Unit Testing con PHPUnit y PyUnit

En la edición Nº3 de Hackers & Developers Magazine hicimos una introducción al desarrollo dirigido por pruebas, prometiendo hablar de PyUnit y PHPUnit en una siguiente entrega. Como lo prometido es deuda, aquí está TDD con Python y PHP.

Escrito por: **Eugenia Bahit** (Arquitecta GLAMP & Agile Coach)

Eugenia es **Arquitecta de Software**, **docente** instructora de tecnologías **GLAMP** (GNU/Linux, Apache, MySQL, Python y PHP) y **Agile coach** (UTN) especializada en Scrum y eXtreme Programming. Miembro de la **Free Software Foundation** e integrante del equipo de **Debian Hackers**.

Webs:
Cursos de programación a Distancia: www.cursosdeprogramacionadistancia.com
Web personal: www.eugeniabahit.com

Redes sociales:
Twitter / Identi.ca: **@eugeniabahit**

Pruebas Unitarias en PHP

Existen varios *frameworks* xUnit para Unit Testing en PHP, pero sin dudas, el único que ha demostrado contar con una gran cobertura de código, estabilidad y buena documentación, es PHPUnit. El manual oficial de PHPUnit (en inglés) puede encontrarse en: http://www.phpunit.de/manual/current/en/index.html

Se puede instalar PHPUnit en sistemas operativos GNU/Linux, mediante PEAR:

```
sudo pear upgrade PEAR
pear config-set auto_discover 1
pear install pear.phpunit.de/PHPUnit
```

Aunque en distribuciones basadas en Debian, también puede hacerse directamente mediante la instalación del paquete `phpunit` con `apt-get`:

```
sudo apt-get install phpunit
```

Métodos Assert de PHPUnit

PHPUnit provee una gran cantidad de métodos `assert`, cuyas referencias podemos encontrar en el Capítulo 4 del manual oficial:

http://www.phpunit.de/manual/3.6/en/writing-tests-for-phpunit.html

Algunas características comunes de los métodos `assert`, son:

- Generalmente, por cada método `assert` existe su opuesto: `assertContains()` y `assertNotContains()`.
- A la vez, cada método `assert` deberá recibir mínimamente un parámetro que será el resultado de ejecutar el código del SUT.
- Adicionalmente, a cada método `assert`, se le puede pasar como parámetro opcional, un mensaje personalizado para ser arrojado en caso de error (generalmente, será el último parámetro).
- Los métodos `assert` que requieren el paso de dos parámetros obligatorios (valores que deben compararse entre sí), generalmente guardan el siguiente orden:

```
metodoAssert($valor_esperado, $valor_recibido)
```

Es decir, que en esos casos, siempre el primer parámetro será el valor esperado y el segundo parámetro, el valor recibido por la ejecución del código SUT.

Veamos algunos ejemplos puntuales:

```
<?php
# Archivo TestCase: BalanceContable_Test.php

require_once 'BalanceContable.php';  # SUT

class BalanceContableTest extends PHPUnit_Framework_TestCase {

    public function setUp() {
        $this->coverage = new BalanceContable();
        $this->coverage->alicuota_iva = 21;
    }

    // AssertEquals($valor_esperado, $valor_recibido)
    public function test_calcular_iva() {
        $this->coverage->importe_bruto = 1500;
        $result = $this->coverage->calcular_iva();
        $this->assertEquals(315, $result);
    }

    // AssertTrue($valor_recibido)
    public function test_alcanzado_por_impuesto_de_importacion_con_160() {
        $this->coverage->importe_bruto = 160;
```

```
        $result = $this->coverage->alcanzado_por_impuesto_de_importacion();
        $this->assertTrue($result);
    }

    // AssertNull($valor_recibido)
    public function test_alcanzado_por_impuesto_de_importacion_con_143() {
        $this->coverage->importe_bruto = 143;
        $result = $this->coverage->alcanzado_por_impuesto_de_importacion();
        $this->assertNull($result);
    }
}

?>


# Código SUT: BalanceContable.php
<?php

class BalanceContable {

    public $importe_bruto;
    public $alicuota_iva;

    # Calcular IVA sobre un importe bruto
    public function calcular_iva() {
        $iva = $this->alicuota_iva / 100;
        $neto = $this->importe_bruto * $iva;
        return $neto;
    }

    # Determinar si un importe paga impuesto de importación
    public function alcanzado_por_impuesto_de_importacion() {
        // importes mayores a 150 USD pagan impuesto
        if($this->importe_bruto > 150) {
            return True;
        }
    }

}

?>
```

Pruebas Unitarias en Python

PyUnit es el *framework* xUnit elegido de forma oficial por Python desde su versión 1.5.2. Si bien existen muchos otros, generalmente están destinados a ampliar los beneficios de PyUnit para la realización de test más complejos, como el caso de PyDoubles[11] -creado por Carlos Blé-. Toda la referencia sobre PyUnit se encuentra en el manual oficial de Python (en inglés): http://docs.python.org/library/unittest.html

PyUnit no necesita ser instalado ya que desde la versión 2.1 forma parte de la librería estándar de Python, a través del módulo unittest.

11 http://www.carlosble.com/category/software-development/pydoubles/

Métodos Assert de PyUnit

Una lista completa de los métodos assert de PyUnit puede encontrarse en la documentación sobre `unittest` de Python en la siguiente URL:

http://docs.python.org/library/unittest.html#assert-methods

Una diferencia particular que existe entre PyUnit y otros frameworks como PHPUnit, es que nos permite efectuar afirmaciones, con una sintaxis bastante simple, sin necesidad de recurrir a métodos `assert` específicos:

```
assert resultado == valor_esperado
```

En un ejemplo más concreto, podríamos verlo así (donde *coverage* será la instancia al objeto del SUT):

```
assert self.coverage.sumar_dos_numeros(5, 15) == 20
```

Otra diferencia fundamental con PHPUnit, es que el método `assert`<*Igualdad*>, posee su nombre en singular:

```
PHPUnit:
    assertEquals($a, $b);

PyUnit:
    asserEqual(a, b)
```

Algunas características comunes de los métodos `assert`, son:

- Al igual que con PHPUnit, generalmente, por cada método `assert` existe su opuesto: `assertEqual()` y `assertNotEqual()`.
- También, siguiendo nuevamente la línea de PHPUnit, cada método `assert` deberá recibir mínimamente un parámetro que será el resultado de ejecutar el código del SUT y opcionalmente, como último parámetro, puede recibir un mensaje personalizado para ser arrojado en caso de error.
- A diferencia de PHPUnit, los métodos `assert` que requieren el paso de dos parámetros obligatorios (valores que deben compararse entre sí), generalmente guardan el siguiente orden:

```
metodoAssert(valor_recibido, valor_esperado)
```

Es decir, que en esos casos, siempre el primer parámetro será el valor recibido

por la ejecución del código SUT y el segundo parámetro, el valor esperado.

Veamos el ejemplo realizado anteriormente en PHP, pero esta vez, en Python con PyUnit:

```
# -*- coding: utf-8 -*-
# Archivo TestCase: test_balance_contable.php
import unittest
from balance_contable import BalanceContable


class BalanceContableTestCase(unittest.TestCase):

    # setUp()
    def setUp(self):
        self.coverage = BalanceContable()
        self.coverage.alicuota_iva = 21

    # assertEqual(valor_recibido, valor_esperado)
    def test_calcular_iva(self):
        self.coverage.importe_bruto = 2500
        result = self.coverage.calcular_iva()
        self.assertEqual(result, 525)

    # AssertTrue(valor_recibido)
    def test_alcanzado_por_impuesto_de_importacion_con_160(self):
        self.coverage.importe_bruto = 160
        result = self.coverage.alcanzado_por_impuesto_de_importacion()
        self.assertTrue(result)

    # AssertIsNone(valor_recibido)
    def test_alcanzado_por_impuesto_de_importacion_con_143(self):
        self.coverage.importe_bruto = 143
        result = self.coverage.alcanzado_por_impuesto_de_importacion()
        self.assertIsNone(result)


# Necesario para correr los test si es llamado por línea de comandos
if __name__ == "__main__":
    unittest.main()
```

Nótese que el método assertEquals de PHPUnit, se denomina assertEqual (en singular) en PyUnit y que en reemplazo del método assertNull, PyUnit propone assertIsNone (esto es debido a que Python no retorna valores nulos como tales, sino como “None”).

```
# -*- coding: utf-8 *-*
# Código SUT: balance_contable.php


class BalanceContable(object):

    def __init__(self):
        self.importe_bruto = 0
        self.alicuota_iva = 0
```

```
    # Calcular IVA sobre un importe bruto
    def calcular_iva(self):
        iva = self.importe_bruto * self.alicuota_iva / 100
        return iva

    # Determinar si un importe paga impuesto de importación
    def alcanzado_por_impuesto_de_importacion(self):
        # importes mayores a 150 USD pagan impuesto
        if self.importe_bruto > 150:
            return True
```

"Descubriendo" Test en Python

Desde la versión 2.7 de Python, ya no es necesario realizar "maniobras" o crear Test Suites, con el único fin de correr todos los Test de nuestra aplicación, de un solo paso. Todos los test de una aplicación, pueden correrse mediante el comando `discover`:

```
eugenia@cocochito:~/proyectos$ python -m unittest discover
```

`discover`, "descubrirá" todos los test, identificándolos por el nombre del archivo: debe comenzar por el prefijo "test" (`discover` utiliza la expresión regular test*.py para identificar Test Cases). Además, debe tenerse en cuenta que el nombre de los métodos de prueba, también deben comenzar por el prefijo test.

Sin embargo, podría pretender ejecutarse solo un TestCase:

```
eugenia@cocochito:~/proyectos$ python test_balance_contable.py
```

O un test en particular de una clase TestCase:

```
eugenia@cocochito:~/proyectos$ python test_balance_contable.py
BalanceContableTestCase.test_calcular_iva
```

También es posible, pasar el parámetro `-v` a fin de obtener un reporte más detallado:

```
eugenia@cocochito:~/proyectos$ python -m unittest discover -v
test_alcanzado_por_impuesto_de_importacion_con_143
(Test.test_balance_contable.BalanceContableTestCase) ... ok
test_alcanzado_por_impuesto_de_importacion_con_160
(Test.test_balance_contable.BalanceContableTestCase) ... ok
test_calcular_iva (Test.test_balance_contable.BalanceContableTestCase) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```