Ruby Tutorial

...o como pasar un buen rato programando

Lección 1

- Introducción
- Instalación
- El Primer Programa
- Números en Ruby
- Strings y diversión
- Variables
- Alcance de las variables

Lección 2

- Introduciendo Datos
- Normas en los nombres

- Los métodos
- Los métodos: argumentos
- 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

contacto

e-mail

Duck Typing

A estas alturas, te habrás dado cuenta de que en Ruby no se declaran los tipos de variables o métodos: todo es un objeto. Los objetos en Ruby pueden ser modificados: siempre se pueden añadir métodos a posteriori. Por lo tanto, el comportamiento del objeto, puede alejarse de aquel suministrado por su clase.

En Ruby, nos fijamos menos en el tipo (o clase) de un objeto y más en sus capacidades. **Duck Typing** se refiere a la tendencia de Ruby a centrarse menos en la clase de un objeto, y dar prioridad a su comportamiento: qué métodos se pueden usar, y qué operaciones se pueden hacer con él.

Se llama "Duck Typing" porque está basado en el Test del Pato (Duck Test):

> Si camina como un pato, nada como un pato y hace "quack", podemos tratarlo como un pato. *James Whitcomb Riley*

Veamos el siguiente ejemplo:

```
# Comprobamos qué objetos responden al método t_str
puts ('Una cadena'.respond_to? :to_str)   # => true
puts (Exception.new.respond_to? :to_str)  # => true
puts (4.respond_to? :to_str) # => false
```

Este ejemplo, es una forma simple de la filosofía "pato typing": si un objeto hace quack como un pato (o actúa como un string), pues trátalo como un pato (o una cadena). Siempre hay que tratar a los objetos por lo que pueden hacer, mejoer que hacerlo por las clases de las que proceden o los módulos que incluyen.

Las **excepciones** (Exceptions) son un tipo de string que tienen información extra asociada con ellas. Sin embargo, aunque ellas no son una subclase de "String", pueden ser tratadas como tales.

¡Tratémoslos como patos!

```
class Pato
  def quack
    'Quack!'
  end

  def nadar
    'Paddle paddle paddle...'
  end
end

class Ganso
  def honk
    'Honk!' # onomatopia de un pato
  end
  def nadar
    'Splash splash splash...'
  end
end

class GrabadoraDePatos
  def quack
    play
  end

  def play
    'Quack!'
```

```
  end
end

# En este método, la Grabadora
# se comporta como un Pato
def haz_quack(pato)
  pato.quack
end
puts haz_quack(Pato.new)
puts haz_quack(GrabadoraDePatos.new)

# Para este método, el Ganso
# se comporta como un Pato
def haz_nadar(pato)
  pato.nadar
end
puts haz_nadar(Pato.new)
puts haz_nadar(Ganso.new)
```


Other interesting sites

Echo Bazaar

Come to Fallen London

Handbook

The Wikidot Handbook

iSchool High

TENNESSEESPOKES