

Literate Doctest

Marius Gedminas
<marius@pov.lt>

Programmers of Vilnius
<http://pov.lt/>



EuroPython 2006

doctests

imagine an interactive Python
session

```
>>> 2 + 2
```

```
4
```

```
>>> if 2 * 2 == 4:
```

```
...     print 'My Python is sane today'
```

```
My Python is sane today
```

copy & paste

run and verify

doctests serve two purposes

documentation and tests

documentation

you develop a Python package

you create a README.txt file

documentation for users of this
package

documentation should have
examples

(I'm a user. I love examples.)

if you put doctests into the
README, then you have real,
working examples

(and they get tested)

example

moneylaundry

=====

This is a package that helps you launder money.

```
>>> from moneylaundry import launder  
>>> money = '$1,000,000.00'  
>>> launder(money)  
Decimal("1000000.00")
```

some advice

focus on documentation

tell a story

do not put complicated setup
code into the README

do not put comprehensive tests
for all imaginable corner cases
into the README

hooking things up

unittest is not a bad framework

```
# tests.py
```

```
import unittest  
import doctest
```

```
def test_suite():  
    return doctest.DocFileSuite(  
        'README.txt')
```

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

DocFileSuite is for plain text files

*(there's also DocTestSuite that I'll
mention later)*

Complicated setup code?

```
def setUp(test):
```

```
...
```

```
def tearDown(test):
```

```
...
```

```
def test_suite():
```

```
    return doctest.DocFileSuite(  
        'README.txt',  
        setUp=setUp,  
        tearDown=tearDown)
```

you can put things into
test.globs

do not abuse

make the README easy to
understand

useful technique for API design:

write the README first

then implement it

you can do this in many iterative
steps, large or small

documentation (again)

short examples in docstrings

```
def blend(color1, color2, alpha=0.5):  
    """Blend two colors together.
```

```
>>> blend('#ff0000', '#ffffff', 0.5)  
'#ff776f'
```

```
>>> blend('#ff0000', '#ffffff', 0.75)  
'#ffbbaa'
```

```
"""
```

short

testing these

```
# tests.py
import unittest
import doctest

def test_suite():
    return unittest.TestSuite([
        doctest.DocFileSuite('README.txt'),
        doctest.DocTestSuite('colors'),
    ])

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

DocFileSuite is for plain text files

*DocTestSuite is for docstrings in
Python modules*

enough about documentation

testing

you can use traditional unittest
TestCases

or you can use doctests

or even both

doctests have some advantages

doctests invite English
descriptions

plus there are some nice doctest
features to make life easier

general pattern of doctests

short paragraph that explains what the next chunk is all about

```
>>> short chunk  
>>> of Python code
```

another

```
>>> another
```

etc.

You can process such text files
with a ReStructuredText
processor

Alternative: put doctests into
Python modules in a subpackage

```
# tests.py
import unittest
import doctest

def doctest_this():
    """Test this

    If X then Y

    >>> this(X)
    Y

    """
```


tests.py, continued

```
def doctest_that():
```

```
    """
```

```
    If Z then Q
```

```
        >>> this(Z)
```

```
        Q
```

```
    """
```

tests.py, continued

```
def test_suite():  
    return unittest.TestSuite([  
        doctest.DocFileSuite('README.txt'),  
        doctest.DocTestSuite('colors'),  
        doctest.DocTestSuite(),  
        # no module name = this file  
    ])  
  
if __name__ == '__main__':  
    unittest.main(defaultTest='test_suite')
```

which is better?

it's up to you

just don't mix long tests and
code in the same module

about those nice features

wildcards

-- real life --

Python 2.4.3 (#2, Apr 27 2006, 14:43:58)

```
>>> object()  
<object object at 0xb7d88448>
```

-- doctest --

Call a class to make an object

```
>>> object()          # doctest: +ELLIPSIS  
<object object at ...>
```

diffs


```
>>> print '\n'.join(sorted(some_dict))  
...      # doctest: +REPORT_NDIFF  
alpha  
beta  
gama
```

Failed example:

```
print '\n'.join(sorted(some_dict))
# doctest: +REPORT_NDIFF
```

Differences (ndiff with -expected +actual):

```
    alpha
-  beta
+  delta
-  gama
+  gamma
?      +
```

sadly diffs + ellipses do not mix
well

there's also whitespace
normalization

doctest:

+NORMALIZE_WHITESPACE

useful when you compare, say,
generated HTML

makes diffs even harder to read

:(

flags can be specified globally


```
doctest.DocFileSuite('README.txt',  
                      optionflags=doctest.ELLIPSIS  
                        |doctest.REPORT_NDIFF)
```

real-life example

functional test of a web app with zope.testbrowser

When we look at the front page, we see a welcome message

```
>>> browser = Browser()
>>> browser.open('http://localhost/')
>>> 'Welcome' in browser.contents
True
```

There's a login link

```
>>> browser.getLink('Log in').click()
```

We can now type the username and password, and log in

```
>>> browser.getControl(  
...     'User Name').value = 'marius'  
>>> browser.getControl(  
...     'Password').value = 'sekrit'  
>>> browser.getControl('Log in').click()
```

And now my home page shows that I've got mail

```
>>> print browser.contents
```

```
<BLANKLINE>
```

```
...
```

```
<h1>Hello, Marius</h1>
```

```
...
```

```
<div class="info">
```

```
  You have mail!
```

```
</div>
```

```
...
```

testbrowser works with Zope 3,
and with real HTTP servers

I'm sure you could hook it up
with WSGI and other things

so, do doctests always win?

there are some downsides

you cannot step through a
doctest with pdb

(nothing else comes into mind)

summary

documentation (README.txt,
docstrings)

tests (unit, functional)

keep them separate

acknowledgements

Jim Fulton's PyCon 2004 talk

Literate Testing: Automated Testing with doctest

<http://www.python.org/pycon/dc2004/papers/4/PyCon2004DocTestUnit.pdf>

Phillip J. Eby's weblog

Stream of Consciousness Testing

<http://dirtsimple.org/2004/11/stream-of-consciousness-testing.html>

that's it
question time

... is there any time left?