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

<table>
<thead>
<tr>
<th>id2</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>2.0</td>
</tr>
<tr>
<td>y</td>
<td>3.0</td>
</tr>
<tr>
<td>z</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Types

- int
- float
- bool
- str

Classes

- Point3
- RGB
- Turtle
- Window
Classes Have Folders Too

Object Folders

- Separate for each instance

<table>
<thead>
<tr>
<th>id2</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>2.0</td>
</tr>
<tr>
<td>y</td>
<td>3.0</td>
</tr>
<tr>
<td>z</td>
<td>5.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id3</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>5.0</td>
</tr>
<tr>
<td>y</td>
<td>7.2</td>
</tr>
<tr>
<td>z</td>
<td>-0.5</td>
</tr>
</tbody>
</table>

Class Folders

- Data common to all instances

Point3

????
Name Resolution for Objects

• ⟨object⟩.⟨name⟩ means
  ▪ Go the folder for object
  ▪ Find attribute/method name
  ▪ If missing, check class folder
  ▪ If not in either, raise error

• What is in the class folder?
  ▪ Data common to all objects
  ▪ First must understand the class definition

```
<table>
<thead>
<tr>
<th></th>
<th>p</th>
<th>id3</th>
<th>q</th>
<th>id4</th>
</tr>
</thead>
<tbody>
<tr>
<td>id3</td>
<td>Point3</td>
<td></td>
<td></td>
<td>Point3</td>
</tr>
<tr>
<td>x</td>
<td>5.0</td>
<td></td>
<td></td>
<td>7.4</td>
</tr>
<tr>
<td>y</td>
<td>2.0</td>
<td></td>
<td></td>
<td>0.0</td>
</tr>
<tr>
<td>z</td>
<td>3.0</td>
<td></td>
<td></td>
<td>0.0</td>
</tr>
</tbody>
</table>
```
The Class Definition

**Keyword class**
Beginning of a class definition

```
class <class-name>(object):
    """Class specification""
    <function definitions>
    <assignment statements>
    <any other statements also allowed>
```

Goes inside a module, just like a function definition.

Do not forget the colon!

More on this later

...but not often used

**Example**

```
class Example(object):
    """The simplest possible class.""
    pass
```

Python creates after reading the class definition.
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:** \(\text{print } e.a\)
  § But assigning it creates object attribute
  § **Example:** \(e.a = 10\)

• **Rule:** check object first, then class
class Worker(object):

    """An instance is a worker in an organization.

Instance has basic worker info, but no salary information.

ATTRIBUTES:

    lname: Worker’s last name.  [str]
    ssn: Social security no.  [int in 0..999999999]
    boss: Worker's boss.  [Worker, or None if no boss]
Method Definitions

• Looks like a function def
  ▪ But indented *inside* class
  ▪ The first parameter is always called `self`

• In a method call:
  ▪ Parentheses have one less argument than parameters
  ▪ The object in front is passed to parameter `self`

• **Example**: `a.distanceTo(b)`

```python
class Point3(object):
    """Instances are points in 3d space
    x: x coord [float]
y: y coord [float]
z: z coord [float]  ""

def distanceTo(self, q):
    """Returns: dist from self to q
    Precondition: q a Point3""
    assert type(q) == Point3
    sqrdst = ((self.x-q.x)**2 +
               (self.y-q.y)**2 +
               (self.z-q.z)**2)
    return math.sqrt(sqrdst)
```

```python
self q
```
• **Example:** `a.distanceTo(b)`

```python
class Point3(object):
    """Instances are points in 3d space
    x: x coord [float]
y: y coord [float]
z: z coord [float]  ""

    def distanceTo(self,q):
        """Returns: dist from self to q
        Precondition: q a Point3""
        assert type(q) == Point3
        sqrdst = ((self.x-q.x)**2 +
                  (self.y-q.y)**2 +
                  (self.z-q.z)**2)
        return math.sqrt(sqrdst)
```

**Example:**
- `a`:
  - `id2`
  - `x: 1.0`
  - `y: 2.0`
  - `z: 3.0`

- `b`:
  - `id3`
  - `x: 0.0`
  - `y: 3.0`
  - `z: -1.0`

- `a.distanceTo(b)`
Special Method: __init__

def __init__(self, n, s, b):
    """Initializer: creates a Worker
    Has last name n, SSN s, and boss b
    Precondition: n a string, s an int in range 0..999999999, and b either a Worker or None.
    self.lname = n
    self.ssn = s
    self.boss = b"

w = Worker('Obama', 1234, None)

don’t forget self
two underscores

Called by the constructor

<table>
<thead>
<tr>
<th>id8</th>
<th>Worker</th>
</tr>
</thead>
<tbody>
<tr>
<td>lname</td>
<td>'Obama'</td>
</tr>
<tr>
<td>ssn</td>
<td>1234</td>
</tr>
<tr>
<td>boss</td>
<td>None</td>
</tr>
</tbody>
</table>
Evaluating a Constructor Expression

Worker('Obama', 1234, None)

1. Creates a new object (folder) of the class Worker
   - Instance is initially empty
2. Puts the folder into heap space
3. Executes the method `__init__`
   - Passes folder name to self
   - Passes other arguments in order
   - Executes the (assignment) commands in initializer body
4. Returns the object (folder) name
Aside: 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 buck 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 = Point3()`  # (0,0,0)
  - `p = Point3(1,2,3)`  # (1,2,3)
  - `p = Point3(1,2)`  # (1,2,0)
  - `p = Point3(y=3)`  # (0,3,0)
  - `p = Point3(1,z=2)`  # (1,0,2)

```python
class Point3(object):
    
    ""
    Instances are points in 3d space
    x: x coord [float]
    y: y coord [float]
    z: z coord [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
        
        ...""
```