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 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

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
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 = 'abra cadabra'`
    - `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

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)`

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

Basic Types vs. Class Types

<table>
<thead>
<tr>
<th>Basic Types</th>
<th>Class Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Built into Python</td>
<td>Provided by modules</td>
</tr>
<tr>
<td>Instantiate with literals</td>
<td>Instantiate w/ constructors</td>
</tr>
<tr>
<td>Are all immutable</td>
<td>Can alter attributes</td>
</tr>
<tr>
<td>Can ignore the folders</td>
<td>Must represent with folders</td>
</tr>
</tbody>
</table>

2/6/14: Objects