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 $x_0, y_0, z_0$ for first point
  - Vars $x_1, y_1, z_1$ 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 Function:
  - Same name as the class
    - 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
Exercise: Attribute Assignment

• Recall, q gets name in p
  >>> p = tuple3d.Point(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

Point 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 \(x,y,z \geq 0\)
  § Just like we saw for strings
    * \(a = 'abracadabra'\)
    * \(a.index('s')\)
  § 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
  * \(x.replace('v','y')\) evaluates to \('yy'\) but \(x\) is still \('foo'\)
  * So we can ignore the folder

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

Class Objects

• Use name \(class\ object\) to distinguish from other values
  * Not int, float, bool, str

• Class objects are mutable
  * You can change them
  * Methods can have effects besides their return value

• Example:
  * \(p = Point(3,-3,0)\)
  * \(p.clamp(-1,1)\)

Example: Files

• Use name \(class\ object\) to distinguish from other values
  * Not int, float, bool, str

• Class objects are mutable
  * You can change them
  * Methods can have effects besides their return value

• Example:
  * \(p = Point(3,-3,0)\)
  * \(p.clamp(-1,1)\)

A1: The Module urllib2

• Module urllib2 is used to read web pages
  * Function urlopen creates a url object
    * \(u = urllib2.urlopen('http://www.cornell.edu')\)

• url has a method called read()
  * Returns contents of web page
  * Usage: \(s = u.read()\) # \(s\) is a string