# CME 193: Introduction to Scientific Python

## Lecture 8: Unit testing, more modules, wrap up

Sven Schmit

`stanford.edu/~schmit/cme193`

# Contents

# Unit testing

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

For example, for factorial function, test

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

# Test driven development

Some write **tests before code**

Reasons:

- Focus on the requirements
- Don't write too much
- Safely refactor 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 hard to get started

# Alternatives

- Nose
- Pytest

# Pytest

- Easy no-boilerplate testing
- Automatically discovers tests

```
$ 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!

# 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.

Save Python object to file with `dump`

Load Python object from file `load`

Very simple and extremely useful.

`cPickle` C implementation: faster

# Pickle

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

Save Python object to file with `dump`

Load Python object from file `load`

Very simple and extremely useful.

`cPickle` C implementation: faster

# Regular expressions

A *regular expression* (RE) specify a set of strings to match.

This module can be used to check whether a particular string matches the RE

I.e.: **Find text pattern in string / document**

see also:

`https://docs.python.org/2/howto/regex.html#regex-howto`

Very powerful, and not just for Python

# Regular expressions

. ^ $ * + ? { } [ ] \ ( )|

[ and ] are used for specifying a *character class*, e.g.[aeiou], [A-Z], [A-z], [0-5]

∧ is the **complement character**, e.g. [∧ 5] matches all except for a '5'.

\ used to signal various special sequences, including the use of metacharacters, e.g. \∧ to match ∧.

characters usually map to characters: 'test' - 'test'

\d matches any decimal digit: '\d' - '[0-9]'

# Regular expressions

Suppose you want to find phone numbers:

- 1234567890
- 123-456-7890
- (123) 465 7890
- (123) 456-7890
- etc

How to find all these?

# Regular expressions

Pattern:

- Maybe a bracket: \(?
- 3 numbers: \d{3}
- Maybe a bracket or a hyphen: [-\)]?
- Maybe a whitespace: \s?
- 3 numbers: \d{3}
- Maybe a hyphen or a whitespace: [-\s]?
- 4 numbers: \d{4}
- End: $

Extract the numbers by placing brackets: (\d{3}) around numbers

```
'\(?(\d{3})[-\)]?\s?(\d{3})[-\s]?(\d{4})$'
```

# Regular expressions

Pattern:

- Maybe a bracket: \(?
- 3 numbers: \d{3}
- Maybe a bracket or a hyphen: [-\)]?
- Maybe a whitespace: \s?
- 3 numbers: \d{3}
- Maybe a hyphen or a whitespace: [-\s]?
- 4 numbers: \d{4}
- End: $

Extract the numbers by placing brackets: (\d{3}) around numbers

```
'\(?(\d{3})[-\)]?\s?(\d{3})[-\s]?(\d{4})$'
```

# Regular expressions

```
import re

pat = '\(?(\d{3})[-\)]?\s?(\d{3})[-\s]?(\d{4})$'
repat = re.compile(pat)
string = '(123) 456-7890'
search =  repat.search(string)
if search:
    print search.groups()
else:
    print 'not found'
```

How to test?

Unit testing is invented for these kind of problems!

# Regular expressions

```
import re

pat = '\(?(\d{3})[-\)]?\s?(\d{3})[-\s]?(\d{4})$'
repat = re.compile(pat)
string = '(123) 456-7890'
search =  repat.search(string)
if search:
    print search.groups()
else:
    print 'not found'
```

How to test?

Unit testing is invented for these kind of problems!

# Regular expressions

```
import re

pat = '\(?(\d{3})[-\)]?\s?(\d{3})[-\s]?(\d{4})$'
repat = re.compile(pat)
string = '(123) 456-7890'
search =  repat.search(string)
if search:
    print search.groups()
else:
    print 'not found'
```

How to test?

Unit testing is invented for these kind of problems!

# Requests

HTTP library for Python.

```
import requests

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

print r.text
```

Alternative: urllib, urllib2

# 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

# 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.

# Scikits

Additional packages that extend Scipy:

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

# Scikit learn

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

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

# 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()
```

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/`

# PyMC

A framework for Monte Carlo simulations

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

# Contents

# Zen of Python

```
import this
```

# Have you fallen in love with Python?

'There's nothing wrong with falling in love with a programming language for her looks. I mean, let's face it - Python does have a rockin' body of modules, and a damn good set of utilities and interpreters on various platforms. Her whitespace-sensitive syntax is easy on the eyes, and it's a beautiful sight to wake up to in the morning after a long night of debugging. The way she sways those releases on a consistent cycle - she knows how to treat you right, you know?...'

`https://www.quora.com/Have-I-have-fallen-in-love-with-Python-because-she-is-beautiful`

# Project and portfolio

Please remember: projects and portfolios due next Thursday at noon.

Coursework checklist:

- Project code (**py** file(s) or **zip**).
- Project write-up (**pdf** file (no word!)).
- All scripts you wrote for class (**zip** with all **py** files). No write up necessary.

# Feedback

Thanks a lot!

Hope you enjoyed the class and learnt a lot!

Another feedback form:

`goo.gl/3onaCL`

or via course website

`Questions?`

# Contents

- Unit testing
- More modules
- Wrap up
- Exercises

# Exercises

See course website for exercises for this week.

Let me know if you have any question about exercises or project

Feel free to email me with questions about project