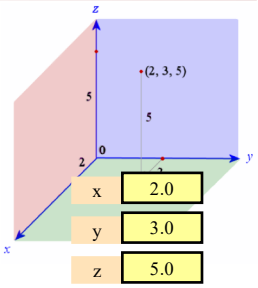
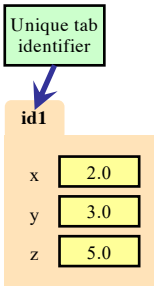
Type: Set of values and the operations on them

- 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?



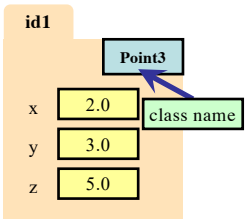
Objects: Organizing Data in Folders

- 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



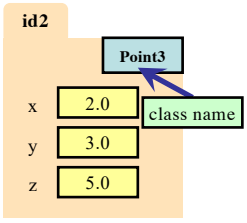
Classes: Types for Objects

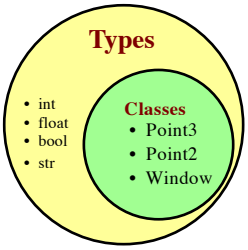
- Values must have a type
  - An object is a **value**
  - Type of object is its **class**
- **Modules** provide classes
  - Will show how later
- **Example**: introcs
  - Part of CornellExtensions
  - Just need to import it
  - Classes: `Point2`, `Point3`



The Old Way: Classes vs Types

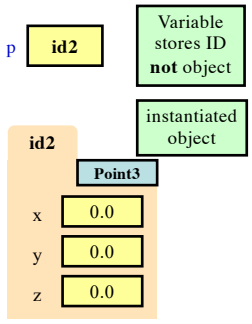
- Values must have a type
  - An object is a **value**
  - Object type is a **class**
- Classes are how we add new types to Python





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
```

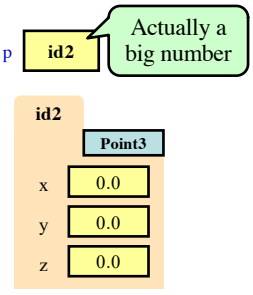
Need to import module that has Point class.

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

Constructor is function. Prefix w/ module name.

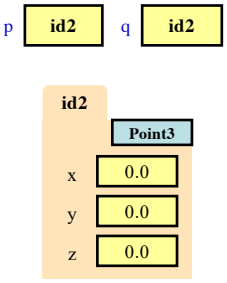
```
>>> id(p)
```

Shows the ID of p.



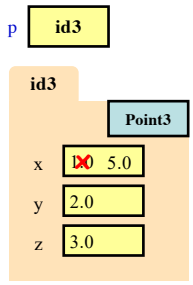
Object Variables

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



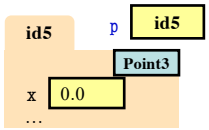
Call Frames and Objects

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

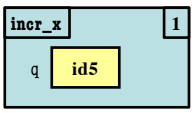
```
def incr_x(q):
1 |   q.x = q.x + 1
```

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

Global **STUFF**



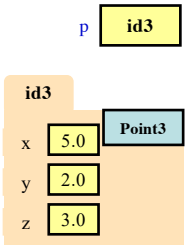
Call Frame



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?

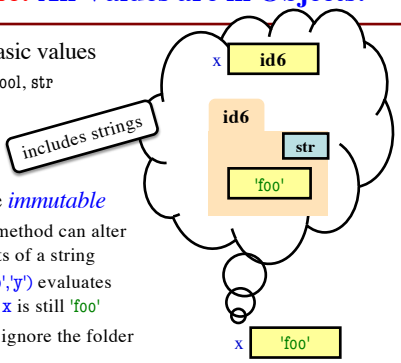


Surprise: All Values are in Objects!

- 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



Base Types vs. Classes

Base Types	Classes
• Built-into Python	• Provided by modules
• Refer to instances as *values*	• Refer to instances as *objects*
• Instantiate with *literals*	• Instantiate w/ *constructors*
• Are all immutable	• Can alter attributes
• Can ignore the folders	• Must represent with folders