

Lecture 5

Objects

Announcements for this Lecture

Last Call

- Quiz: About the Course
- Take it by tomorrow
- Also remember survey



Assignment 1

- Assignment 1 is live
 - Posted on web page
 - Due Wednesday, Sep. 25th
 - Due in place of Lab 4
- Still not ready for it...
 - Testing is a major part
 - Will cover this on Tues
 - But can still get started

One-on-One Sessions

- Starting next week: **1/2-hour one-on-one sessions**
 - Bring computer and work with instructor, TA or consultant
 - Hands on, dedicated help with Lab 2 and/or Lab 3
 - To prepare for assignment, **not for help on assignment**
- **Limited availability: we cannot get to everyone**
 - **Students with experience or confidence should hold back**
- Sign up online in CMS: first come, first served
 - Choose assignment One-on-One
 - Pick a time that works for you; will add slots as possible
 - Can sign up starting at 1pm **TODAY**

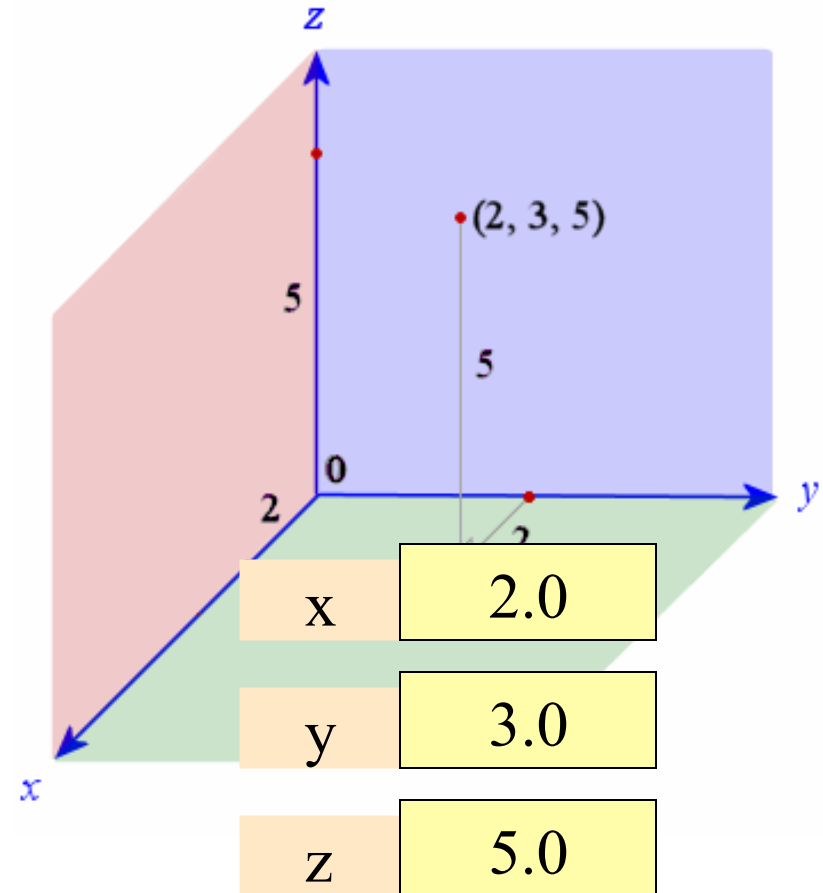
Type: Set of values and the operations on them

- Type **int**:
 - **Values**: integers
 - **Ops**: +, −, *, /, %, **
- Type **float**:
 - **Values**: real numbers
 - **Ops**: +, −, *, /, **
- Type **bool**:
 - **Values**: **True** and **False**
 - **Ops**: not, and, or
- Type **str**:
 - **Values**: string literals
 - Double quotes: "abc"
 - Single quotes: 'abc'
 - **Ops**: + (concatenation)

Are the the only types that exist?

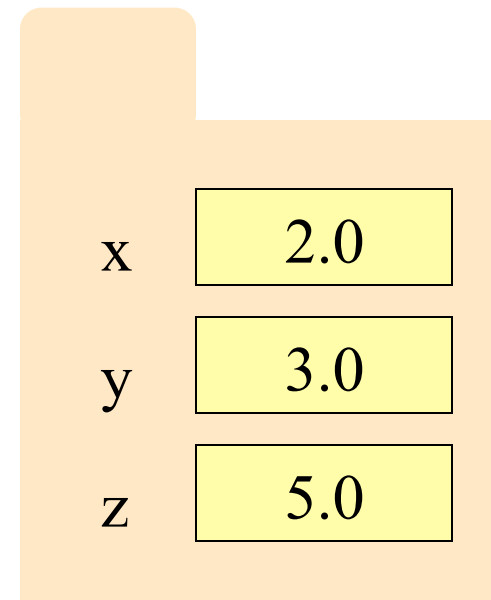
Type: Set of values and the operations on them

- 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?



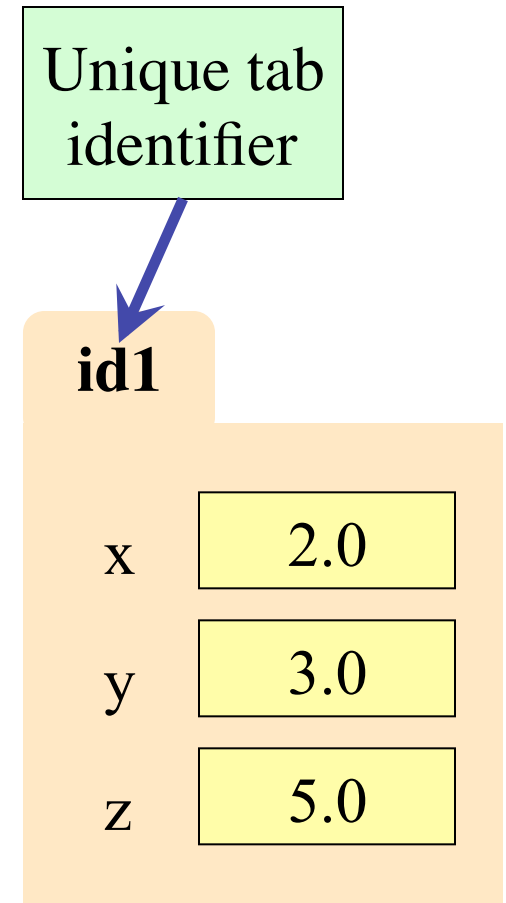
Type: Set of values and the operations on them

- 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**



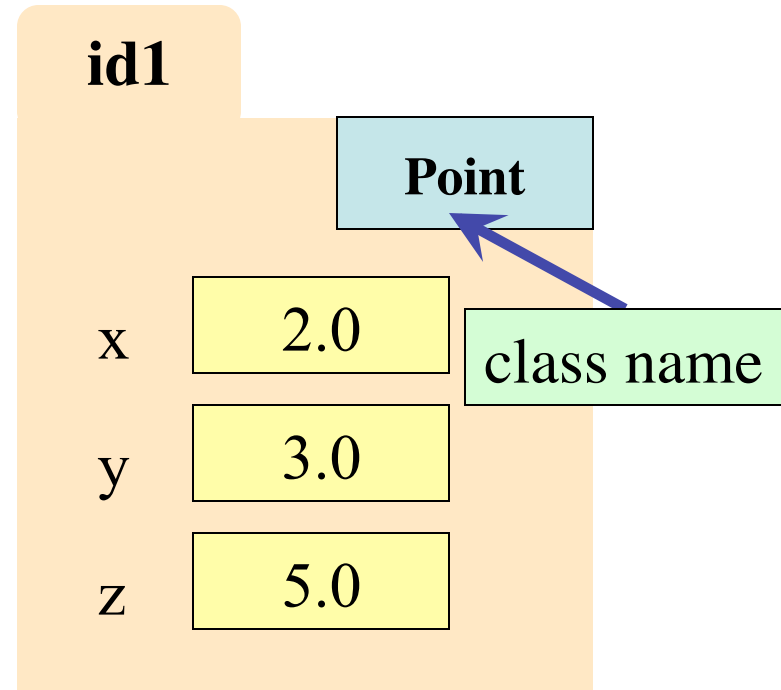
Objects: Organizing Data in Folders

- 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



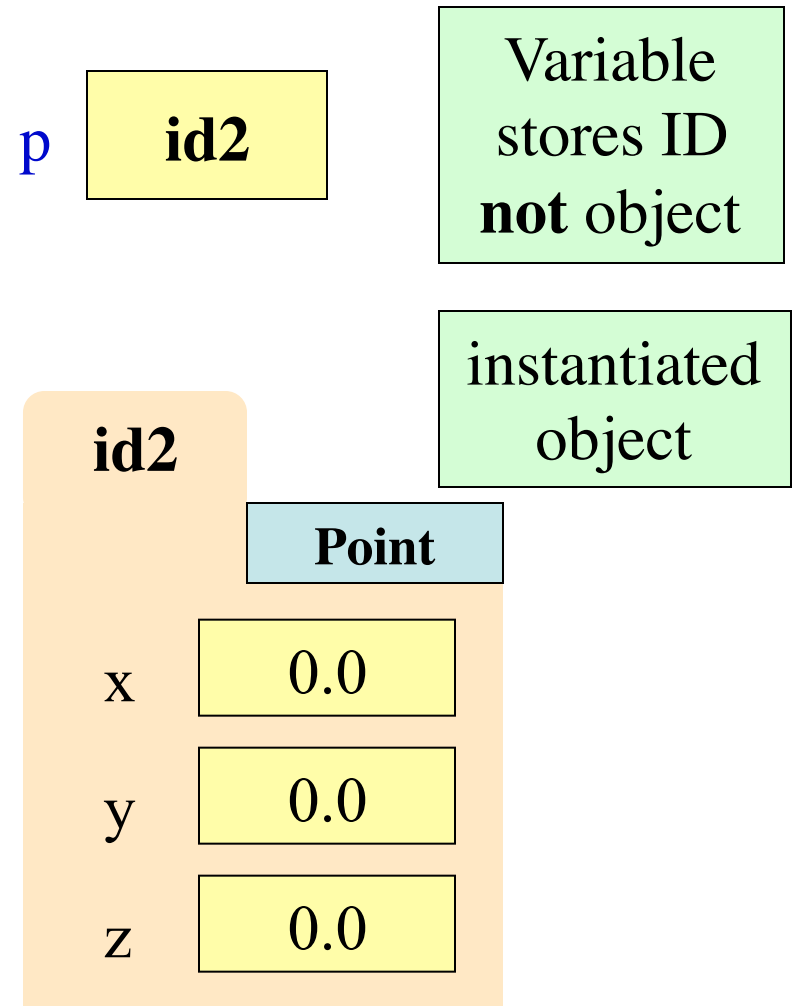
Classes: Types for Objects

- Values must have a type
 - An object is a **value**
 - Object type is a **class**
- **Modules** provide classes
 - Will show how later
- **Example:** tuple3d
 - Part of CornellExtensions
 - Just need to import it
 - Classes: Point, Vector



Constructor: Function to make Objects

- How do we create objects?
 - Other types have **literals**
 - **Example:** 1, "abc", true
 - No such thing for objects
- **Constructor Function:**
 - Same name as the class
 - **Example:** Point(0,0,0)
 - Makes an object (manila folder)
 - Returns folder ID as value
- **Example:** p = Point(0, 0, 0)
 - Creates a Point object
 - Stores object's ID in p



Constructors and Modules

```
>>> import tuple3d
```

Need to import module that has Point class.

```
>>> p = tuple3d.Point(0,0,0)
```

Constructor is function. Prefix w/ module name.

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

Shows the ID of p.

p

id2

Actually a big number

id2

Point

x

0.0

y

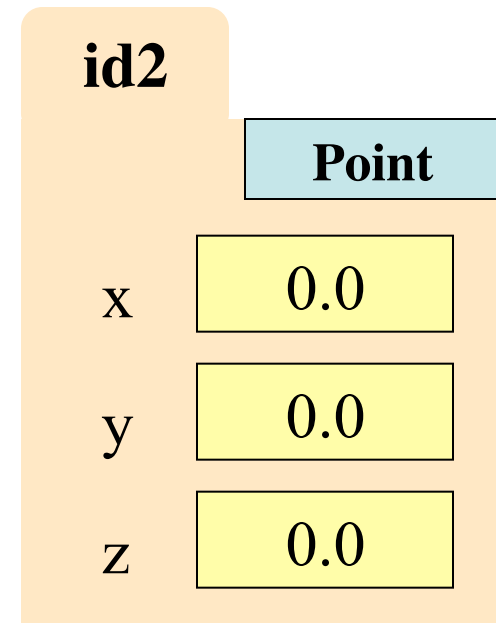
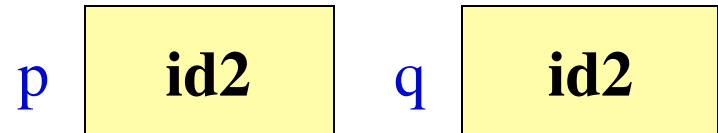
0.0

z

0.0

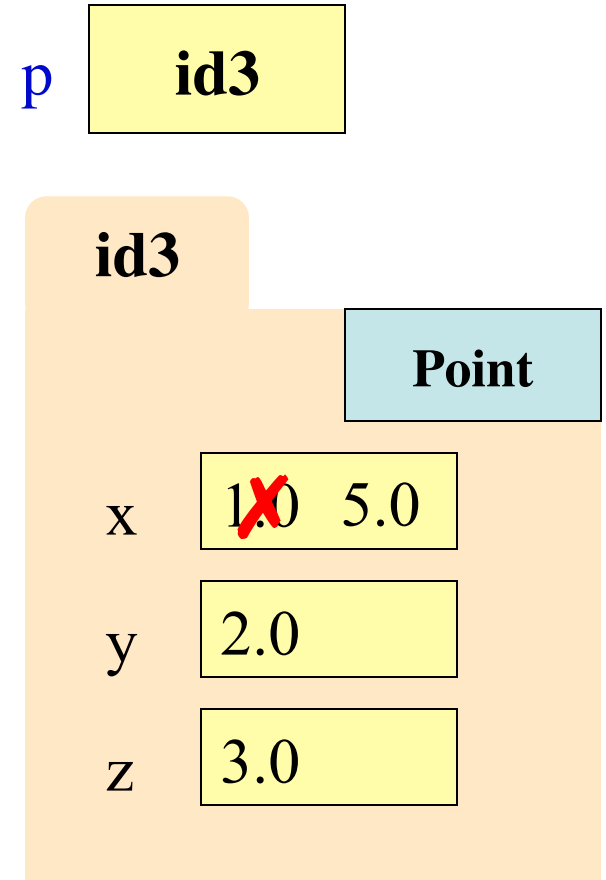
Object Variables

- Variable stores object name
 - **Reference** to the object
 - Reason for folder analogy
- Assignment uses object name
 - **Example:** $q = p$
 - Takes name from p
 - Puts the name in q
 - Does not make new folder!
- This is the cause of many mistakes in this course



Objects and Attributes

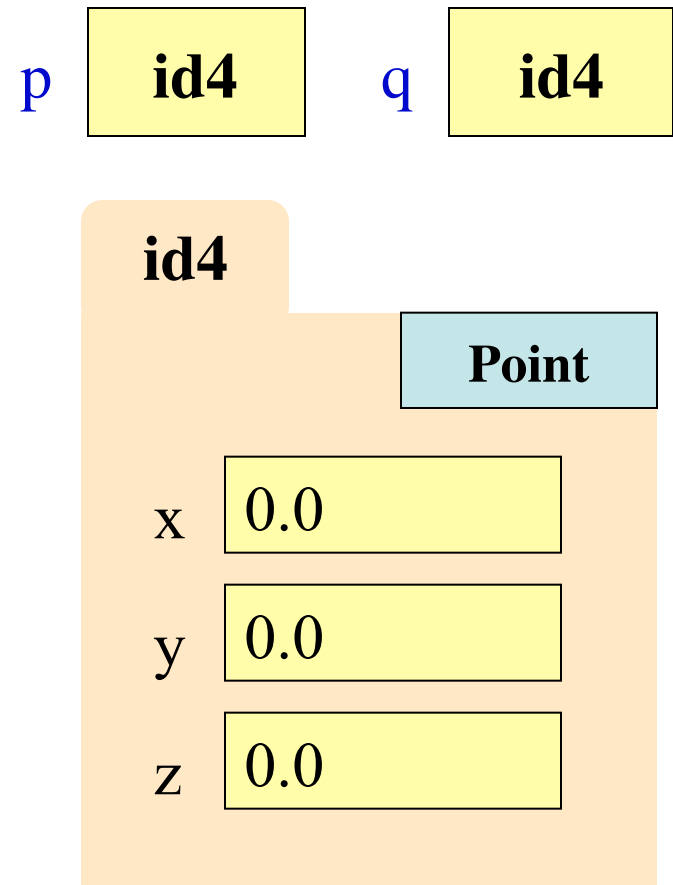
- Attributes are variables that live inside of objects
 - Can **use** in expressions
 - Can **assign** values to them
- **Access:** `<variable>.<attr>`
 - **Example:** `p.x`
 - Look like module variables
- Putting it all together
 - `p = tuple3d.Point(1,2,3)`
 - `p.x = p.y + p.z`



Exercise: Attribute Assignment

- Recall, q gets name in p
 - >>> p = tuple3d.Point(0,0,0)
 - >>> q = p
- Execute the assignments:
 - >>> p.x = 5.6
 - >>> q.x = 7.4
- What is value of p.x?

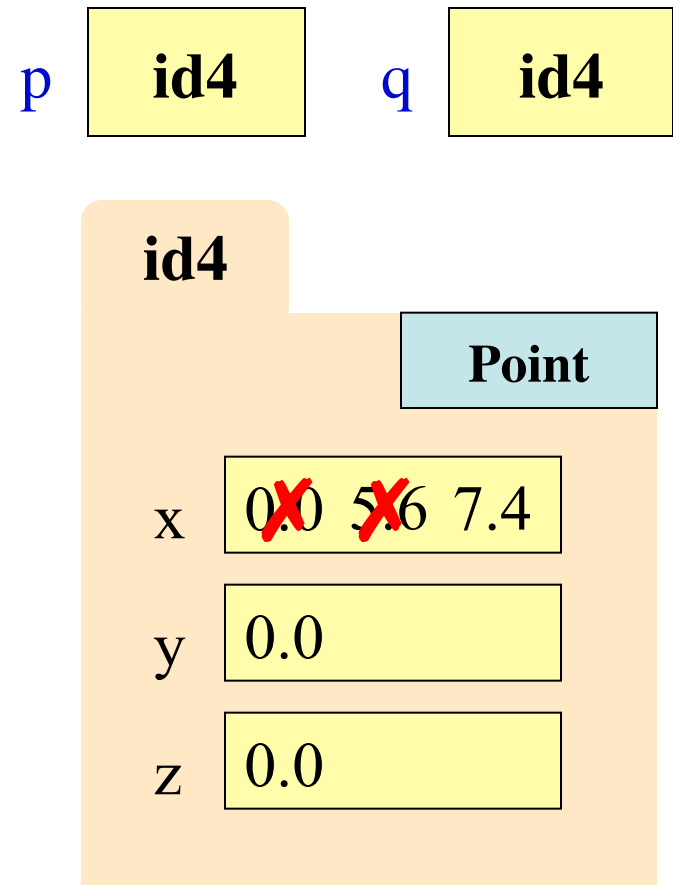
- A: 5.6
- B: 7.4
- C: id4
- D: I don't know



Exercise: Attribute Assignment

- Recall, q gets name in p
 - `>>> p = tuple3d.Point(0,0,0)`
 - `>>> q = p`
- Execute the assignments:
 - `>>> p.x = 5.6`
 - `>>> q.x = 7.4`
- What is value of `p.x`?

A: 5.6
B: 7.4 **CORRECT**
C: id4
D: I don't know



Call Frames and Objects

- Mutable objects can be altered in a function call
 - Object vars hold names!
 - Folder accessed by both global var & parameter

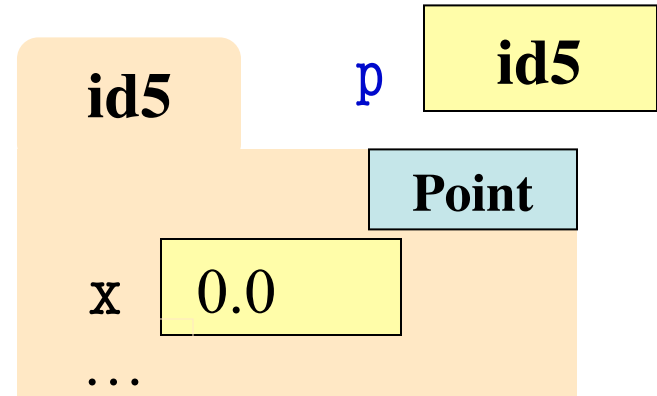
- **Example:**

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

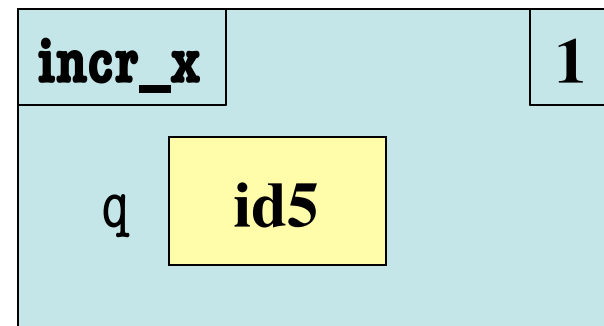
```
>>> p = Point(0,0,0)
```

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

Global **STUFF**



Call Frame



Call Frames and Objects

- Mutable objects can be altered in a function call
 - Object vars hold names!
 - Folder accessed by both global var & parameter

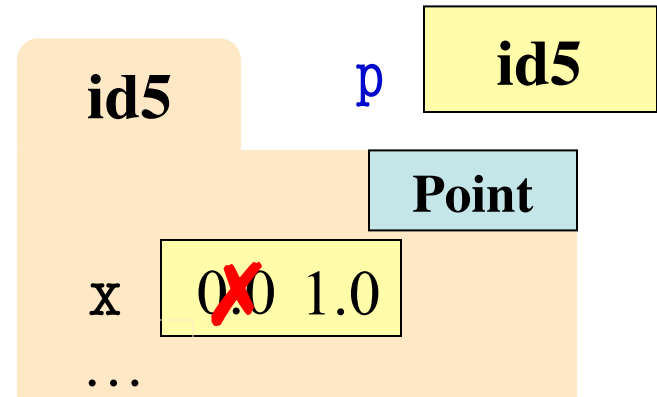
- **Example:**

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

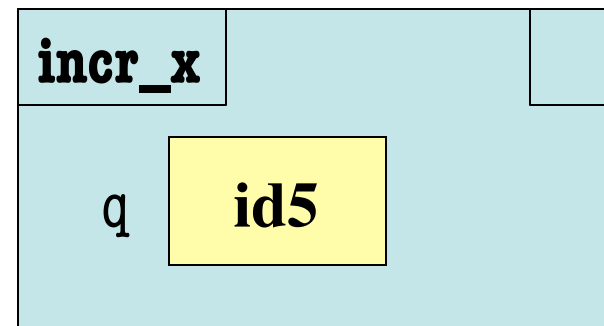
```
>>> p = Point()
```

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

Global STUFF



Call Frame



Call Frames and Objects

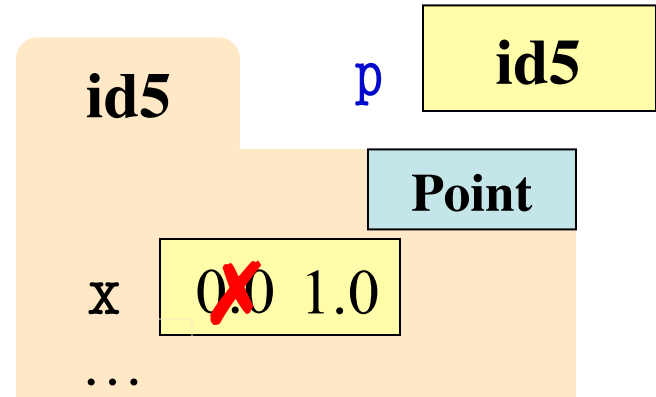
- Mutable objects can be altered in a function call
 - Object vars hold names!
 - Folder accessed by both global var & parameter
- **Example:**

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

```
>>> p = Point()
```

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

Global **STUFF**



Call Frame

Methods: Functions Tied to Objects

- **Method:** function tied to object

- Method call looks like a function call preceded by a variable name:

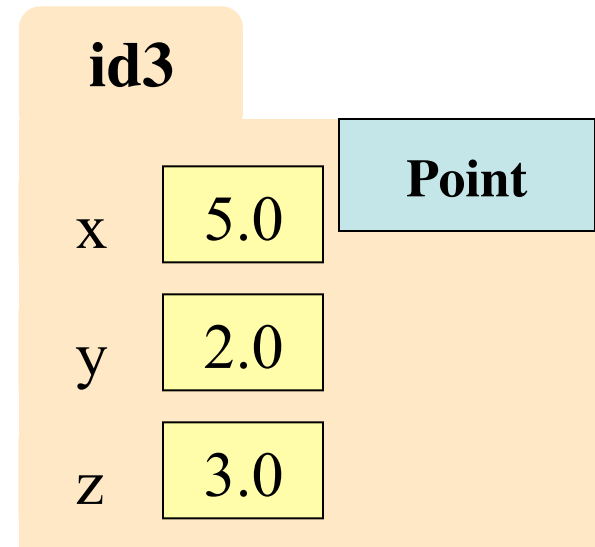
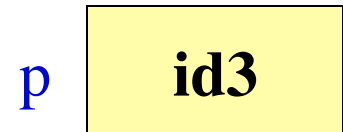
<variable>.<method>(<arguments>)

- **Example:** `p.distanceTo(q)`
- **Example:** `p.abs()` # makes $x, y, z \geq 0$

- Just like we saw for strings

- `s = 'abracadabra'`
- `s.index('a')`

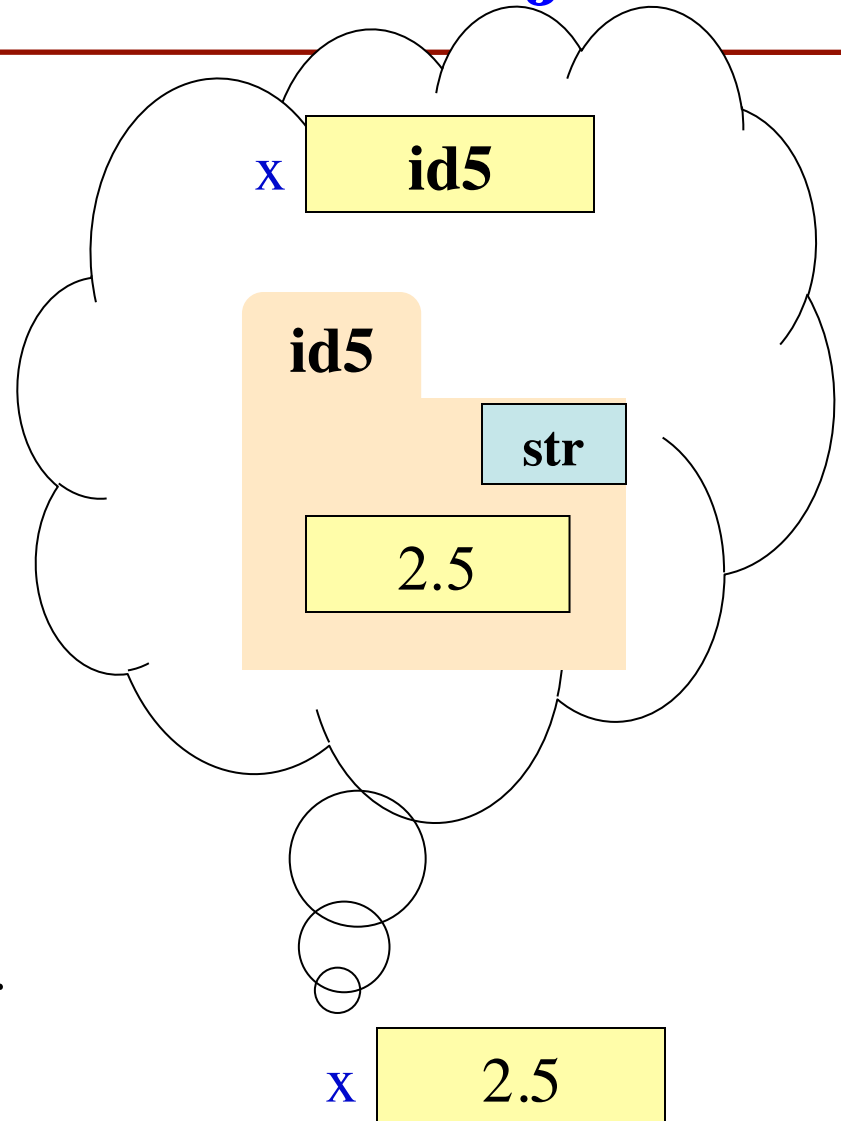
- Are strings objects?



Surprise: All Values are in Objects!

- Including basic values
 - int, float, bool, str
- **Example:**

```
>>> x = 2.5
>>> id(x)
```
- But they are *immutable*
 - Contents cannot change
 - Distinction between *value* and *identity* is immaterial
 - So we can ignore the folder

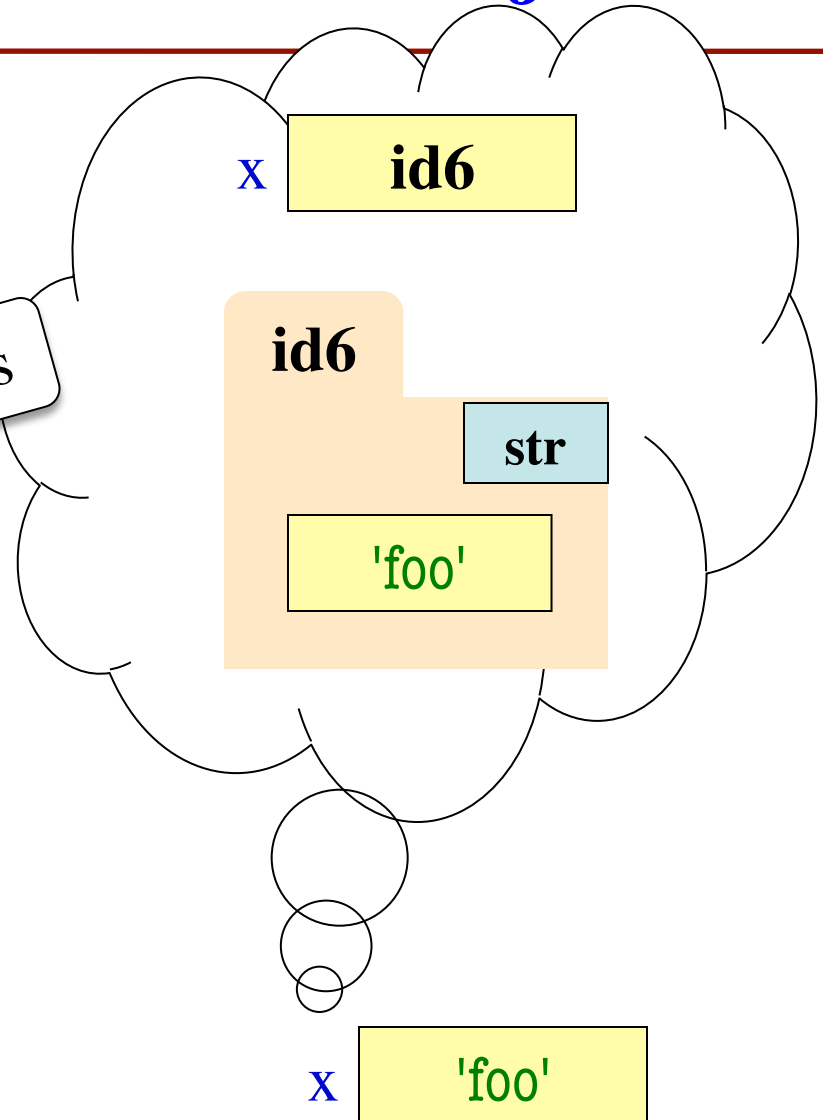


Surprise: All Values are in Objects!

- Including basic values
 - int, float, bool, str
- **Example:**

```
>>> x = 'foo'
>>> id(x)
```
- But they are *immutable*
 - No string method can alter the contents of a string
 - `x.replace('o','y')` evaluates to `'fyy'` but `x` is still `'foo'`
 - So we can ignore the folder

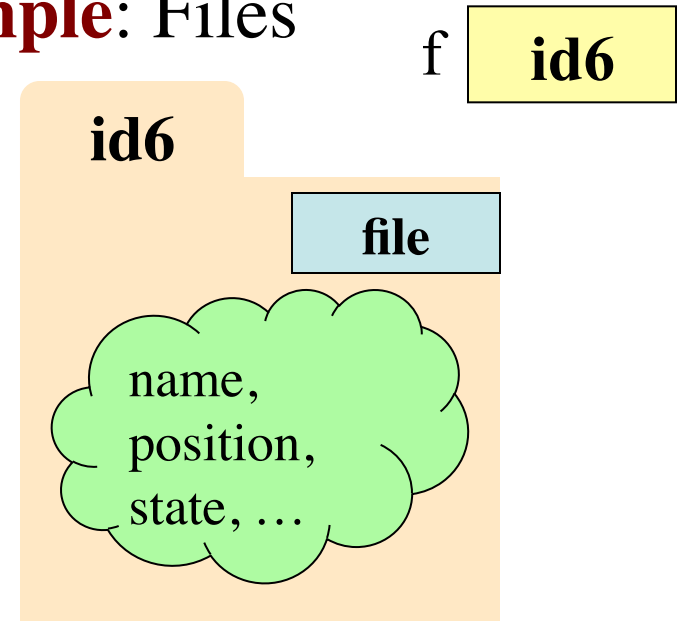
includes strings



Class Objects

- Use name **class object** to distinguish from other values
 - Not int, float, bool, str
- Class objects are **mutable**
 - You can change them
 - Methods can have effects besides their return value
- **Example:**
 - `p = Point(3,-3,0)`
 - `p.clamp(-1,1)`

Example: Files



```
f = open('jabber.txt')  
s = f.read()  
f.close()
```

Opens a file on your disk; returns a **file object** you can read

Base Types vs. Classes

Base Types

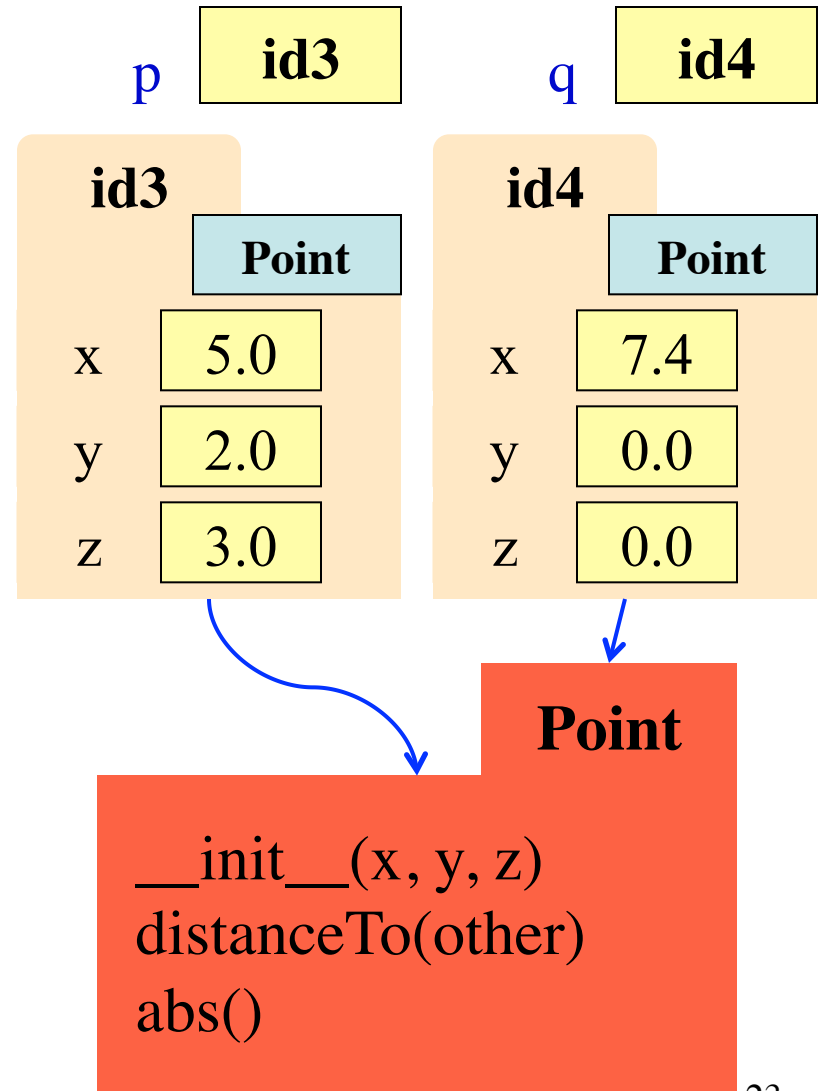
- Built-into Python
- Refer to instances as *values*
- Instantiate with *literals*
- Are all immutable
- Can ignore the folders

Classes

- Provided by modules
- Refer to instances as *objects*
- Instantiate w/ *constructors*
- Can alter attributes
- Must represent with folders

Aside: Name Resolution

- `<object>.<name>` means
 - Go the folder for *object*
 - Look for attr/method *name*
 - If missing, check *class folder*
- Class folder is a **shared folder**
 - Only one for the whole class
 - Shared by all objects of class
 - Stores common features
 - Typically where methods are
- Do not worry about this yet



Where To From Here?

- Right now, just try to understand **objects**
 - All Python programs use objects
 - Most small programs use objects of classes that are part of the Python Library
- OO Programming is about **creating classes**
 - Eventually you will make your own classes
 - Classes are the primary tool for organizing more complex Python programs
 - But we need to learn other basics first