Thinking About Assignment 2

- A3: three color models
  - RGB: 3 ints 0 to 255
  - CMYK: 4 floats 0.0 to 1.0
  - HSV: 3 floats, mult. bounds
  - Represented 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 how we add new types to Python
  - Values look like dicts
    - Represent as a folder
    - Content variables 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

Using Classes in Python

- Modules provide classes
  - Import to use the class
  - Will show contents later
- Example: colormodel
  - Color classes for A3
  - RGB, CMYK, HSV
- Example: geom
  - Geometry classes
  - Point2, Point3

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

```python
>>> import geom
```

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

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

Actually a big number
**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 = geom.Point3(1,2,3)
  - p.x = p.y + p.z

**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
  - 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.distanceTo(q)
  - 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

**Base Types vs. Classes**

<table>
<thead>
<tr>
<th>Base Types</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Built-into Python</td>
<td>Provided by modules</td>
</tr>
<tr>
<td>Refer to instances as <strong>values</strong></td>
<td>Refer to instances as <strong>objects</strong></td>
</tr>
<tr>
<td>Instantiate with <strong>literals</strong></td>
<td>Instantiate w/ <strong>constructors</strong></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>