Lecture 7:
Objects
(Chapter 15)

CS 1110

Introduction to Computing Using Python

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

• Optional 1-on-1 with a staff member to help just you with course material. Sign up for a slot on CMS under “SPECIAL: one-on-ones“.

• A1: updates on course website—see orange text on cover page of A1 on website. We encourage you to use Ed Discussions

• Want more examples or practice questions on string functions? See archive on course website.
Be sure to start A1 now

• **Start A1 now 😊**
  - Give yourself time to think through any difficult parts
  - Consulting/office hours not too busy now—can get help fast
  - There’s time to schedule a 1-on-1 appt

  Rewarding learning experience

• **Start A1 the night before due date**
  - *No time to deal with “sudden” difficulties*
  - *Consulting/office hours very crowded—long wait time*

  Stressful experience undermines learning
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?
Can we stick them together in a "folder"?

Motivation for objects

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?
Analogy: A folder is used to store info (data)
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

Unique tab identifier

id1

<table>
<thead>
<tr>
<th>x</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>3</td>
</tr>
<tr>
<td>z</td>
<td>5</td>
</tr>
</tbody>
</table>
Classes: user-defined types for Objects

- 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

You just need to **use** (have) the file `shapes.py`; no need to read its code for now. You can read the docstring though to learn about the `Point3` class. **Later** in the course you will learn how to write such class files.
Constructor: Function to make Objects

- How do we create objects?
  - Other types have literals
  - No such thing for objects
- Call a 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
Storage in Python

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

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

- **Call Frames**
  - Parameters
  - Other variables local to function
  - Lasts until function returns
Constructors and Modules

>>> import shapes

Need to import module that has Point3 class.

- This is what’s actually happening
- Python Tutor draws this.
- Knowing this will help you debug.

CS 1110 doesn’t draw module variables & module folders (also skips all the built-in functions)

→ makes your diagrams cleaner
Constructors and Modules

```python
>>> import shapes

Need to import module that has `Point3` 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:** \( \langle \text{variable} \rangle.\langle \text{attribute} \rangle \)

- **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:

```python
p = shapes.Point3(1, 2, 3)
p.x = p.x + 3
```
## Object Variables

- Variable stores object *id*
  - **Reference** to the object
  - Reason for folder analogy
- Assignment uses object *id*
  - **Example:**
    ```python
    p1 = shapes.Point3(0, 0, 0)
p2 = p1
    ```
  - Takes contents from *p1*
  - Puts contents in *p2*
  - *Does not make new folder!*

This is the cause of many mistakes when starting to use objects
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
```
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:**

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

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

22
• 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
```

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

---

**Call Frame**

```
Global Space

Heap Space

Point3

x 2
...
```

```
incr_x
```

```
q  id5
```

```
RETURN
```
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)
```

```python
id5
```

```
i incr_x
q id5
```

```
Global Space
p
id5
Heap Space
id5
Point3
x 2
...
```

```
Call Frame
```

```
RETURN
NONE
```

```
RETURN
NONE
```
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)
def swap_x(p, q):
    t = p.x
    p.x = q.x
    q.x = t
swap_x(p, q)

What is in p.x at the end of this code?
A: 1
B: 2
C: 3
D: I don’t know
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
def swap(p, q):
    t = p
    p = q
    q = t
swap(p, q)

What is in global p after calling swap?
A: id1
B: id2
C: I don’t know

Heap Space

Global Space

<table>
<thead>
<tr>
<th>id1</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>y</td>
<td>2</td>
</tr>
<tr>
<td>z</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id2</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>4</td>
</tr>
<tr>
<td>z</td>
<td>5</td>
</tr>
</tbody>
</table>
Methods: Functions Tied to Classes

• **Method**: function tied to object
  - Method call looks like a function call preceded by a variable name:

    \[(\text{variable}) . (\text{method})(\text{arguments})\]

**Example:**

```python
import shapes
u = shapes.Point3(4,2,3)
u.greet()
```

“Hi! I am a 3-dimensional point located at (4,2,3)”

Where else have you seen this??
Recall: String Methods

- \texttt{s1.upper()}
  - Returns an upper case version of \texttt{s1}

- \texttt{s1.strip()}
  - Returns a copy of \texttt{s} with white-space removed at ends

- \texttt{s1.index(s2)}
  - Returns position of the first instance of \texttt{s2} in \texttt{s1}
  - \texttt{error} if \texttt{s2} is not in \texttt{s1}

- \texttt{s1.count(s2)}
  - Returns number of times \texttt{s2} appears inside of \texttt{s1}
# 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>

37
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!