CME 193: Introduction to Scientific Python

Winter 2017

Lecture 8: Writing tests in Python

Blake Jennings

`stanford.edu/~bmj/cme193`

Contents

- Homework, Portfolio and Project
- Unit testing
- More modules
- Wrap up

Last Lecture!

This is the last lecture of the course!

Everything (portfolio + HW2 or Project) due Thursday 2/16

Portfolio: Complete 2/3 of each section – OK to do more from some sections than others.

Office hours – what works best? T/Th same time or something else?

You can always ask questions via Canvas or email me directly.

Contents

Unit testing

Unit tests: test **individual** pieces of code

For example, for factorial function, test

- $0! = 1$
- $3! = 6$
- etc.

Unit testing

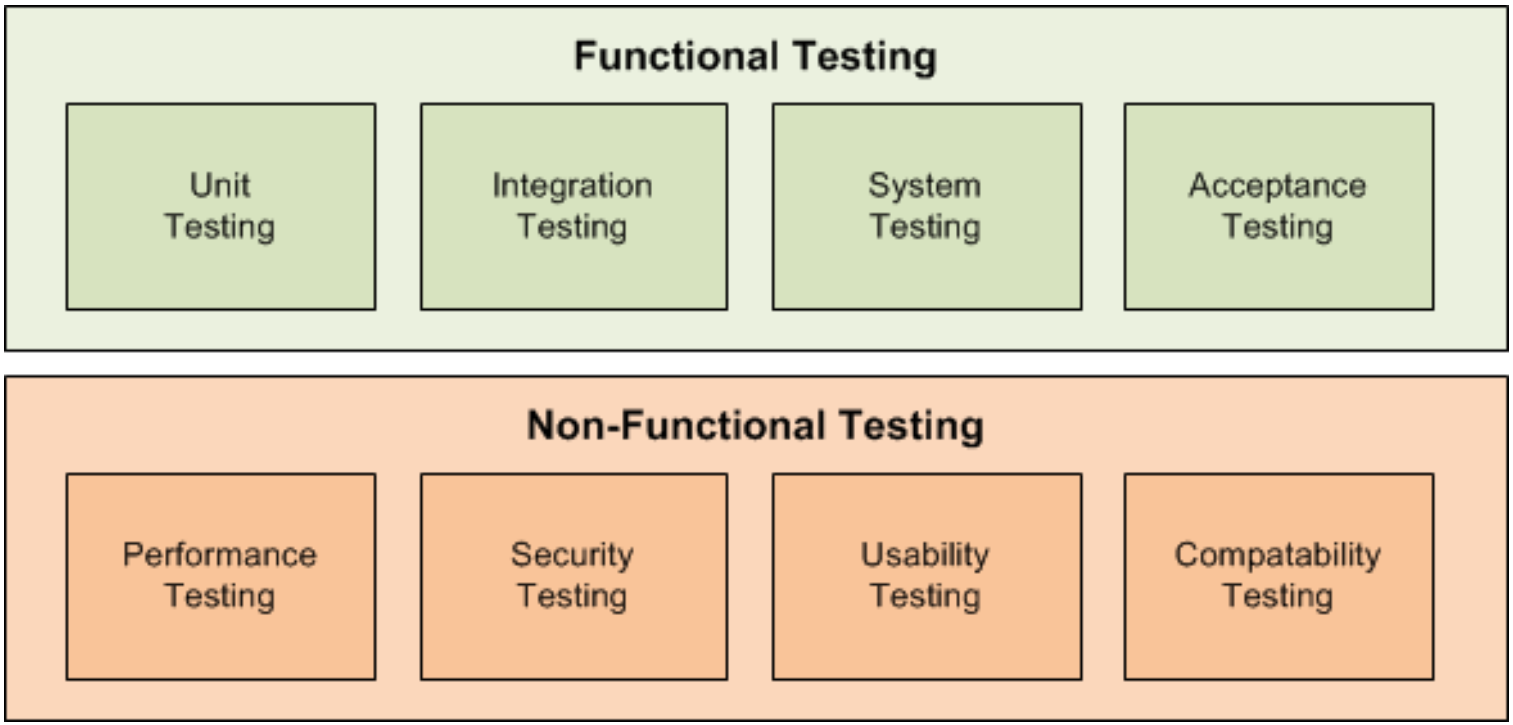

Figure: Test hierarchy

Test driven development

Some write **tests before code**

Reasons:

- Focus on the requirements
- Don't write too much
- Safely restructure/optimize code
- When collaborating: don't break other's code
- Faster

Test cases

How to construct test cases?

A *test case* should answer a **single question** about the code,

A test case should

- **Run by itself**, no human input required
- **Determine on its own** whether the test has passed or failed
- Be **separate** from other tests

What to test

- Known values
- Sanity check (for conversion functions for example)
- Bad input
  - Input is too large?
  - Negative input?
  - String input when expected an integer?
- etc: very dependent on problem

unittest

The standard Python module `unittest` helps you write unit tests.

```
import unittest
from my_script import is_palindrome

class KnownInput(unittest.TestCase):
    knownValues = (('lego', False),
                   ('radar', True))

    def testKnownValues(self):
        for word, palin in self.knownValues:
            result = is_palindrome(word)
            self.assertEqual(result, palin)
```

Not complicated, but can be hard to get started

unittest

A testcase is created by subclassing `unittest.TestCase`

Individual tests are defined with methods whose names start with the letters `test`. (Allows the test runner to identify the tests)

Each test usually calls an assert method to run the test - many assert options.

A few different ways to run tests (see documentation). Easiest way is to run `unittest.main()` for example if the test script is the main program.

assert

We can use a number of methods to check for failures:

- `assertEqual`
- `assertNotEqual`
- `assertTrue`, `assertFalse`
- `assertIn`
- `assertRaises`
- `assertAlmostEqual`
- `assertGreater`, `assertLessEqual`
- etc. (see Docs)

Alternatives

- nose2
- Pytest

`http://nose2.readthedocs.io/en/latest/differences.html`

Pytest

- Easy testing
- Automatically discovers tests
- No need to remember all assert functions, keyword `assert` works for everything
- Informative failure results

```
$ pip install -U pytest
```

Test discovery: (basics)

- Scans files starting with `test_`
- Run functions starting with `test_`

Example: primes

Create two files in a directory:

- primes.py – Implementation
- test_primes.py – Tests

Initial code

primes.py

```
def is_prime(x):
    return True
```

test_primes.py

```
from primes import is_prime

def test_is_three_prime():
    assert is_prime(3)

def test_is_four_prime():
    assert not is_prime(4)
```

Pytest output

```
$ py.test
============================= test session starts ==============================
platform darwin -- Python 2.7.9 -- py-1.4.27 -- pytest-2.7.1
rootdir: /Users/sps/Dropbox/cc/cme193/demo/unit_testing, inifile:
collected 2 items

test_primes.py .F

=================================== FAILURES ===================================
_____________________________ test_is_four_prime ______________________________

    def test_is_four_prime():
>       assert not is_prime(4)
E       assert not True
E        +  where True = is_prime(4)

test_primes.py:7: AssertionError
====================== 1 failed, 1 passed in 0.03 seconds ======================
```

Fixing is_prime

Simplest solution that passes tests:

primes.py

```
def is_prime(x):
    for i in xrange(2, x):
        if x % i == 0:
            return False
    return True
```

'Premature optimization is the root of all evil' - Donald Knuth

Pytest output

```
$ py.test
============================= test session starts ==============================
platform darwin -- Python 2.7.9 -- py-1.4.27 -- pytest-2.7.1
rootdir: /Users/sps/Dropbox/cc/cme193/demo/unit_testing, inifile:
collected 2 items

test_primes.py ..

========================== 2 passed in 0.01 seconds ===========================
```

Add more tests

```
from primes import is_prime

def test_is_zero_prime():
    assert not is_prime(0)

def test_is_one_prime():
    assert not is_prime(1)

def test_is_two_prime():
    assert is_prime(2)

def test_is_three_prime():
    assert is_prime(3)

def test_is_four_prime():
    assert not is_prime(4)
```

Pytest output

```
============================= test session starts ==============================
platform darwin -- Python 2.7.9 -- py-1.4.27 -- pytest-2.7.1
rootdir: /Users/sps/Dropbox/cc/cme193/demo/unit_testing, inifile:
collected 5 items

test_primes.py FF...

=================================== FAILURES ===================================
_____________________________ test_is_zero_prime _____________________________

    def test_is_zero_prime():
>       assert not is_prime(0)
E       assert not True
E        +  where True = is_prime(0)

test_primes.py:4: AssertionError
_____________________________ test_is_one_prime ______________________________

    def test_is_one_prime():
>       assert not is_prime(1)
E       assert not True
E        +  where True = is_prime(1)

test_primes.py:7: AssertionError
====================== 2 failed, 3 passed in 0.05 seconds ======================
```

Some more tests

- Negative numbers
- Non integers
- Large prime
- List of known primes
- List of non-primes

When all tests pass...

- First make sure all tests pass
- Then optimize code, making sure nothing breaks

Now you can be confident that whatever algorithm you use, it still works as desired!

Exercise

Recall the rational numbers class we made earlier. What are some unit tests that you would use to test that class?

Exercise

With the unittest module, write a test that tests that we do not allow rational numbers with denominator equal to zero.

Exercise

```
import exception_rational_fix
import unittest

class TestMethods(unittest.TestCase):
  def test_denomZero(self):
    self.assertRaises(ZeroDivisionError)

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

Writing good tests

- Utilize automation and code reuse
- Know the type and scope - your module or somebody else's?
- A single test should focus on a single thing
- Functional tests must be deterministic
- leave no trace - safe setup and clean up

Contents

More modules

Quickly go over some useful modules

What else is there?

Also, some nice resources to explore

Pickle

Module for *serializing* and *deserializing* objects in Python.

Serializing (pickling) saves a Python object as a byte stream.

Deserializing (unpickling) is inverse operation.

Save Python object to file with `dump`

Load Python object from file `load`

Very simple and extremely useful.

`cPickle` C implementation: faster

Pickle Example

Recall the `grades.txt` file we used in an example during the file i/o lecture. Each line held the name of a student as well as a list of grades.

Let's convert our `grades.txt` file to binary using pickle.

```
import pickle

with open('grades.txt', 'r') as fin:
  with open('grades.bin', 'wb') as fout:
    lines = fin.readlines()
    n = len(lines)
    pickle.dump(n, fout)
    for line in lines:
      student = line.split()
      name = student[0]
      grades = [int(student[i]) for i in range(1,len(student))]
      pickle.dump(name, fout)
      pickle.dump(grades, fout)
```

Pickle Example

We can investigate `grades.bin` using `hexdump` command in linux.

Now let's use Pickle to read that file, compute the averages and output those to a new text file.

```
import pickle

with open('grades.bin', 'rb') as fin:
  with open('grades_avg_v2.txt', 'w') as fout:
    n = pickle.load(fin)
    for i in range(n):
      name = pickle.load(fin)
      grades = pickle.load(fin)
      avg = float(sum(grades))/len(grades)
      fout.write('{} {:.2f}\n'.format(name,avg))
```

Comments on Pickle

There are definitely other options for binary file i/o in Python. Just use the tags `'rb'` and `'wb'` - but Pickle makes it easier to deal with conversion of objects to byte streams.

As you see in previous example, we kept track of how many students we had. There are ways to not know exactly how many things you need to load (try block or a clever while loop).

Easiest way to deal with unknown size of loads is to just save all data in one big data structure and load everything at once (though may be infeasible if you are working with a lot of data).

Warning: Pickle not secure against erroneous or maliciously constructed data!

Speeding up Python

Compared to C or Fortran, Python can be slow.

Ways to improve execution time:

- `Pypy`: no need to change any code, simply run your code using `pypy script.py`. However, does not work with Numpy etc.
- `Numba`: A little more work, but works with numpy
- `Cython`: Most work, fastest

Requests

HTTP library for Python.

```
import requests

r = requests.get('http://google.com')

print r.text
```

Alternative: urllib, urllib2

Beautiful soup

Useful for scraping HTML pages.

Such as: finding all links, or specific urls.

Get data from poorly designed websites.

Alternative: `Scrapy`

APIs

There are several modules that you can use to access APIs of websites

Twitter	python-twitter, Tweepy
Reddit	PRAW
	...

Able to get data or create apps for the ambitious.

Flask

Flask is a "microframework" for web development using Python

```
from flask import Flask
app = Flask(__name__)

@app.route("/<name>")
@app.route("/")
def hello(name="World"):
    return "Hello {}!".format(name)

if __name__ == "__main__":
    app.run(debug=True)  # only for debugging!
```

Run the above script, then browse to `http://127.0.0.1:5000/`

In-depth tutorial: `http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world`

Django

Another web development framework using Python

`https://www.djangoproject.com/`

Scikit learn

Large Scikit package with a lot of functionality. Sponsored by INRIA (and Google sometimes)

- Classification
- Regression
- Clustering
- Dimensionality reduction
- Model selection
- Preprocessing

scikit-*

Additional scikit packages that extend Scipy:

- skikit-aero
- scikit-image
- cuda
- odes

PyMC

A framework for Monte Carlo simulations

Tutorial: `https://camdavidsonpilon.github.io/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/`

Selenium

Selenium Python bindings provides a simple API to write functional/acceptance tests using Selenium WebDriver.

Tutorial: `http://selenium-python.readthedocs.io/`

Contents

Zen of Python

Very easy to write code.

A ton of packages already exist to help do most any tasks you like.

Once you know basics, very easy to pick up everything else - and a ton of sources as well!

Feedback

Thanks a lot!

Hope you enjoyed the class, learned a lot and will continue using Python!

Please fill out feedback forms at the end of the quarter - or feel free to let me know any feedback you have.

`Questions?`