Recall: Objects as Data in Folders

- An object is like a **manila folder**
- It contains other variables
  - Variables are called **attributes**
  - Can change values of an attribute (with assignment statements)
- It has a "tab" that identifies it
  - Unique number assigned by Python
  - Fixed for lifetime of the object

<table>
<thead>
<tr>
<th>Unique tab identifier</th>
<th><code>id2</code></th>
<th><code>x</code></th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><code>y</code></td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>z</code></td>
<td>5.0</td>
</tr>
</tbody>
</table>

Classes Have Folders Too

<table>
<thead>
<tr>
<th>Object Folders</th>
<th>Class Folders</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Separate for each <em>instance</em></td>
<td>- Data common to all instances</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>id2</code></th>
<th><code>point</code></th>
<th><code>id3</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x</code></td>
<td>2.0</td>
<td><code>x</code></td>
</tr>
<tr>
<td><code>y</code></td>
<td>3.0</td>
<td><code>y</code></td>
</tr>
<tr>
<td><code>z</code></td>
<td>5.0</td>
<td><code>z</code></td>
</tr>
</tbody>
</table>

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
- For most Python objects
  - Attributes are in *object* folder
  - Methods are in *class* folder
  - But rules can be broken…

<table>
<thead>
<tr>
<th><code>id3</code></th>
<th><code>point</code></th>
<th><code>id4</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x</code></td>
<td>2.0</td>
<td><code>x</code></td>
</tr>
<tr>
<td><code>y</code></td>
<td>2.0</td>
<td><code>y</code></td>
</tr>
<tr>
<td><code>z</code></td>
<td>3.0</td>
<td><code>z</code></td>
</tr>
</tbody>
</table>

The Class Definition

- `class <class-name>(object):`
- **"Class specification"**
- **"Function definitions"**
- **"Assignment statements"**
- **"Any other statements also allowed"**

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

The Class Specification

- `class Worker(object):
  """An instance is a worker in an organization."
  Instance has basic worker info, but no salary information."
- **Attributes**
  - `red`: the red channel [int in range 0.0-1.0]
  - `green`: the green channel [int in range 0.255]
  - `blue`: the blue channel [int in range 0.255]
  - `alpha`: the alpha channel [int in range 0.255]""

| `Example` |
| `a`       |
| 29        |

Instances and Attributes

- Assignments add object attributes
  - `<object> = <expression>`
  - **Example**: `e.b = 42`
- Assignments can add class attributes
  - `<class> = <expression>`
  - **Example**: `Example.a = 29`
- Objects can access class attributes
  - **Example**: `print e.a`
  - But assigning it creates object attribute
  - **Example**: `e.a = 10`
- Rule: check object first, then class
Recall: Objects can have Methods

- **Method**: function tied to object
  - Function call: `<function-name>(<arguments>)`
  - Method call: `<object-variable>.(<function-call>)`
- **Example**: `p.distanceTo(q)`
  - Both `p` and `q` act as arguments

Methods Calls

- **Example**: `a.distanceTo(b)`

Methods are organized with the class

Methods are tied to objects. Method calls are syntactic sugar for function calls.

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?

Evaluating a Constructor Expression

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__`
   - Parses folder name to self
   - Parses other arguments in order
   - Executes the (assignment) commands in constructor body
4. Returns the object (folder) name

Special Method: `__init__`

Called by the constructor

In Python, `__init__` is a special method called by the constructor.

Initializing the Fields of an Object (Folder)

- Creating a new `Worker` is a multi-step process:
  - `w = new Worker()`
    - Instance is empty
  - `w.lname = 'White'`
  - ...
- Want to use something like
  - `w = Worker('White', 1234, None)`
  - Create a new `Worker` and assign fields
    - `lname` to 'White', `ssn` to 1234, and `boss` to None
  - Need a custom constructor