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

Lecture 7: Objects

(Chapter 15)

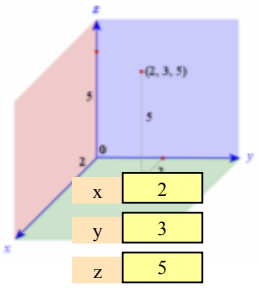
CS 1110

Introduction to Computing Using Python

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

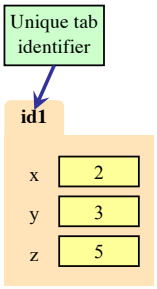
Built-in Types are not "Enough"

- 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?



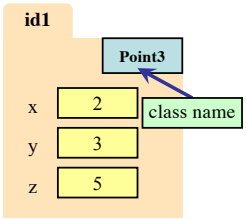
Objects: Organizing Data in Folders

- 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



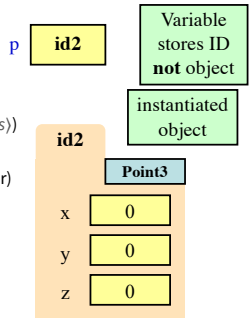
Classes: user-defined types

- 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



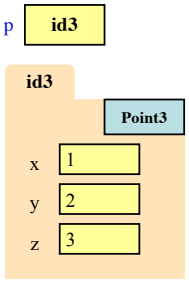
Constructor: Function to make Objects

- 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



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

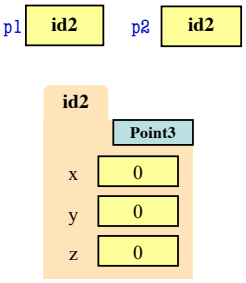


Object Variables

- 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
```

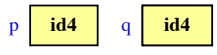
  - Takes contents from `p1`
  - Puts contents in `p2`
  - Does not make new folder!



This is the cause of many mistakes in this course

Attribute Assignment (Question)

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

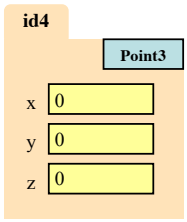


- Execute the assignments:

```
>>> p.x = 5
>>> q.x = 7
```

- What is value of `p.x`?

A: 5
B: 7
C: **id4**
D: I don't know



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?

Swap (Question)

```
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
```

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

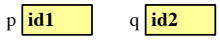
A: 1
B: 2
C: 3
D: I don't know

Global p (Question)

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

```
def swap(p, q):
1   t = p
2   p = q
3   q = t
```

Before calling `swap(p, q)`:



What is in global `p` after calling swap?

A: **id1**
B: **id2**
C: I don't know

Built-in Types vs. Classes

Built-in types	Classes
• Built-into Python	• Provided by modules
• Refer to instances as *values*	• Refer to instances as *objects*
• Instantiate with *literals*	• Instantiate w/ *constructors*
• Can ignore the folders	• Must represent with folders