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

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

contacto

e-mail

Clases y Objetos

Desde hace tiempo el estilo de programación funcional (que se usa por ejemplo en el lenguaje C) es usado para programar. En este tipo de programación, hay que centrarse en los pasos para realizar la tarea, y nos olvidamos de como se manejan los datos.

Sin embargo, en la programación orientada a objetos, los objetos son los agentes, el universo de tu programa: se presta atención a la estructra de los datos. Cuando se diseña una clase, se piensa en los objetos que serán creados por esa clase: en las cosas que podrá hacer ese objeto, y las características que lo definen.

Un **objeto** es un contenedor de datos, que a su vez controla el acceso a dichos datos. Asociados a los objetos está una serie de variables que lo definen: sus **atributos**. Y también un conjunto de funciones que crean un interfaz para interactuar con el objeto: son los **métodos**.

> Un objeto es una combinación de estado y de métodos que cambian ese estado.

Una **clase** es usada para construir un objeto. Una clase es como un molde para objetos. Y un objeto, una instancia de la clase. Por ejemplo, se puede usar la clase `Button` para hacer docenas de botones, cada botón con su propio color, tamaño, forma,…

Nuestra primera clase

```
# define la clase Perro
class Perro

  # método inicializar clase
  def initialize(raza, nombre)
    # atributos
    @raza = raza
    @nombre = nombre
  end

  # método ladrar
  def ladrar
    puts 'Guau! Guau!'
  end

  # método saludar
  def saludar
    puts "Soy un perro de la raza #{@raza} y mi nombre es #
{@nombre}"
  end
end

# para hacer nuevos objetos,
# se usa el método new
d = Perro.new('Labrador', 'Benzy')
puts d.methods.sort
puts "La id del ojbeto es #{d.object_id}."

if d.respond_to?("correr")
  d.correr
else
  puts "Lo siento, el objeto no entiende el mensaje 'correr'"
end

d.ladrar
```

```
d.saludar

# con esta variable, apuntamos al mismo objeto
d1 = d
d1.saludar

d = nil
d.saludar
```

El método **new** se usa para crear un nuevo objeto de la clase Perro. Los objetos son creados en el momento y el espacio de memoria donde se guardan, se asignan a una variable, en este caso la variable **d**, que se conoce como variable de referencia.

Un método recién creado no es un espacio en blanco: un objeto recién creado, puede responder un montón de mensajes. Para ver la lista de esos mensajes o métodos de forma ordenada (.sort):

```
puts d.methods.sort
```

El resultado es una lista de todos los mensajes o métodos que el objeto recién creado puede responder. De todos esos métodos, los `object_id` y `respond_to?` son importantes.

object_id, respond_to?

Cada objeto en Ruby tiene un único número **id** asociado con él. Se puede ver dicho número mediante el método **`object_id`**. En nuestro ejemplo:

```
puts "La id del ojbeto es #{d.object_id}."
```

Se puede conocer de antemano, si un objeto será capaz de responder a un mensaje; o dicho de otra forma, si un objeto posee cierto método. Para ellos se usa **`respond_to?`**. En nuestro ejemplo:

```
if d.respond_to?("correr")
  d.correr
else
  puts "Lo siento, el objeto no entiende el mensaje 'correr'"
end
```

class, instance_of?

Puedes saber a qué clase pertenece un objeto mediante el método **`class`**. En nuestro ejemplo si

ponemos:

```
d = Perro.new('Alsatian', 'Lassie')
puts d.class.to_s # obtenemos Perro
```

`instance_of?` nos devuelve `true` si un objeto es instancia de una clase determinada. Por ejemplo:

```
num = 10
puts (num.instance_of? Fixnum) # true
```

La clase Class

En [este diagrama] se vió como todas las clases descendían de la clase Class.

Las clases en Ruby son instancias de la clase **`Class`**. Cuando una nueva clase se define (p.e. `class NombreClase ... end`), un objeto de la clase `Class` es creado y asignado a una constante (en este caso `NombreClase`). Cuando se usa `NombreClase.new` para construir un nuevo objeto, se usa el método de la clase `Class` para crear nuevas instancias; y después se usa el método inicializador de la propia clase `NombreClase`: la construcción y la inicialización de un objeto son cosas distintas, y pueden modificarse.

Constructores literales

Significa que se puede usar una notación especial, en vez de usar `new` para crear un nuevo objeto de esa clase. Las clases que un constructor literal puede crear, están en la siguiente tabla: cada vez que usas uno de estos constructores, creas un nuevo objeto.

Clase	**Constructor Literal**	**Ejemplo**
String	' ó "	`"nuevo string"` o `'nuevo string'`
Símbolo	:	`:símbolo` ó `:"símbolo con espacios"`
Array	[ ]	[1, 2, 3, 4, 5]
Hash	{ }	`{"Nueva Yor" => "NY", "Oregon" => "OR"}`
Rango	.. ó ...	`0...10` ó `0..9`
Expresiones regulares	/	`/([a-z]+)/`

Por esto, y aunque a veces no lo parezca, siempre estamos creando objetos. En Ruby, todo es un objeto.

Colector de basura

La instrucción:

```
d = nil
```

hace que d apunte a `nil`, lo que significa que no se refiere a nada. Si después de eso, añado:

```
d1 = nil
```

entonces el objeto Perro dejará de estar apuntado por las variables y será objetivo del **colector de basura**. El colector de basusra de Ruby es del tipo marcar-y-borrar (mark-and-sweep):

- en la fase "mark" el colector verifica si el objeto está en uso. Si un objeto es apuntado por una variable, podría ser usado, y por tanto ese objeto se marca para ser conservado.
- si no hay variable que apunte al objeto, entonces el objeto no es marcado. Y en la fase "sweep" se borran los objetos en desuso para liberar memoria.

Ruby usa un mark-and-sweep conservativo: no hay garantía de que un objeto sea eliminado por el colector, antes de que termine el programa. Si almacenas algo en un array, y se mantiene el array, todo dentro del array es marcado. Si almacenas algo en una constante o variable global, entonces se marca para siempre.

Métodos de clase

La idea de los **métodos de clase** es mandar el mensaje a la clase, en vez de una de sus instancias. Los métodos de clase se usan porque algunas operaciones que pertenecen a una clase, no pueden ser realizadas por sus instancias. **`new`** es un buen ejemplo. La tarea de crear nuevos objetos sólo la puede hacer la clase; los objetos no pueden crear nuevos objetos.

page_revision: 6, last_edited: 3 Jun 2009, 04:55 GMT-05 (545 days ago)

Other interesting sites

Windycity

MUX

The Inkwell

A fountain pen ink and paper wiki

UR Mission Wiki

LOA FTW

Karma-Lab

KARMA Technology and the products incorporating it