Lecture 8

Classes & Objects
Type: Set of values and the operations on them

- **Type int:**
  - **Values:** integers
  - **Ops:** +, −, *, /, %, **

- **Type float:**
  - **Values:** real numbers
  - **Ops:** +, −, *, /, **

- **Type bool:**
  - **Values:** True and False
  - **Ops:** not, and, or

- **Type str:**
  - **Values:** string literals
    - Double quotes: "abc"
    - Single quotes: 'abc'
  - **Ops:** +, slicing

- **Type list:**
  - **Values:** list of values
    - Indicate with []
  - **Ops:** +, slicing
Type: Set of values and the operations on them

- **Type int:**
  - Values: integers
  - Ops: +, −, *, /, %, **

- **Type float:**
  - Values: real numbers
  - Ops: +, −, *, /, **

- **Type bool:**
  - Values: True and False
  - Ops: not, and, or

- **Type str:**
  - Values: string literals
  - Ops: +, slicing
    - Double quotes: "abc"
    - Single quotes: 'abc'

- **Type list:**
  - Values: list of values
  - Indicate with []
  - Ops: +, slicing

Are these the only types that exist?
Thinking About Assignment 2

- **A3**: three color models
  - RGB: 3 ints 0 to 255
  - CMYK: 4 floats 0.0 to 100.0
  - 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

Types
- int
- float
- bool
- str
- list

Classes
- RGB
- CMYK
- HSV

**id2**
- RGB
- 255
- red
- 128
- green
- 0
- blue

class name
Classes are Customized Types

- Classes are how we add new types to Python
- Values look like dicts
  - Represent as a folder
  - Content variables named

Class values are called objects

- int
- float
- bool
- str
- list

RGB
- R
- G
- B

CMYK
- C
- M
- Y
- K

HSV
- H
- S
- V

id2
- red: 255
- green: 128
- blue: 0

RGB class name
Why Are They Better Than dicts?

- Can add new variables
- Does not check bounds of the content variables

10/5/15

Classes & Objects
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

<table>
<thead>
<tr>
<th>id2</th>
<th>dict</th>
<th>RGB</th>
</tr>
</thead>
<tbody>
<tr>
<td>'red'</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>'green'</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>'blue'</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Designed for the purpose of **safety**
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

>>> import geom

Need to import module that has Point class.

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

Constructor is function.
Prefix w/ module name.

>>> id(p)

Shows the ID of p.

Actually a big number

id2

x 0.0
y 0.0
z 0.0

Point3
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>.

  Example: `p.x`
  ▪ Look like module variables

• Putting it all together
  ▪ `p = geom.Point3(1,2,3)`
  ▪ `p.x = p.y + p.z`
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
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 CORRECT
  C: id4
  D: I don’t know
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  CORRECT
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:
    \( \text{⟨variable⟩} . \langle \text{method} \rangle (\langle \text{arguments} \rangle) \)
  - **Example**: `p.distanceTo(q)`
  - **Example**: `p.abs()` # makes \(x, y, z \geq 0\)

- 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:**
  ```python
  >>> 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:**
  ```python
  >>> x = 'foo'
  >>> id(x)
  `x` is still 'foo'
  ```
- 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

10/5/15 Classes & Objects 20
# 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 <em>values</em></td>
<td>• Refer to instances as <em>objects</em></td>
</tr>
<tr>
<td>• Instantiate with <em>literals</em></td>
<td>• Instantiate w/ <em>constructors</em></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>

10/5/15 Classes & Objects 21