Recall: Classes are Types for Objects

- Values must have a type
  - An object is a value
  - Object type is a class
- Classes are how we add new types to Python

Types

- int
- float
- bool
- str

Recall: Objects can have Methods

- Method: function tied to object
  - Function call: \(<function-name>(<arguments>)\)
  - Method call: \(<object-variable>.<function-call>\)
  - Use of a method is a method call
  - Both \(p\) and \(q\) act as arguments
  - Very much like \(\text{distanceTo}(p, q)\)

Name Resolution for Objects

- \(\langle\text{object}\rangle.\langle\text{name}\rangle\) means
  - Go the folder for \(\text{object}\)
  - Find attribute/method \(\text{name}\)
  - If missing, check \(\text{class}\) folder
  - If not in either, raise error

- For most Python objects
  - Attributes are in \(\text{object}\) folder
  - Methods are in \(\text{class}\) folder
  - Rules can be broken...
    (but not in this class)

The Class Definition

- \(\text{class <class-name>(<object>):}\)
  - **“Class specification”**
    - \(\langle\text{function definitions}\rangle\)
    - \(\langle\text{assignment statements}\rangle\)
    - \(\langle\text{other statements also allowed}\rangle\)
  - Example \(\text{Example(object):}\)
    - “The simplest possible class.”
    - \(\text{pass}\)

Instances and Attributes

- Assignments add object attributes
  - \(<\text{object}>.:<\text{att}\> = <\text{expression}>\)
  - Example: \(e.b = 42\)
- Assignments can add class attributes
  - \(<\text{class}>.:<\text{att}\> = <\text{expression}>\)
  - Example: \(\text{Example.a} = 29\)
- Objects can access class attributes
  - Example: \(\text{print a}\)
    - But assigning it creates object attribute
  - Example: \(a.a = 10\)
- Rule: check object first, then class

Invariants

- Properties of an attribute that must be true
- Works like a precondition:
  - If invariant satisfied, object works properly
  - If not satisfied, object is “corrupted”
- Examples:
  - \(\text{Point}\) class: all attributes must be floats
  - \(\text{RGB}\) class: all attributes must be ints in \(0..255\)
- Purpose of the class specification
The Class Specification

class Worker(object):
    # An instance is a worker in an organization.
    Instance has basic worker info, but no salary information.

    Attributes:
    - name: Worker's last name. [str]
    - ssn: Social security no. [int in 0.999999999]
    - boss: Worker's boss. [Worker, or None if no boss]

    Methods Calls

    Example: a.distanceTo(b)

    def distanceTo(self,q):
        # Returns: dist from self to q
        # Precondition: q a Point
        assert type(q) == Point
        sqrdst = ((self.x-q.x)**2 + (self.y-q.y)**2 + (self.z-q.z)**2)
        return math.sqrt(sqrdst)

    Class Methods

    Special Method: __init__

    Called by the constructor

ÀAside: The Value None

- The boss field is a problem.
  - boss refers to a Worker object
  - Some workers have no boss
  - Or maybe not assigned yet
    (the book stops there)
- Solution: use value None
  - None: Lack of (folder) name
  - Will reassign the field later!
- Be careful with None values
  - var3.x gives error!
  - There is no name in var3
  - Which Point to use?

Making Arguments Optional

- We can assign default values to __init__ arguments
  - Write as assignments to parameters in definition
  - Parameters with default values are optional
- Examples:
  - p = Point() # (0,0,0)
  - p = Point(1,2,3) # (1,2,3)
  - p = Point(y=4) # (1,0,0)
  - p = Point(x=8) # (1,0,0)

class Point(object):
    # Instances are points in 3d space
    x: float
    y: float
    z: float

    def __init__(self, x=0, y=0, z=0):
        # Initializer: makes a new Point
        # Precondition: x,y,z are numbers
        self.x = x
        self.y = y
        self.z = z