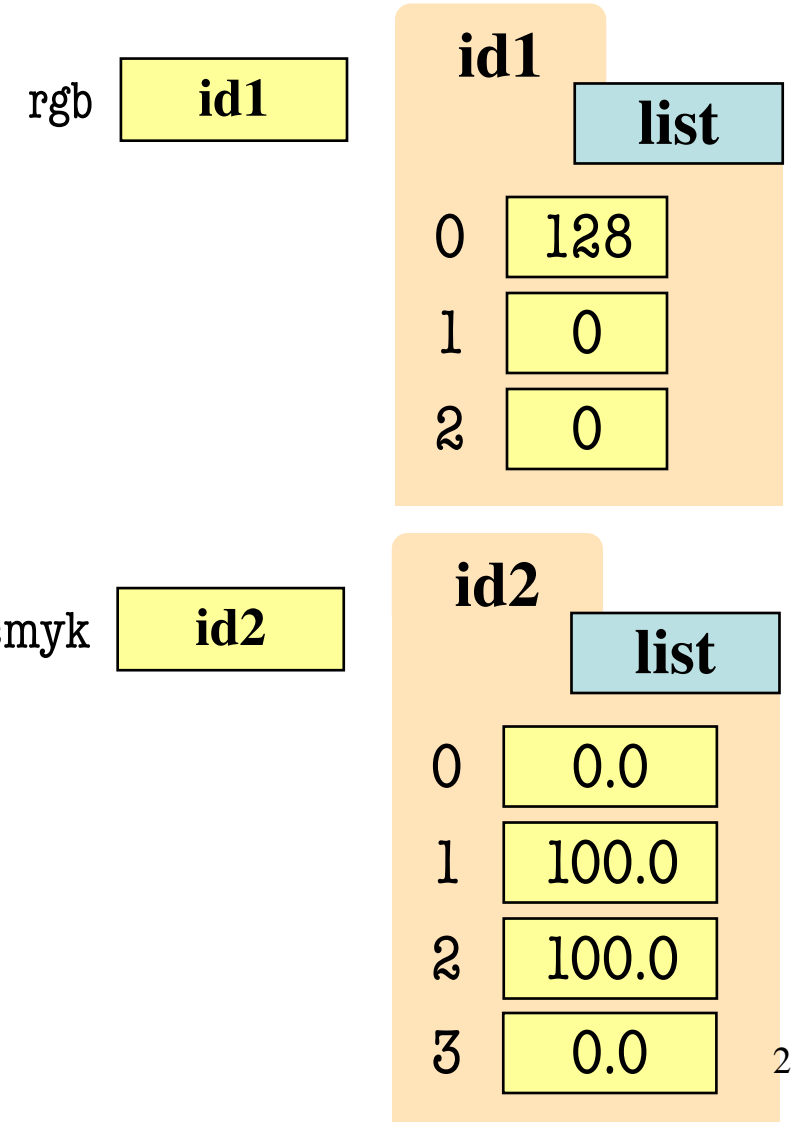
Mini-Lecture 16

Objects

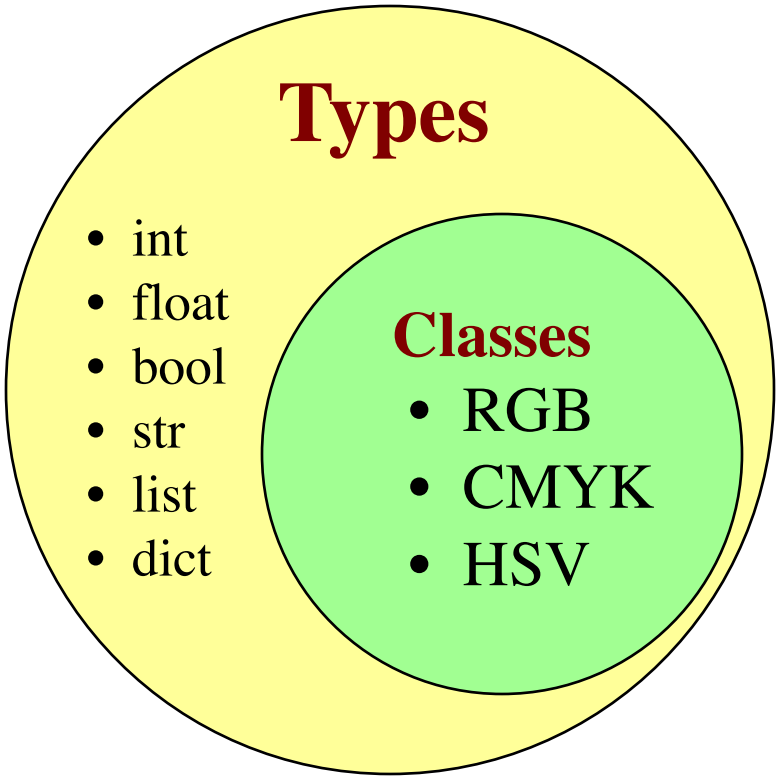
Thinking About Assignment 2

- **A2**: three color models
  - `RGB`: 3 ints 0 to 255
  - `CMYK`: 4 floats 0.0 to 100.0
  - `HSV`: 3 floats, mult. bounds
  - We could represent as lists
- Can get really confusing
  - Easy to mix-up models
  - Easy to go out of bounds
- **We want custom types**
  - One for each color model
  - Motivation for ***classes***

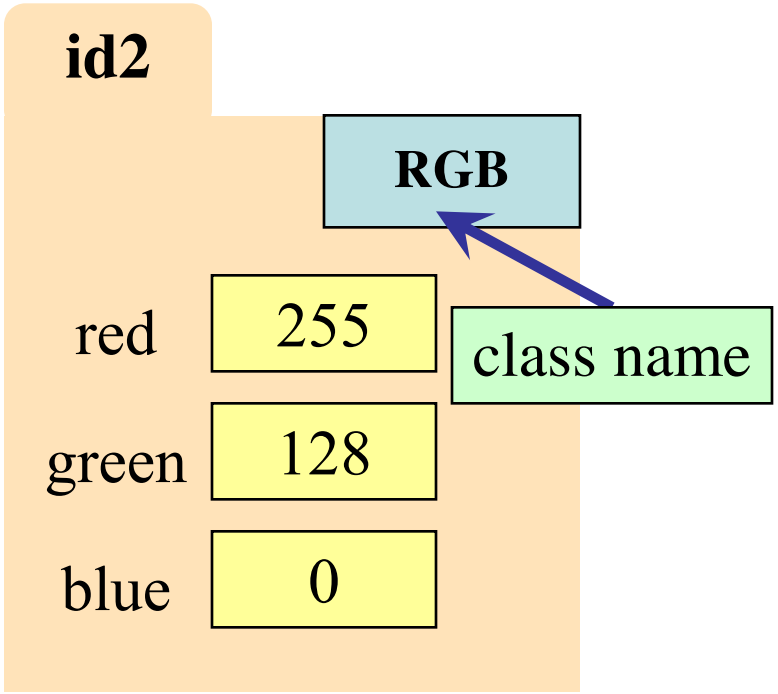


Classes are Customized Types

- Classes are any type not already built-into Python



- Values look like **dicts**
  - Represent as a folder
  - Variables are named

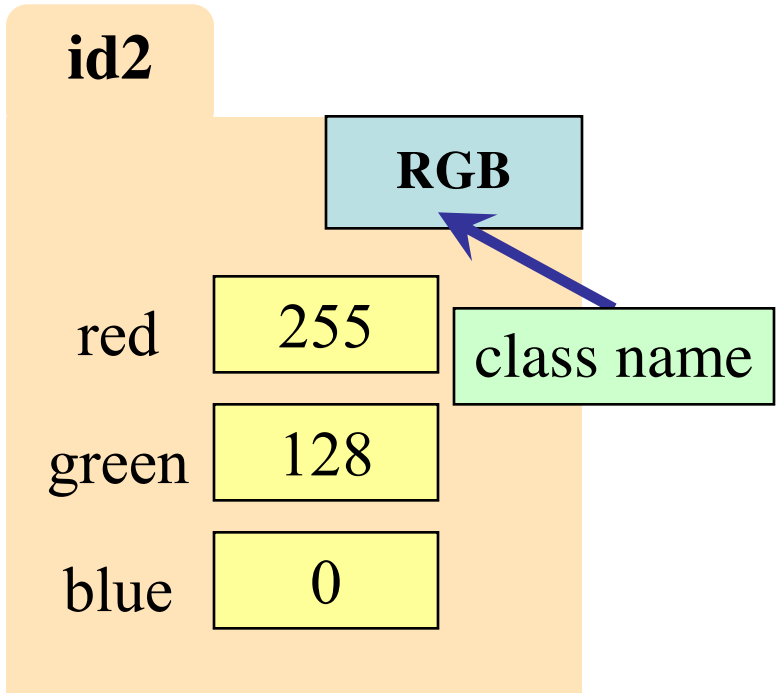


Classes are Customized Types

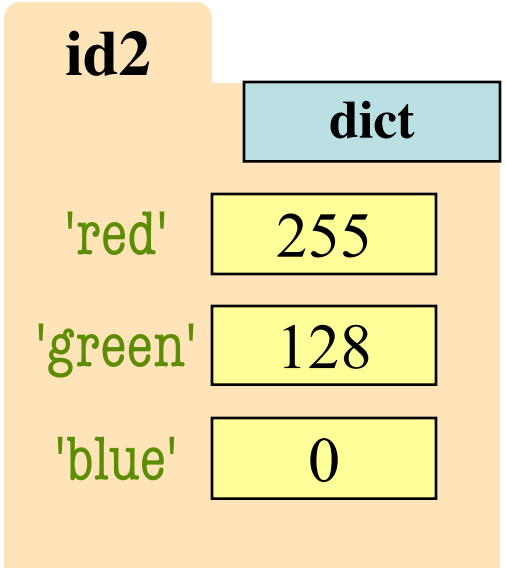
- Classes are any type not already built-into Python
  - int
  - float
  - bool
  - str
  - list
  - dict
  - RGB
  - CMYK
  - HSV

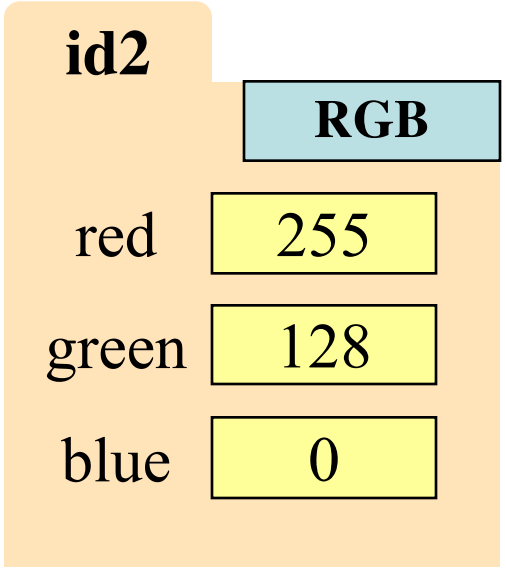
Class values are called **objects**

- Values look like **dicts**
  - Represent as a folder
  - Variables are named



Why Are They Better Than dicts?

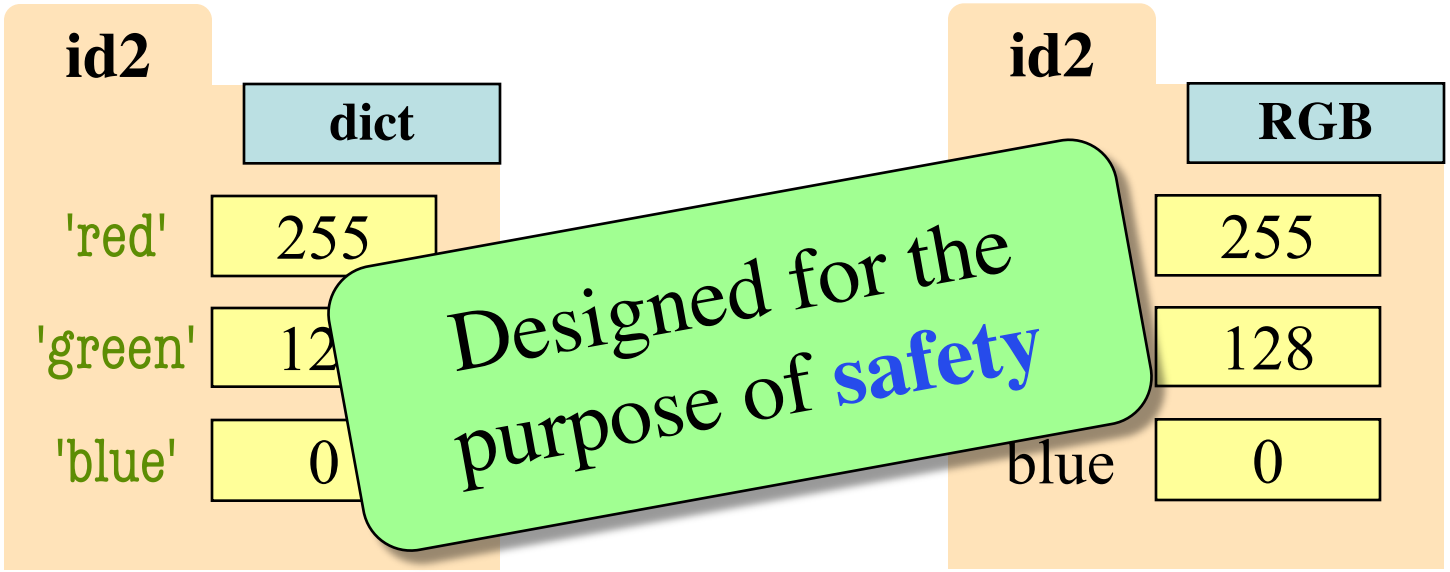




- Can add new variables
- Does not check bounds of the content variables

- Variables fixed (sort-of)
- Possibly checks bounds of the content variables

Why Are They Better Than dicts?

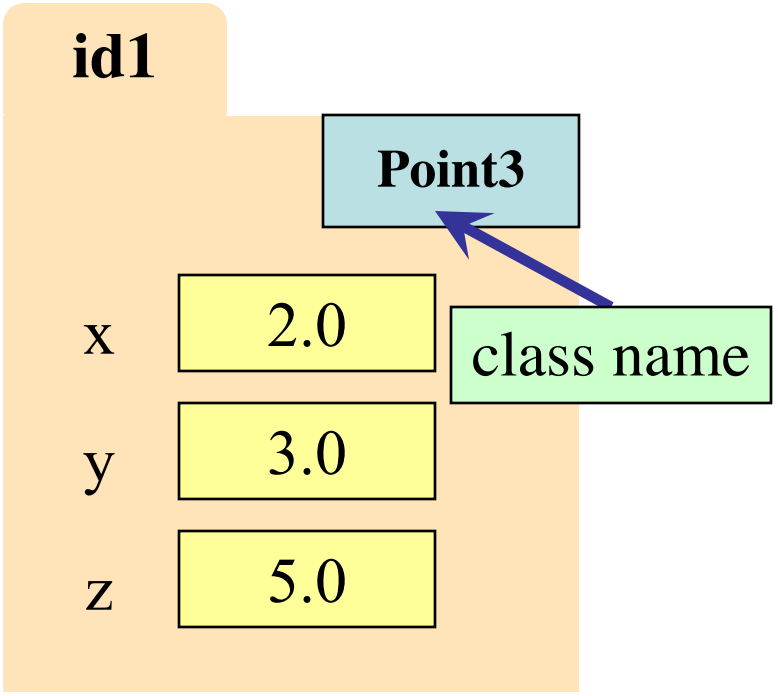


- Can add new variables
- Does not check bounds of the content variables

- Variables fixed (sort-of)
- Possibly checks bounds of the content variables

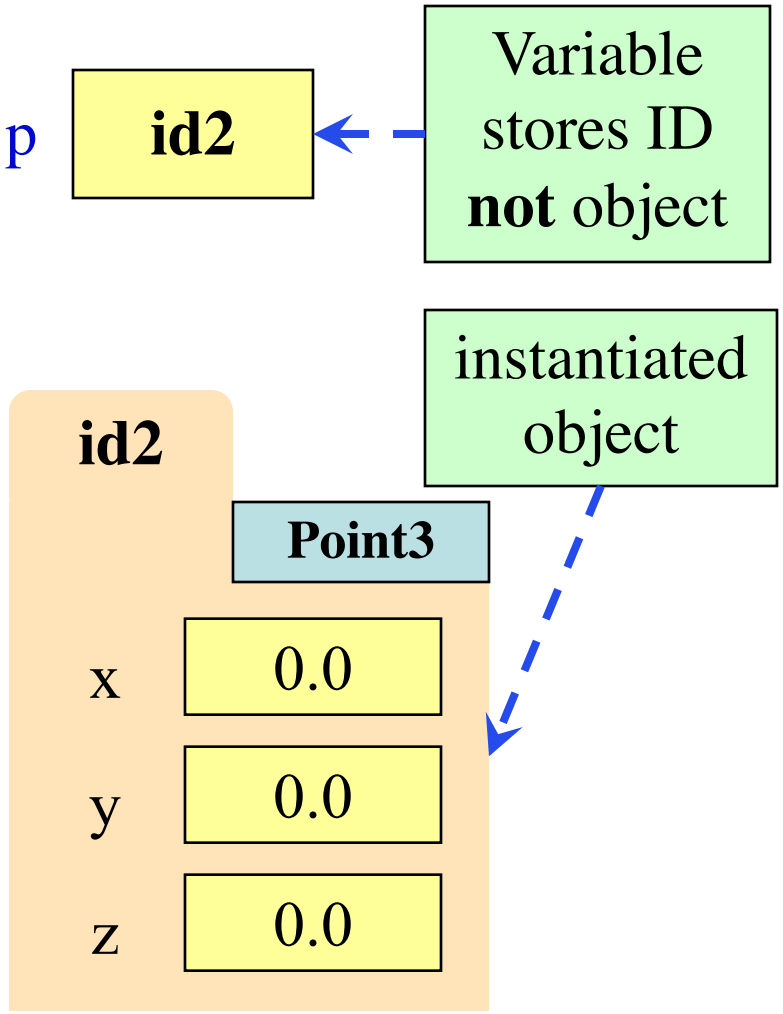
Using Classes in Python

- **Modules** provide classes
  - Import to use the class
  - Will show contents later
- **Example**: `introcs`
  - Color classes for A2: `RGB`, `CMYK`, `HSV`
  - Geometry classes: `Point2`, `Point3`
- Will make our own later



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**: `Point3(0,0,0)`
  - Makes an object (manila folder)
  - 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 introcs
```

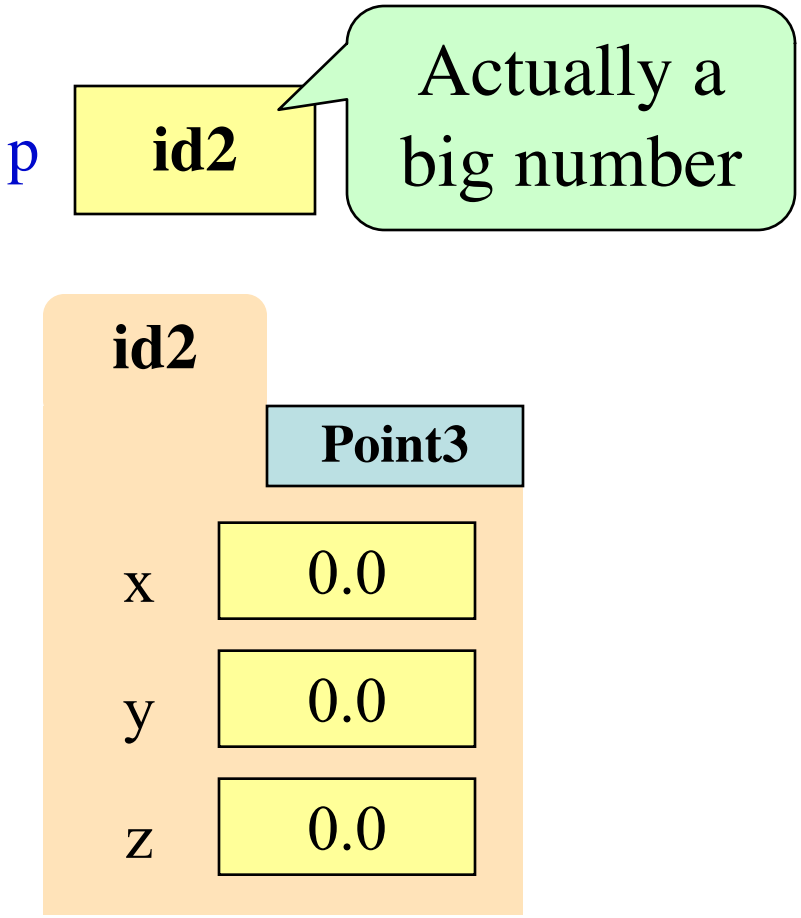
Need to import module that has Point class.

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

Constructor is function. Prefix w/ module name.

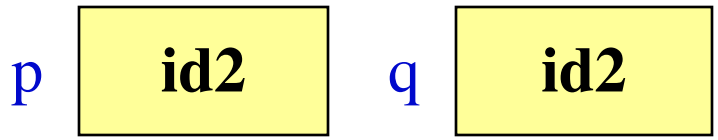
```
>>> id(p)
```

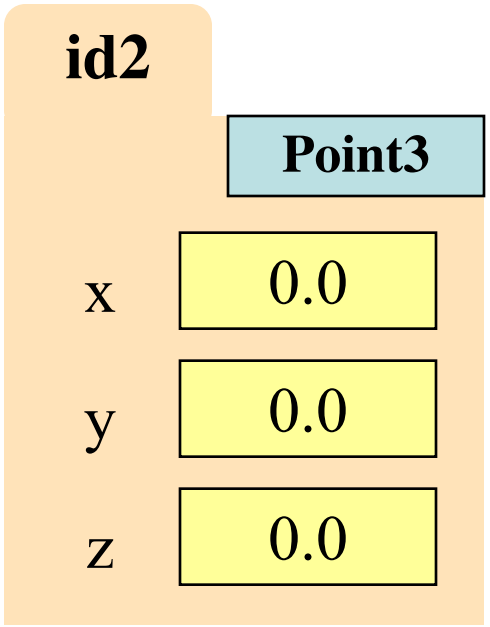
Shows the ID of p.



Object Variables

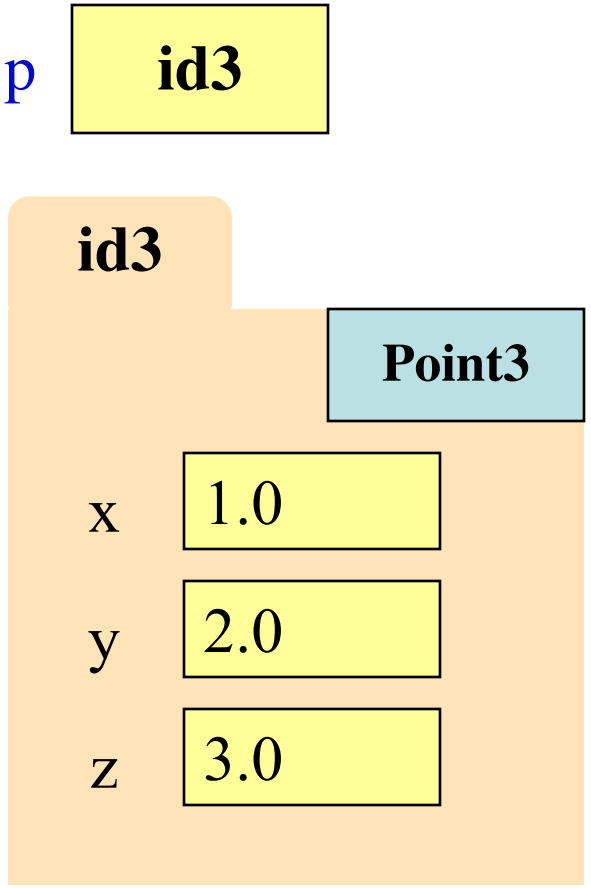
- Variable stores object name
  - **Reference** to the object
  - Reason for folder analogy
- Assignment uses object name
  - **Example**: q = p
  - Takes name from p
  - Puts the name in q
  - Does not make new folder!
- **Like we saw with lists**
  - Reason for using folders





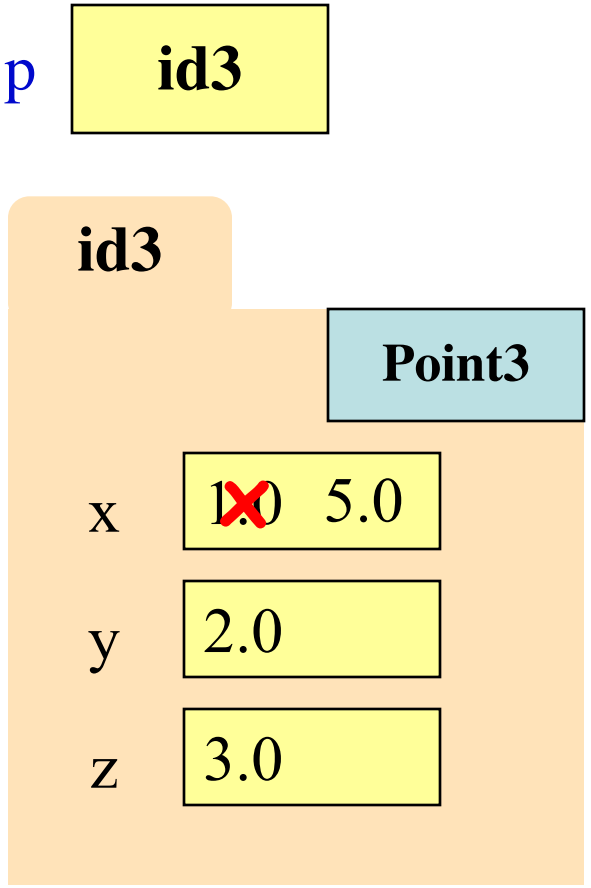
Objects and Attributes

- 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 = introcs.Point3(1,2,3)
  - p.x = p.y + p.z



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 = introcs.Point3(1,2,3)
  - p.x = p.y + p.z



Exercise: Attribute Assignment

- Recall, q gets name in p

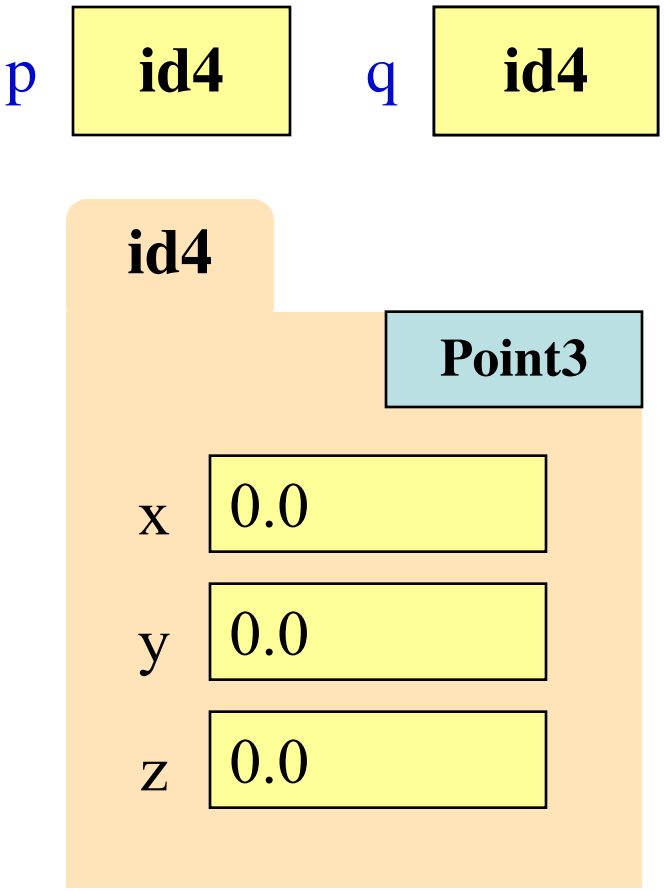
```
>>> p = cornell.Point3(0,0,0)
>>> q = p
```

- 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

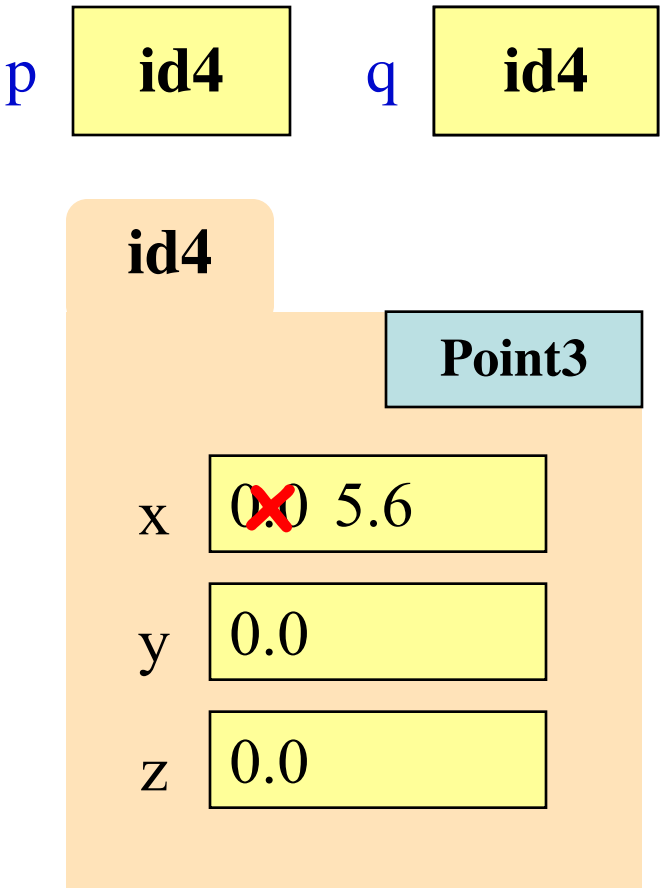
```
>>> p = geom.Point3(0,0,0)
>>> q = p
```

- 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



Exercise: Attribute Assignment

- Recall, q gets name in p

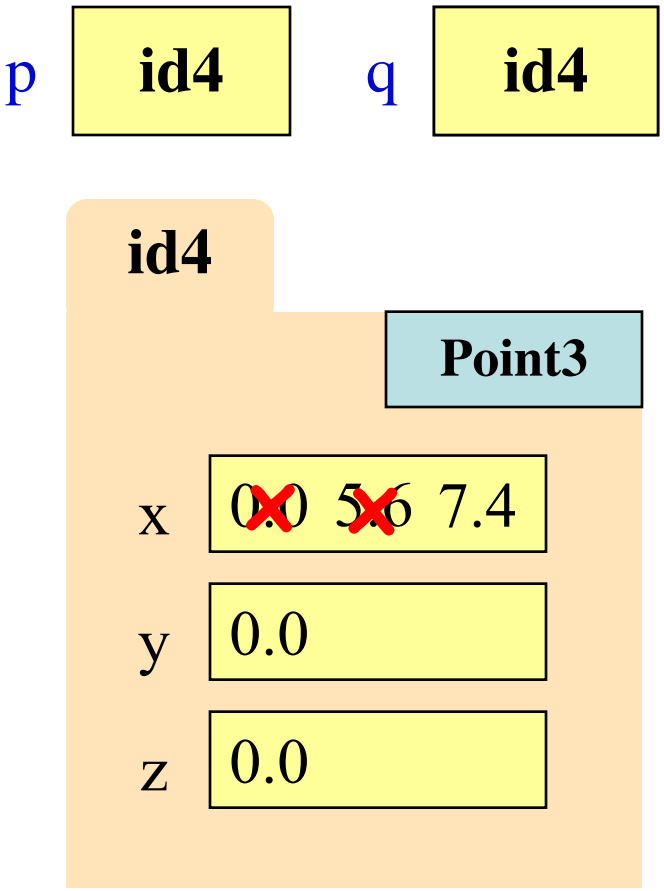
```
>>> p = geom.Point3(0,0,0)
>>> q = p
```

- 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

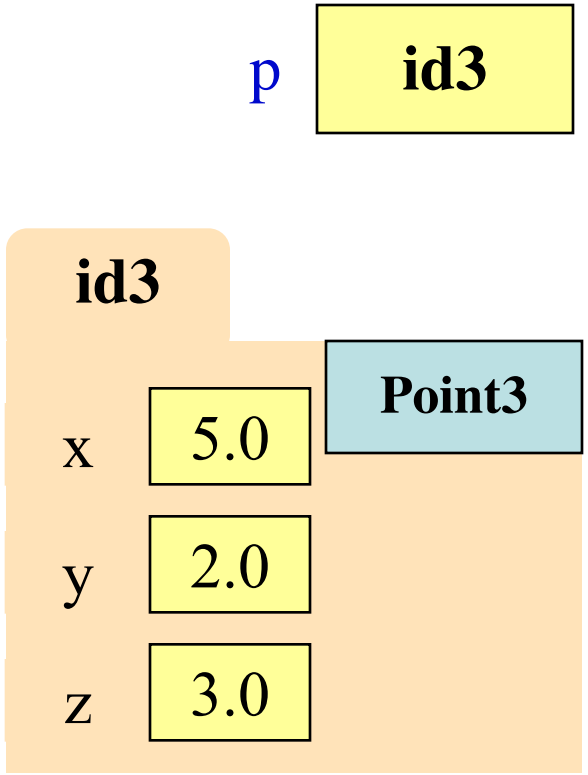


Methods: Functions Tied to Objects

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

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

  - **Example**: `p.distance(q)`
  - **Example**: `p.abs()` # makes x,y,z ≥ 0
- Object acts like an argument
  - Distance p to q: `p.distance(q)`
  - Distance x to y: `x.distance(y)`
  - Different objects, different values

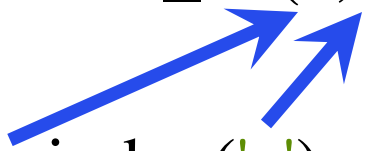


Strings Have Methods Too

```
>>> from introcs import index_str, count
>>> s = 'Hello'
>>> index_str(s,'e')
2
>>> s.index('e')
2
>>> count_str(s,'l')
2
>>> s.count('l')
2
```

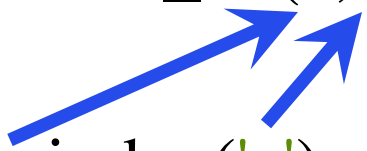
Strings Have Methods Too

```
>>> from introcs import index_str, count
>>> s = 'Hello'
>>> index_str(s,'e')
2
>>> s.index('e')
2
>>> count_str(s,'l')
2
>>> s.count('l')
2
```



Strings Have Methods Too

```
>>> from introcs import index_str, count
>>> s = 'Hello'
>>> index_str(s,'e')
2
>>> s.index('e')
2
>>> count_str(s,'l')
2
>>> s.count('l')
2
```



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