Taller de Ruby Básico

Guillermo Valdés Lozano

20 de noviembre de 2009

Documento protegido por GFDL

A cerca de Ruby

- Es un lenguaje de programación interpretado y orientado a objetos.
- En ruby **todo es un objeto**.
- Creado por el programador japonés **Yukihiro "Matz" Matsumoto**, quien comenzó a trabajar en Ruby en 1993, y lo presentó públicamente en 1995.
- La última versión estable es la **1.8.6**, publicada en diciembre de 2007.
- Es Software Libre.

Reflexión de Yukihiro “Matz” Matsumoto

A menudo la gente, especialmente los ingenieros en informática, se centran en las máquinas. Ellos piensan, *“Haciendo esto, la máquina funcionará más rápido. Haciendo esto, la máquina funcionará de manera más eficiente. Haciendo esto...”* Están centrados en las máquinas, pero en realidad necesitamos centrarnos en las personas, en cómo hacen programas o cómo manejan las aplicaciones en los ordenadores. Nosotros somos los jefes. Ellos son los esclavos.

Comezando

Averigua la versión de Ruby instalada.

```
$ ruby -v
ruby 1.8.7 (2009-06-12 patchlevel 174) [i686-linux]
```

Podríamos usarlo intectivamente, pero no lo recomendamos.

```
$ ruby
puts "Hola Mundo !"
^D
Hola Mundo !
```

irb, la terminal de Ruby

Ingrese escribiendo el comando...

```
$ irb
```

Y tenga la libertad de ingresar instrucciones...

```
def suma(n1, n2)
  n1 + n2
end
suma(3, 4)           -> 7
suma('cat', 'dog') -> 'catdog'
```

Para salir escriba...

```
exit
```

Programas de Ruby en archivos rb

Se puede ejecutar un archivo de Ruby **.rb** pasándole como parámetro el nombre del mismo.

```
$ ruby mi_programa.rb
```

O bien, que la primera línea declare el intérprete.

```
#!/usr/bin/env ruby
```

Y cambie el modo a ejecutable.

```
$ chmod a+x mi_programa.rb
```

Así podrá ejecutarlo directamente.

```
$ ./mi_programa.rb
```

La sintaxis de Ruby es simple

- No necesitas ; después de cada instrucción.
- Puedes poner comentarios con #
- El espaciado no afecta al programa.
- No hay que declarar variables antes de usarlas.

Métodos de un objeto

Practiquemos algunos **métodos** sobre un texto, que es un **objeto**.

```
"Guillermo".length          -> 9
"Guillermo".index('r')      -> 6
"Guillermo".reverse         -> "omrelliuG"
"Guillermo".reverse.upcase -> "OMRELLIUG"
```

Método simple

Con **def** y **end** indicamos el inicio y término de un **método** creado por nosotros.

```
def buenas_noches(nombre)
  entregar = "Buenas Noches " + nombre
  return entregar
end
puts buenas_noches('memo') -> Buenas Noches memo
puts buenas_noches('rosy') -> Buenas Noches rosy
```

Sustitución de variables con comillas dobles

Los textos con comillas dobles pueden sustituir variables por sus valores.

```
def buenas_noches(nombre)
  entregar = "Buenas Noches #{nombre}"
  return entregar
end
puts buenas_noches('memo') -> Buenas Noches memo
puts buenas_noches('rosy') -> Buenas Noches rosy
```

Expresiones complejas

Dentro de las llaves podrán llamarse a métodos o efectuarse operaciones. Además, como el resultado de la última instrucción del método es lo que entrega, nos ahorramos el **return**.

```
def buenas_noches(nombre)
  "Buenas Noches #{nombre.capitalize}"
end
puts buenas_noches('memo') -> Buenas Noches Memo
puts buenas_noches('rosy') -> Buenas Noches Rosy
```

Arreglos

Un **arreglo** es un conjunto ordenado de objetos. El primer elemento tiene como índice el 0.

```
a = [1, 'cat', 3.4]
a[0] -> 1
a[2] -> 3.4
a[2] = nil
a -> [ 1, 'cat', nil]
```

Para declarar un arreglo vacío, puede usar:

```
a = Array.new
```

Arreglos de textos

Hay dos formas de definir un arreglo que contiene palabras, la tradicional.

```
a = ['hormiga', 'abeja', 'gato', 'perro']
```

Y la simplificada

```
a = %w{hormiga abeja gato perro}
```

Hash

En un hash, cada elemento se accede por una clave única.

```
instrumentos_musicales = {
  'violin'   => 'cuerda',
  'tambor'   => 'percución',
  'trompeta' => 'viento'
}
instrumentos_musicales['violin']   -> "cuerda"
instrumentos_musicales['director'] -> nil
```

Para declarar un hash vacío, use:

```
directorio = Hash.new
```

Estructura de control if

Si el resultado de la expresión lógica es verdadera se ejecutan las instrucciones.

```
if nombre.length > 10
    puts "¡Tiene #{nombre.length} caracteres tu nombre!"
end
```

Use **elsif** para efectuar otra prueba si la anterior resultó falsa. Y **else** por si todo lo anterior dio falso.

```
if nombre.length > 10
    puts "¡Tiene #{nombre.length} caracteres tu nombre!"
elsif nombre.length > 0
    puts "Creo que #{nombre} es corto."
else
    puts "El nombre está vacío."
end
```

Una sola instrucción bajo una condición

Una estructura if con una sola instrucción como la siguiente...

```
if radiacion > 3000
  puts "¡Peligro! ¡Peligro!"
end
```

Puede simplificarse de esta forma...

```
puts "¡Peligro! ¡Peligro!" if radiacion > 3000
```

Estructura de control while

Por medio de **while** se hace un bucle que continúa mientras la prueba lógica sea verdadera.

```
puts "Tabla de multiplicar del 2"
contador = 0
while contador < 20
  contador += 1
  puts "2 x #{contador} = #{2*contador}"
end
```

Una sola instrucción bajo una while

Si una sola instrucción va estar dentro de un **while**...

```
cuadrado = 2
while cuadrado < 1000
    cuadrado = cuadrado * cuadrado
end
puts cuadrado
```

Entonces puede simplifcarse de esta forma.

```
cuadrado = 2
cuadrado = cuadrado * cuadrado while cuadrado < 1000
puts cuadrado
```

Expresiones regulares

Las **expresiones regulares** son patrones que buscan algo en un texto.

```
if lenguaje ~= /Perl|Python/
    puts "¡Qué gusto que usa #{lenguaje}!"
end
```

Caracteres especiales

Dentro de una expresión regular pueden usarse estos códigos:

- + para indicar uno o más caracteres.
- * para indicar ninguno o más caracteres
- **s** un caracter de espaciado (espacio, tabulador, avance de línea).
- **d** un caracter dígito.
- **w** un caracter alfabético.
- **.** (punto) para cualquier caracter.

Ejemplos:

```
puts "La hora es correcta" if hora ~= /\d\d:\d\d/
puts "Es un archivo AVI" if archivo ~= /.+\.avi/
```

Reemplazo de texto

Para sustituir la primer aparición de *Perl* por *Ruby*.

```
linea.sub(/Perl/, 'Ruby')
```

Para sustituir todos los *Perl*.

```
linea.gsub(/Perl/, 'Ruby')
```

Para sustituir todos los *Perl* y *Phyton*

```
linea.gsub(/Perl|Phyton/, 'Ruby')
```

Bloques

Un bloque es un conjunto de intrucciones entre llaves...

```
{ puts "Gracias por asistir al taller." }
```

O entre un **do** y un **end**...

```
do
    contador += 1
    puts "Van #{contador} ciclos."
end
```

Un bloque puede será ejecutado por una *invocación*.

Bloques

En este ejemplo, el bloque es ejecutado en cada **yeild**.

```
def hacer_documento
    puts "Incio del documento"
    yeild
    yeild
    puts "Fin del documento"
end
hacer_documento { puts "contenido" }
```

Iteradores

Un **iterador** es un método que entrega elementos de alguna clase de colección. Necesita un **bloque** con las instrucciones a realizar.

```
animales = %w{ hormiga abeja gato perro ratón }
animales.each { |animal| puts animal }
```

Muchas formas de **bucles** pueden hacerse por iteradores.

```
5.times { puts "¡Gracias por tomar este taller!" }
1.upto(10) { |n| puts "2 x #{n} = #{2*n}" }
('a'..'z').each { |letra| print letra }
```

Clases

Una clase es un conjunto de métodos con propiedades.

```
class Persona
    def initialize(parametro)
        @nombre = parametro
    end
    def cantidad
        @nombre.length
    end
    def al_reves
        @nombre.reverse
    end
end
```

Objetos

Creamos **objetos** o **instancias** de una clase con **new**.

```
per1 = Persona.new('David')
per2 = Persona.new('Ricardo')
per3 = Persona.new('Rosa María')

tot = per1.cantidad + per2.cantidad + per3.cantidad
rev = "#{per1.al_reves}, #{per2.al_reves}, #{per3.al_reves}"

puts "La suma de las cantidades de letras es #{tot}."
puts "Y los nombres al revés son #{rev}."
```

Referencias

Ruby Language `http://www.ruby-lang.org/`

Programming Ruby The Pragmatic Programmers' Guide.
`http://www.ruby-doc.org/docs/ProgrammingRuby/`

Wikipedia `http://es.wikipedia.org/wiki/Ruby`