# Introduction to Python
# Decorators, Context Managers,
# Packages and Packaging

Christopher Barker

UW Continuing Education

December 3, 2013

# Table of Contents

## Review of Previous Class

- Magic methods
- Iterators
- Generators
- (wxPython)

## Lightning Talks

Lightning talks today:

Harlan AuBuchon

Brian Schmitz

## Review

Questions about labs?

My Solutions?

## A diversion...

A number of you are already using `iPython`

It's a very useful tool

And the `iPython` notebook is even cooler .. paticularly for in-class demos.

So I'll use it some today:

```
http://ipython.org/ipython-doc/dev/
interactive/notebook.html
```

## Decorators

Decorators are wrappers around functions

They let you add code before and after the execution of a function

Creating a custom version of that function

## Decorators

# Syntax:

```
@logged
def add(a, b):
    """add() adds things"""
    return a + b
```

Demo and Motivation:
`code/decorators/basic_math.py` [ipnb]

PEP: http://www.python.org/dev/peps/pep-0318/

## Decorators

@ decorator operator is an abbreviation:

```
@f
def g:
    pass
```

same as

```
def g:
    pass
g = f(g)
```

"Syntactic Sugar" – but really quite nice

## Decorators

demo:

`decorator.py`

## Decorator examples

Examples from the stdlib:

Does this structure:

```
def g:
    pass
g = f(g)
```

look familiar from last class?

## Decorator examples

```
staticmethod()


class C(object):
    def add(a, b):
        return a + b
    add = staticmethod(add)
```

## Decorator examples

staticmethod()

Decorator form:

```
class C(object):
    @staticmethod
    def add(a, b):
        return a + b
```

( and classmethod )

## examples

# property()

```
class C(object):
    def __init__(self):
        self._x = None
    def getx(self):
        return self._x
    def setx(self, value):
        self._x = value
    def delx(self):
        del self._x
    x = property(getx, setx, delx,
                "I'm the 'x' property.")
```

becomes...

## Decorator examples

```
class C(object):
    def __init__(self):
        self._x = None
    @property
    def x(self):
        return self._x
    @x.setter
    def x(self, value):
        self._x = value
    @x.deleter
    def x(self):
        del self._x
```

Puts the info close to where it is used

## examples

# CherryPy

```
import cherrypy
class HelloWorld(object):
    @cherrypy.expose
    def index(self):
        return "Hello World!"
cherrypy.quickstart(HelloWorld())
```

## examples

# Pyramid

```
@template
def A_view_function(request)
    .....
@json
def A_view_function(request)
    ......
```

so you don't need to think about what your view is returning...

## decorators...

For this class:

Mostly want to you to know how to use decorators that someone else has written

Have a basic idea what they do when you do use them

But writing a couple will help you "get" it, and help cement your Python knowledge...

## Writing Decorators

So how to you write one?

demo in iPython notebook

code\decorators\DecoratorDemo.py

For more detail: (and talks about closures...):
http://simeonfranklin.com/blog/2012/jul/1/
python-decorators-in-12-steps/

## LAB

- Re-write the properties from last week's `Circle` class to use
  the decorator syntax (see a couple slides back for an example)
  (`circle_properties.py` and
  `test_circle_properties.py`)
- Write a decorator that can be used to wrap any function that
  returns a string in a `<p>` element – auto-generation of simple
  html. (`p_wrapper.py`)
- Try using a class to make a decorator that will wrap a
  specified tag around a function that returns a string:

  ```
  @tag_wrapper('h1')
  def func2(x, y=4, z=2):
      return "the sum of %s and %s and %s is %s"%(x, y
  >>> print func2(3,4)
  <h1> the sum of 3 and 4 and 2 is 9 </h1>
  ```

Lightning Talk

Lightning Talk:

Harlan AuBuchon

## Context Managers

the `with` statement

A class with `__enter__()` and `__exit__()` methods.

`__enter__()` is run before your block of code

`__exit__()` is run after your block of code

Can be used to setup/cleanup before and after: open/closing files, db connections, etc

## Context Managers

"PEP 343: the `with` statement"
    – A.M. Kuchling
http://docs.python.org/dev/whatsnew/2.6.html#
pep-343-the-with-statement

"Understanding Python's `with` statement"
    – Fredrik Lundh
http://effbot.org/zone/python-with-statement.htm

"The Python `with` Statement by Example"
    – Jeff Preshing
http://preshing.com/20110920/
the-python-with-statement-by-example

## Context Managers

Use syntax:

```
with manager as something:
    a = block_of_code
    use_something_here(something)
    ...
```

manager is the context manager: i.e. has an __enter__ and
__exit__ method – if __enter__ returns an object, it gets
assigned to something

## Context Managers

The file object is also a context manager:

```
with open(filename) as the_file:
    for line in the_file:
        work_with(line)
        ...
    ...
```

In this case, the file will automatically be closed when you leave that block, regardless of errors, etc.

Most commonly used context manager – by far!

## Context Managers

You also may hav seen this in some of my unit tests:

```
with pytest.raises(ZeroDivisionError):
    some_test_code_here
    1/0
```

Context Managers can also catch Exceptions....

## LAB

See if you can write a context manger that will time
some code.
When using it, you can do:

```
with timer:
    this_is_some_code_to_run()
    how_long_might_it_take
```

and you'll get something like:

```
this code took 0.12 seconds
```

See: context_manager\timer_context.html
(timer_context.py)

## Lightning Talk

Lightning Talk:

Brian Schmitz

## Modules and Packages

A module is a file with python code in it

A package is a directory with an `__init__.py` file in it

And usually other modules, packages, etc...

```
my_package
    __init__.py
    module_a.py
    module_b.py

import my_package

runs my_package/__init__.py
```

## Modules and Packages

```
import sys

for p in sys.path:
    print p
```

(demo)

## Installing Python

Linux:
Usually part of the system – just use it

Windows:

Use the `python.org` version:

System Wide

Can install multiple versions if need be

Third party binaries for it.

## Installing Python

OS-X:

Comes with the system, but:

- Apple has never upgraded within a release
- There are non-open source components
- Third party packages may or may not support it
- Apple does use it – so don't mess with it.
- I usually recommend the python.org version

(Also Macports, Fink, Home Brew...)

## Distributions

There are also a few "curated" distributions:

These provide python and a package management system for hard-to-buid packages.

Widely used by the scipy community (lots of hard to build stuff that needs to work together...)

- Anoconda
  (https://store.continuum.io/cshop/anaconda/)
- Canopy
  (https://www.enthought.com/products/canopy/)
- ActivePython
  (http://www.activestate.com/activepython)

## Installing Packages

Every Python installation has its own stdlib and
`site-packages` folder

`site-packages` is the default place for third-party
packages

## Finding Packages

The Python Package Index:

PyPi

`http://pypi.python.org/pypi`

## Installing Packages

From source (`setup.py install`)

With the system installer (apt-get, yum, etc...)

From binaries:

Windows: MSI installers

OS-X: dmg installers

And now: binary wheels
(make sure to get compatible packages)

`easy_install` and `pip`

## Installing Packages

In the beginning, there was the `distutils`:

. . . .

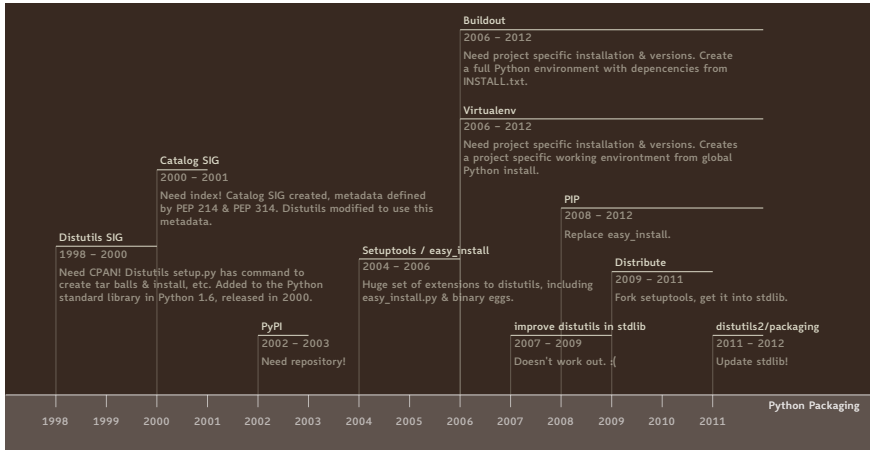But `distutils` is missing some key features:

- package versioning
- package discovery
- auto-install

- And then came `PyPi`
- And then came `setuptools`
- But that wasn't well maintained...
- Then there was `distribute`/`pip`
- Which has now been merged back into `setuptools`

## Installing Packages

Actually, it's still a bit of a mess

But getting better...

# Packaging Time line

## Packaging Tools

## Current State of Packaging

To build packages: distutils
`http://docs.python.org/2/distutils/`

For more features: setuptools
`http://pythonhosted.org/setuptools/`

To install packages: pip
`http://www.pip-installer.org/en/latest/`

For binary packages: wheels
`http://www.python.org/dev/peps/pep-0427/`

## Compiled Packages

Biggest issue is with compiled extensions
$$(C/C++, etc)$$

– You need the right compiler set up

Dependencies

– Here's were it gets really ugly

– Particularly on Windows

## Compiled Packages

Linux

Pretty straightforward:

1) Is there a system package

(rpm, deb, apt-get, etc...)?

2) Install the dependencies, build from source:

`python setup.py build ; python setup.py instal`

( Or maybe `pip install` will just work )

## Compiled Packages

Windows
Sometimes simpler:

1) A lot of packages have Windows binaries:

- Usually for `python.org` builds

- Excellent source:
`http://www.lfd.uci.edu/~gohlke/pythonlibs/`

- Make sure you get 32 or 64 bit consistent

2) But if no binaries:

- Hope the dependencies are available!

- Set up the compiler (MS VS2008 Express works)

## Compiled Packages

OS-X

Lots of Python versions:

- Apple's built-in (different for each version of OS)

- `python.org` builds.
    - 32 bit PPC+Intel
    - 32+64 bit Intel

- Macports - Homebrew

Binary Installers (dmg or wheel) have to match python version

## Compiled Packages

OS-X

If you have to build it yourself:

Xcode compiler (the right version):

- Version 3.* for 32 bit PPC+Intel

- Version 4.* for 32+64 bit Intel

If extra dependencies:

- macports or home brew often easiest way to build them

## Final Recommendation

First try: `pip install`

If that doesn't work:

Read the docs of the package you want to install

Do what they say

## virtualenv

virtualenv is a tool to create isolated Python environments.

Very useful for developing multiple apps

Or deploying more than one on one system

http://www.virtualenv.org/en/latest/index.html

(Cris will get into more detail with this next class)

## Distributing

What if you need to distribute you own:

Scripts

Libraries

Applications

## Scripts

Often you can just copy, share, or check in the script to source control and call it good.

But only if it's a single file, and doesn't need anything non-standard

## Scripts

When the script needs more than just the stdlib
(or your company standard environment)

You have an application, not a script

## Libraries

When you read the distutils docs, it's usually libraries they're talking about

Scripts + library is the same...

(http://docs.python.org/distutils/)

## distutils

distutils makes it easy to do the easy stuff:

Distribute and install to multiple platforms, etc.

Even binaries, installers and compiled packages

(Except dependencies)

(http://docs.python.org/distutils/)

## distutils basics

It's all in the `setup.py` file:

```
from distutils.core import setup
setup(name='Distutils',
      version='1.0',
      description='Python Distribution Utilities',
      author='Greg Ward',
      author_email='gward@python.net',
      url='http://www.python.org/sigs/distutils-sig/',
      packages=['distutils', 'distutils.command'],
      )
```

(http://docs.python.org/distutils/)

## distutils basics

Once your setup.py is written, you can:

```
python setup.py ...

build          build everything needed to install
install        install everything from build directory
sdist          create a source distribution
               (tarball, zip file, etc.)
bdist          create a built (binary) distribution
bdist_rpm      create an RPM distribution
bdist_wininst  create an executable installer for MS Windows
upload         upload binary package to PyPI
```

## More complex packaging

For a complex package:

You want to use a well structured setup:

`http://guide.python-distribute.org/creation.html`

## develop mode

While you are developing your package, Installing it is a pain.

But you want your code to be able to import, etc. as though it were installed

`setup.py develop` installs links to your code, rather than copies – so it looks like it's installed, but it's using the original source

`python setup.py develop`

You need `setuptools` to use it.

## Applications

For a complete application:

- Web apps
- GUI apps

Multiple options:

- Virtualenv + VCS
- zc.buildout ( http://www.buildout.org/ )
- System packages (rpm, deb, ...)
- Bundles...

## Bundles

Bundles are Python + all your code + plus all the
dependencies – all in one single "bundle"
Most popular on Windows and OS-X

```
py2exe
py2app
pyinstaller
```
. . .

User doesn't even have to know it's python

Examples:

http://www.bitpim.org/
http://response.restoration.noaa.gov/nucos

## LAB

Write a setup.py for a script of yours

- Ideally, your script relies on at least one other module
- At a minimum, you'll need to specify `scripts`
- and probably `py_modules`
- try:
    - `python setup.py build`
    - `python setup.py install`
    - `python setup.py sdist`
    - `python setup.py bdist_wininst`
- EXTRA: install setuptools
    - use: `from setuptools import setup`
    - try: `python setup.py develop`

  (my example: capitalize Package)

## Homework

Finish any labs...

Your project

Next week:

Cris Ewing will come and talk about the next quarter