Announcements

Prelim 1
...can be picked up in lab this week.
Solutions will be posted this week, after all makeups are complete.

Regrades
If you find an error in grading, write down the issue clearly on a separate note, attach it to your exam book, and hand it to us in class before March 29.
Recall: Objects as Data in Folders

• An object is like a **manila folder**
• It can contain variables
  - Variables are **attributes**
  - Can change values of an attribute (with assignment statements)
• It has a “tab” that identifies it
  - Unique identifier assigned by Python
  - This is fixed for the lifetime of the object
Recall: **Classes are Types for Objects**

- Objects must have types
  - Some types are built in (float, int, file, list, …)
  - Other types are defined by **classes**
  - Classes are how we add new types to Python

<table>
<thead>
<tr>
<th>id1</th>
<th>Point</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>id8</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>6</td>
<td>5</td>
<td><strong>id1</strong></td>
</tr>
</tbody>
</table>

- **Types**
  - int
  - float
  - bool
  - str

- **Classes**
  - Point
  - RGB
  - Turtle
  - Window

A class object contains only named attributes.

A list contains indexed items.
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*

- **Example**: `p.distanceTo(q)`
  - Both `p` and `q` act as arguments
  - Very much like `distanceTo(p, q)`
Machinery vs. use of machinery

- Classes in Python provide some very simple machinery, and very few constraints on how you use it.
- Learning to program with classes in Python means learning two things:
  1. how the machinery works (this lecture)
  2. some ways to use the machinery effectively (next lecture)
The 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.

**Example**

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

- You can create instances of the class:
  
  e = Example()  
  - Creates a new, empty object

- and access attributes of the class:

  Example.a = 29
  print Example.a  
  - Writing to one creates a new attribute in the class

- and access attributes of an instance:

  e.b = 42
  print e.b  
  - Rule: look first in the instance, then the class
    - Writing to one creates a new attribute in the instance

- and that’s pretty much it!
Populating a class with methods

Everything defined in the class definition creates attributes of the class.

```python
class Example2(object):
    """A class that defines some things."""

    # This is a class variable.
    a = 29

    # This is a method that writes to an instance variable.
    def set_b(self, x):
        self.b = x

    # This is a method that reads from a class variable and an instance variable.
    def f(self):
        return self.a * self.b
```

Every method has a special first parameter `self` that receives a reference to the instance the method was called on.

A variable that lives in a class is a class variable.

A function that lives in a class defines a method.

This assignment will create an instance variable.

Example2

```
a 29
set_b()
f()
```
Method calls

Given class definition from previous slide:

\[ e = \text{Example2}() \]
- constructor expression assigned to \( e \)
- creates a new instance, stores ID in \( e \)

\[ e.\text{set\_b}(42) \]
- method call has object + one argument
- turns into function call with 2 arguments
- value of \( e \) passed to \( \text{self} \); 42 passed to \( x \)
- assignment to \( \text{self}.b \) creates instance var.

\[ \text{print } e.f() \]
- method call has object + no arguments
- turns into function call with 1 arguments
- value of \( e \) passed to \( \text{self} \)
- attribute references find \( \text{self}.a \) in class, \( \text{self}.b \) in instance
Initializing instances

- Instances are initially empty.
- Usually we want to immediately add some instance variables.
- To make this easy, Python will automatically call a method named `__init__` (if you declared one) right after creating an object, before the constructor call returns.

```python
class Worker(object):
    """An instance is a worker in a certain organization.
    Instances have these variables:
    lname [string]: Last name
    ssn [int]: Social security
    boss [Worker]: Immediate boss
    ""
    def __init__(self, lname, ssn, boss):
        self.lname = lname
        self.ssn = ssn
        self.boss = boss

w = Worker("Obama", 1234, None)
```

this statement creates a new Worker instance, calls `__init__` to set it up, and stores the name into `w`. Note two underscores gives access to the instance being initialized.
Aside: The value None

- The boss field is a problem.
  - boss is supposed to refer to a Worker object
  - But some workers might not have a boss
  - Maybe not assigned yet, maybe the buck stops there.

- **Solution**: use value `None`
  - `None`: Lack of (folder) name
  - Will reassign the field later!

- Be careful with `None` variables
  - `var3`.x gives error!
  - There is no name in `var3`
  - Which Point to use?
Evaluating a Constructor Expression

Worker('Obama', 1234, None)

1. Create a new object (folder) that is an instance of the class
   - Instance is initially empty
2. Call the method `__init__`
   (if it exists)
   - Pass folder ID to self
   - Pass other arguments in order
3. Returns the object (folder) name as final value of expression
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(1,2)`   # (1,2,0)
  ▪ `p = Point(y=3)`   # (0,3,0)
  ▪ `p = Point(1,z=2)`   # (1,0,2)

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

    def __init__(self, x=0, y=0, z=0):
        self.x = float(x)
        self.y = float(y)
        self.z = float(z)
    ...
```
### 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:**
  - \texttt{p = Point()} # (0,0,0)
  - \texttt{p = Point(x=1, y=2)} # (1,2,0)
  - \texttt{p = Point(y=3)} # (0,3,0)
  - \texttt{p = Point(1, z=2)} # (1,0,2)

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

    def __init__(self, x=0, y=0, z=0):
        self.x = float(x)
        self.y = float(y)
        self.z = float(z)
```

- Assigns in order
- Use parameter name when out of order
- Can mix two approaches

Not limited to methods. Can do with any function.
What does `str()` do on class objects?

- Does **NOT** display contents
  ```python
  >>> p = Point(1,2,3)
  >>> str(p)
  '<Point object at 0x1007a90>'
  ```
- To display contents, you must implement a special method called `__str__`
- With the defns. on these slides:
  ```python
class Point(object):
    """Instances are points in 3d space"""
    ...
    def __str__(self):
      """Returns: string with contents"""
      return ('(' + self.x + ',' +
               self.y + ',' +
               self.z + ')')
  ```
  print `Point(3,4,5)`
  produces the output:
  `(3.0,4.0,5.0)`
### Important!

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong> Point(object):</td>
<td><strong>class</strong> Point:</td>
</tr>
<tr>
<td>&quot;&quot;&quot;Instances are 3D points x [float]: x coord y [float]: y coord z [float]: z coord&quot;&quot;&quot;</td>
<td>&quot;&quot;&quot;Instances are 3D points x [float]: x coord y [float]: y coord z [float]: z coord&quot;&quot;&quot;</td>
</tr>
</tbody>
</table>

---

**3.0-Style Classes**
Well-designed

---

**“Classic” Classes**
No reason to use these