Introduction to Python
Decorators – Debugging – Packages and Packaging

Christopher Barker

UW Continuing Education / Isilon

August 22, 2012

Table of Contents

Review of Previous Class

Lightning talk today: Peter

- Some more OO
  - Multiple inheritance / mix-ins
  - Properties
  - `staticmethod` and `classmethod`
  - Special methods ("dunder")
- Iterators
- Generators

Homework review

Who added some classes to some “real” code?

- Multiple inheritance / mix-ins ?
- Property ?
- `staticmethod` or `classmethod` ?
- Special methods ?
- Iterator or Generators ?

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: `basicmath.py`

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

LAB

- Re-write the properties from last week's `Circle` class to use the decorator syntax (see a couple slides back for an example)
- Write a decorator that can be used to wrap any function that returns a string in a <p> element from the html builder from the previous couple classes (the `P Element` subclass).

Lightning Talk

Lightning Talk:

Peter

Debugging

Debugging

Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program

We often spend more time debugging that we do writing the code in the first place

Debugging

Core Message:

Well structured code is less prone to bugs

Well structured code is easier to debug

Types of Bugs

- Syntax Errors
- Run Time Errors
- Logic Errors

(Usually show up in that order)

Syntax Errors

Common Causes

- Mismatched parenthesis, quotes, brackets, etc...
- Missing colons
- "=" vs "=="
- Indentation
- Using a keyword for a variable name

Hint: Make sure you are editing the same file you are running!

Runtime Errors

This may seem obvious, but...

Read the traceback carefully!

- What type of error?
  - ValueError
  - TypeError
  - NameError
  - Think about what that type of error means
- What module/function did it occur in?
- What line did it occur?
- Where was that function called from?

Logic Errors

No hints from the interpreter

- Make sure the code you think is executing is really executing
- Simplify your code
- Boil it down to the simplest version that shows the bug
    - Often you'll find it in the process
- Save (and print) intermediate results from long expressions
- Try out bits of code at the command line (or iPython)

Debugging Tools

Print statements

Interactive debuggers

Logging

Tests

Print Statements

Simple

Easy to understand

Quick (with no compile cycle)

Nice if something fails the 1000th time through a loop...

(I do most of my debugging with print statements)

Logging

"enterprise level print statements"

Standard library logging module

powerful, awesome, and a bit annoying

`http://docs.python.org/library/logging.html`
`http://docs.python.org/howto/logging.html#logging-basic-tutorial`

Logging Module

Using the standard logging module means you can share your logging with third party packages, etc.

- Customized levels
- String interpolation
- On the fly configuration
- etc, etc..

Logging Module

Output options:

- StreamHandler
- FileHandler
- BaseRotatingHandler
- RotatingFileHandler
- TimedRotatingFileHandler
- SocketHandler
- SMTPHandler
- SysLogHandler
- HTTPHandler
- NullHandler
- ...

Tests

Test Suites Find Bugs

And keep them from recurring

You can get closer to the bug by writing more tests

Tests

Test Suites are particularly helpful for Heisenbugs:

heisenbug: /hi-zen-buhg/, n.

A bug that disappears or alters its behavior when one attempts to probe or isolate it.

`http://www.catb.org/jargon/html/H/heisenbug.html`

More on testing next class

Interactive Debuggers

PDB

- in stdlib
- command line
- local
- in process

PDB

(I've never used it much – but ...)

Python Debugging Techniques
`http://aymanh.com/python-debugging-techniques`

Use pdb to debug Django (screencast):
`http://ericholscher.com/blog/2008/aug/31/using-pdb-python-debugger-django-debugging-series-/`

Visual Debuggers

Visual debuggers in many IDEs:

- WingIDE: `http://wingware.com/wingide/debugger`
- PyDEV / Eclipse `http://pydev.org/`
- Spyder `http://code.google.com/p/spyderlib/`
- PyCharm: `http://www.jetbrains.com/pycharm/`
- .....

LAB

PDB lab

Follow this tutorial:

Getting started with pdb:
`http://pythonconquerstheuniverse.wordpress.com/2009/09/10/debugging-in-python/`

Try it on your own code – or class code

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...)

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
(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`

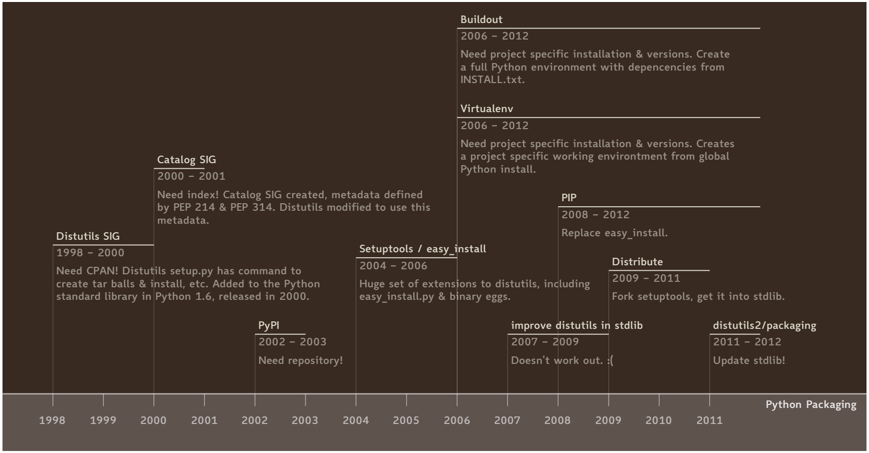
But that wasn't well maintained...

So now there is `distribute/pip`

Installing Packages

Actually, it's a bit of a mess

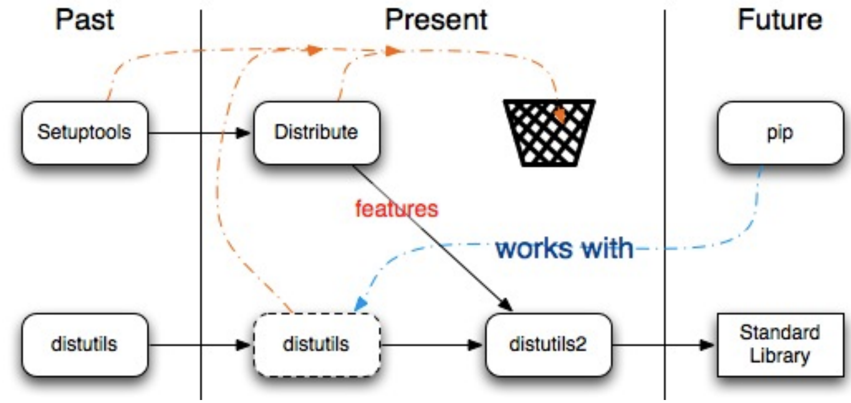
Packaging Time line


Buildout
2006 – 2012
Need project specific installation & versions. Create a full Python environment with depencencies from INSTALL.txt.
Virtualenv
2006 – 2012
Need project specific installation & versions. Creates a project specific working environtment from global Python install.
Catalog SIG
2000 – 2001
Need index! Catalog SIG created, metadata defined by PEP 214 & PEP 314. Distutils modified to use this metadata.
PIP
2008 – 2012
Replace easy_install.
Distutils SIG
1998 – 2000
Need CPAN! Distutils setup.py has command to create tar balls & install, etc. Added to the Python standard library in Python 1.6, released in 2000.
Setuptools / easy_install
2004 – 2006
Huge set of extensions to distutils, including easy_install.py & binary eggs.
Distribute
2009 – 2011
Fork setuptools, get it into stdlib.
PyPI
2002 – 2003
Need repository!
improve distutils in stdlib
2007 – 2009
Doesn't work out. :(
distutils2/packaging
2011 – 2012
Update stdlib!
Python Packaging
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011

Packaging Tools

ATTENTION PYTHON COMRADES
NEW ORDERS FROM THE MINISTRY OF PACKAGING!
SONS AND DAUGHTERS OF THE GLORIOUS
PEOPLE'S PYTHONIC REPUBLIC
USE DISTRIBUTE AND PIP
THAT IS ALL

Current State of Packaging



http://guide.python-distribute.org/introduction.html

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:

```
pythonsetup.py build ; python setup.py install
```

Compiled Packages

Windows

Sometimes simpler:

1) A lot of packages have Windows binaries:

- Usually for `python.org` builds

- Make sure you get 32 or 64 bit correct

2) But if no binaries:

- Hope the dependencies are available!

- Set up the compiler (MS "Express" version usually 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 egg) 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`

virtualenv

keep an eye on

distutils2 / packaging / pysetup

(They are trying to solve the problem!)

(`http://packages.python.org/Distutils2/`)

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 theyre 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

You need `distribute` (or `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 `distribute`
  - use: `from setuptools import setup`
  - try: `python setup.py develop`

Homework

- Find a package or two and install it
- Try to install it "from source" – i.e. `setup.py install|` `Make a nice package of your class project (or something else)`