Lecture 7:
Objects
(Chapter 15)
CS 1110
Introduction to Computing Using Python

http://www.cs.cornell.edu/courses/cs1110/2018sp

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]
Good news about Assignment A1 & Lab 3

- **Now:** Optional one-on-one programming practice with a staff member to help just you. Sign up on CMS under the “Optional 1-on-1” assignment.
- **This week:** You have two weeks to do lab 3, and it helps you with A1.
- **Break:** There will be some consulting hours. Check the calendar.
- **Next week:** No new lab. All Wed. Feb 21 labs are drop-in office hours open to all. (Nobody at the Tuesday labs – break).
CS 1110 Spring 2018, Assignment 1: Class Status Lookup*

February 14, 2018

Navigating links in this pdf. Text in any shade of blue in this document is a click-able link.

Watch the course website for any announcements or updates.

Contents

1 Rules
   1.1 How To Register a Partner (You Only Get One) ............................................. 1
   1.2 What Collaborations Are (Dis-)Allowed And How To Document Them .................. 1
   1.3 Python You Are NOT Allowed To Use In This Assignment .............................. 2

2 Deadlines
   2.1 Is there going to be a chance to revise in response to grader feedback? .......... 2
Partnering (See section 1.1)

• You may do this assignment with \textit{at most one} other person

• Make partnership on official on CMS \textit{before either of you submit any files}

• If your partnership dissolves, there are special “group divorce” procedures you must follow
  - Take this very seriously!
Academic Integrity Rules Gloss (1.2)

• Never look at another else’s code.

• Never show your code or talk about lines of your actual code to someone else (except course staff).

• DO specifically acknowledge by name all help you received, whether or not it was “legal”
Submit early and often

- Submit your first draft to CMS by Monday, February 26 at **2pm**.
- You can replace older submissions with improved ones up to the deadline of 11:59pm.
- Benefits:
  - practice with CMS before deadline is looming
  - alert us during business hours if problems arise
  - You have been warned to submit by 2pm. Do not expect that we will accept work that doesn’t make it onto CMS on time, for whatever reason.
Type: set of values & operations on them

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

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

Type **bool:**
- Values: integers
- Ops: not, and, or

Type **str:**
- Values: string literals
  - Double quotes: “abc”
  - Single quotes: ‘abc’
- Ops: + (concatenation)
Built-in Types are not “Enough”

- 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?
Built-in Types are not “Enough”

- 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

\[
\begin{array}{|c|}
\hline
x & 2 \\
\hline
y & 3 \\
\hline
z & 5 \\
\hline
\end{array}
\]
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
Classes: user-defined types

- Values must have a type
  - An object is a **value**
  - Object type is a **class**
- **Modules** provide classes
- **Example**: shapes.py
  - Defines: Point3, Rectangle classes
Constructor: Function to make Objects

- How do we create objects?
  - Other types have literals
  - No such thing for objects

- Constructor Function:
  - Format: `<class name> (arguments)`
  - Example: `Point3(0,0,0)`
  - Makes a new object (manila folder) with a new id
  - Called an instantiated object
  - 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 shapes

Need to import module that has Point class.

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

Constructor is function. Prefix w/ module name.

>>> id(p)

Shows the id of p
```
Accessing Attributes

- Attributes are variables that live inside of objects
  - Can use in expressions
  - Can assign values to them
- **Format**: `<variable>.<attribute>`
  - **Example**: `p.x`
  - Look like module variables
- To evaluate `p.x`, Python:
  1. finds folder with `id` stored in `p`
  2. returns the value of `x` in that folder
Accessing Attributes Example

• Example:
  - `p = shapes.Point3(1, 2, 3)`
  - `p.x = p.x + 3`
Object Variables

- Variable stores object \textit{id}
  - \textbf{Reference} to the object
  - Reason for folder analogy

- Assignment uses object \textit{id}
  - \textbf{Example}:
    - \texttt{p1 = shapes.Point3(0, 0, 0)}
    - \texttt{p2 = p1}
    - Takes contents from \texttt{p1}
    - Puts contents in \texttt{p2}
    - Does not make new folder!

This is the cause of many mistakes in this course
Attribute Assignment (Question)

```python
>>> p = shapes.Point3(0,0,0)
>>> q = p
• Execute the assignments:
  >>> p.x = 5
  >>> q.x = 7
• What is value of p.x?

A: 5  
B: 7  
C: id4  
D: I don’t know
```
Attribute Assignment (Solution)

>>> p = shapes.Point3(0,0,0)
>>> q = p

• Execute the assignments:
  >>> p.x = 5
  >>> q.x = 7

• What is value of p.x?

A: 5
B: 7  CORRECT
C: id4
D: I don’t know
Call Frames and Objects (1)

- Objects can be altered in a function call
  - Object variables hold *ids*
  - Folder can be accessed from global variable or parameter

**Example:**

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

```python
>>> p = shapes.Point3(1, 2, 3)
```

```python
>>> incr_x(p)
```

```python
id5
```

Global Memory

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

```
incr_x
```

```
q
```

```
p
```

*id5*

```
Point3
```

Call Frame

```
q
```

```
incr_x
```

```
1
```

```
...x
```

```
id5
```

```
1
```

```
Point3
```

```
1
```

```
...p
```

```
1
```

```
id5
```

```
Point3
```

```
1
```

```
id5
```

```
1
```
**Call Frames and Objects (2)**

- Objects can be altered in a function call
  - Object variables hold *ids*!
  - Folder can be accessed from global variable or parameter

**Example:**

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

>>> p = shapes.Point3(1, 2, 3)
>>> incr_x(p)
```

Evaluation:

- `id5
  - p
  - id5

Global Memory

- `id5
  - p
  - id5

Call Frame

- `id5
  - p
  - id5

- `x
  - 2

- `...`

- `incr_x
  - q
  - id5`

- `RETURN
  - NONE`
Objects can be altered in a function call
- Object variables hold *ids*!
- Folder can be accessed from global variable or parameter

**Example:**

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

>>> p = shapes.Point3(1, 2, 3)
>>> incr_x(p)
How Many Folders (Question)

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

Draw everything that gets created. How many folders get drawn?
import shapes
p = shapes.Point3(1, 2, 3)
q = shapes.Point3(3, 4, 5)

Draw everything that gets created. How many folders get drawn?
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)

Draw everything that gets created.
How many folders get drawn?
What else gets drawn?
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)

Draw everything that gets created. How many folders get drawn? What else gets drawn?
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
swap_x(p, q)

def swap_x(p, q):
    t = p.x
    p.x = q.x
    q.x = t

Execute `swap_x` on what we just drew. There should be a call frame. What is in `p.x` at the end?

A: 1
B: 2
C: 3
D: I don’t know
Execute `swap_x` on what we just drew. There should be a call frame. What is in `p.x` at the end?

A: 1  
B: 2  
C: 3  CORRECT  
D: I don’t know
Swap Explanation (1)

```python
def swap_x(p, q):
    t = p.x
    p.x = q.x
    q.x = t
```

```python
import shapes
p = shapes.Point3(1, 2, 3)
q = shapes.Point3(3, 4, 5)
swap_x(p, q)
```
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
swap_x(p, q)

def swap_x(p, q):
    1. t = p.x
    2. p.x = q.x
    3. q.x = t
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
swap_x(p, q)
import shapes
p = shapes.Point3(1, 2, 3)
q = shapes.Point3(3, 4, 5)
swap_x(p, q)

```
def swap_x(p, q):
    t = p.x
    p.x = q.x
    q.x = t
```

**swap_x**

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>id2</td>
</tr>
</tbody>
</table>

**Point3**

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

```
Return

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>id2</td>
</tr>
<tr>
<td>t</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Return: **NONE**
```python
import shapes
p = shapes.Point3(1, 2, 3)
q = shapes.Point3(3, 4, 5)
swap_x(p, q)

def swap_x(p, q):
    t = p.x
    p.x = q.x
    q.x = t
```

**Swap Explanation (5)**

- `p` and `q` are two `Point3` objects.
- `swap_x(p, q)` function is called to swap the x-coordinates of `p` and `q`.
- After the swap, the x-coordinate of `p` becomes 3 and the x-coordinate of `q` becomes 1.
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
swap(p, q)

Before calling \texttt{swap}(p, q):

\begin{itemize}
\item \texttt{p}: \texttt{id1}
\item \texttt{q}: \texttt{id2}
\end{itemize}

What is in global \texttt{p} after calling \texttt{swap}? 

\begin{itemize}
\item \textbf{A:} \texttt{id1}
\item \textbf{B:} \texttt{id2}
\item \textbf{C:} I don’t know
\end{itemize}
Global p (Solution)

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

After calling `swap(p, q)`:  

<table>
<thead>
<tr>
<th>p</th>
<th>id1</th>
<th>q</th>
<th>id2</th>
</tr>
</thead>
</table>

What is in global p after calling swap?  

A: id1  CORRECT  
B: id2  
C: I don’t know
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
swap(p, q)

def swap(p, q):
    t = p
    p = q
    q = t
import shapes
p = shapes.Point3(1, 2, 3)
q = shapes.Point3(3, 4, 5)
swap(p, q)

def swap(p, q):
    t = p
    p = q
    q = t

id1

Point

x 1
y 2
z 3

id2

Point

x 3
y 4
z 5
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
swap(p, q)

def swap(p, q):
  t = p
  p = q
  q = t
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
swap(p, q)

def swap(p, q):
    t = p
    p = q
    q = t

id1  id2
p    q

id1  Point
id2  Point

swap
p  id1 id2  q  id2 id1
x 1
y 2
z 3
x 3
y 4
z 5

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

def swap(p, q):
    t = p
    p = q
    q = t
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
swap(p, q)

Before calling `swap(p, q)`:  

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>id2</td>
</tr>
</tbody>
</table>

What is in global `p` after calling `swap`?

A: id1  
B: id2  
C: I don’t know
Global p Revisited (Solution)

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

def swap(a, b):
    t = a
    a = b
    b = t

After calling swap(p, q):

p id1 q id2

What is in global p after calling swap?

A: id1 CORRECT
B: id2
C: I don’t know

Same question as the previous one, (except we re-named the parameters) but this time it seems obvious that global p is unaffected.
Methods: Functions Tied to Classes

- **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:

```python
import shapes
p = shapes.Point3(1,2,3)
p.greet()

"Hi! I am a 3-dimensional point located at (4,2,3)"
```
# Built-in Types vs. Classes

<table>
<thead>
<tr>
<th>Built-in 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>• Can ignore the folders</td>
<td>• Must represent with folders</td>
</tr>
</tbody>
</table>
Where To From Here?

• First, Understand **objects**
  - All Python programs use objects
  - Most small programs use objects of classes that are part of the Python Library

• Eventually, create your own **classes:**
  - the heart of OO Programming
  - the primary tool for organizing Python programs

• But we need to learn more basics first!