Python testing tools

Tomáš Ehrlich

29. srpna. 2013

assert statement

```
"""
examples/assert/euler01.py
Project Euler, problem 1

"""
def multiples35(upper_limit):
    # TODO: This is *not* the real answer
    if upper_limit == 10:
        return [3, 5, 6, 9]
    return []

# Check known outputs
assert multiples35(10) == [3, 5, 6, 9]
assert sum(multiples35(10)) == 23

# TODO: Get real answer
print(sum(multiples35(1000)))
```

- Jednoduché

- Jednoduché
- Bez konfigurace

- Jednoduché
- Bez konfigurace
- Může být součástí kódu (odstraněno při optimalizaci `-O`)

Jednoduchý type checking:

```
# redis_collections/base.py

def _create_new(self, data=None, key=None, ...):
    assert not isinstance(data, RedisCollection), \
        "Not atomic. Use '_data()' within ..."

    cls = cls or self.__class__
    ...
```

unittest

```
""" examples/unittest/test_crash.py """
import unittest
from factory import Car

class TestCar(unittest.TestCase):
    def setUp(self):
        self.car = Car()
        car.hit_wall(velocity=120)

    # def tearDown(self): pass

    def test_crash(self):
        self.assertTrue(car.is_damaged)

if __name__ == '__main__':
    unittest.main()
```

```
% python examples/unittest/test_crash.py
.
----------------------------------------
Ran 1 test in 0.000s

OK
```

```
F
======================================
FAIL: test_crash (__main__.TestCar)
--------------------------------------
Traceback (most recent call last):
  File "examples/unittest/test_crash.py", line 13, ...
    self.assertFalse(self.car.is_damaged)
AssertionError: True is not false

--------------------------------------
Ran 1 test 0.001s

FAILED (failures=1)
```

- Součástí standartní knihovny

- Součástí standartní knihovny
- Od verze 2.7/3.2 obsahuje `test discovery`

- Součástí standartní knihovny
- Od verze 2.7/3.2 obsahuje `test discovery`
- Sada testů (assertEqual, assertIn, ...)

- Součástí standartní knihovny
- Od verze 2.7/3.2 obsahuje `test discovery`
- Sada testů (assertEqual, assertIn, ...)
- Může se využít dědičnost tříd (django db backends)

```
class TestCar(unittest.TestCase):
    model = Car

    def setUp(self):
        self.car = self.model()
        ...

class TestBlueCar(TestCar):
    model = BlueCar

class TestHovercraft(TestCar):
    model = Hovercraft

    def test_flying(self):
        ...
```

Komplexních sady testů a porovnání (asserts) s podrobným chybovým výstupem

nose

```
def setup_func():
    global car
    car = Car()
    car.hit_wall(velocity=120)

@with_setup(setup_func)
def test_crash():
    global car
    assert car.is_damaged == True
```

- Jednoduchý zápis

- Jednoduchý zápis
- Obsahuje (hlavně) `test discovery`

- Jednoduchý zápis
- Obsahuje (hlavně) `test discovery`
- Může spouštět i TestCase z unittestu

Množství rozšiřujících modulů:

- Logcapture

Množství rozšiřujících modulů:

- Logcapture
- Test coverage

Množství rozšiřujících modulů:

- Logcapture
- Test coverage
- Paralelní spouštění testů

Množství rozšiřujících modulů:

- Logcapture
- Test coverage
- Paralelní spouštění testů
- Výběr testů podle atributů

Množství rozšiřujících modulů:

- Logcapture
- Test coverage
- Paralelní spouštění testů
- Výběr testů podle atributů
- ...

Nose není testovací framework, ale test runner, který spouštění testů usnadňuje a doplňuje.

py.test

```python
def multiples35(upper_limit):
    # TODO: This is *not* the real answer
    if upper_limit == 10:
        return [3, 5, 6, 9]
    return []

def test_known_inputs():
    # Check known outputs
    assert multiples35(10) == [3, 5, 6, 9]
    assert sum(multiples35(10)) == 23

def test_solution():
    assert sum(multiples35(1000)) != 0
```

```
% py.test euler01.py
== test session starts ==
platform linux2 -- Python 3.2.3 -- pytest-2.3.5
collected 2 items

euler01.py .F

== FAILURES ==
__ test_solution __

    def test_solution():
>       assert sum(multiples35(1000)) != 0
E       assert 0 != 0
E        +  where 0 = sum([])
E        +    where [] = multiples35(1000)

euler01.py:20: AssertionError
== 1 failed, 1 passed in 0.05 seconds ==
```

- Jednoduchý zápis

- Jednoduchý zápis
- Podrobnější chybový výstup než nose/assert statement

- Jednoduchý zápis
- Podrobnější chybový výstup než nose/assert statement
- Testy můžou být seskupeny do tříd

Jednoduchost `assert statementu` doplněná o podrobnější chybový výstup jako od `unittestu`.

unittest vs. nose vs. py.test

- unittest má testovací třídy (TestCase s assert*)

- unittest má testovací třídy (TestCase s assert*)
- nose má pluginy (coverage, multiprocessing)

- unittest má testovací třídy (TestCase s assert*)
- nose má pluginy (coverage, multiprocessing)
- py.test má skvělý chybový výstup a lightweight zápis

- Nose podporuje unittesty, doctesty i py.test

- Nose podporuje unittesty, doctesty i py.test
- Ideální kombinace s unittest a/nebo py.test

lettuce

```
Feature: Compute factorial
    In order to play with Lettuce
    As beginners
    We'll implement factorial

    Scenario: Factorial of 0
        Given I have the number 0
        When I compute its factorial
        Then I see the number 1
```

```
from lettuce import *

@step('I have the number (\d+)')
def have_the_number(step, number):
    world.number = int(number)

@step('I compute its factorial')
def compute_its_factorial(step):
    world.number = factorial(world.number)

@step('I see the number (\d+)')
def check_number(step, expected):
    expected = int(expected)
    assert world.number == expected, \
        "Got %d" % world.number

def factorial(number):
    return -1
```

- Inspirováno **Ruby** knihovnou `Cucumber`
- Test je popsán v čistém textu
- Musí se definovat parsery jednotlivých kroků

Co dál?

- **Testy** automatizují ověřování, že kód funguje

- **Testy** automatizují ověřování, že kód funguje
- Co potřebujeme teď, je **automatické** spouštění testů.

- **Testy** automatizují ověřování, že kód funguje
- Co potřebujeme teď, je **automatické** spouštění testů.
- **Tox** – testování v různých prostředích

- **Testy** automatizují ověřování, že kód funguje
- Co potřebujeme teď, je **automatické** spouštění testů.
- **Tox** – testování v různých prostředích
- **CI** (Jenkins, Travis) – spuštění testů při commitu do vcs

- **Testy** automatizují ověřování, že kód funguje
- Co potřebujeme teď, je **automatické** spouštění testů.
- **Tox** – testování v různých prostředích
- **CI** (Jenkins, Travis) – spuštění testů při commitu do vcs
- etc.

Děkuji za pozornost, trpělivost a všechny ty ryby.

Otázky?

tomas.ehrlich@gmail.com

Twitter: @tomas_ehrlich