Welcome

Welcome to East Bay Django Meetup meeting (2012.11.13)

- Topic: *Best practices (with examples) for Django/Python testing* (6:15-6:45)
- Presenter: Raymond Yee (@rdhyee). I work for unglue.it and occasionally teach at the School of Information, UC Berkeley. Author of *Pro Web 2.0 Mashups*
- You can get source for this talk at: https://github.com/rdhyee/django_testing_tutorial

Outline

I. Unit testing using Python standard libraries: using factorial as example
II. brief excursion into Django testing

BTW, iPython Notebook is great!

Using iPython Notebook to display materials for talk

In [ ]:
```
# % magic commands in case I forget
%reset
%whos
```

Calculating factorial

$n! = \prod_{k=1}^{n} k!$

$n! = \begin{cases} 1 & \text{if } n = 0, \\ (n-1)! \times n & \text{if } n > 0. \end{cases}$

e.g.,

$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

Factorial in Python standard library

In [ ]:
```
# http://docs.python.org/2.7/library/math.html

import math

math.factorial(5)
```

(fac1) factorial implementation 1

Of course many ways to calculate factorial in Python -- see, for your viewing pleasure, http://www.artima.com/forums/flat.jsp?forum=181&thread=75931

In [ ]:
```
# factorial implementation 1

def fac1(n):
    f = 1
    for i in xrange(1, n+1):
        f = f*i
    return f

print fac1(5)
```

Python Test fixture, case, suite, runner

From http://docs.python.org/2.7/library/unittest.html:

test fixture
A *test fixture* represents the preparation needed to perform one or more tests, and any associate cleanup actions. This may involve, for example, creating temporary or proxy databases, directories, or starting a server process.
test case
A *test case* is the smallest unit of testing. It checks for a specific response to a particular set of inputs. `unittest` provides a base class, `TestCase`, which may be used to create new test cases.
test suite
A *test suite* is a collection of test cases, test suites, or both. It is used to aggregate tests that should be executed together.
test runner
A *test runner* is a component which orchestrates the execution of tests and provides the outcome to the user. The runner may use a graphical interface, a textual interface, or return a special value to indicate the results of executing the tests.

TestCase for fac1

In [ ]:
```
# TestCase for fac1

import unittest
from unittest import TestCase

class Fac1Test(TestCase):
    def test_6(self):
        self.assertEqual(120, fac1(5))
```

fac1 implementation, testcase, and suite

In [ ]:
```
# testing fac1

import unittest
from unittest import TestCase


# factorial implementation 1

def fac1(n):
    f = 1
    for i in xrange(1, n+1):
        f *= i
    return f


# TestCase for fac1

class Fac1Test(TestCase):
    def test_6(self):
        self.assertEqual(120, fac1(5))
    def test_0(self):
        self.assertEqual(1, fac1(0))

# define which tests to run

def suite():

    testcases = [Fac1Test]

    suites = unittest.TestSuite([unittest.TestLoader().loadTestsFromTestCase(te
    return suites

# use the TextTestRunner to run suite
testresults = unittest.TextTestRunner().run(suite())


# intepret results of tests

print "number of tests run", testresults.testsRun
# http://docs.python.org/2/library/unittest.html#unittest.TestResult.wasSuccess
print "tests successful?", testresults.wasSuccessful()

print "number of errors and failures: {0}, {1}".format(len(testresults.errors),

if len(testresults.errors):
    print "errors: "
    for error in testresults.errors:
        print error[0], error[1]
```

```
if len(testresults.failures):
    print "failures: "
    for failure in testresults.failures:
        print failure[0], failure[1]
```

Let's mimick math.factorial for Errors

In [ ]:
```
# negative n

import math

try:
    math.factorial(-1)
except Exception as e:
    print "math.factorial(-1) -> ", type(e), e.message

# non-integer real n
try:
    math.factorial(1.5)
except Exception as e:
    print "math.factorial(1.5) -> ", type(e), e.message

# non-numeric

try:
    math.factorial("a")
except Exception as e:
    print 'math.factorial("a") -> ', type(e), e.message
```

Testcase for this exception handling in math.factorial

In [ ]:
```
# expanded testcase for fac2

import math
import unittest
from unittest import TestCase

class Fac2Test(TestCase):
    def test_6(self):
        self.assertEqual(120, fac2(5))
    def test_0(self):
```

```
        self.assertEqual(1, fac2(0))
    def test_range(self):
        for i in xrange(10):
            self.assertEqual(math.factorial(i), fac2(i))
    def test_neg(self):
        self.assertRaises(ValueError, fac2, -1)
    def test_non_integer(self):
        self.assertRaises(ValueError, fac2, 1.5)
    def test_non_numeric(self):
        self.assertRaises(TypeError, fac2,"a")
```

Testing fac1 on this behavior

In [ ]:

```
# testing fac2

import unittest
from unittest import TestCase
import math


# factorial implementation 1

def fac1(n):
    f = 1
    for i in xrange(1, n+1):
        f *= i
    return f

# factorial implementation 2

fac2 = fac1

# TestCase for fac1

class Fac1Test(TestCase):
    def test_6(self):
        self.assertEqual(120, fac1(5))
    def test_0(self):
        self.assertEqual(1, fac1(0))

# TestCase for fac2

class Fac2Test(TestCase):
    def test_6(self):
        self.assertEqual(120, fac2(5))
    def test_0(self):
        self.assertEqual(1, fac2(0))
    def test_range(self):
        for i in xrange(10):
            self.assertEqual(math.factorial(i), fac2(i))
```

```
    def test_neg(self):
        self.assertRaises(ValueError, fac2, -1)
    def test_non_integer(self):
        self.assertRaises(ValueError, fac2, 1.5)
    def test_non_numeric(self):
        self.assertRaises(TypeError, fac2,"a")

# define which tests to run

def suite():

    testcases = [Fac1Test, Fac2Test]

    suites = unittest.TestSuite([unittest.TestLoader().loadTestsFromTestCase(te
    return suites

# use the TextTestRunner to run suite
testresults = unittest.TextTestRunner().run(suite())


# intepret results of tests

print "number of tests run", testresults.testsRun
# http://docs.python.org/2/library/unittest.html#unittest.TestResult.wasSuccess
print "tests successful?", testresults.wasSuccessful()

print "number of errors and failures: {0}, {1}".format(len(testresults.errors),

if len(testresults.errors):
    print "errors: "
    for error in testresults.errors:
        print error[0], error[1]

if len(testresults.failures):
    print "failures: "
    for failure in testresults.failures:
        print failure[0], failure[1]
```

Running Fact2Test on math.factorial

In [ ]:

```
# testing fac2

import unittest
from unittest import TestCase
import math


# factorial implementation 1

def fac1(n):
```

```
    f = 1
    for i in xrange(1, n+1):
        f *= i
    return f

# factorial implementation 2

fac2 = math.factorial

# TestCase for fac1

class Fac1Test(TestCase):
    def test_6(self):
        self.assertEqual(120, fac1(5))
    def test_0(self):
        self.assertEqual(1, fac1(0))

# TestCase for fac2

class Fac2Test(TestCase):
    def test_6(self):
        self.assertEqual(120, fac2(5))
    def test_0(self):
        self.assertEqual(1, fac2(0))
    def test_range(self):
        for i in xrange(10):
            self.assertEqual(math.factorial(i), fac2(i))
    def test_neg(self):
        self.assertRaises(ValueError, fac2, -1)
    def test_non_integer(self):
        self.assertRaises(ValueError, fac2, 1.5)
    def test_non_numeric(self):
        self.assertRaises(TypeError, fac2,"a")

# define which tests to run

def suite():

    testcases = [Fac1Test, Fac2Test]

    suites = unittest.TestSuite([unittest.TestLoader().loadTestsFromTestCase(te
    return suites

# use the TextTestRunner to run suite
testresults = unittest.TextTestRunner().run(suite())


# intepret results of tests

print "number of tests run", testresults.testsRun
# http://docs.python.org/2/library/unittest.html#unittest.TestResult.wasSuccess
print "tests successful?", testresults.wasSuccessful()

print "number of errors and failures: {0}, {1}".format(len(testresults.errors),

if len(testresults.errors):
```

```
    print "errors: "
    for error in testresults.errors:
        print error[0], error[1]

if len(testresults.failures):
    print "failures: "
    for failure in testresults.failures:
        print failure[0], failure[1]
```

Possible code to mimic math.factorial exception handling

In [ ]:
```
n = 1.5


try:
    n1 = int(n)
except:
    raise TypeError
else:
    if (n1 <> n) or n < 0:
        raise ValueError
```

new fac2 now passes tests

In [ ]:
```
# testing fac2

import unittest
from unittest import TestCase
import math


# factorial implementation 1

def fac1(n):
    f = 1
    for i in xrange(1, n+1):
        f *= i
    return f

# factorial implementation 2

def fac2(n):
    try:
```

```
        n1 = int(n)
    except:
        raise TypeError
    else:
        if (n1 <> n) or n < 0:
            raise ValueError

    f = 1
    for i in xrange(1, n+1):
        f *= i
    return f

# TestCase for fac1

class Fac1Test(TestCase):
    def test_6(self):
        self.assertEqual(120, fac1(5))
    def test_0(self):
        self.assertEqual(1, fac1(0))

# TestCase for fac2

class Fac2Test(TestCase):
    def test_6(self):
        self.assertEqual(120, fac2(5))
    def test_0(self):
        self.assertEqual(1, fac2(0))
    def test_range(self):
        for i in xrange(10):
            self.assertEqual(math.factorial(i), fac2(i))
    def test_neg(self):
        self.assertRaises(ValueError, fac2, -1)
    def test_non_integer(self):
        self.assertRaises(ValueError, fac2, 1.5)
    def test_non_numeric(self):
        self.assertRaises(TypeError, fac2,"a")

# define which tests to run

def suite():

    testcases = [Fac1Test, Fac2Test]

    suites = unittest.TestSuite([unittest.TestLoader().loadTestsFromTestCase(te
    return suites

# use the TextTestRunner to run suite
testresults = unittest.TextTestRunner().run(suite())


# intepret results of tests

print "number of tests run", testresults.testsRun
# http://docs.python.org/2/library/unittest.html#unittest.TestResult.wasSuccess
print "tests successful?", testresults.wasSuccessful()
```

```
print "number of errors and failures: {0}, {1}".format(len(testresults.errors),

if len(testresults.errors):
    print "errors: "
    for error in testresults.errors:
        print error[0], error[1]

if len(testresults.failures):
    print "failures: "
    for failure in testresults.failures:
        print failure[0], failure[1]
```

BTW Rel'n between factorial and Gamma function

http://en.wikipedia.org/wiki/Gamma_function

$\Gamma(z) = \int_0^\infty e^{-t} t^{z-1} dt$

and

$\Gamma(n) = (n-1)!$

In [ ]:

```
# requires scipy

import math
from scipy.special import gamma as Gamma


for i in xrange(10):
    print i, math.factorial(i), Gamma(i+1)

print Gamma(2.5), Gamma(-0.5)
print Gamma(-1)
```

Django tests on unglue.it + Jenkins

Testing Django

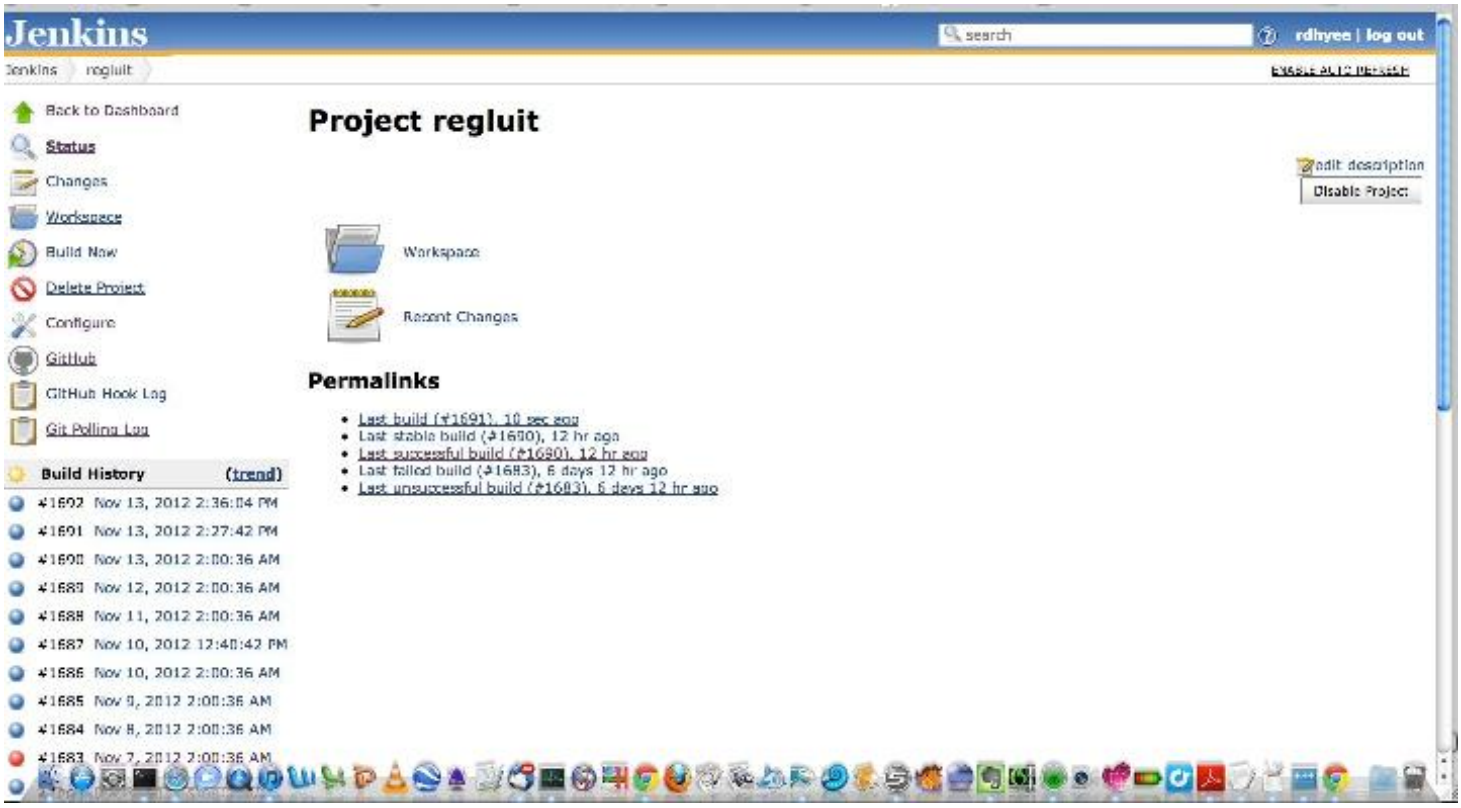
screenshot of output from unglue.it tests

```
(regluitepd1)raymond-yees-computer:regluit raymondyee$ django-admin.py test core api frontend
payment
Creating test database for alias 'default'...
..........x.....................................
----------------------------------------------------------------------
Ran 45 tests in 420.914s

OK (expected failures=1)
Destroying test database for alias 'default'...
(regluitepd1)raymond-yees-computer:regluit raymondyee$
```

screenshot of Jenkins installation used by unglue.it

http://jenkins-ci.org/



References

http://wiki.python.org/moin/PythonTestingToolsTaxonomy gives a wide survey of Python testing frameworks

Test Driven Development -- I showed in miniature.

possible enhancements/alternatives to `unittest` and the Django default test `client`

- nose
- django-nose
- Prefer WebTest to Django's test client for functional tests

Talk from DjangoCon 2012 that might be helpful: https://speakerdeck.com/zeeg/lessons-in-testing-djangocon-2012 / http://www.youtube.com/watch?v=9_39Vbjx23Y