

# Testing

K. Jarrod Millman  
Helen Wills Neuroscience Institute  
University of California, Berkeley  
`millman@berkeley.edu`

Applied Mathematics Perspectives 2011  
Reproducible Research: Tools and Strategies for Scientific Computing  
July 13, 2011

## Computing is error-prone

*In ordinary computational practice by hand or by desk machines, it is the custom to check every step of the computation and, when an error is found, to localize it by a backward process starting from the first point where the error is noted.*

— Norbert Wiener (1948)

## Software crisis

*The major cause [of the software crisis] is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.*

— Edsger W. Dijkstra (1972)

# Testing and debugging

- debugging is what you do when you know a program is broken
- testing is a determined, systematic attempt to break a program
- writing tests is more interesting work than debugging

# Program correctness

*Program testing can be used to show the presence of bugs,  
but never to show their absence!*  
— Edsger W. Dijkstra (1969)

## In the imperfect world ...

- write code as simple as possible
- avoid cleverness
- avoid writing code if possible
- use code to generate code
- program languages play an important role

## Testing and reproducibility

*In the good old days physicists repeated each other's experiments, just to be sure. Today they stick to FORTRAN, so that they can share each other's programs, bugs included.*  
— Edsger W. Dijkstra (1975)

# Assert invariants

```
if i%3 == 0:
    print 1
elif i%3 == 1:
    print 2
else:
    assert i%3 == 2
    print 3
```

## Pre- and post-condition tests

- what must be true *before* a method is invoked
- what must be true *after* a method is invoked
- use assertions

# Program defensively

- out-of-range index
- division by zero
- error returns

# Be systematic

- incremental
- simple things first
- know what to expect
- compare independent implementations

# Regression tests

- ensure that changes don't break existing functionality
- verify conservation
- **unit tests** (white box testing)
- measure test coverage

# Test fixtures

- create self-contained tests
- setup: open file, connect to a DB, create datastructures
- teardown: tidy up afterward

# Interface and implementation

- an **interface** is how something is used
- an **implementation** is how it is written

# Test-Driven Development (TDD)

- 1 **Add a Failing Test.** This focuses attention on the code interface rather than its implementation, and it provides examples of code use that can highlight intended use.
- 2 **Write Code to Pass Test.** “Do the simplest thing that could possibly work.”
- 3 **Refactor.** As code is evolved and refactored these tests give rapid confirmation of correct behavior.

## Improves code quality

- it is easy to get lost in implementation details
- unit tests help redirect your attention by making you thinking about use cases for the code
- difficult to test huge functions with both output and side effects

## Improves documentation

- an example is often better than an explanation
- tests don't get out-of-date

## More robust code

- TDD leads to quicker isolation of bugs
- that leads to shorter debugging
- facilitates change
- simplifies integration

# Enthought Python Distribution

http://www.enthought.com/products/edownload.php


ENTHOUGHT  
SCIENTIFIC COMPUTING SOLUTIONS

EPD repository | code.enthought.com | www.scipy.org | downloads | blog

PRODUCTS CONSULTING TRAINING SECTORS COMPANY CONTACT US

## PRODUCTS

- Overview
- Enthought Python Distribution
- Support Levels
- Purchase/Download**
- Package Index
- FAQ
- Getting Started
- License
- Repository
- Changelog



### Enthought Python Distribution Academic Download

The academic version of EPD is a fully functional installation of the software that can be used indefinitely by students and employees at degree-granting institutions. See [license terms](#) for details.

To download epd-7.1-1, please enter your academic email address below. A download link will be sent to the address you provide. For details on this release, see [EPD release notes](#).

ACADEMIC E-MAIL

# Landscape

- errors, exceptions, and assert
- doctests and unittests
- nose

# Exceptions

```
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division by zero
>>> factorial
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'factorial' is not defined
>>> '1'+1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int'
```

# Exception handling

```
>>> try:
...     file=open('test.txt')
... except IOError:
...     print 'No such file'
...
No such file
```

# Raising exceptions

```
>>> def newfunction():  
...     raise NotImplementedError  
...  
>>> newfunction()  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
  File "<stdin>", line 2, in newfunction  
NotImplementedError
```

# Assert

```
>>> 1 == 1
```

```
True
```

```
>>> assert 1 == 1
```

```
>>> assert 1 >= 0
```

```
>>> from math import factorial
```

```
>>> assert factorial(5) == 120
```

```
>>> assert factorial(4) == 120
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
AssertionError
```

# doctests

```
def factorial2(n):  
    """  
    >>> [factorial2(n) for n in range(5)]  
    [1, 1, 2, 6, 24]  
    >>> factorial2(-1)  
    Traceback (most recent call last):  
      File "<stdin>", line 1, in <module>  
    ValueError: factorial() not defined  
    """  
  
    from math import factorial  
    return factorial(n)
```

# Running doctests

```
$ python -m doctest -v myfactorial.py
Trying:
    [factorial2(n) for n in range(5)]
Expecting:
    [1, 1, 2, 6, 24]
ok
Trying:
    factorial2(-1)
Expecting:
    Traceback (most recent call last):
        ...
    ValueError: factorial() not defined
ok
```

## Running doctests, cont.

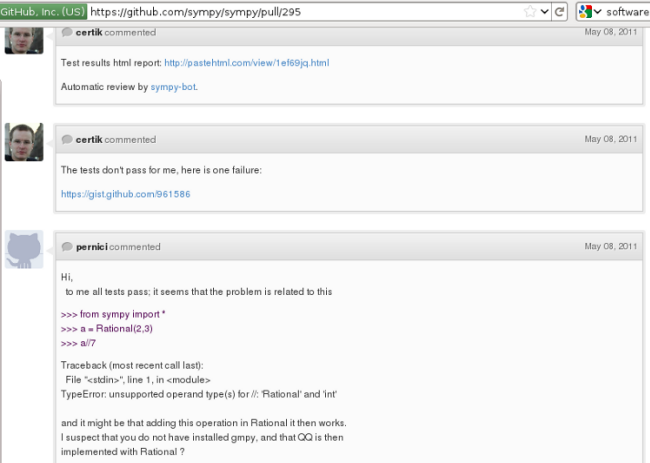
```
1 items had no tests:  
    myfactorial  
1 items passed all tests:  
    2 tests in myfactorial.factorial2  
2 tests in 2 items.  
2 passed and 0 failed.  
Test passed.
```




# sympy-bot


`https://github.com/sympy/sympy-bot`

`https://github.com/sympy/sympy/pull/295`


## sympy-bot




GitHub, Inc. (US) <https://github.com/sympy/sympy/pull/295>    software

 **certik** commented May 08, 2011

Test results html report: <http://pastehtml.com/view/1ef69jq.html>  
Automatic review by [sympy-bot](#).

 **certik** commented May 08, 2011

The tests don't pass for me, here is one failure:  
<https://gist.github.com/961586>

 **pernici** commented May 08, 2011

Hi,  
to me all tests pass; it seems that the problem is related to this

```
>>> from sympy import *
>>> a = Rational(2,3)
>>> a/7
```

Traceback (most recent call last):  
File "<stdin>", line 1, in <module>  
TypeError: unsupported operand type(s) for /: 'Rational' and 'int'

and it might be that adding this operation in Rational it then works.  
I suspect that you do not have installed gmpy, and that QQ is then implemented with Rational ?



# Test coverage

```
$ easy_install coverage
...
$ coverage run my_program.py arg1 arg2
blah blah ..your program's output.. blah blah

$ coverage report -m

$ coverage html
```

# Installing

```
$ easy_install nose
```

# Test runner

- `nosetests`
- **test discovery**: any callable beginning with `test` in a module beginning with `test`

# Writing tests

create a new module named `test_myfactorial` with the following content:

```
from myfactorial import factorial2

def check_factorial2():
    assert factorial2(5) == 150
```

# Test generators

```
def check_badvalue(n, MyError):  
    with assert_raises(MyError):  
        factorial2(n)  
  
def test_factorial2():  
    for n in range(20):  
        yield check_factorial2, n, factorial(n)
```

# Test coverage

`--with-coverage`

Enable plugin Coverage

`--cover-tests`

Include test modules in coverage report

`--cover-inclusive`

Include all python files under working directory

`--cover-html`

Produce HTML coverage information

## Learn more

<http://software-carpentry.org/>  
<http://docs.python.org/library/exceptions.html>  
<http://docs.python.org/library/doctest.html>  
<http://docs.python.org/library/unittest.html>  
<http://nedbatchelder.com/code/coverage/> <http://somethingaboutorange.com/mrl/projects/nose>