Testing for Pythonistas: nose and pytest

- Why nose and pytest are better than unittest
- How to write and run pythonic tests
- Using plugins to solve real-world problems

hello

- Wes Winham @weswinham, github.com/winhamwr
- Responsible for product development at PolicyStat
- ~7 year old Django app, ~1 million monthly page views
- 0-4700 "unit" tests, 0-200 selenium tests

Testing Matters


www.JJ.AM

Today

- I'll show you how to write and run tests
- Randy- How to test effectively
- Randy- How to test difficult things
- Rob- Keep things fast and scalable
- Calvin- Run those awesome tests

Write Tests!

you'll get...

- Better code design
- Quicker implementation to working code
- Refactoring with abandon
- Fearless updates

Good Tools Matter

Master testers can...

- Eliminate boilerplate
- Write maintainable tests (tests are code)
- Pick the right tool

Part 1: Pythonic Testing

```
# add_one.py
def add_one(x):
    return x + 1
```

Zen of python

- Simple is better than complex.
- Flat is better than nested.

```
import unittest
from add_one import add_one

class AddOneTest(unittest.TestCase):
    def test_negative_one(self):
        self.assertEqual(add_one(-1), 0)

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

Is this Java?

nose or pytest!

```
# test.py
from add_one import add_one

def test_negative_one():
    assert add_one(-1) == 0
```

Complex tests

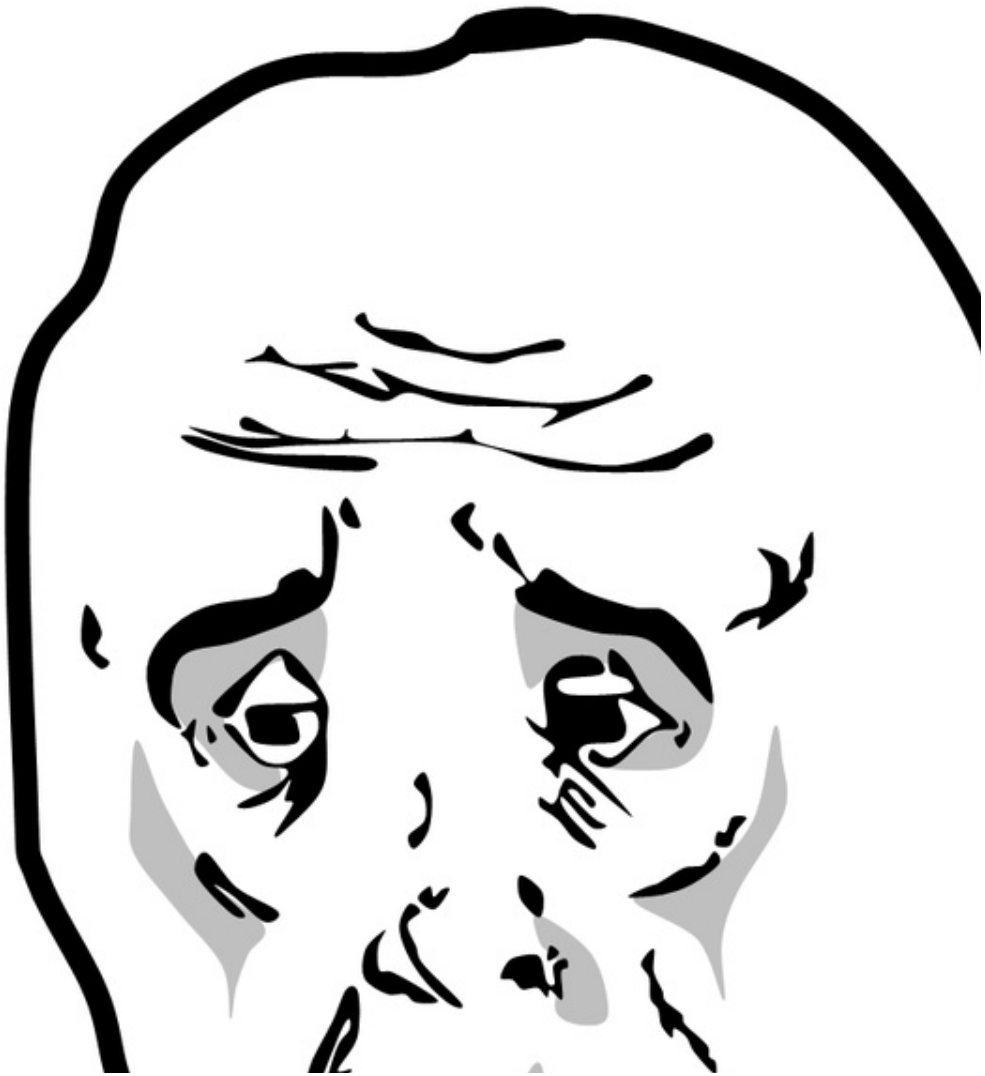
- Test fixtures
- Test cleanup
- Stdout
- Filesystem

Example: Easy filesystem tests

```
# test_tmpdir.py
def test_needsfiles(tmpdir):
    print (tmpdir)
    assert 0
```

History of python testing

In the beginning, unittest



History: Rise of nose

- pytest first, easy test discovery
- ~2007 nose is easy to install
- Works with unittest for free
- More extensive plugin system

History: pytest resurgence

- Feature parity since 2009-2011
- Better documentation
- Better parameterized fixtures
- Both good plugin ecosystems

Installation

- $ virtualenv tfp
- $ pip install nose pytest

Why you should use them

Better: Test discovery and running

```
import unittest

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

They run everything

- unittest cases
- test_FOO modules
- test functions

Discovery options: 1

- $ nosetests only_test_this.py
- $ nosetests test.module
- $ nosetests another.test:TestCase.test_method
- $ nosetests a.test:TestCase
- $ nosetests /path/to/test/file.py:test_function

Discovery options: 2

- $ py.test only_test_this.py
- $ nosetests
- $ py.test


Runs doctests

```
# add_one.py
def add_one(x):
    """
    >>> add_one(-1)
    0
    """
    return x + 1
```

Test type: class

```
from add_one import add_one

class AddOneTest:
    def test_negative_one(self):
        assert add_one(-1) == 0
```

Test type: generator

```
def test_evens():
    for i in range(0, 5):
        yield check_even, i, i*3

def check_even(n, nn):
    assert n % 2 == 0 or nn % 2 == 0
```

Setup/Teardown types

- Module
- Class
- Function (decorators)

```
from add_one import add_one

class AddOneTest:
    def set_up(self):
        # Do work

    def test_negative_one(self):
        assert add_one(-1) == 0
```

Configuration: nose

```
# setup.cfg, .noserc or nose.cfg
[nosetests]
verbosity=3
with-doctest=1
```

Configuration: pytest

```
# pytest.ini, tox.ini or setup.cfg
[pytest]
python_files=*.py
norecursedirs = _build
```

pytest: Fixture injection

- Examples from Danny Greenfield

```
# test_fixtures.py
from pytest import fixture

@fixture
def complex_data():
    class DataTypes(object):
        string = str
        list = list
    return DataTypes()

def test_types(complex_data):
    assert isinstance(
        "Elephant", complex_data.string)
    assert isinstance(
        [5, 10, 15], complex  data.list)
```

pytest: Tests for tests

```
# test_fixtures.py
# note: this version of test_fixtures.py is built c

def test_complex_data(complex_data):
    assert isinstance(complex_data, object)
    assert complex_data.string == str
    assert complex_data.list == list
```

Debugging: PDB on failures

- $ nosetests --pdb
- $ py.test --pdb

Debugging: PDB when you want

```
from nose.tools import set_trace
from pytest import set_trace

...
set_trace()
```

Plugins

Plugins: For (almost) everything

- Your favorite web framework
- Code coverage
- Tox
- Parallel test running
- xUnit output

Plugins: Sources

- pytest-plugs
- nose builtin
- nose 3rd party

Plugin: Tox

- Test combinations of things
- Different python versions
- Different requirements

Plugin: Parallel Test Running

- $ nosetests --processes=2
- $ pip install pytest-xdist
- $ py.test -n 2

Plugin: Shared-nothing Distributed Test Running

- distributed-nose
- $ export NOSE_NODES=2;
- $ export NOSE_NODE_NUMBER=1;
- $ nosetests long_test_suite

Plugin: xUnit output

- $ py.test --junitxml=path
- $ nosetests --with-xunit

Plugin authoring

- All the hooks!
- Plugin with a plugin system
- e.g. fulltext search setup/teardown

We're hiring!