CS1110

Lecture 5: Objects

Announcements

Grades for Lab 2 should all be posted in CMS. Please verify that you have a 1 if you checked off the lab. Let course staff know if your grade is missing!

Read Piazza about the surprise wrinkle in Lab 2 Q4.

Install troubles? Post on Piazza! Including on Linux —install procedures vary but are usually simple.

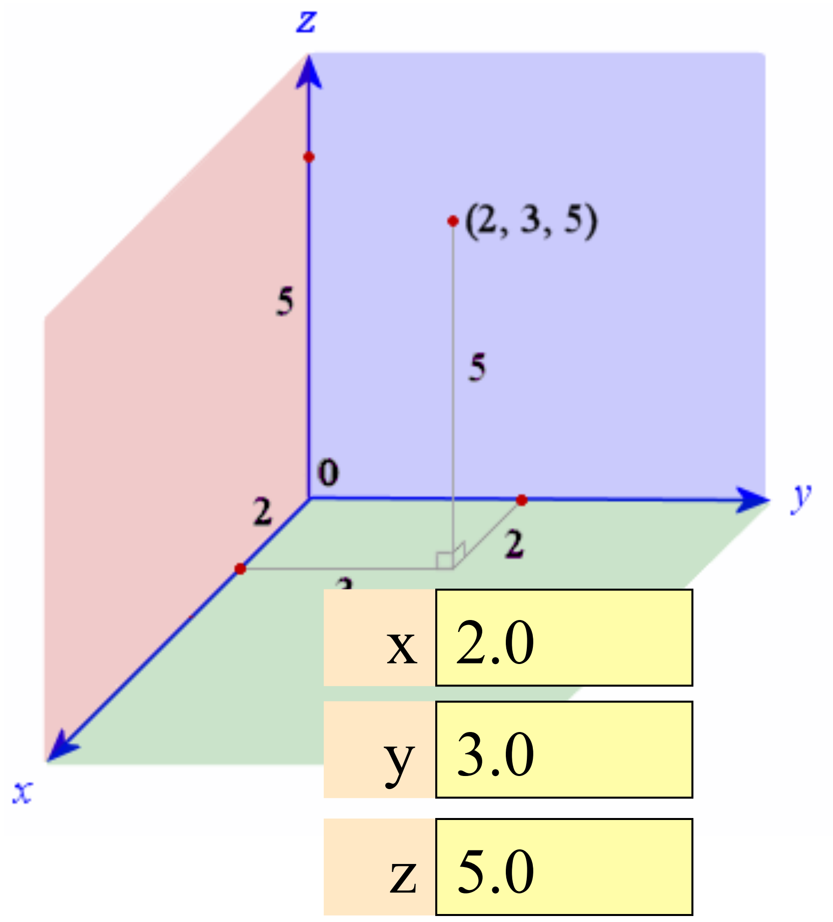
Reading for next time:

3.7–3.13 on functions and function calls

Slides by D. Gries, L. Lee, S. Marschner, W. White

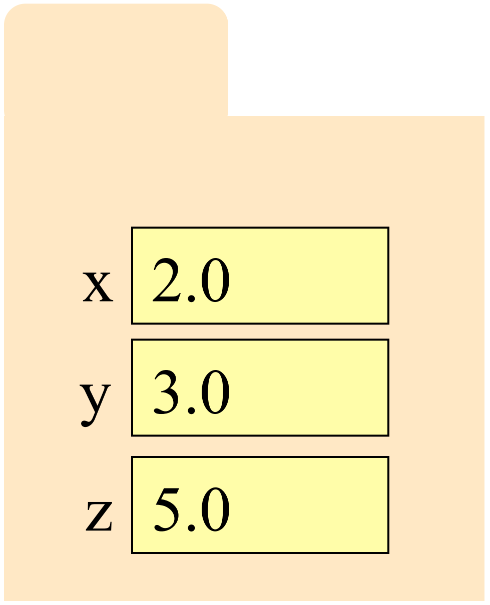
Example: Points in 3D space

- Want a point in 3D space
  - We need three variables
  - *x*, *y*, *z* coordinates
- What if we have many 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?



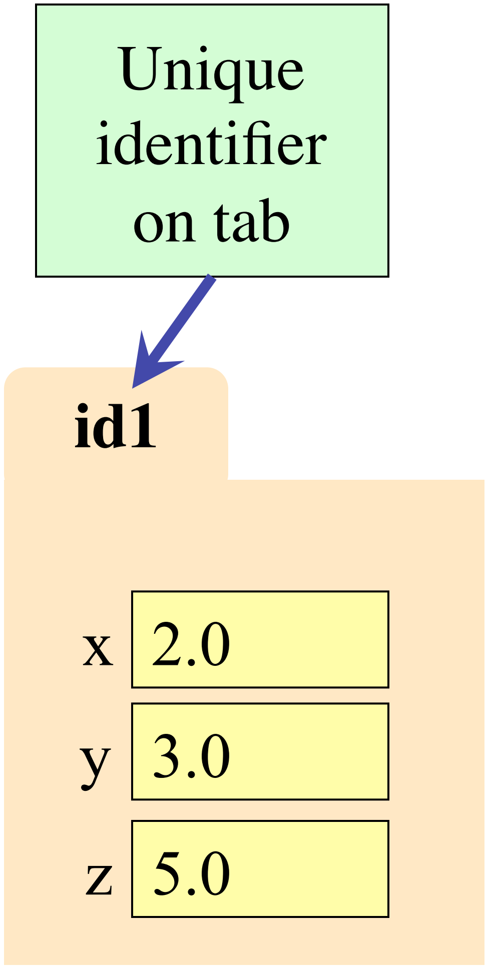
Example: Points in 3D space

- Want a point in 3D space
  - We need three variables
  - *x*, *y*, *z* coordinates
- What if we have many 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?



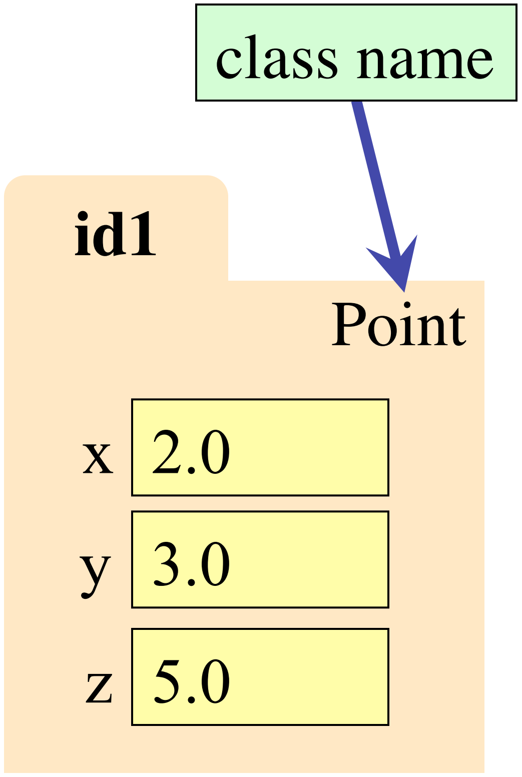
Objects: Organizing Data in Folders

- An object is like a **manila folder**
- It contains variables
  - These variables are **attributes**
  - Their values can change
- It has an **ID** that identifies it
  - Unique number assigned by Python (just like a NetID for a Cornellian)
  - Does not ever change
  - Has no meaning—only identifies



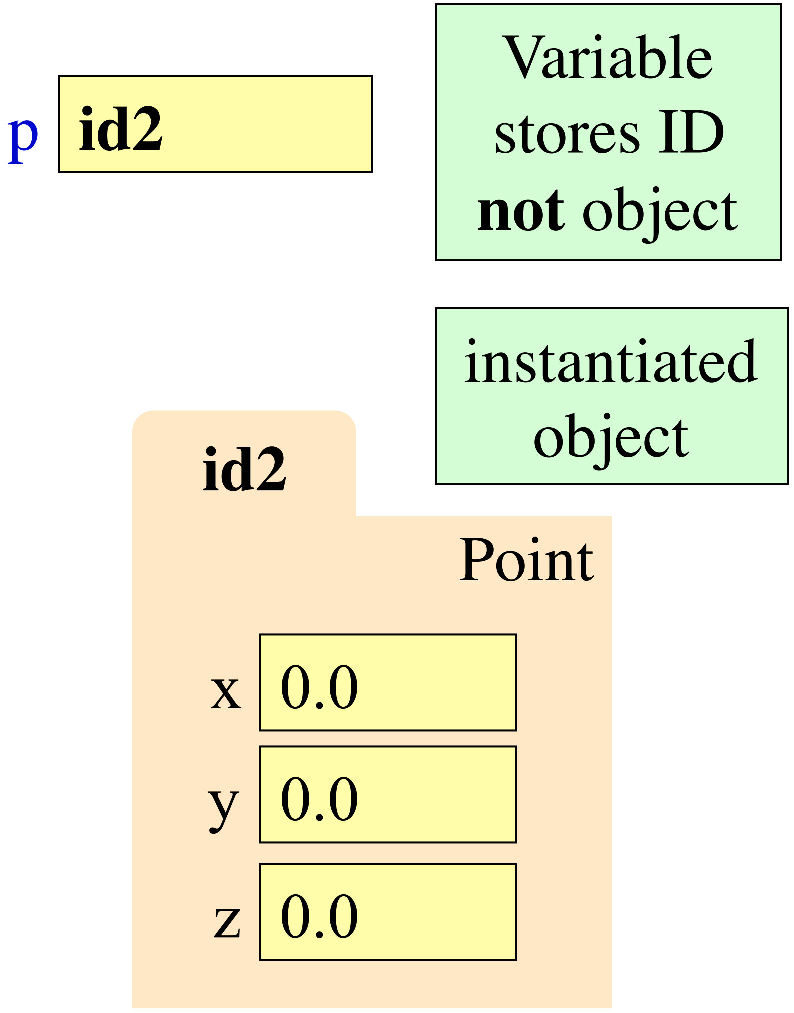
Classes: Types for Objects

- Everything needs a type
  - An object's type is a **class**
- Modules provide classes
  - **Example**: `point.py`
  - Import to use `Point`
- We'll learn how to define classes later
  - Do not try to understand the contents of `point.py`
  - Lots more to learn first



Constructor: Function to Make Objects

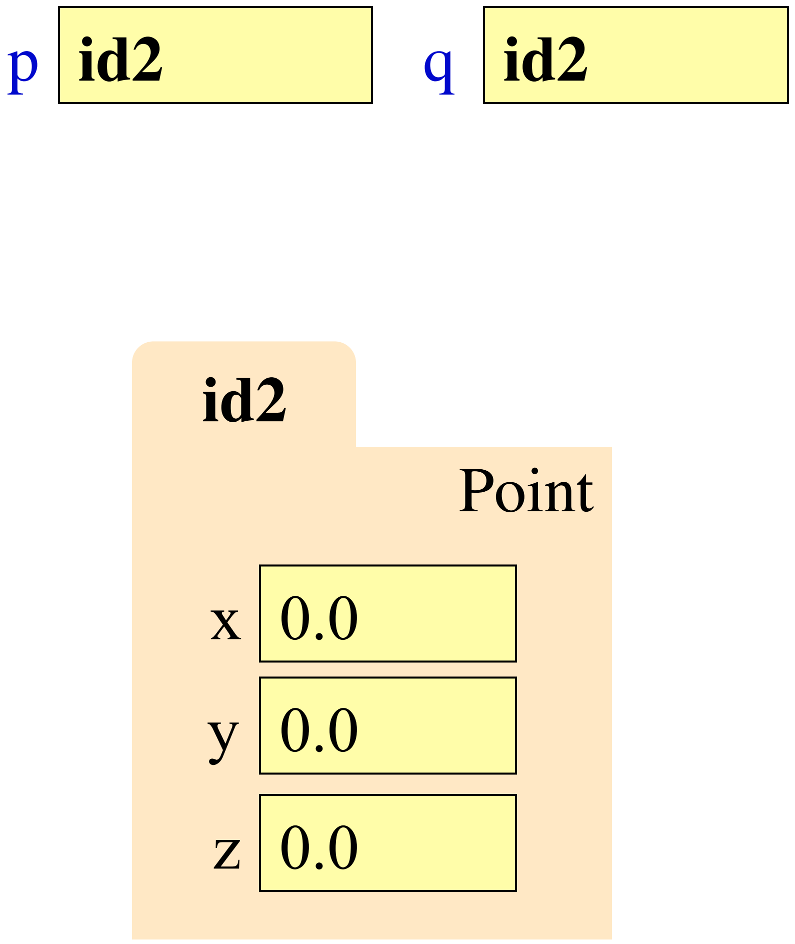
- How do we create objects?
  - Other types have *literals*
  - **Example**: `1`, `"abc"`, `True`
- **Constructor Function**:
  - Same name as the class
  - Example: `Point(0, 0, 0)`
  - Makes an object (manila folder)
  - Returns folder ID as its value
- Example: `p = Point(0, 0, 0)`
  - Creates a Point object
  - Stores object's ID in p



Referencing Objects With Variables

- Variable stores object ID
  - **Reference** to the object
  - Reason for folder analogy
- Assignment uses object ID
  - Example: `q = p`
  - Takes ID from p
  - Puts the ID in q
  - **Does not** make new folder!
- Use `id()` to see folder IDs
  - `id(p)` and `id(q)` evaluate to **id2**

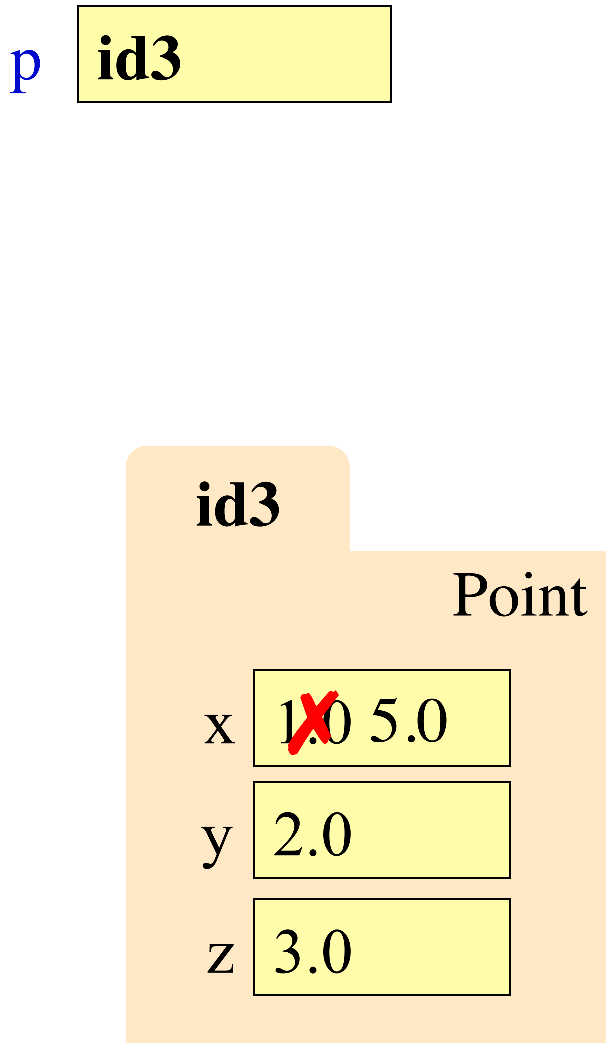
Actually some big number



Objects and Attributes

- Attributes are **variables** that live in objects
  - Can **use** in expressions
  - Can **assign** values to them
- Access: ⟨*variable*⟩.⟨*attribute*⟩
  - Example: `p.x`
  - Same syntax as accessing a variable in a module: `math.pi`
- Putting it all together

```
p = Point(1, 2, 3)
p.x = p.y + p.z
```



Exercise: Attribute Assignment

- Create point; name into q and p

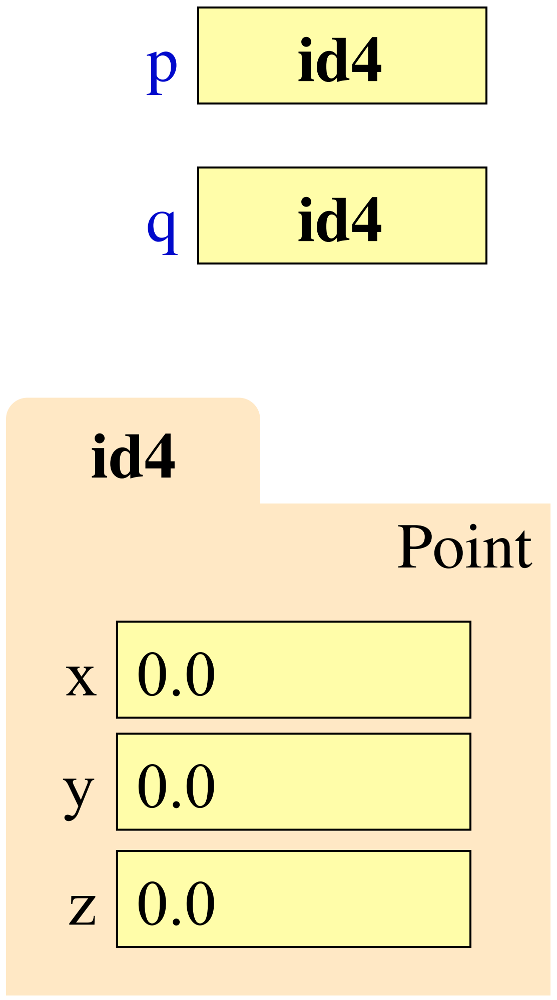
```
p = Point(0,0,0)
q = p
```

- 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

- Create point; name into q and p

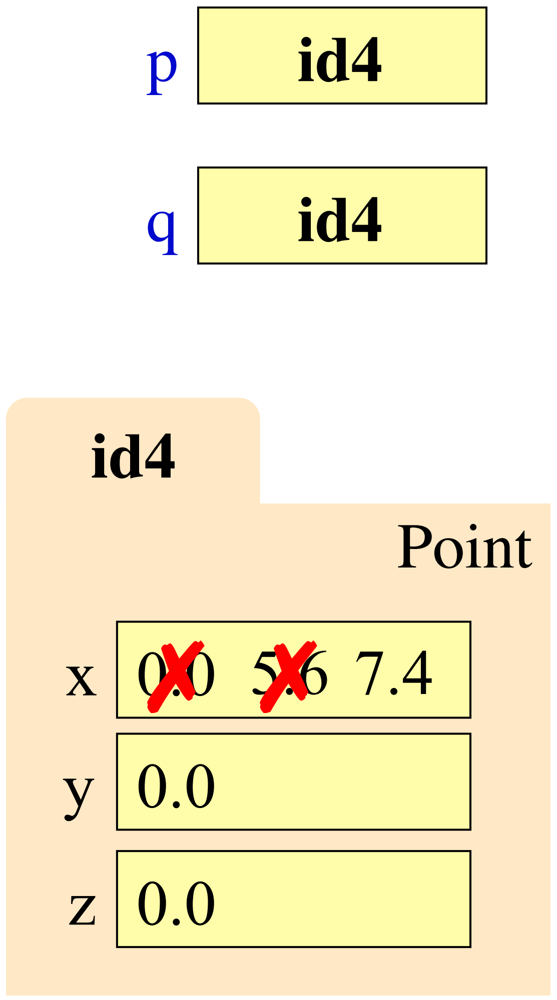
```
p = Point(0,0,0)
q = p
```

- Execute the assignments:

```
p.x = 5.6
q.x = 7.4
```

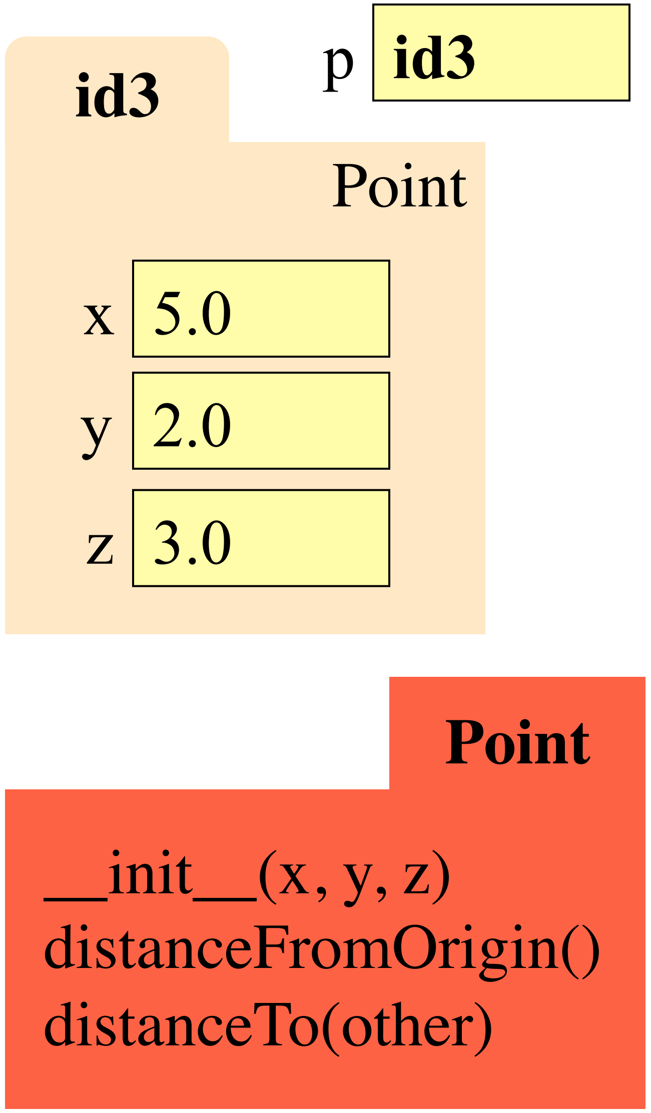
- What is value of `p.x`?

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



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.distanceFromOrigin()`
  - Example: `p.distanceTo(q)`
- Name resolution
  - ⟨*object*⟩.⟨*name*⟩ means "go to *object* and look for something called *name*."
  - Python looks first in the object's folder, then in the object's class



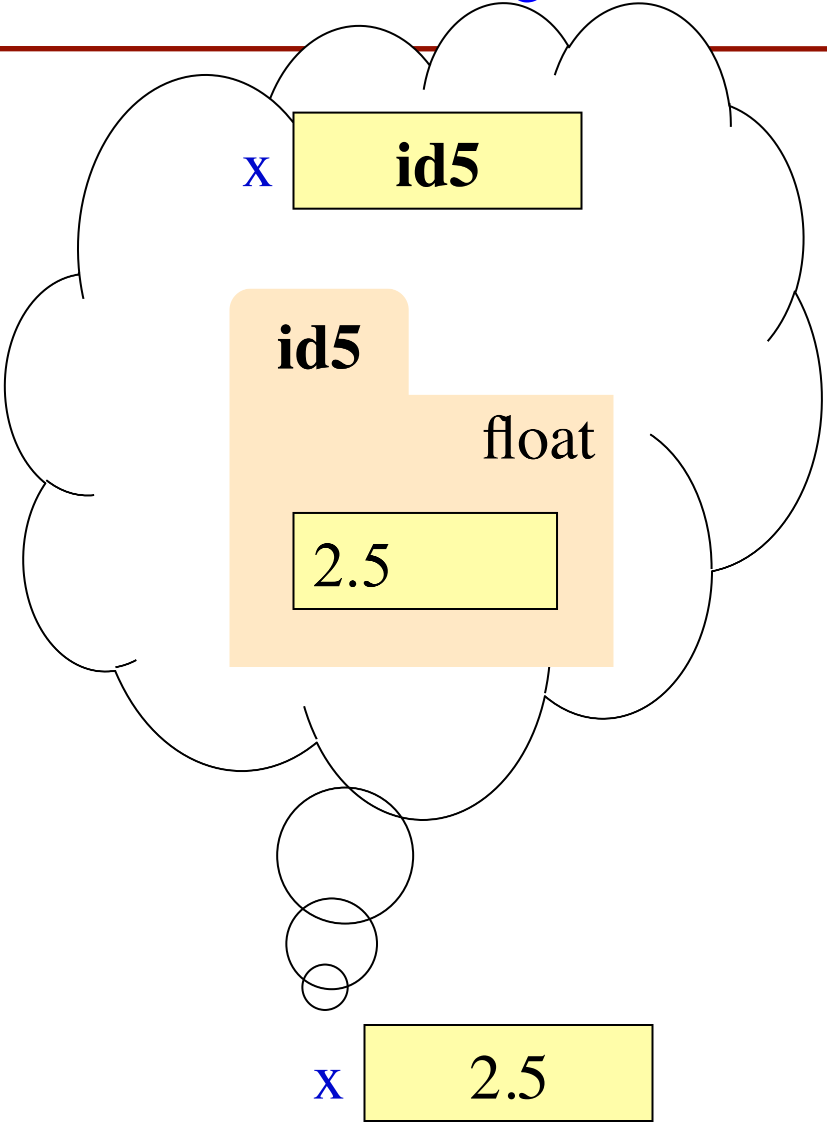
Surprise: All Values are in Objects!

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

```
>>> x = 2.5

>>> id(x)
```

- But they are special
  - They are *immutable* (contents cannot change)
  - Distinction between *value* and *identity* is immaterial
  - So we can ignore the folder



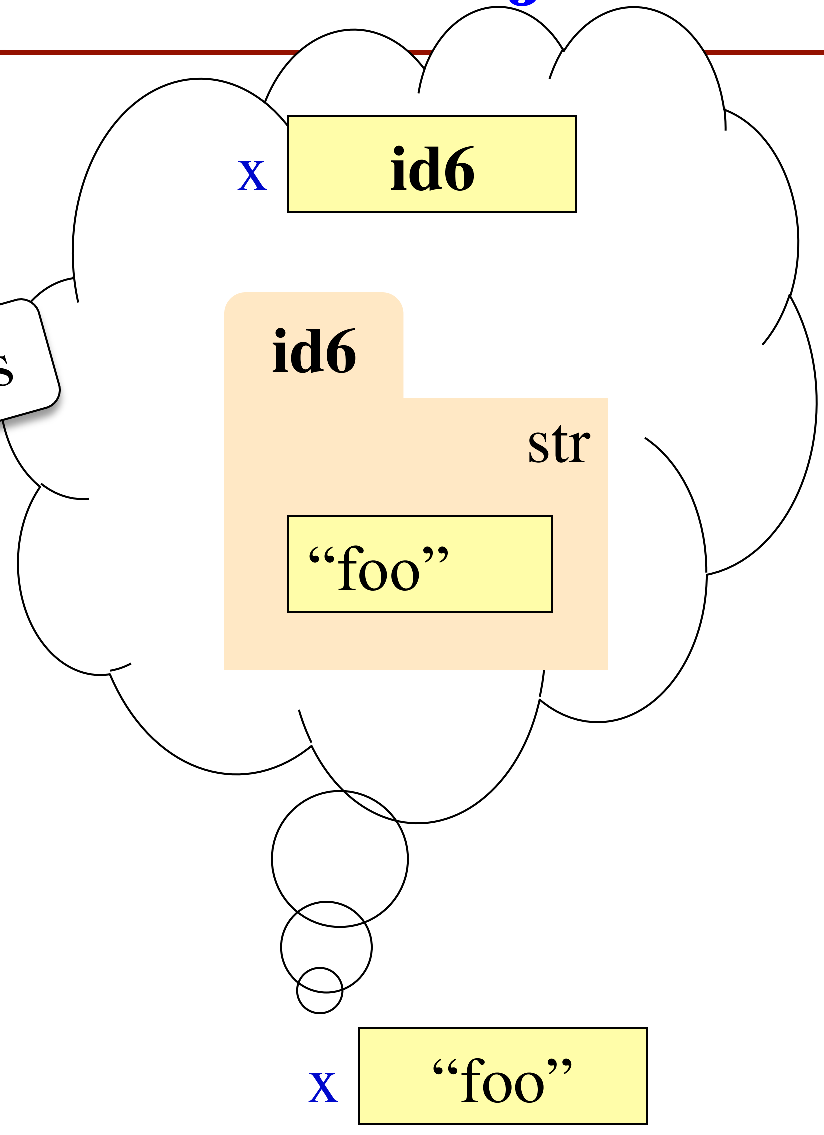
Surprise: All Values are in Objects!

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

```
>>> x = "foo"
>>> id(x)
```

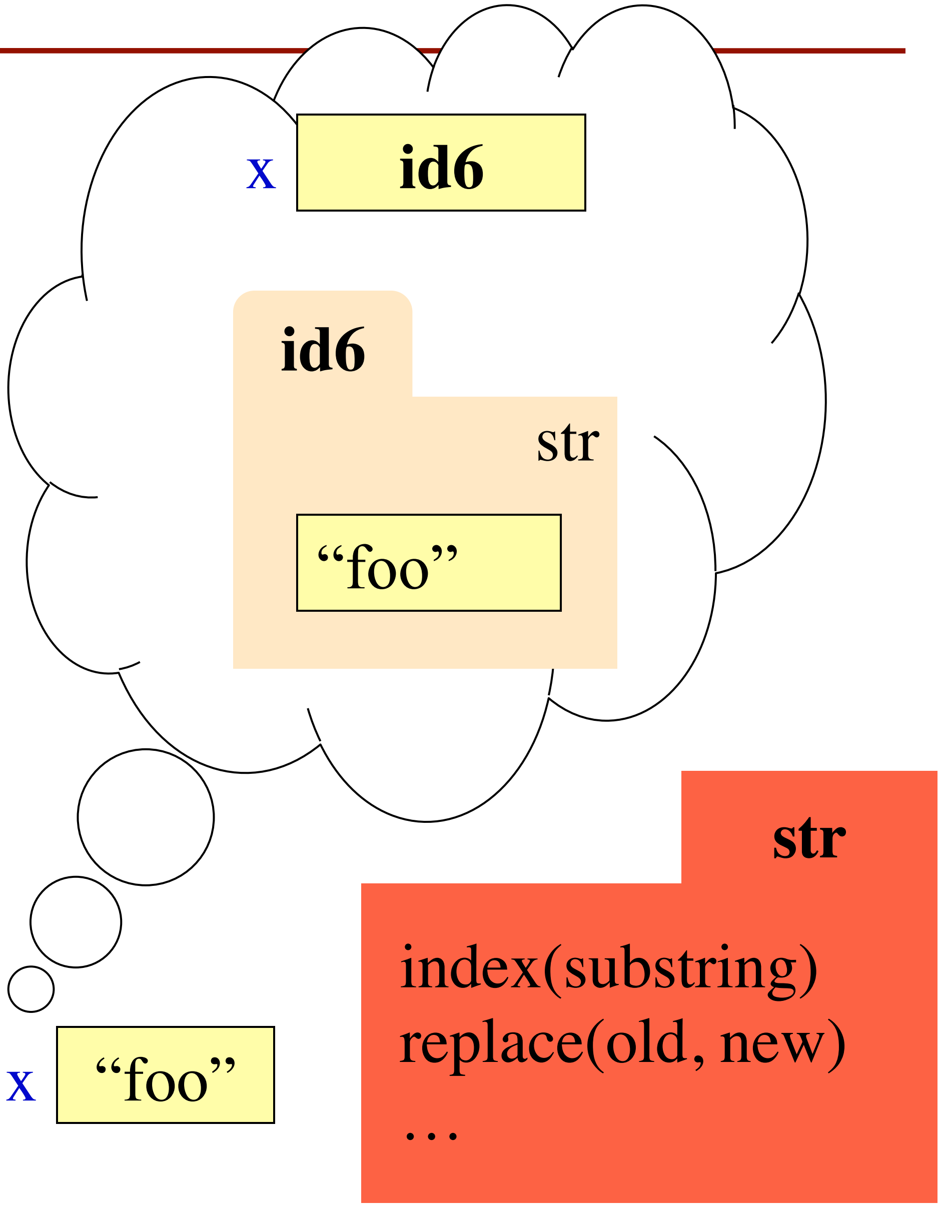
includes strings

- But they are special
  - They are *immutable* (contents cannot change)
  - Distinction between *value* and *identity* is immaterial
  - So we can ignore folder



Strings Have Methods Too

- We have seen expressions like `s.index('a')`
- Now we can recognize them as method calls
- String methods do not change the string
  - Can’t: strings immutable
  - “Modifications” made by returning a *new* string
  - `s.replace('o','uh')` evaluates to `'Helluh Wuhld!'` but s is still `'Hello World'`



Class Objects are Mutable

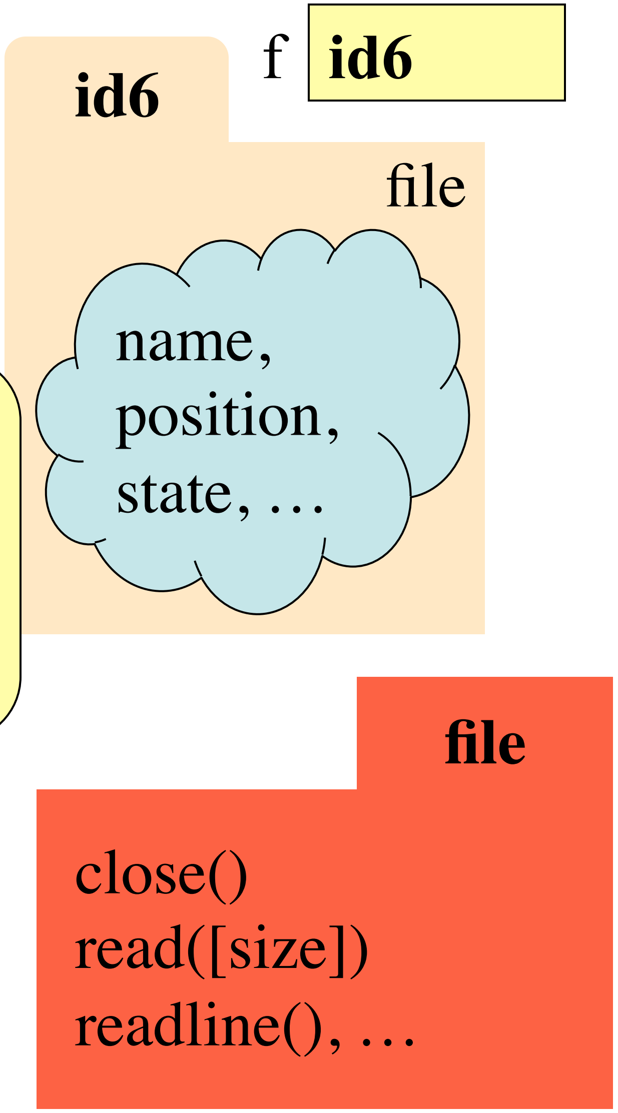
- Unlike int, str, etc., objects of class type (and some others) are *mutable*
  - You can change them
  - Methods can have effects besides their return value
- Example:

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

Opens a file on your hard disk, returns a file object you can read from

- Example: `p.projectToFloor()`

http://docs.python.org/2/library/stdtypes.html#file-objects



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 defined by the Standard Library or other libraries.
- 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