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]

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*

http://www.cs.cornell.edu/courses/cs1110/2018sp/assignments/assignment1/a1.pdf

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**

1.1 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**

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 *at most one* other person
- Make partnership on official on CMS *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:**

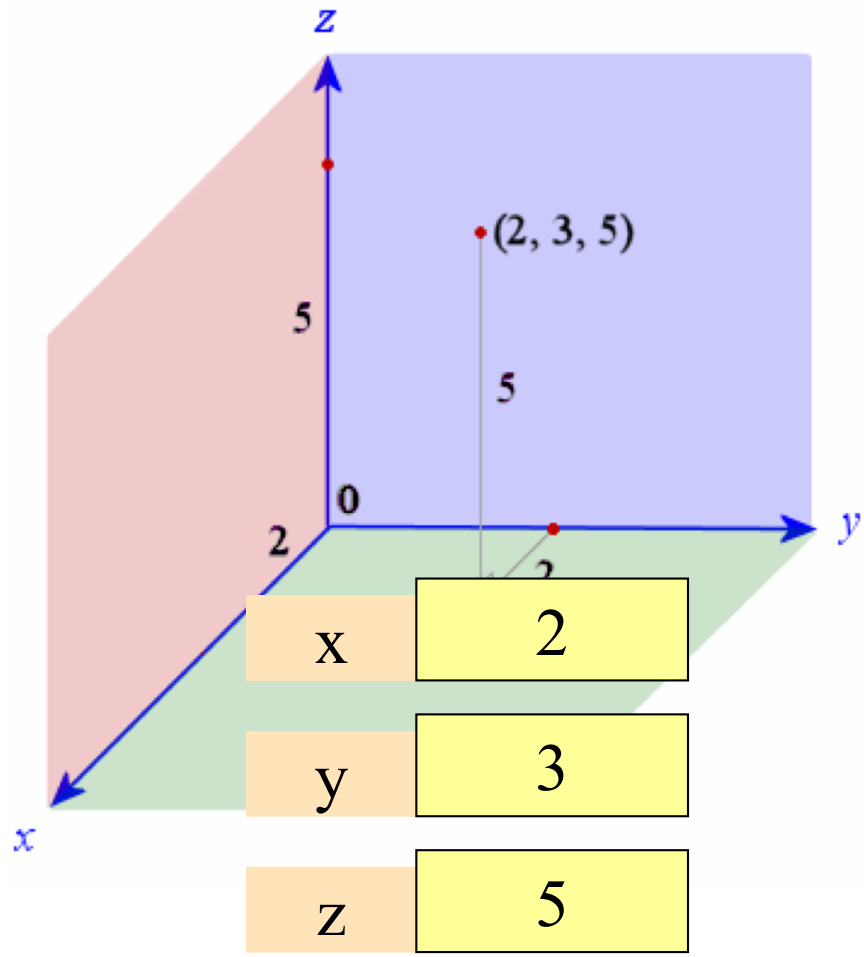
- Values: integers
- Ops: not, and, or

Type **str:**

- Values: string literals
  - Double quotes: **“abc”**
  - Single quotes: **‘abc’**
- Ops: + (concatenation)

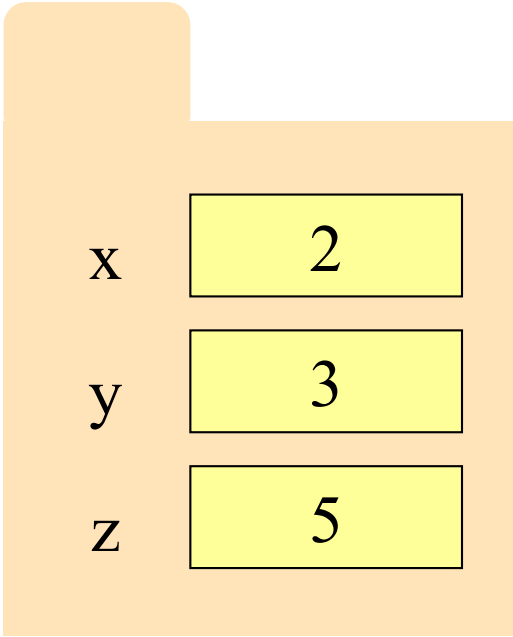
Built-in Types are not "Enough"

- Want a point in 3D space
  - We need three variables
  - x, y, z coordinates
- What if have a lot of points?
  - Vars 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?



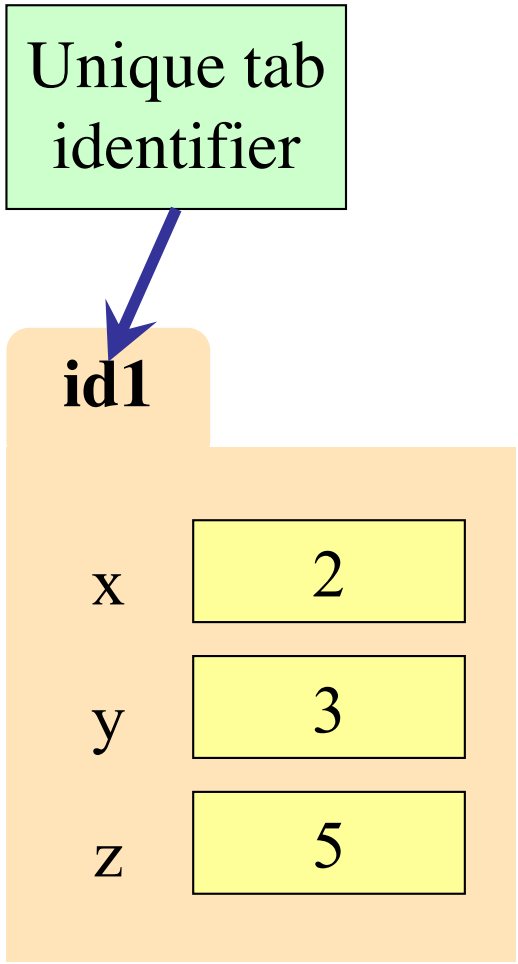
Built-in Types are not "Enough"

- 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?
- Can we stick them together in a "folder"?
- Motivation for **objects**



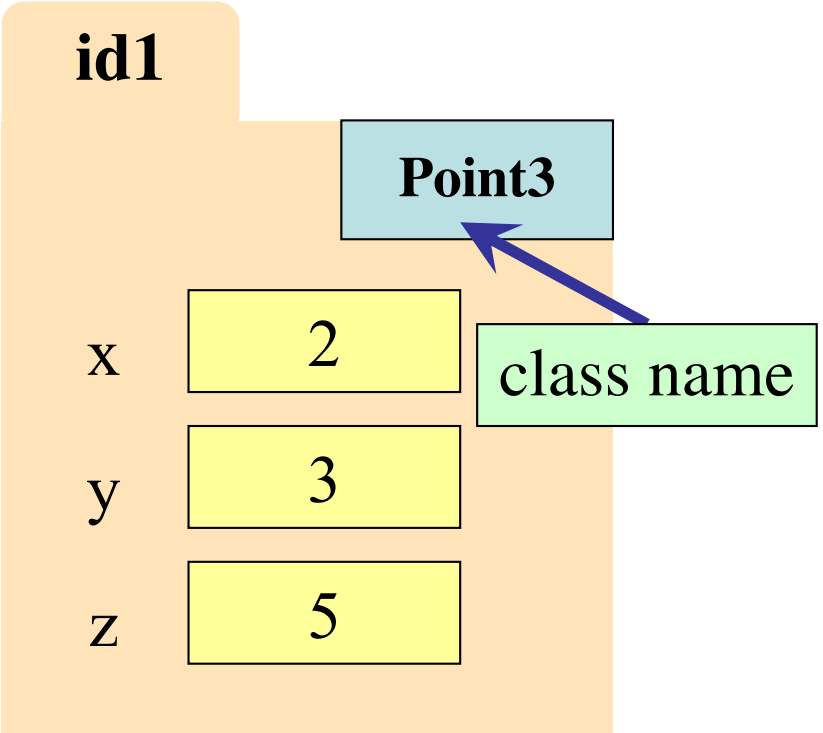
Objects: Organizing Data in Folders

- 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



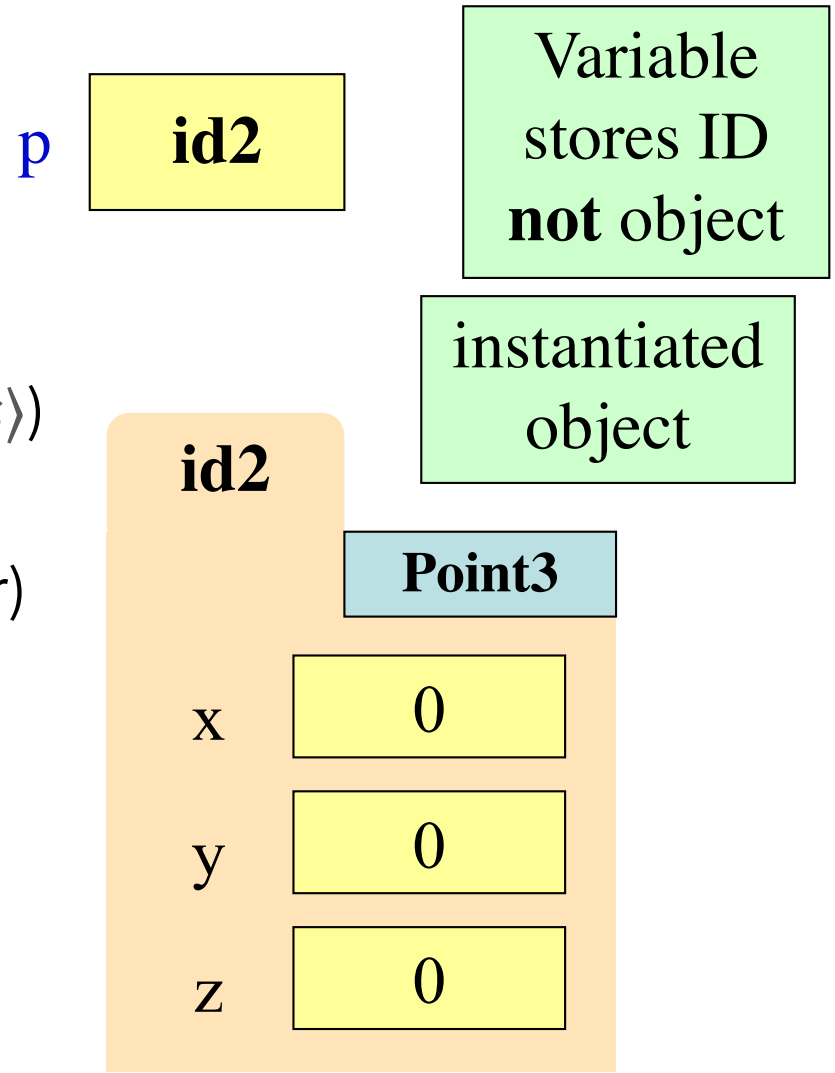
Classes: user-defined types

- 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

```
>>> import shapes
```

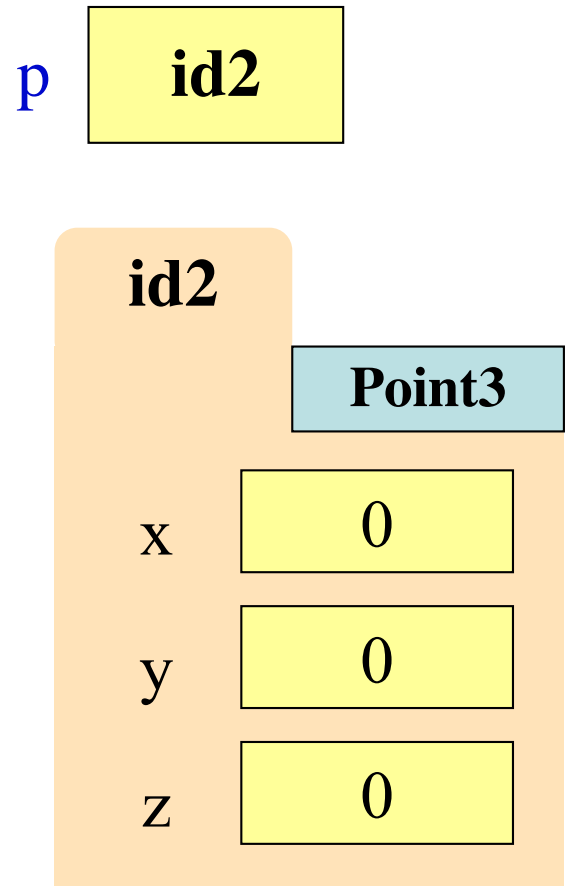
Need to import module that has Point class.

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

Constructor is function. Prefix w/ module name.

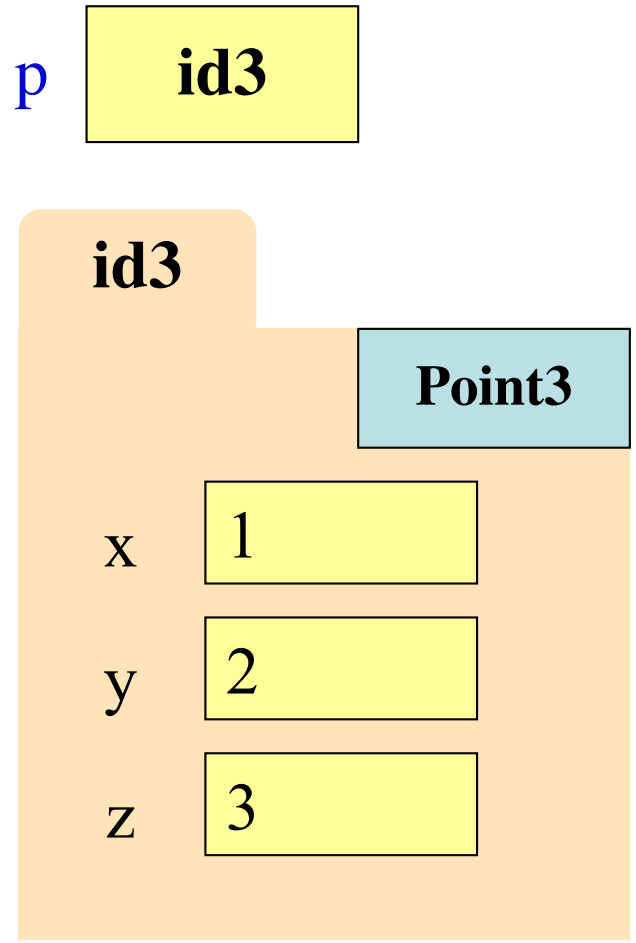
```
>>> id(p)
```

Shows the ***id*** of p



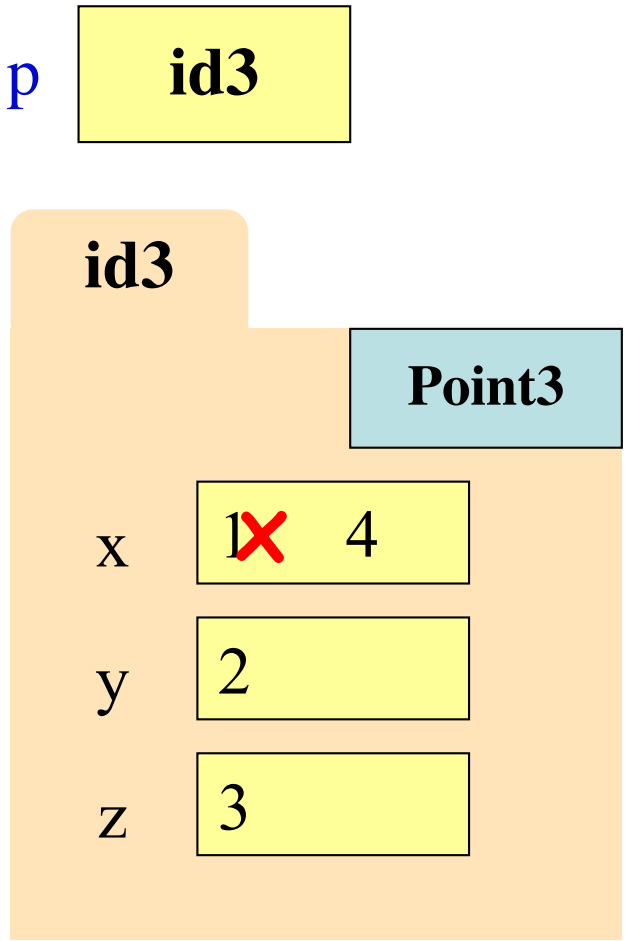
Accessing Attributes

- 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

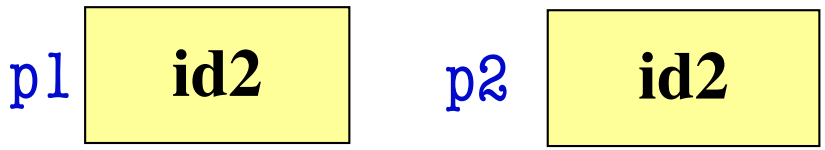


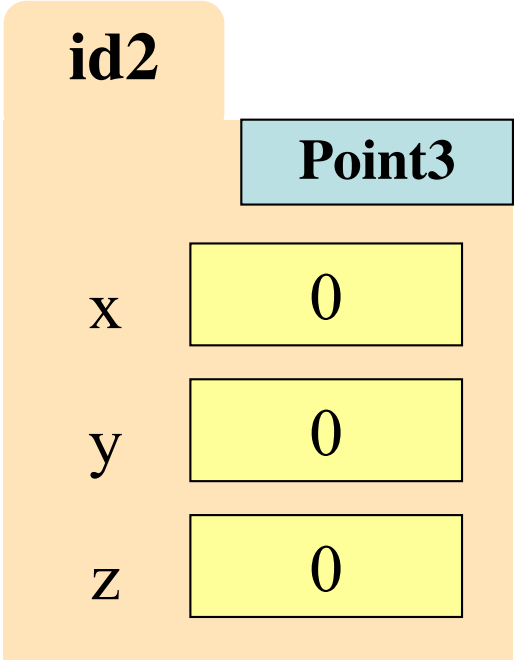
Object Variables

- Variable stores object ***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)

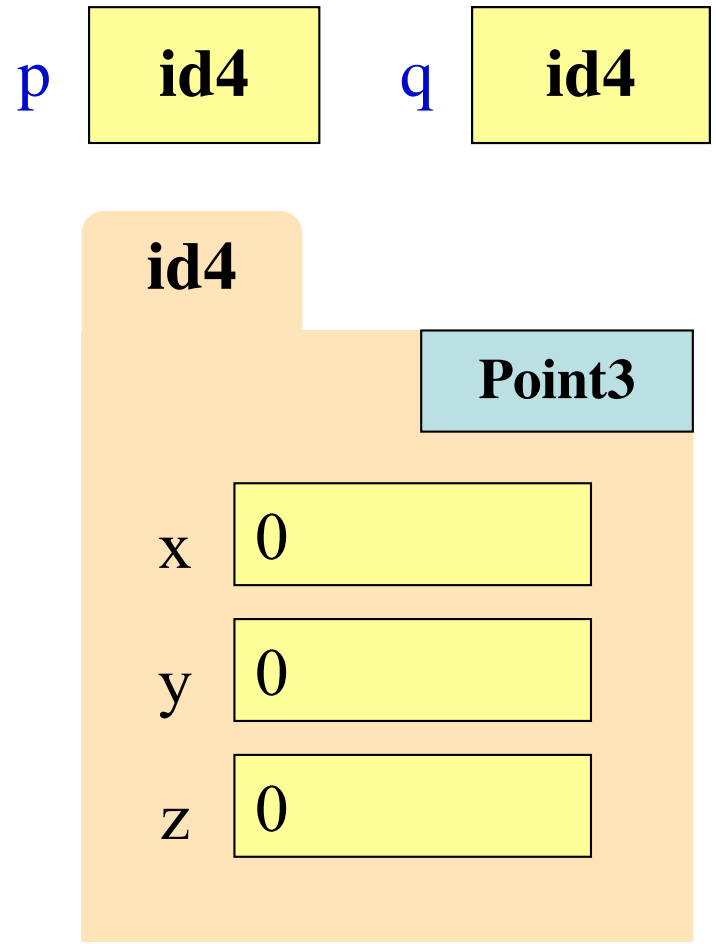
```
>>> p = shapes.Point3(0,0,0)
>>> q = p
```

- 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)

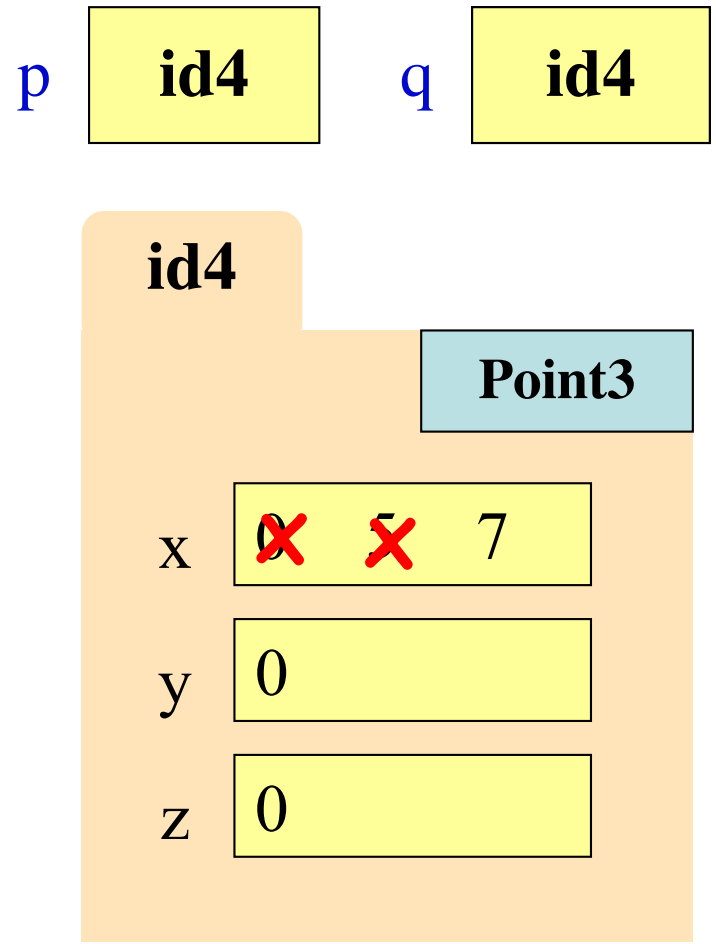
```
>>> p = shapes.Point3(0,0,0)
>>> q = p
```

- 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



Call Frames and Objects (1)

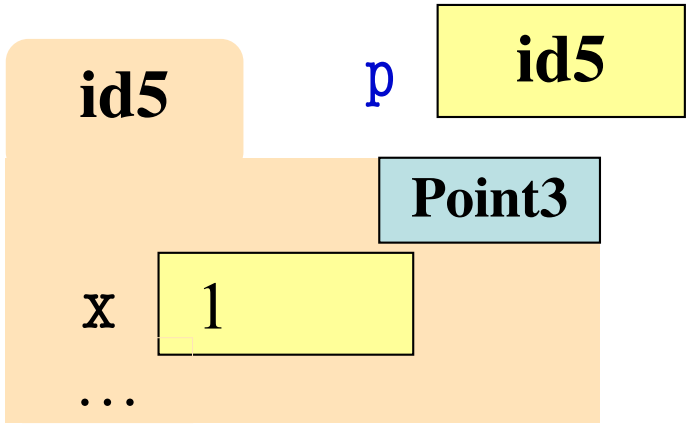
- Objects can be altered in a function call
  - Object variables hold ***id***s!
  - Folder can be accessed from global variable or parameter
- **Example**:

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

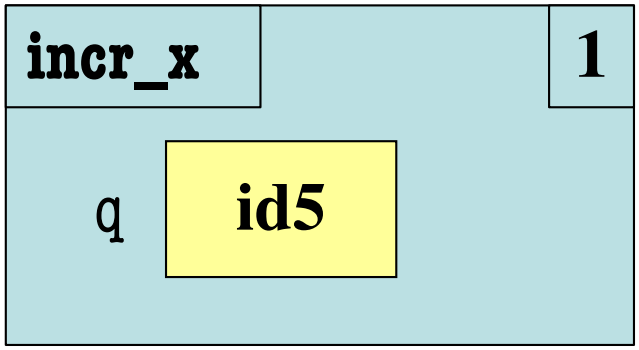
1

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

Global Memory



Call Frame



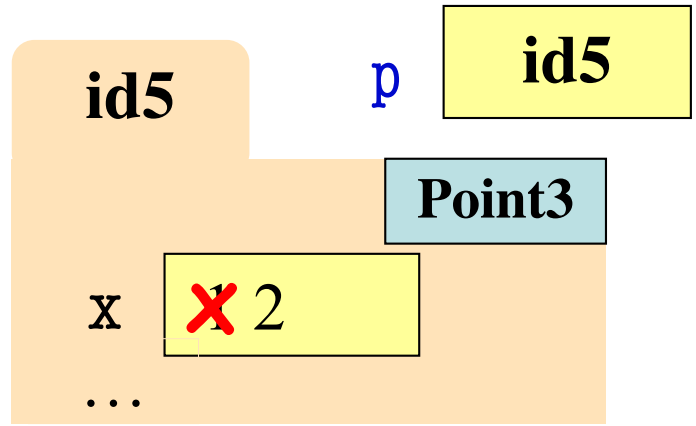
Call Frames and Objects (2)

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

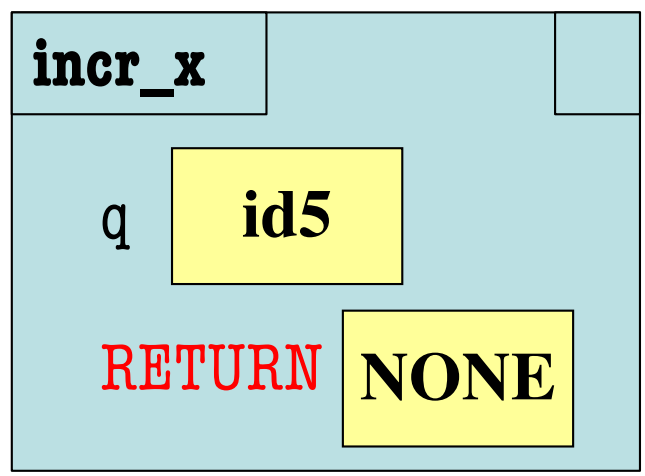
```
def incr_x(q):
    q.x = q.x + 1
```

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

Global Memory



Call Frame

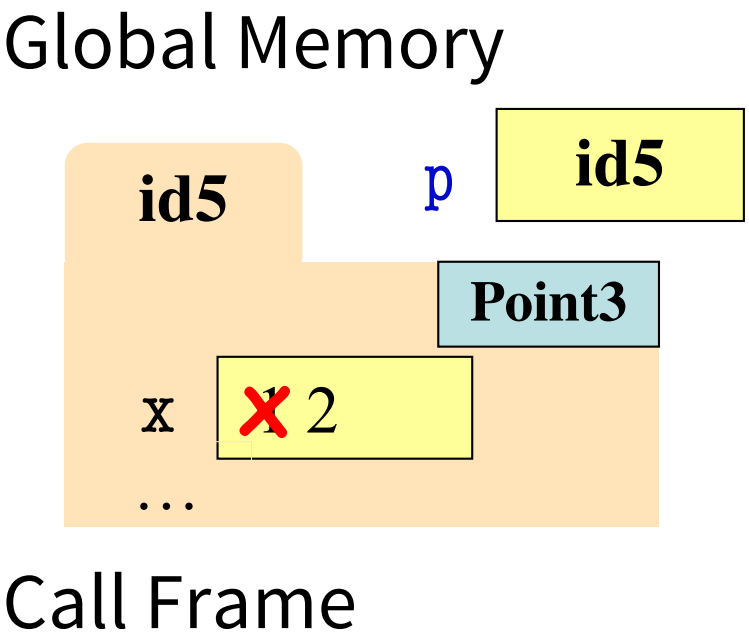


Call Frames and Objects (3)

- Objects can be altered in a function call
  - Object variables hold ***id***s!
  - 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)

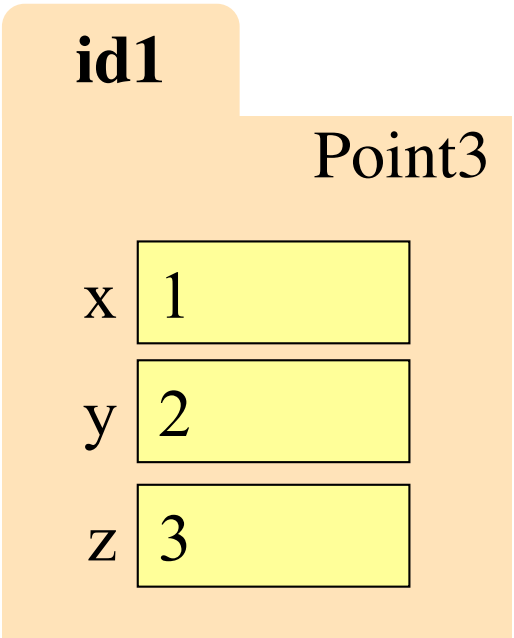
```
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
```

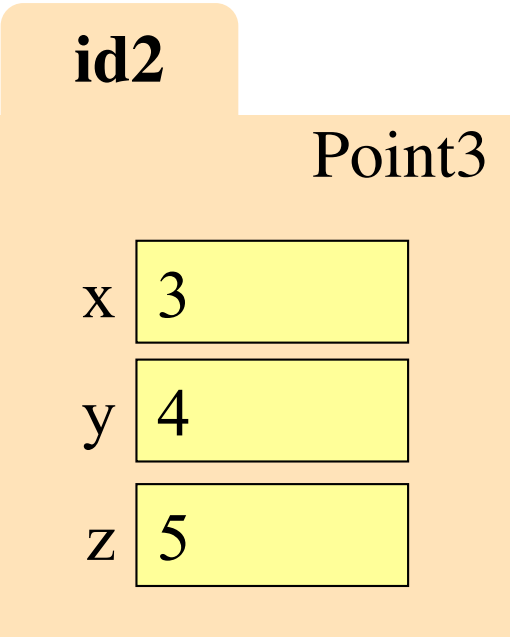
Draw everything that gets created.
How many folders get drawn?

How Many Folders (Solution)

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

Draw everything that gets created.
How many folders get drawn?

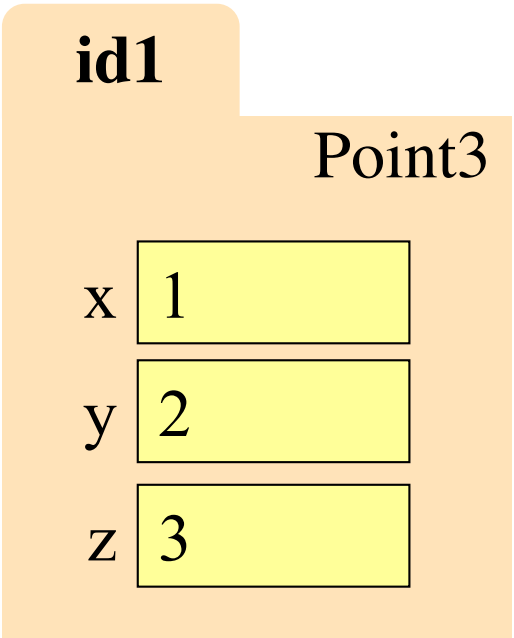


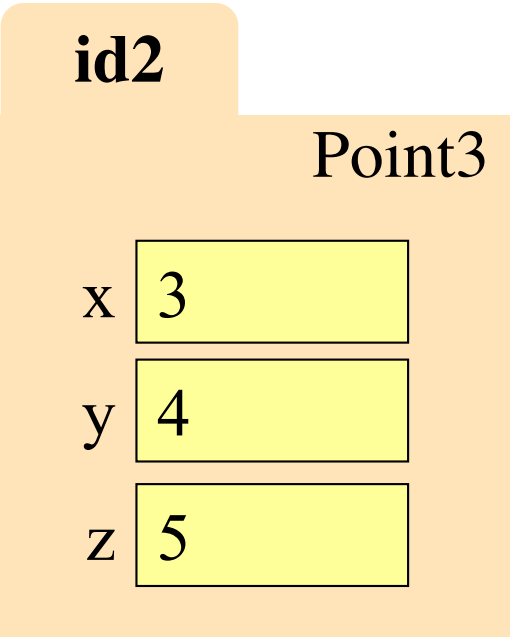


What Else? (Question)

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

Draw everything that gets created.
How many folders get drawn?
What else gets drawn?

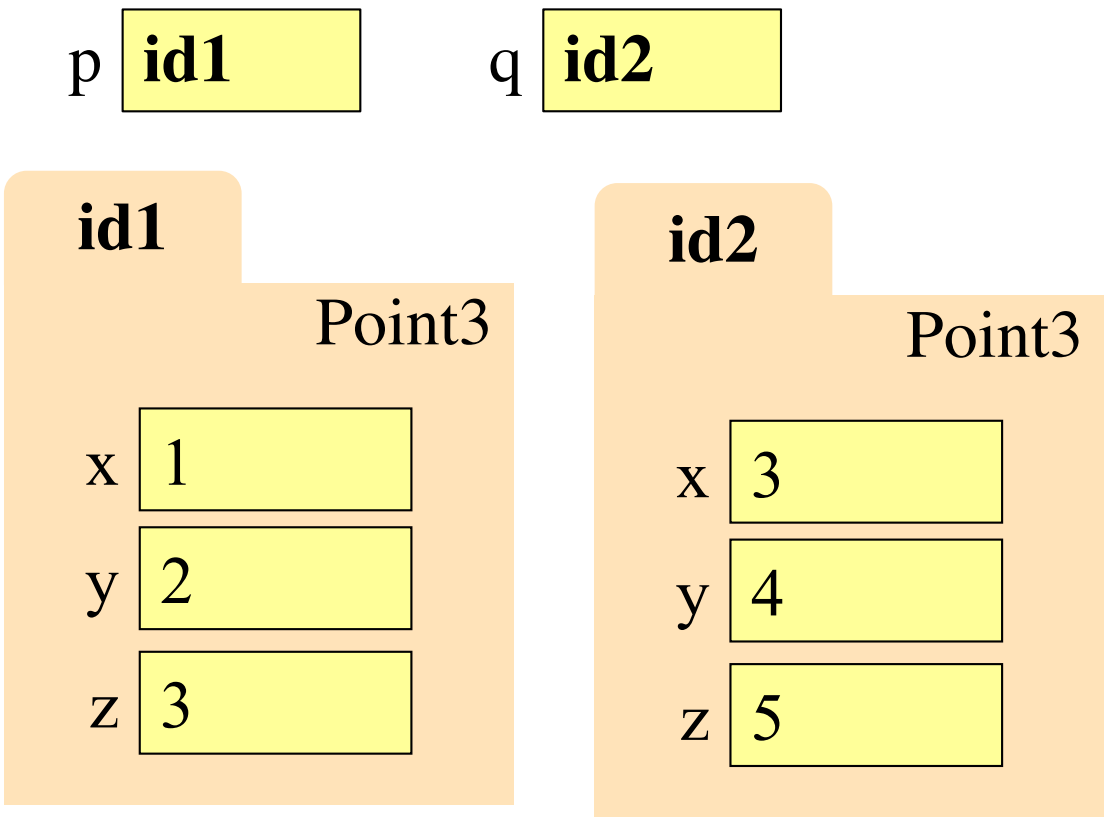




What Else? (Solution)

```
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?



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):
    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

Swap (Solution)

```
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 **CORRECT**
D: I don't know

Swap Explanation (1)

```
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
```

p **id1** q **id2**

id1 Point3

x 1
y 2
z 3

id2 Point3

x 3
y 4
z 5

swap_x 1

p **id1** q **id2**

Swap Explanation (2)

```
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
```

p **id1** q **id2**

id1 Point3

x	1
y	2
z	3

id2 Point3

x	3
y	4
z	5

swap_x 2

p **id1** q **id2**

t **1**

Swap Explanation (3)

```
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
```

p **id1** q **id2**

id1 Point3

x ~~1~~ 3
y 2
z 3

id2 Point3

x 3
y 4
z 5

swap_x 3

p **id1** q **id2**
t **1**

Swap Explanation (4)

```
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
```

p **id1** q **id2**

id1 Point3

x	~~1~~ 3
y	2
z	3

id2 Point3

x	~~3~~ 1
y	4
z	5

swap_x

p	**id1**	q	**id2**
t	**1**		
Return	**NONE**		

Swap Explanation (5)

```
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
```

p **id1** q **id2**

id1 Point3

x	~~1~~ 3
y	2
z	3

id2 Point3

x	~~3~~ 1
y	4
z	5

ERASE WHOLE FRAME

Global p (Question)

```
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
```

Before calling swap(p, q):

p **id1** q **id2**

What is in global p after calling swap?

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

Global p (Solution)

```
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
```

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

Global p Explanation (1)

```
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
```

p **id1** q **id2**

id1 Point

x 1
y 2
z 3

id2 Point

x 3
y 4
z 5

swap 1

p **id1** q **id2**

Global p Explanation (2)

```
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
```

p **id1** q **id2**

id1 Point

x 1
y 2
z 3

id2 Point

x 3
y 4
z 5

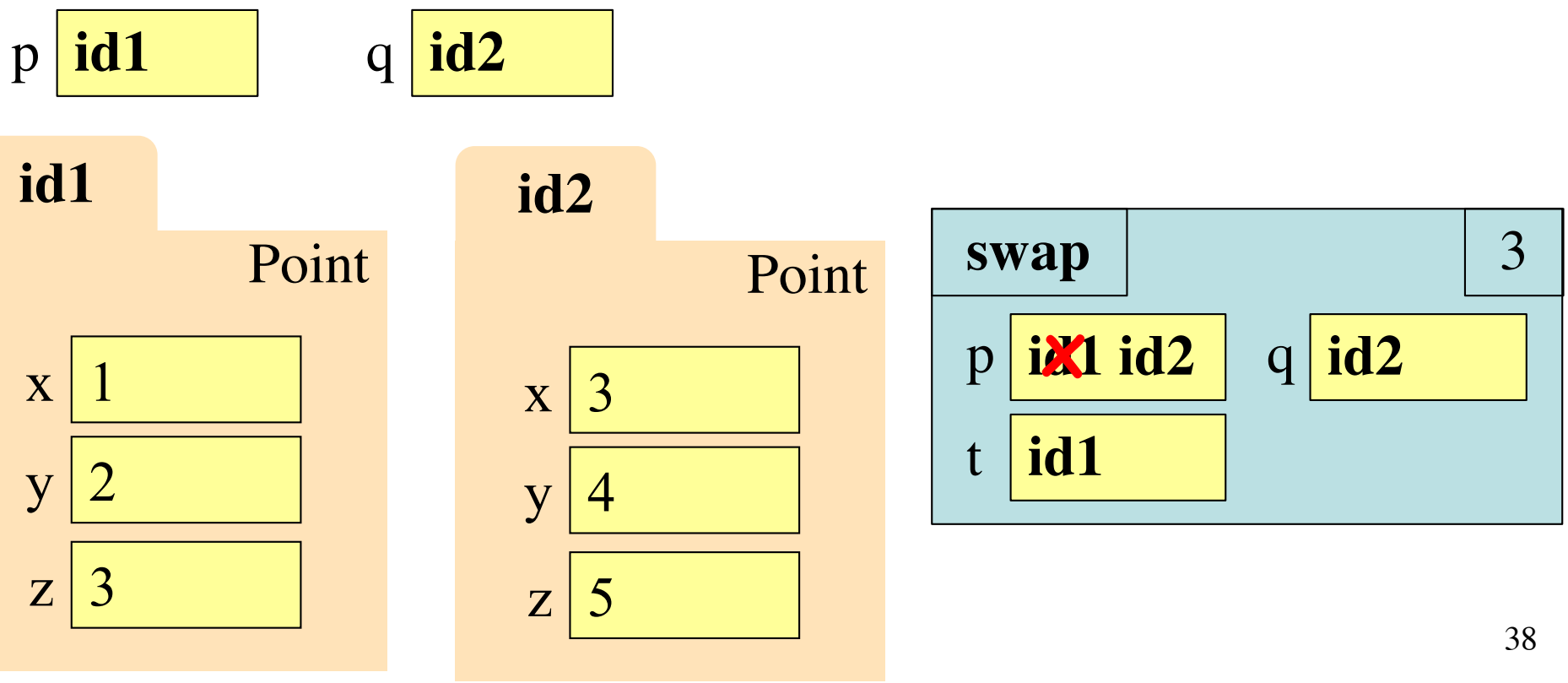
swap 2

p **id1** q **id2**
t **id1**

Global p Explanation (3)

```
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
```



Global p Explanation (4)

```
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
```

p **id1** q **id2**

id1

Point

x	1
y	2
z	3

id2

Point

x	3
y	4
z	5

swap

p ~~**id1**~~ **id2** q ~~**id2**~~ **id1**

t **id1**

RETURN **NONE**

Global p Explanation (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
```

p **id1** q **id2**

id1

Point

x	1
y	2
z	3

id2

Point

x	3
y	4
z	5

ERASE WHOLE FRAME

Global p Revisited (Question)

```
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
```

Before calling swap(p, q):

p **id1** q **id2**

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

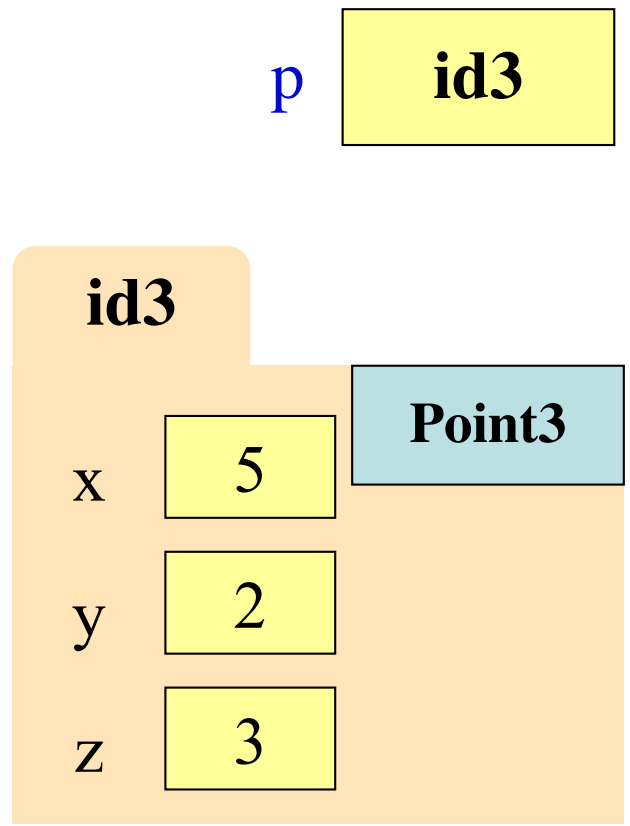
- **Method**: function tied to object
  - Method call looks like a function call preceded by a variable name:

    ⟨*variable*⟩.⟨*method*⟩(⟨*arguments*⟩)

Example:

```
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

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

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!