

# Introduction to Python

## More OO: Special methods, magic methods

## Iterators and Generators

Christopher Barker

UW Continuing Education

November 19, 2013

# Table of Contents

- 1 Review/Questions
- 2 Properties
- 3 Special Attributes
- 4 Iterators / Generators

## Schedule...

Three more classes (including this one)!

No class next week: Thanksgiving!

Extra time to work on project...

# Desktop GUIs

A number of people are interested in desktop GUIs

No time to cover that in class

Extra class T-day week on wxPython?

# Review of Previous Class

- Object oriented programing
- Classes, subclasses, instances.
- The html generator

# Lightning Talks

Lightning talks today:

Luke Cypret

Blane Moore

Brent Parrish

# Accessing Attributes

One of the strengths of Python is lack of clutter

Simple attributes:

```
In [5]: class C(object):  
        def __init__(self):  
            self.x = 5
```

```
In [6]: c = C()
```

```
In [7]: c.x
```

```
Out[7]: 5
```

```
In [8]: c.x = 8
```

# Getter and Setters?

What if you need to add behavior later?

- do some calculation
- check data validity
- keep things in sync



## Getter and Setters?

```
class C(object):
    def get_x(self):
        return self.x
    def set_x(self, x):
        self.x = x

>>> c = C()
>>> c.get_x()
>>> 5
>>> c.set_x(8)
>>> c.get_x()
>>> 8
```

Ugly and verbose – Java?

<http://dirtsimple.org/2004/12/python-is-not-java.html>

# properties

When (and if) you need them:

```
class C(object):  
    def _getx(self):  
        return self._x  
    def _setx(self, value):  
        self._x = value  
    def _delx(self):  
        del self._x  
    x = property(_getx, _setx, _delx, doc="docstring")
```

Interface is still like simple attribute access

(demo: properties\_sample.py )

## staticmethod

A method that doesn't get self!

```
class C(object):  
    def add(a, b):  
        return a + b  
    add = staticmethod(add)  
  
>>> C.add(3,4)  
7  
>>> c = C()  
>>> c.add(2, 2)  
4
```

When you don't need self – can be used from either an instance or the class itself

see: `static_method.py`

## classmethod

Method gets the class object, rather than an instance, as the first argument

```
class C(object):  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
    def a_class_method(klass, y):  
        print "in a_class_method", klass  
        return klass( y, y**2 )  
    a_class_method = classmethod(a_class_method)
```

When you need the class object rather than an instance – plays well with subclassing

see: `class_method.py`

# dict.fromkeys()

classmethod often used for alternate constructors:

```
>>> d = dict([1,2,3])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot convert dictionary update
sequence element #0 to a sequence
>>> d = dict.fromkeys([1,2,3])
>>> d
{1: None, 2: None, 3: None}
```

# dict.fromkeys()

```
class Dict: ...
    def fromkeys(klass, iterable, value=None):
        "Emulate dict_fromkeys() in dictobject.c"
        d = klass()
        for key in iterable:
            d[key] = value
        return d
    fromkeys = classmethod(fromkeys)
```

See also `datetime.datetime.now()`, etc....

For a low-level look:

<http://docs.python.org/howto/descriptor.html>

## LAB

Write a simple “Circle” class:

```
In [13]: c = Circle(radius=3)
```

```
In [15]: c.diameter
```

```
Out[15]: 6
```

```
In [16]: c.diameter = 8
```

```
In [17]: c.radius
```

```
Out[17]: 4.0
```

```
In [18]: c.area
```

```
Out[18]: 50.26548245743669
```

Use properties so you can keep the radius and diameter in sync, and the area computed on the fly.

Extra Credit: use a class method to make an alternate constructor that takes the diameter instead.

code/circle.py and code/test\_circle1.py

# Lightning Talks

Lightning Talk:

Luke Cypret

Blane Moore



## “magic” methods

Python’s Duck typing:

Defining special (or magic) methods in your classes  
is how you make your class act like standard classes

## special methods

We've seen at least one:

```
__init__
```

It's all in the double underscores...

Pronounced “dunder” (or “under-under”)

try: `dir(2)` or `dir(list)`

## special methods

### Emulating Numeric types

```
object.__add__(self, other)
object.__sub__(self, other)
object.__mul__(self, other)
object.__floordiv__(self, other)
object.__mod__(self, other)
object.__divmod__(self, other)
object.__pow__(self, other[, modulo])
object.__lshift__(self, other)
object.__rshift__(self, other)
object.__and__(self, other)
object.__xor__(self, other)
object.__or__(self, other)
```

## special methods

### Emulating container types:

```
object.__len__(self)
object.__getitem__(self, key)
object.__setitem__(self, key, value)
object.__delitem__(self, key)
object.__iter__(self)
object.__reversed__(self)
object.__contains__(self, item)
object.__getslice__(self, i, j)
object.__setslice__(self, i, j, sequence)
object.__delslice__(self, i, j)
```

# special methods

Example – to define addition:

```
def __add__(self, v):  
    """  
    redefine + as element-wise vector sum  
    """  
    assert len(self) == len(v)  
    return vector([x1 + x2 for x1, x2 in zip(self, v)])
```

( from a nice complete example in code/vector.py )

## special methods

You only need to define the ones that are going to get used

But you probably want to define at least these:

`object.__str__`: Called by the `str()` built-in function and by the print statement to compute the informal string representation of an object.

`object.__repr__`: Called by the `repr()` built-in function and by string conversions (reverse quotes) to compute the official string representation of an object.

(ideally: `eval( repr(something) ) == something` )

## special methods

When you want your class to act like a "standard" class in some way:

Look up the magic methods you need and define them

<http://docs.python.org/reference/datamodel.html#special-method-names>

<http://www.rafekettler.com/magicmethods.html>

## LAB

Extend your “Circle” class:

Add `__str__` and `__repr__` methods

Write an `__add__` method so you can add two circles  
(and multiply by a number....)

```
In [22]: c1 = Circle(3)
```

```
In [23]: c2 = Circle(4)
```

```
In [24]: c3 = c1+c2
```

```
In [25]: c3.radius
```

```
Out[25]: 7
```

```
In [26]: c1*3
```

```
Out[26]: Circle(9)
```

If you have time: compare them... (`c1 > c2`, etc)

code/circle.py and code/test\_circle2.py



# Lightning Talk

Lightning Talk:

Brent Parrish

# Iterators

Iterators are one of the main reasons Python code is so readable:

```
for x in just_about_anything:  
    do_stuff(x)
```

you can loop through anything that satisfies the iterator protocol

<http://docs.python.org/library/stdtypes.html#iterator-types>

# Iterator Protocol

An iterator must have the following methods:

```
iterator.__iter__()
```

Return the iterator object itself. This is required to allow both containers and iterators to be used with the `for` and `in` statements.

```
iterator.next()
```

Return the next item from the container. If there are no further items, raise the `StopIteration` exception.

## Example Iterator

A simple version of xrange()

```
class IterateMe_1(object):
    def __init__(self, stop=5):
        self.current = 0
        self.stop = stop
    def __iter__(self):
        return self
    def next(self):
        if self.current < self.stop:
            self.current += 1
            return self.current
        else:
            raise StopIteration
```

# itertools

`itertools` is a collection of utilities that make it easy to build an iterator that iterates over sequences in various common ways

<http://docs.python.org/library/itertools.html>

## LAB

- Extend (`iterator_1.py`) to be more like `xrange()` – add three input parameters:  
`iterator_2(start, stop, step=1)`
- See what happens if you break out in the middle of the loop:

```
it = IterateMe_2(2, 20, 2)
for i in it:
    if i > 10: break
    print i
```

And then pick up again:

```
for i in it:
    print i
```

- Does `xrange()` behave the same?  
– make yours match `xrange()`.

# generators

Generators give you the iterator immediately: no access to the underlying data ... if it even exists

## **Conceptually:**

iterators are about various ways to loop over data,  
generators generate the data on the fly

## **Practically:**

You can use either either way (and a generator is one type of iterator)

Generators do some of the book-keeping for you.

# yield

yield is a way to make a quickie generator with a function:

```
def a_generator_function(params):  
    some_stuff  
    yield(something)
```

Generator functions "yield" a value, rather than returning it

State is preserved in between yields



# yield

A function with `yield` in it is a “factory” for a generator

Each time you call it, you get a new generator:

```
gen_a = a_generator()  
gen_b = a_generator()
```

Each instance keeps its own state.

Really just a shorthand for an iterator class that does the book keeping for you.

# yield

An example: like xrange()

```
def y_xrange(start, stop, step=1):  
    i = start  
    while i < stop:  
        yield i  
        i += step
```

Real World Example: FloatCanvas

# yield

Note:

```
In [164]: gen = y_xrange(2,6)
```

```
In [165]: type(gen)
```

```
Out[165]: generator
```

```
In [166]: dir(gen)
```

```
Out[166]:
```

```
...
```

```
  '__iter__',
```

```
...
```

```
  'next',
```

So the generator **is** an iterator

# yield

A generator function can also be a method in a class

More about iterators and generators:

<http://www.learningpython.com/2009/02/23/iterators-iterables-and-generators-oh-my/>

`yield_example.py`

# generator comprehension

another way to make a generator:

```
>>> [x * 2 for x in [1, 2, 3]]  
[2, 4, 6]  
>>> (x * 2 for x in [1, 2, 3])  
<generator object <genexpr> at 0x10911bf50>  
>>> for n in (x * 2 for x in [1, 2, 3]):  
...     print n  
... 2 4 6
```

More interesting if `[1, 2, 3]` is also a generator

# LAB

## Generator lab:

Write a few generators:

- Sum of integers
- Doubler
- Fibonacci sequence
- Prime numbers

`code/generatorLAB.html`  
`code/test_generator.py`

# Homework

Finish the labs

Project Proposals!

You should have a good start on your project by the end of this week

Remember: no class next week!