \text{Point3}
  \item \text{__init__}(x, y, z)
  \item \text{distanceTo}(other)
  \item \text{abs()}
\end{itemize}
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
  ▪ \texttt{u = urllib2.urlopen('http://www.cornell.edu')} 

• url has a method called read()
  ▪ Returns contents of web page
  ▪ \textbf{Usage: } \texttt{s = u.read()} \# \texttt{s} is a string
A1: The Module urllib2

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

- url has a method called read()
  - Returns contents of web page
  - \textbf{Usage}: \texttt{s = u.read()} # s is a string

\textbf{9/13/16 Objects 28}