Ruby Tutorial

...o como pasar un buen rato programando

Lección 1

Lección 2

- Rangos
- Arrays

Lección 3

- Bloques
- Más malabares con strings
- Expresiones Regulares
- Condicionales
- Bucles
- Números Aleatorios

Lección 4

- Clases y Objetos
- Accesores
- Ficheros: lectura/escritura
- Cargando librerías
- Herencia de clases
- Modificando clases
- Congelando objetos
- Serializando objetos

Lección 5

- Control de acceso
- Excepciones
- Módulos
- Constantes
- Hashes y Símbolos
- La clase Time

Lección 6

- self
- Duck Typing
- Azúcar Sintáctico
- Test de unidades

contacto

e-mail

Unit Testing

vía Gluttonous

Test de unidades

El **test de unidades** es un método para testear el código en pequeños trozos.

Por qué

- Significa que nunca tendrás el problema de crear un error mientras solucionas otro.
- Significa que no tendrás que ejecutar tu programa y jugar con él (lo que es lento) para arreglar los errores. El testeo de unidades es mucho más rápido que el "testeo manual".
- Conociendo cómo usar las unidades de test, abre el mundo al Desarrollo Guiado por Pruebas (Test Driven Development, TDD).

Requisitos

- Cargar la biblioteca 'test/unit'

- Hacer que la clase a testear sea una subclase de `Test::Unit::TestCase`
- Escribir los métodos con el prefijo `test_`
- Afirmar (`assert`) las cosas que decidas que sean ciertas.
- Ejecutar los tests y corregir los errores hasta que desaparezcan.

```
require 'test/unit'

class MiPrimerTest < Test::Unit::TestCase
  def test_de_verdad
    assert true
  end
end
```

Cada afirmación, es un método heredado de la clase `Test::Unit::TestCase`: Hay que echar un ojo al listado de las posibles afirmaciones (asserts) que podemos comprobar.

Ejemplo

Supongamos que queremos escribir una clase sencilla, `Mates`, que implemente operaciones aritméticas básicas. Queremos hacer distintos tests para comprobar que la suma, la resta, el producto y la división funcionan.

```
require 'mates'
require 'test/unit'

class TestDeMates < Test::Unit::TestCcase
  def test_suma
    assert_equal 4, Mates.run("2+2")
    assert_equal 4, Mates.run("1+3")
    assert_equal 5, Mates.run("5+0")
    assert_equal 0, Mates.run("-5 + 5")
  end

  def test_resta
    assert_equal 0, Mates.run("2-2")
    assert_equal 1, Mates.run("2-1")
    assert_equal -1, Mates.run("2-3")
  end
end
```

Si ejecutamos el programa, aparecerán siete puntos '…….'. Cada **.** es un test que se ha ejecutado, **E** es un error y cada **F** un fallo.

```
Started
```

```
.......
Finished in 0.015931 seconds.
7 tests, 13 assertions, 0 failures, 0 errors
```

Unidades de test negativas

Además de los tests positivos, también se pueden escribir unidades de tests negativas intentando romper el código. Esto puede incluir el testeo para excepciones que surgan de usar entradas como Mates.run("a + 2") o Mates.run("4/0").

```
def test_para_no_numericos
    assert_raises(ErrorNoNumerico) do
      Mates.run("a + 2")
    end
  end

  def test_division_por_cero
    assert_raises(ErrorDivisionPorZero) do
      Mates.run("4/0")
    end
  end
```

Automatizando tests: setup, teardown y rake

Algunas veces necesitamos que ocurran cosas antes y después de cada test. Los métodos **`setup`** y **`teardown`** son tus compañeros en esta aventura. Cualquier código escrito en `setup` será ejecutado antes del código, y el código escrito en `teardown` será ejecutado a posteriori.

Si estás escibiendo tests para todo tu código (como debería ser), el número de ficheros a testear empieza a crecer. Una cosa que puede facilitarte la vida, es automatizar los tests, y **`rake`** es la herramienta para este trabajo.

fichero_rake

```
require 'rake'
require 'rake/testtask'

task :default => [:test_units]

desc "Ejecutando los tests"
Rake::TestTask.new("test_units") { |t|
  t.pattern = 'test/*_test.rb'  # busca los ficheros acabados en '_test.rb'
  t.verbose = true
```

```
  t.warning = true
}
```

Básicamente, un **fichero_rake** define las tareas que `rake` puede hacer. En el fichero_rake, la tarea por defecto (la que sucede cuando se ejecuta rake en un directorio con un fichero_rake en él) es configurada hacia la tarea `tests_units`. En la tarea tests_units, `rake` es configurado para buscar ficheros en el directorio que terminen en "_test.rb" y los ejecute. Resumiendo: puedes poner todos los tests en un directorio y dejar que `rake` haga el trabajo.

Other interesting sites

Soymilk Linkshell

Professional, drama-free dynamis since 2005!

Librairie d'Orléans

Lords von Dante

Portland ImmunoResearch Group