Testing in Python

It is more fun to write tests on a weekday than it is to look for bugs during the weekend.

-Jacob Kaplan Moss

Overview

Thinking about programs in different ways

Pytest

Where to learn more

Thinking about programs in different ways.

Tests help you think about your code in specific ways.

They

- invite you to keep your code decoupled (generally a good practice)
- run your code in new ways
- help other programmers understand where to hook into your code

Automated tests help you

- decrease time to run tests
- make sure you don't break something while fixing a bug

Thinking about programs in different ways.

How many ways can python interpret the following program?

```
# hello.py
def greet(msg="Students and professionals, good evening!"):
    "Print a greeting to standard output"
    print(msg)
```

Python has at least half dozen modules for interpreting python code, each designed to provide another perspective to the program.

Thinking about programs in different ways

```
# hello.py
def greet(msg="Students and professionals, good evening!"):
    "Print a greeting to standard output"
    print(msg)
```

as documentation and as source code

```
$ python -m pydoc hello

Help on module hello:

NAME
    hello

FILE
    /home/buckles/Documents/learn/python/hello.py

FUNCTIONS
    hello()
        Print a greeting to standard output

$
```

Thinking about programs in different ways

```
# hello.py
def greet(msg="Students and professionals, good evening!"):
    "Print a greeting to standard output"
    print(msg)
```

as currently running code in a REPL session

```
$ python -i hello.py
Students and professionals, good evening!
>>> greet("This presentation is on testing in python.")
This presentation is on testing in python.
>>>
```

Thinking about programs in different ways

```
# hello.py
def greet(msg="Students and professionals, good evening!"):
    "Print a greeting to standard output"
    print(msg)
```

as a series of function calls to be optimized

```
$ python -m profile hello.py
Students and professionals, good evening!
       4 function calls in 0.001 seconds

    Ordered by: standard name

    ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    1    0.001    0.001    0.001    0.001 :0(setprofile)
    1    0.000    0.000    0.000    0.000 hello.py:2(<module>)
    1    0.000    0.000    0.000    0.000 hello.py:2(greet)
    1    0.000    0.000    0.001    0.001 profile:0(<code object <module> at 0x7f7
    0    0.000             0.000          profile:0(profiler)
```

Thinking about programs in different ways

```
# hello.py
def greet(msg="Students and professionals, good evening!"):
    "Print a greeting to standard output"
    print(msg)
```

as code to debug

```
$ python -m pdb hello.py
> /home/buckles/Documents/learn/python/hello.py(2)<module>()
-> def greet():
(Pdb) list
  1     # hello.py
  2  -> def greet():
  3          "Print a greeting to standard output"
  4          print("Students and professionals, good evening!")
  5
  6      def test_hello():
  7          # how do we check IO?
  8          greet()
  9
 10      if "__main__" == __name__:
 11          greet()
(Pdb)
```

Thinking about programs in different ways

```
# hello.py
def greet(msg="Students and professionals, good evening!"):
    "Print a greeting to standard output"
    print(msg)
```

as code with tests to run

```
$ python -m unittest hello

----------------------------------------------------------------------
Ran 0 tests in 0.000s

OK
$
```

(This would be more interesting if there was a test to run.)

Pytest: a small taste

- discovery
- detailed error output
- parametrization

(Small gotcha: for historical reasons, pytest is run with a dot between py and test: `py.test`)

Pytest discovery is simple: if it starts with "test_" pytest will pick it up and run it.

test discovery (i.e. let py.test figure out what needs to run)

```
# hello.py
def greet(who="Students and professionals"):
    "Create a greeting"
    return "{}, good evening!".format(who)


def test_hello():
    greeting = greet()
    assert greeting == "Students and professionals, good evening!"


if "__main__" == __name__:
    greet()
```

```
$ py.test hello.py
============================== test session starts ==============================
platform linux2 -- Python 2.7.6 -- py-1.4.27 -- pytest-2.6.3
plugins: capturelog
collected 1 items

hello.py .

=========================== 1 passed in 0.02 seconds ============================
$
```

detailed failure output

```
# hello.py
def greet(who="Students and professionals"):
    "Create a greeting"
    return "{}, good evening!".format(who)


def test_hello():
    greeting = greet("Ladies and Gentlemen")
    assert greeting == "Students and professionals, good evening!"


if "__main__" == __name__:
    greet()
```

detailed failure output

```
$ py.test hello.py
============================= test session starts ==============================
platform linux2 -- Python 2.7.6 -- py-1.4.27 -- pytest-2.6.3
plugins: capturelog
collected 1 items

hello.py F

=================================== FAILURES ===================================
_____________________________________ test_hello _____________________________________

    def test_hello():
            greeting = greet("Ladies and Gentlemen")
            >       assert greeting == "Students and professionals, good evening!"
            E       assert 'Ladies and G...good evening!' == 'Students and ...good
            E         - Ladies and Gentlemen, good evening!
            E         + Students and professionals, good evening!

hello.py:9: AssertionError
=========================== 1 failed in 0.04 seconds ===========================
$
```

parametrizing tests

Some tips:

- Put your tests into a different module/file from the code they test.
- For bigger projects, put them in a whole different package/directory.

```
# hello.py
def greet(who="Students and professionals"):
    "Create a greeting"
    return "{}, good evening!".format(who)

if "__main__" == __name__:
    greet()
```

```
# test_hello.py

import pytest
import hello

@pytest.mark.parametrize('who', 'Boss')
def test_hello(who):
    greeting = hello.greet(who)
    assert greeting == "{}, good evening!".format(who)
```

parametrizing tests

```
$ py.test
============================= test session starts ==============================
platform linux2 -- Python 2.7.6 -- py-1.4.27 -- pytest-2.6.3
plugins: capturelog
collected 4 items

test_hello.py ....

=========================== 4 passed in 0.05 seconds ===========================
$
```

Hey, look, now that our tests are in `test_hello.py` py.test can find it without being told what file to look in!

parametrizing tests

```
$ py.test -v
============================= test session starts ==============================
platform linux2 -- Python 2.7.6 -- py-1.4.27 -- pytest-2.6.3 -- /home/buckles/.vir
plugins: capturelog
collected 4 items

test_hello.py::test_hello[B] PASSED
test_hello.py::test_hello[o] PASSED
test_hello.py::test_hello[s] PASSED
test_hello.py::test_hello[s] PASSED

=========================== 4 passed in 0.05 seconds ===========================
$
```

Whoops. Forgot that strings act as lists of letters.

parametrizing tests

Let's try that again with a real list this time.

```
# test_hello.py

import pytest
import hello


@pytest.mark.parametrize('who', [
            "Ladies and gentlemen",
            "Students and professionals",
            "Hackers and engineers"])
def test_greet(who):
    greeting = hello.greet(who)
    assert greeting == "{}, good evening!".format(who)
```

parametrizing tests

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.6 -- py-1.4.27 -- pytest-2.6.3 -- /home/buckles/.vir
plugins: capturelog
collecting ... collected 3 items

test_hello.py::test_greet[Ladies and gentlemen] PASSED
test_hello.py::test_greet[Students and professionals] PASSED
test_hello.py::test_greet[Hackers and engineers] PASSED

=========================== 3 passed in 0.05 seconds ===========================
```

That looks more like it.

Resources

Sites

Python Tools Taxonomy

PythonTesting.net Blog

pyvideo.org

People (Look for their talks on pyvideo)

Holger Krekel - py.test guy, located in Germany

Michael Foord Creator of the Python "Mocks" library

Mailing Lists

Python testing mailing list

Mailing List Archives