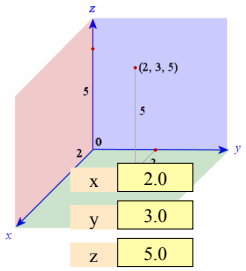
One-on-One Sessions

- Still ongoing: 1/2-hour one-on-one sessions
  - To help prepare you for the assignment
  - **Primarily for students with little experience**
- There are still some spots available
  - Sign up for a slot in CMS
- Will keep running after **September 18**
  - Will open additional slots after the due date
  - Will help students revise Assignment 1

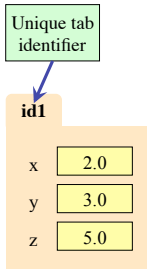
Type: Set of values and the operations on them

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



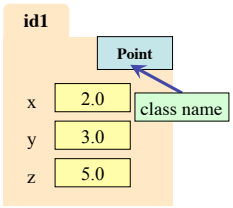
Objects: Organizing Data in Folders

- 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



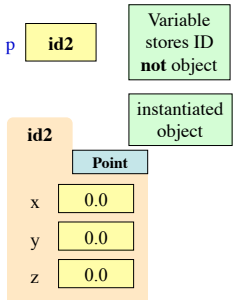
Classes: Types for Objects

- 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
```

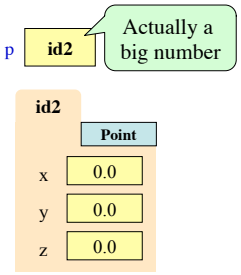
Need to import module that has Point class.

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

Constructor is function. Prefix w/ module name.

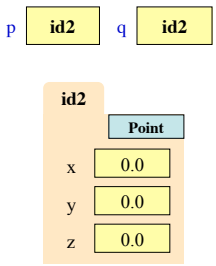
```
>>> id(p)
```

Shows the ID of p.



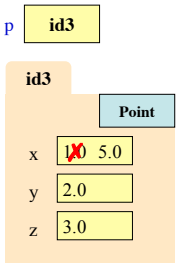
Object Variables

- 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

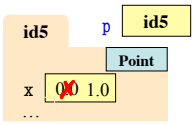


Call Frames and Objects

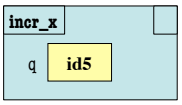
- Mutable objects can be altered in a function call
  - Object vars hold names!
  - Folder accessed by both global var & parameter
- **Example**:

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

Global **STUFF**

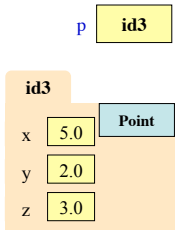


Call Frame



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 ≥ 0
- Just like we saw for strings
  - s = 'abracadabra'
  - s.index('a')
- Are strings objects?

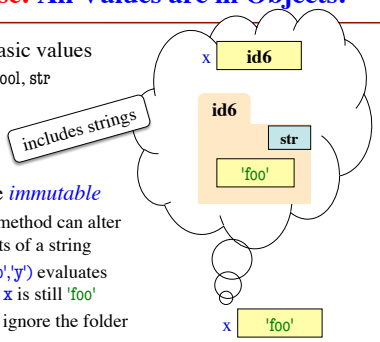


Surprise: All Values are in Objects!

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

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

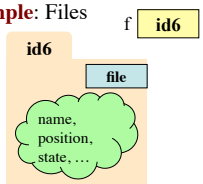
- But they are *immutable*
  - No string method can alter the contents of a string
  - x.replace('o','y') evaluates to 'fyy' but x is still 'foo'
  - So we can ignore the folder



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