Lecture 7: Objects (Chapter 15)

CS 1110

Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]
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 $x0, y0, z0$ for first point
  - Vars $x1, y1, z1$ for next point
  - ...
  - This can get really messy
- How about a single variable that represents a point?
• 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
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

```
Point3

  x  2
  y  3
  z  5
```
**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`
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 Point class.

- This **is** what’s actually happening
- Python Tutor draws this.

CS 1110 omits module variables & folders
(also omit all the built-in functions)

→ makes your diagrams cleaner
Constructors and Modules

>>> 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**: \(<\text{variable}\>_.<\text{attribute}\>
  - **Example**: p.x
  - Look like module variables

- To evaluate p.x, Python:
  1. finds folder with \text{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 *id*
  - **Reference** to the object
  - Reason for folder analogy

- Assignment uses object *id*
  - **Example:**
    - `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 in this course
Attribute Assignment (Question)

>>> 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
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 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)
```
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:**

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

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

1. `q.x = q.x + 1`

```
RETURN
```
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?

Heap Space

id1
Point3
x 1
y 2
z 3

id2
Point3
x 3
y 4
z 5
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?

Heap Space

id1

Point3

x 1
y 2
z 3

id2

Point3

x 3
y 4
z 5
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?

Global Space

Heap Space

id1

Point3

x 1
y 2
z 3

id2

Point3

x 3
y 4
z 5
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_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  CORRECT
D: I don’t know

Heap Space

Global Space

```
p  id1
q  id2
```

```
Point3

id1

x 1
y 2
z 3

id2

x 3
y 4
z 5
```
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

Global Space

p id1
q id2

Heap Space

id1
Point3
x 1
y 2
z 3

id2
Point3
x 3
y 4
z 5
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  CORRECT
B: id2
C: I don’t know
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:**
```
import shapes
p = shapes.Point3(1,2,3)
p.greet()
```

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

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

- **$s_1.upper()$**
  - Returns an upper case version of $s_1$

- **$s_.strip()$**
  - Returns a copy of $s$ with white-space removed at ends

- **$s_1.index(s_2)$**
  - Returns position of the first instance of $s_2$ in $s_1$
  - **error** if $s_2$ is not in $s_1$

- **$s_1.count(s_2)$**
  - Returns number of times $s_2$ appears inside of $s_1$
# Built-in Types vs. Classes

## Built-in types
- Built-into Python
- Refer to instances as *values*
- Instantiate with *literals*
- Can ignore the folders

## Classes
- Provided by modules
- Refer to instances as *objects*
- Instantiate w/ *constructors*
- Must represent with folders
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!