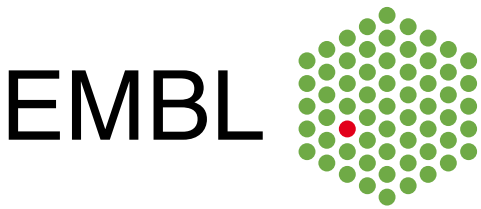
Unit testing using Python

Luis Pedro Coelho
§
On the web: http://luispedro.org
On twitter: @luispedrocoelho

European Molecular Biology Laboratory

June 17, 2014



Python exceptions

(We need this for the rest of the session)

```
def read_data():
    import os
    values = []
    for line in open('data.txt'):
        line_data = [float(elem) for elem in line.split()]
        values.append(line_data)
    return line_data
```

Python exceptions

```
def read_data():
    import os
    if not os.exists('data.txt'):
        raise ValueError('Data file is missing')
    values = []
    for line in open('data.txt'):
        line_data = [float(elem) for elem in line.split()]
        values.append(line_data)
    return line_data

try:
    data = read_data()
except:
    print 'Error in processing'
```

Motivation

Scientific code must not just produce nice looking output, but actually be correct.
http://bit.ly/testing-science

Motivation (II)

Some recent code-related scientific catastrophes:

- Geoffrey Chang
- Abortion reduces crime? maybe not so much once you fix the bug
- Rogoff's "Growth in a Time of Debt" paper is a famous example (even if the bug itself is only a small part of the counter-argument)
- Ariane 5 & NASA Mars Climate Orbiter

Why do things go wrong?

1. Your code is correct, but input files are wrong/missing/, the network goes down ...
2. Your code is buggy.

Never fail silently!

- The worst thing is to fail silently.
- Fail loudly and clearly

(This is partially why Unix tradition is to produce no output when things go well)

Defensive Programming

Defensive programming means writing code that will catch bugs early.

Assertions

```
def stddev(values):
    '''
    S = stddev(values)

    Compute standard deviation
    '''
    assert len(values) > 0, 'stddev: got empty list.'
    ...
```

Assertions

```
def stddev(values):
    '''
    S = stddev(values)

    Compute standard deviation
    '''
    if len(values) <= 0:
        raise AssertionError(
            'stddev: got empty list.')
    ...
```

Preconditions

In computer programming, a precondition is a condition or predicate that must always be true just prior to the execution of some section of code.

(Wikipedia)

Preconditions

Other Languages

- C/C++ #include <assert.h>
- Java assert pre-condition
- Matlab assert() (in newer versions)
- ... ...

Assertions Are Not Error Handling!

- Error handling protects against outside events; assertions protect against programmer mistakes.
- Assertions should never be false.

Programming by Contract

1. pre-conditions.
2. post-conditions.
3. invariants.

Pre-condition

What must be true before calling a function.

Post-condition

What is true after calling a function.

Testing

Do you test your code?

Unit Testing

- Python: nosetest
- Java: JUnit, ...
- C++: Boost test,...
- ...

Wikipedia has a Listofunittestingframeworkhttp://en.wikipedia.org/wiki/List_of_-unit_testing_frameworks

Unit Testing

```
def test_stddev_const():
    assert stddev([1]*100) < 1e-3

def test_stddev_positive():
    assert stddev(range(20)) > 0.
```

Nosetest

Nose software testing framework:

- Tests are named test_something.
- Conditions are asserted.

Software Testing Philosophies

1. Write tests first.
2. Write tests after.
3. Regression testing.

Test driven development is “Write tests first”

1. Write tests
2. Write code until all tests pass
3. Done

Regression Testing

Make sure bugs only appear once!

Practical Session: some preliminaries

statistics.py

```
def stddev(xs):
    ...
```

test_statistics.py

```
def test_stddev_const():
    assert stddev([1]*100) < 1e-3

def test_stddev_positive():
    assert stddev(range(20)) > 0.
```

What to test?

- Test behaviour, not implementation.
- Break code into separate functions.

Types of tests

- Smoke test: just check it runs
- Corner/edge cases: check “complex” cases.
- Case testing: test a “known case”
- Regression testing: create a test when you find a bug.
- Integration test: test that different parts work together.

Example tests

```
def test_simple():
    assert robust.average([1,2,3]) ==  2
    assert robust.average([1,2,30]) ==  2

def test_const():
    assert robust.average([2,2,2,2,2,2,2]) ==  2
```

More advanced unit testing

- setup run some code before the test
- teardown run some code after the test

More advanced unit testing

- setup run some code before the test
- teardown run some code after the test

- setup create input data, set up mock objects
- teardown remove output, cleanup databases, ...

Example of setup/teardown

```
import os
from nose import with_setup

_filename = 'output.txt'
def _remove_file():
    import os
    if os.path.exists(_filename):
        os.unlink(_filename)

@with_setup(teardown=_remove_file)
def test_writing_data():
    ...
```

Write code that is testable

- Separate I/O from processing
- Use functions that are units that can be tested
- Pure functions are better for testing
  (but are not always possible or even appropriate).

Links for testing

- http://nose.readthedocs.org/