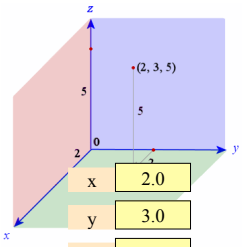
Type: Set of values and the operations on them

- 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?



x	2.0
y	3.0
z	5.0

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

Unique tab identifier

id1

x	2.0
y	3.0
z	5.0

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`

id1

Point

x	2.0
y	3.0
z	5.0

class name

Constructor: Function to make Objects

- How do we create objects?
  - Other types have **literals**
  - **Example**: 1, `"abc"`, **true**
  - No such thing for objects
- **Constructor Expression**:
  - "Call" the class like a function
  - **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

p **id2**

Variable stores ID **not** object

instantiated object

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

p **id2** q **id2**

id2

Point

x	0.0
y	0.0
z	0.0

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`

p **id3**

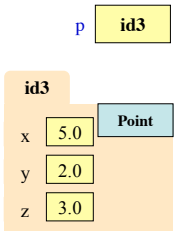
id3

Point

x	~~1.0~~ 5.0
y	2.0
z	3.0

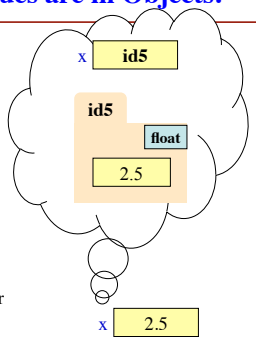
Methods: Functions Tied to Classes

- **Method**: function tied to a class
  - 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?



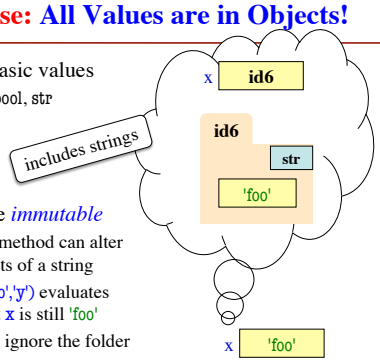
Surprise: All Values are in Objects!

- Including basic values
  - int, float, bool, str
- **Example**:
  >>> x = 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!

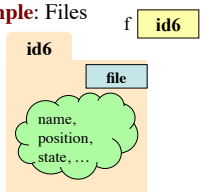
- Including basic values
  - int, float, bool, str
- **Example**:
  >>> x = 'foo'
  >>> id(x)
- But they are *immutable*
  - No string method can alter the contents of a string
  - x.replace('o','y') evaluates to 'fyy' but x is still 'foo'
  - So we can ignore the folder



Class Instance Objects

- Use name **class instance** to distinguish from other values
  - Not int, float, bool, str
- Class instances are **mutable** objects
  - 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

Basic Types vs. Class Types

Basic Types	Class Types
Built into Python	Provided by modules
Instantiate with *literals*	Instantiate w/ *constructors*
Are all immutable	Can alter attributes
Can ignore the folders	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