Introduction to Python: Object Oriented Programming

Christopher Barker

UW Continuing Education / Isilon

August 01, 2012

Table of Contents

Lightning Talks

Lightning Talks today:

Brett and Matt

Review of Previous Class

- Built an HTTP server
- Very basics of sockets
- Basics of HTTP protocol
- Got a server working (even proto-CGI!)

review of my `http_serve8.py`

Object Oriented Programming

More about Python implementation than OO design/strengths/weaknesses

One reason for this:
Folks can't even agree on what OO "really" means

The Quarks of Object-Oriented Development - Deborah J. Armstrong:
`http://agp.hx0.ru/oop/quarks.pdf`

Object Oriented Programming

Is Python a "True" Object-Oriented Language?

(Doesn't support full encapsulation, doesn't require objects, etc...)

Object Oriented Programming

I don't Care!

Good software design is about code-reuse, clean separation of concerns, refactorability, testability, etc...

OO can help with all that, but:

- it doesn't guarantee it
- it can get in the way

Object Oriented Programming

Python is a Dynamic Language

That clashes with "pure" OO

Think in terms of what makes sense for you project – not any one paradigm of software design.

Object Oriented Programming

OO for this class:

"Objects can be thought of as wrapping their data within a set of functions designed to ensure that the data are used appropriately, and to assist in that use"

`http://en.wikipedia.org/wiki/Object-oriented_programming`

Object Oriented Programming

Even simpler:

Objects are data and the functions that act on them in one place.

In Python: just another namespace.

Object Oriented Programming

The OO buzzwords:

- data abstraction
- encapsulation
- messaging
- modularity
- polymorphism
- inheritance

Object Oriented Programming

You can do OO in C

(see the GTK+ project)

"OO languages" give you some handy tools to make it easier (and safer).

- polymorphism (duck typing gives you this anyway)
- inheritance

Object Oriented Programming

OO is the dominant model for the past couple decades

You will need to use it:

– It's a good idea for a lot of problems

– You'll need to work with OO packages

Object Oriented Programming

Some definitions

class A category of objects: particular data and behavior: A circle (same as a type in python)

instance A particular object of a class: a specific circle

object the general case of a instance – really any value (in Python anyway)

attribute something that belongs to an object (or class) – generally thought of as a variable, or single object, as apposed to a ...

method a function that belongs to a class

Python Classes

The `class` statement

`class` creates a new type object:

```
In [4]: class C(object):
    pass
   ...:
In [5]: type(C)
Out[5]: type
```

It is created when the statement is run – much like `def`

(note on "new style" classes)

Python Classes

Note about the book (TP):

Chapters 15 and 16 use a style that generally isn't recommended:

```
In [6]: class Point(object):
   ...:     pass
In [7]: p = Point()
In [8]: p.x = 4
In [9]: p.y = 2
```

Python is Dynamic – you can do this, but you generally want more structure, defaults, etc.

(it used to be a quick and dirty "struct" – but use a named tuple now)

Python Classes

Basic Structure

```
class Point(object):
#( everything defined in here is in the class namespace)
    def __init__(self, x, y):
        self.x = x
        self.y = y

## create an instance of that class
p = Point(3,4)

## access the attributes
print "p.x is:", p.x
print "p.y is:", p.y
```

see: `simple_class` in code dir

Python Classes

The Initializer

The `__init__` special method is called when a new instance of a class is created.

You can use it to do any set-up you need

```
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

It gets the arguments passed to the class constructor

Python Classes

`self`

The instance of the class is passed as the first parameter for every method.

"`self`" is only a convention – but you DO want to use it.

```
class Point(object):
    def a_function(self, x, y):
...
```

Does this look familiar from C-style procedural programming?