Announcements

**TA Midterm Evals**
The University wants your feedback about the TAs. You should hear from them about this mid-term survey. Please participate!

**Upcoming assignments**
A tentative schedule of due dates is now posted. A3 is planned to go out this week.
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>
```

```
id8
<table>
<thead>
<tr>
<th></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>id1</td>
</tr>
</tbody>
</table>
```

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

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

**Example**

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

• You can create *instances* of the class:

  ```python
e = Example()
  ```

  • Creates a new, empty object

  • and access *attributes* of the class:

    ```python
    Example.a = 29
    print Example.a
    ```

    • Writing to one creates a new attribute in the class

  • and access *attributes* of an instance:

    ```python
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!

  • a “constructor expression”

  • not the way we normally create class attributes! …more later

  • not the way we normally create instance attributes! …more later
Everything defined in the class definition creates attributes of the class.

Every method has a special first parameter \texttt{self} that receives a reference to the instance the method was called on.

```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
```

A variable that lives in a class is a \textit{class variable}.

A function that lives in a class defines a \textit{method}.

This assignment will create an \textit{instance variable}.

Example 2

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

Given class definition from previous slide:

```
e = Example2()

- constructor expression assigned to e
- creates a new instance, stores ID in e
```

```
e.set_b(42)

- method call has object + one argument
- turns into function call with 2 arguments
- value of e passed to self; 42 passed to x
- assignment to self.b creates instance var.
```

```
print e.f()

- method call has object + no arguments
- turns into function call with 1 arguments
- value of e passed to self
- attribute references find self.a in class, 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.)
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)
```

- 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 \texttt{str()} do on class objects?

- Does **NOT** display contents
  
  \[
  \begin{align*}
  &>>> \texttt{p} = \texttt{Point(1,2,3)} \\
  &\texttt{str(p)} \\
  &'\texttt{<Point object at 0x1007a90>}'
  \end{align*}
  \]

- To display contents, you must implement a special method called \texttt{__str__}

- With the defns. on these slides:
  
  class Point(object):
  
  \begin{quote}
  "Instances are points in 3d space"
  \end{quote}
  
  def \texttt{__str__}(self):
  
  \begin{quote}
  "Returns: string with contents"
  \end{quote}
  
  \begin{verbatim}
  return ('(' + self.x + ',' + self.y + ',' + self.z + ')')
  \end{verbatim}

  print \texttt{Point(3,4,5)}

  produces the output:

  (3.0,4.0,5.0)
Important!

YES

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

"""Classic""" Classes
Well-designed

NO

```python
class Point:
    """Instances are 3D points
    x [float]: x coord
    y [float]: y coord
    z [float]: z coord"
```

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