

Programming in Python

Week 10

Survey of Python testing tools

Syllabus:

http://briandorsey.github.com/uwpython_onsite/

Brian Dorsey
brian@dorseys.org
<http://briandorsey.info>

Introductions,
comments
(...)

my turn?

bit stressed this week

how do I prepare a testing
tools overview for a room
full of professional
testers?

Unit A (25)

Python testing tools overview

testing tools overview

can't really do this

too many

you are

I'm not the expert, but...

preaching to choir, right?

testing is awesome

so let's try

» PythonTestingToolsTaxonomy

» PythonTesti...olsTaxonomy

Contents

1. [Unit Testing Tools](#)
2. [Mock Testing Tools](#)
3. [Fuzz Testing Tools](#)
4. [Web Testing Tools](#)
5. [Acceptance/Business Logic Testing Tools](#)
6. [GUI Testing Tools](#)
7. [Source Code Checking Tools](#)
8. [Code Coverage Tools](#)
9. [Continuous Integration Tools](#)
10. [Automatic Test Runners](#)
11. [Test Fixtures](#)
12. [Miscellaneous Python Testing Tools](#)

lots and lots of tools

**lots of testing
methods & philosophies**

small tests vs. big tests?

some testing words

black box
white box
unit
integration
regression
acceptance

...

performance
usability
security
internationalization

...

test driven development
behavior driven development

...

look back at
testing tools taxonomy

what did you notice?
seems useful?
want to try?
weird / surprising?

» PythonTestingToolsTaxonomy

» PythonTesti...olsTaxonomy

Contents

1. [Unit Testing Tools](#)
2. [Mock Testing Tools](#)
3. [Fuzz Testing Tools](#)
4. [Web Testing Tools](#)
5. [Acceptance/Business Logic Testing Tools](#)
6. [GUI Testing Tools](#)
7. [Source Code Checking Tools](#)
8. [Code Coverage Tools](#)
9. [Continuous Integration Tools](#)
10. [Automatic Test Runners](#)
11. [Test Fixtures](#)
12. [Miscellaneous Python Testing Tools](#)

-
- unittest
 - doctest
 - mock (3.3+)
 - restructured
text (docutils)
 - sphinx docs
 - nose / py.test
 - coverage.py
 - pyflakes
 - pep8

`unittest`

`stdlib`

based on JUnit & XUnit

the standard

a bit verbose

write classes & methods

```
import random
import unittest

class TestSequenceFunctions(unittest.TestCase):

    def setUp(self):
        self.seq = range(10)

    def test_shuffle(self):
        # make sure the shuffled sequence does not lose any elements
        random.shuffle(self.seq)
        self.seq.sort()
        self.assertEqual(self.seq, range(10))

        # should raise an exception for an immutable sequence
        self.assertRaises(TypeError, random.shuffle, (1,2,3))

    def test_choice(self):
        element = random.choice(self.seq)
        self.assertTrue(element in self.seq)

    def test_sample(self):
        with self.assertRaises(ValueError):
            random.sample(self.seq, 20)
        for element in random.sample(self.seq, 5):
            self.assertTrue(element in self.seq)

if __name__ == '__main__':
    unittest.main()
```

doctest

stdlib

unique to Python?
literate programming
verify examples in docs
docstring or file

```
def get(self):
    """get() -> return TestClass's associated value.

>>> x = _TestClass(-42)
>>> print x.get()
-42
"""

return self.val
```

```
The ``example`` module
```

```
=====
```

```
Using ``factorial``
```

```
-----
```

```
This is an example text file in reStructuredText format. First import  
``factorial`` from the ``example`` module:
```

```
>>> from example import factorial
```

```
Now use it:
```

```
>>> factorial(6)  
120
```

Running `doctest.testfile("example.txt")` then finds the error in this documentation:

```
File "./example.txt", line 14, in example.txt
```

```
Failed example:
```

```
    factorial(6)
```

```
Expected:
```

```
    120
```

```
Got:
```

```
    720
```

"As mentioned in the introduction, doctest has grown to have three primary uses:

- Checking examples in docstrings.
- Regression testing.
- Executable documentation

These uses have different requirements, and it is important to distinguish them. In particular, filling your docstrings with obscure test cases makes for bad documentation."

`unittest.mock`

adding in version 3.3
available for previous vers
stubs
action → assertion
patch modules
insulate your tests

ReST (the text format)

in docutils package
plain text → markup
used for Python docs, etc
similar to Markdown, etc

```
:mod:`doctest` --- Test interactive
```

```
=====
```

The `:mod:`doctest`` module searches Python sessions, and then executes exactly as shown. There are several ways to use doctest:

- * To check that a module's docstrings and interactive examples still work.
- * To perform regression testing by verifying that test file or a test object work as expected.
- * To write tutorial documentation with input-output examples. Depending on whether the examples or the documentation are emphasized, this has the flavor of “literate testing” or “executable documentation”.

Here's a complete but small example:

```
"""
```

This is the "example" module.

The example module supplies one function:

```
>>> factorial(5)
```

```
120
```

25.2. `doctest` — Test interactive examples ¶

The `doctest` module searches for pieces of text that look like interactive sessions, and then executes those sessions to verify that they work exactly as shown. There are several ways to use doctest:

- To check that a module's docstrings are up-to-date and that its interactive examples still work as documented.
- To perform regression testing by verifying that a test file or a test object work as expected.
- To write tutorial documentation for a package, including examples. Depending on whether the examples or the documentation are emphasized, this has the flavor of “literate testing” or “executable documentation”.

Here's a complete but small example module:

```
"""
```

This is the "example" module.

The example module supplies one function, `factorial`:

```
>>> factorial(5)
```

```
120
```

Sphinx (the doc tool)

ReST to
html, singlehtml, pickle,
json, htmlhelp, qthelp,
devhelp, epub, latex,
latexpdf, text
can run all doctests

Sphinx

Footnotes

- [1] Examples containing both expected output and an exception are not supported. Trying to guess where one ends and the other begins is too error-prone, and that also makes for a confusing test.

 Python v2.7.3 documentation » The Python Standard Library » 25.

[previous](#) | [next](#) | [modules](#)

Development Tools »

© [Copyright](#) 1990-2012, Python Software Foundation.

The Python Software Foundation is a non-profit corporation. [Please donate.](#)

Last updated on Jun 06, 2012. [Found a bug?](#)

Created using [Sphinx](#) 1.0.7.



Read the Docs

Create, host, and browse documentation.

[Sign up](#)

or [Log in](#)

What is this place?

Read the Docs hosts documentation, making it fully searchable and easy to find. You can import your docs using any major version control system, including Mercurial, Git, Subversion, and Bazaar. We support [webhooks](#) so your docs get built when you commit code. There's also support for versioning so you can build docs from tags and branches of your code in your repository. A [full list of features](#) is available.

It's free and simple. Read the [Getting Started](#) guide to get going!

nose / pytest

"no API" unit testing
assert instead of assert??
print & output capture
client code reads cleanly
better runner for unittest

```
# content of test_sample.py
def func(x):
    return x + 1

def test_answer():
    assert func(3) == 5
```

```
# unittest style  
self.assertRaises(FooException, func, a, b)
```

```
# py.test style  
with pytest.raises(FooException):  
    func(a, b)
```

nose vs pytest?

`coverage.py`

uses debugging hook to
see which lines of code
are actually executed

plugins exist for
most (all?) test runners

Quick start

Getting started is easy:

1. Install coverage.py from the [coverage page on the Python Package Index](#), or by using "easy_install coverage". For a few more details, see [Installation](#).
2. Use `coverage run` to run your program and gather data:

```
$ coverage run my_program.py arg1 arg2
blah blah ..your program's output.. blah blah
```

3. Use `coverage report` to report on the results:

```
$ coverage report -m
Name           Stmts   Miss  Cover   Missing
-----
my_program      20      4    80%   33-35, 39
my_other_module 56      6    89%   17-23
-----
TOTAL          76     10    87%
```

4. For a nicer presentation, use `coverage html` to get annotated HTML listings detailing missed lines:

```
$ coverage html
```

Then visit `htmlcov/index.html` in your browser, to see a [report like this](#).

pyflakes

check code for (some) errors

fast enough to run every
time you save a file

awesome

PEP 8

PEP: 8
Title: Style Guide for Python Code
Version: 24d02504e664
Last-Modified: 2012-04-28 00:51:37 -0700 (Sat, 28 Apr 2012)
Author: Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>
Status: Active
Type: Process
Content-Type: text/x-rst
Created: 05-Jul-2001
Post-History: 05-Jul-2001

Contents

- [Introduction](#)
- [A Foolish Consistency is the Hobgoblin of Little Minds](#)
- [Code lay-out](#)
 - [Indentation](#)
 - [Tabs or Spaces?](#)
 - [Maximum Line Length](#)
 - [Blank Lines](#)
 - [Encodings \(PEP 263\)](#)
 - [Imports](#)
- [Whitespace in Expressions and Statements](#)
 - [Pet Peeves](#)
 - [Other Recommendations](#)
- [Comments](#)
 - [Block Comments](#)
 - [Inline Comments](#)
 - [Documentation Strings](#)
- [Version Bookkeeping](#)
- [Naming Conventions](#)
 - [Descriptive: Naming Styles](#)
 - [Prescriptive: Naming Conventions](#)
 - [Names to Avoid](#)
 - [Package and Module Names](#)

pep8.py

tells you when you have
code which doesn't
follow PEP 8

awesome

PEP 712

<http://www.revsys.com/blog/2011/oct/20/pep712-proposal-make-unittest2-more-accurate/>

PEP712 - Proposal to make unittest2 more accurate

PEP: 712

Title: Proposal to make unittest2 more accurate

Version: a58437babcaa

Last- 2011-10-20T14:40:01.661119 (Thu, 20 Oct 2011)

Modified:

Authors: Frank Wiles <frank@revsys.com>, Jacob Kaplan-Moss <jacob@jacobian.org>,
Jeff Triplett <jeff@revsys.com>

Status: Draft

Type: Humor

Created: 20-Oct-2011

Python- 2.7

Version

Introduction

This PEP describes a proposal to make unittest2 output more accurate and fun.

The Proposed Solution

Upon having more than 7 failing tests in a test run replace all 'F' character output with 'U's for the remainder of the test run.

Rationale

This output formatting more accurately describes the mental state of the developer.

Usage

```
nosetests --with-f7u12
```

Example

```
$ nosetests --with-f7u12
```

```
.....FFFFFFFFFFUUUUUUUUUUUUUUUUUU..U..U..U..U..U..U.
```

```
=====
FAIL: test_f7u12.TestGeneratesLotsOfFailures.test_generates_failures(7,
```

```
-----
Traceback (most recent call last):
```

```
File
```

```
  "/Users/mpirnat/Documents/code/python/nose-f7u12/lib/python2.7/site-
  packages/nose-1.1.2-py2.7.egg/nose/case.py",
    line 197, in runTest
```

Lab A
(20)

Lab A

- `git clone https://github.com/gregmalcolm/python_koans.git`
- `cd "python_koans/python 2"`
- `python contemplate_koans.py`

Unit B (25) Challenge!

lots of cool stuff in that
last section, eh?

let's use ALL of it

Lab B
(20)

```
git pull
```

Lab B

```
py.test --doctest-glob=*.txt --pep8 --cov=fetcher
```

- your turn. ;)
- grab the code we just made
- try to run it all
- (convert to nose?)
- make changes, add stuff, break stuff, etc
- unittest
- doctest
- nose / py.test
- coverage.py
- pyflakes
- pep8

Unit C (25)

Discuss.

wrapup

the end

thanks for everything!

Lab C
(20)

Lab C

- fill out feedback forms
- I'll be hanging out downstairs