# Mythbusters: Tests

Naty Bidart

PyCon Argentina 2010

15-16 de Octubre de 2010

# Outline

# Outline

# Testing vs. Tests

## Testing

- Etapa del proceso de desarrollo en cascada
- Independiente del desarrollo de código
- Posterior a la implementación

## Tests

- Parte del diseño
- Parte de la implementación
- Parte del testing!
- En esta charla hablaremos de **Tests**

# Por qué?

- Construir sobre bases sólidas
- Confianza (*as in Confidence, not trust*)
- Responsabilidades compartidas
- Menos stress!
- Traen calma, actúan como un arnés
- Son adictivos

## Es un camino de ida

Como  Python!

# Para qué?

- Calidad
- Corrección (y reducción de riesgo)
- Re-Usabilidad
- Comprensión
- Documentación
- Ejemplo de uso!

# Cómo?

- Todos los test juntos o un test a la vez?
  - Lista de tests pendientes (ToDo): Hoy, Semana, Mes
- De afuera hacia adentro? O de adentro hacia afuera?
- Chequear comportamiento o chequear estado?
- Fixture de datos particular a cada test o a nivel proyecto?

# Cuándo?

- Después?
- Antes!

**TDD**

*Test Driven Development*

- Siempre con la luz verde
- Si encontramos/nos reportan un bug (*Regression tests*)

# Qué se testea?

- TODO
- Para conjunto de datos grande, se elijen casos extremos (`None`, `[]`, `0`, etc.)

# Outline

# Colores

- Rojo: *red bar*
- Verde: *green bar*

# `unittest`

- Módulo de la biblioteca standard de Python
- **`unittest`** vs. **`unittest2`**
- Provee un marco de trabajo (*framework*) para escribir tests
- Permite agrupar test en **`TestCase`**s
- Permite agrupar test cases en **`TestSuite`**s
- Provee un conjunto de métodos para chequear propiedades
- Provee un *runner*

# `assert` vs. `self.assert*`

## `assert`

- Predicado que debe valer
- La veracidad es obvia
- Si no vale, las suposiciones nuestras están equivocadas

## `self.assert*`

- Aserciones sobre funcionalidad deseada
- Guían el desarrollo del código

# Error vs. Failure

## ERROR

- El código no ”compila”
- Typo!
- Algo que debía valer, no vale (por ej AssertionError)
- Excepciones inesperadas
- Un objeto no tiene un método

## FAILURE

- El código no cumple una característica deseada
- Queremos de éstas!

# Tests que sean...

- Aislados, sin efectos residuales
- Desacoplados
- Independientes del orden en que se corren
- Reproducibles (nada de aleatoriedad)
- Autocontenido, auto explicativo

# Y además

- Fáciles de correr
  - Toda la test suite, o un test solo, o un grupo particular
- Fáciles de mantener
- Fáciles de expandir/modificar
- Fáciles de determinar el estado (red/green bar)
- Rápidos de correr
- Poder volver a un estado verde anterior
- Entornos pristinos (**`setUp`**, **`tearDown`**)

# `setUp` y `tearDown`

- Los test se agrupan en test cases
- Cada test case tiene su **`setUp`** y **`tearDown`**
- Se pueden customizar según cómo agrupemos

## `setUp`

- Se llama siempre, siempre, antes de cada test

## `tearDown`

- Se llama siempre, siempre, después de cada test
- Aunque el test falle
- Pero no si **`setUp`** falla

# Dependencia entre tests

- El resto de la suite se asume correcto

# Dependencia con módulos

- Monkey patching
  - **twisted.trial.unittest.TestCase.patch**
  - **mock.patch**
  - custom!

```
def patch(self, obj, name, value):
    current = getattr(obj, name)
    setattr(obj, name, value)
    self.addCleanup(setattr, obj, name,
                    current)
    return current
```

- Mocks (python-mocker, python-mock)
- Fakes (your owns!)

# `self.addCleanup` rocks

```
self.addCleanup(f1)
self.addCleanup(f2)
# ...
self.addCleanup(fn)
```

- Todas las *fn* se ejecutan siempre
- Incluso si **`self.setUp`** falla
- Es una LIFO

# Sugiero fuertemente...

## Beautiful is better than ugly

- pylint
- pep8

## Now is better than never

- Leer projectos que tengan una buena test suite
- Creer!
- Experimentar

# Buenas prácticas

## No hacer...

- Duplicación de código
- ~~Duplicación de código~~
- Uso de literales
- ~~Uso de literales~~
- Reinventar la rueda
- ~~Reinventar la rueda~~
- Reinventar patrones
- ~~Reinventar patrones~~

# Outline

# Referencias


- Test Driven Development by Example (Kent Beck)
- xUnit Test Patterns: Refactoring Test Code (Gerard Meszaros)
- unittest
- unittest2
- Rudolf
- py.test
- nose
- PyCon 2010: Tests and Testability
- PyCon 2010: New *and* Improved: Coming changes to unittest, the standard library test framework
- PyCon 2010: py.test - Rapid Testing with Minimal Effort

# Outline

# El patrón de escritura de tests

- Crear un objeto $X$
- Ejercitar un método de $X$
- Chequear una propiedad del estado de $X$

# Estructura en disco

```
my_package/
    __init__.py
    my_module.py
    tests/
        __init__.py
        test_my_module.py
```

# Errors should never pass silently

- Sí, también hay que testear los logs!
  - Que el código loguee en situaciones críticas/inesperadas
  - Que la info logueada transmita datos claves
  - Y ojo, que el formateo del log no falle:

```
import logging
logging.error('Got unexpected value %r.',
              a_variable)
```

- Es muy importante testear que se loguean las excepciones
  - **`logging.error`** vs. **`logging.exception`**

# Para los greedies

- **`unittest.skip`**
- **`unittest.skipIf`**
- **`unittest.skipUnless`**
- **`unittest.expectedFailure`**
- **`twisted.trial.unittest.ToDo`**
- **`xvfb-run`** para correr test suites que usen el X server

# Adivina adivinador

```
[FAIL]: my_package.tests....

twisted.trial.unittest.FailTest: not equal:
a = '9999 bytes used of 12345 bytes (0.0%)'
b = '9999 bytes used of 12345 bytes (81.0%)'
```