# 

# Python Testing and Automation Script

Mike Jackson, The Software Sustainability Institute

This work is licensed under the Creative Commons Attribution License

Copyright © Software Carpentry and The University of Edinburgh 2010-2012

See http://software-carpentry.org/license.html for more information.



## Prerequisites

Python, nose.

## Introduction

Use wget or curl e.g.

wget --no-check-certificate https://.../test-materials.zip

curl --insecure https://.../test-materials.zip > test-materials.zip

To get test-materials.zip from BitBucket and unzip these.

utilities.py contains functions, test functions and calls to test functions.

Run the tests.

python utilities.py

Modularity is a design principle so separate out functions and test functions. Create test\_utilities.py and move test functions and calls to test functions to this file.

Add to test\_utilities.py

from utilities import sum\_list

from utilities import calc\_mean

Run tests.

python test\_utilities.py

Test data is outside functions but validation is within. Put test data into test functions e.g.

numbers = [1, 2, 3, 4]

def test\_calc\_mean(numbers):

assert calc\_mean(numbers) == 2.5, 'mean of [1, 2, 3, 4] is not 2.5'

test\_calc\_mean(numbers)

becomes

def test\_calc\_mean():

numbers = [1, 2, 3, 4]

assert calc\_mean(numbers) == 2.5, 'mean of [1, 2, 3, 4] is not 2.5'

test\_calc\_mean()

Run.

python test\_utilities.py

Every time we add a test we need to add a call to that test function. Automate!

nosetests test\_utilities.py

nosetests looks for functions beginning with “test\_”.

Remove the test function calls.

nosetests test\_utilities.py

nosetests reports progress – “.”s – time taken and number of tests run.

nosetests reports failures e.g. introduce a bug into calc\_mean so it adds 1 to the total.

nosetests test\_utilities.py

Correct calc\_mean and retest just that function.

nosetests test\_utilities.py:test\_calc\_mean

nosetests can output an “xUnit” test report.

nosetests --with-xunit test\_utilities.py

cat nosetests.xml

xUnit framework by Kent Beck. JUnit, CUnit, fUnit etc. Can present results in different ways.

unittest module XXXXXXXXX

## Automated build-and-test

Version control + automated testing => automated build and test.

EPCC oncology project optimized and paralleled medical code. Initial run to get expected results. Create overnight test job to check out code, run code, compare to expected results. Optimize and parallelize in confidence.

VTK test dashboard, built using CDash. <http://open.cdash.org/index.php?project=VTK>

Continuous integration tools detect version control commits, check out code, build, run tests, and publish, or run every few minutes and publish.

MICE test dashboard uses Jenkins continuous integration server. Python code and tests, run using nosetests. <https://micewww.pp.rl.ac.uk/tab/show/maus>.

Faster you see a failure, faster you can fix it. Public shame is a motivator too!

OGSA-DAI uses Jenkins, Java code and JUnit tests, <http://ogsadai-public.epcc.ed.ac.uk:8080/jenkins/>

## How much testing is enough?

Learn by experience. Analogous to when to finish a proof reading a paper.

If you find bugs when you use your code, you did too little.

Tests, like code, should be reviewed.

Helps avoid tests that:

* Pass when they should fail.
* Fail when they should pass.
* Don't test anything. For example,

def test\_example():

pass

## Test driven development

Common to write code then write tests.

Test-driven development – test first code second.

Red-green-refactor:

* Red - write tests based on requirements. They fail as there is no code!
* Green - write/modify code to get tests to pass.
* Refactor code - clean it up.

Think about what the code should do, before we write it, not what we know it does.

## Conclusion

Cover Testing.ppt.