# 

TEMAS:

* Instalar Ruby
* Tipos de datos
* Variables
* Operaciones Matemáticas
* Estructuras de control
* Todos son Objetos
* Estructuras de control
* Array & Hashes
* Ciclo y iteradores
* Blocks, Procs y Lambdas

OBJETIVO:

Aprender qué es Ruby y su funcionamiento básico.

MARCO TEÓRICO

## Lectura: Aprende a Programar con Ruby

Cuando programas una computadora, necesitas "hablar" en la forma que tu computadora entienda: un lenguaje de programación. Existen diferentes lenguajes y muchos de ellos son excelentes. En esta guía voy a usar mi lenguaje de programación favorito: *Ruby*.

Además de ser mi favorito, Ruby es también el lenguaje más fácil de aprender que he visto (y eso que he visto unos cuantos). De hecho, esa es mi verdadera motivación para escribir esta guía: No decidí escribir una guía y luego escogí Ruby porque es mi lenguaje favorito; fue la simplicidad de Ruby lo que me motivó a escribirla (Escribir una guía similar en otro lenguaje, como C++ o Java, hubiese requerido cientos y cientos de páginas). ¡Pero no creas que Ruby es un lenguaje para principiantes sólo porque es fácil! Es un lenguaje poderoso y de calibre profesional, si es que existe alguno así.

Cuando escribes algo en un lenguaje humano, lo que se escribe es texto. Cuando escribes algo en un lenguaje de computadora, lo que se escribe es *código*. He incluido muchos ejemplos de código Ruby en esta guía, muchos de ellos son programas completos que puedes ejecutar en tu equipo. Para hacer el código más fácil de leer, he marcado partes del código de distintos colores y formatos.

Si encuentras algo que no entiendes o si tienes una pregunta que no has podido resolver, ¡escríbela y sigue leyendo! Es posible que la respuesta llegue en un capítulo posterior. Sin embargo, si tu pregunta no ha sido resuelta al finalizar la guía, te voy a mostrar donde preguntar. Hay muchas personas dispuestas a ayudar; sólo necesitas saber donde están.

## 

## Instalación

Antes de comenzar, necesitas descargar e instalar Ruby en tu computadora.

Para instalar Ruby en Windows, descarga Ruby Installer. Luego, ejecuta el instalador, sigue los pasos de instalación y listo!

Si estás en Mac OS X, ya tienes instalado Ruby. Pero si quieres la última versión de Ruby, puedes tenerla a través de Homebrew:

$ brew install ruby

Por último, si usas Linux, puedes usar el manejador de paquetes de preferencia. Por ejemplo, en Debian o Ubuntu, puedes instalar Ruby a través de apt:

$ sudo apt-get install ruby

En CentOS, Fedora o RHEL, puedes usar yum:

$ sudo yum install ruby

## Números

Ahora que tienes todo instalado, ¡vamos a escribir un programa! Abre tu editor de texto favorito y escribe lo siguiente:

puts 1 + 2

Guarda tu programa (sí, ¡eso es un programa!) como calc.rb (el .rb es lo que usualmente ponemos al final de los programas escritos en Ruby). Ahora ejecuta tu programa escribiendo ruby calc.rb en la línea de comandos. Deberías obtener un 3 en tu pantalla. ¿Lo ves?, programar no es tan difícil, ¿cierto?

Introducción a puts

¿Cómo es que funciona ese programa? Seguramente puedes adivinar qué es lo que 1 + 2 hace; nuestro programa es básicamente lo mismo que:

puts 3

puts simplemente escribe en la pantalla lo que sea que escribamos a continuación.

Enteros y flotantes

En la mayoría de los lenguajes de programación (y Ruby no es la excepción) los números sin decimales son llamados enteros (traducción de integers en su versión en Inglés), y los números con punto decimal normalmente son llamados números de punto flotante (o llamados sólo flotantes de ahora en adelante ya que proviene de su original en inglés floats).

Aquí hay algunos enteros:

5  
-205  
9999999999999999999999999  
0

Y aquí hay algunos flotantes:

54.321  
0.001  
-205.3884  
0.0

En la práctica, la mayoría de los programas no usan flotantes, sólo enteros (después de todo, nadie quiere leer 7.4 emails, o navegar 1.8 páginas, o escuchar 5.24 de sus canciones favoritas...) Los flotantes se usan más con propósitos académicos (como por ejemplo experimentos científicos) y para gráficos en 3D. Incluso la mayoría de los programas financieros usan enteros, ¡simplemente llevan un registro de los centavos!

## Aritmética simple

Hasta ahora, tenemos todo lo que necesita una calculadora simple (las calculadoras siempre usan flotantes, así que si quieres que tu computadora actúe como una calculadora, también deberías usar flotantes). Para suma y resta, usamos + y -, como ya lo vimos. Para la multiplicación, usamos \*, y para la división usamos /. La mayoría de los teclados tienen estas teclas en el teclado numérico a la derecha. Intentemos expandir un poco nuestro programa calc.rb. Escribe lo siguiente y ejecútalo:

puts 1.0 + 2.0  
puts 2.0 \* 3.0  
puts 5.0 - 8.0  
puts 9.0 / 2.0

Esto es lo que retorna el programa:

3.0  
6.0  
-3.0  
4.5

Los espacios en el programa no son importantes; simplemente hacen que el código sea más fácil de leer. Bueno, eso no fue muy sorprendente. Ahora probemos con enteros:

puts 1 + 2  
puts 2 \* 3  
puts 5 - 8  
puts 9 / 2

Básicamente lo mismo, ¿no?

3  
6  
-3  
4

Uh... ¡excepto por el último! Pero cuando se hace aritmética con enteros, se obtienen enteros. Cuando tu computadora no puede obtener la respuesta "correcta", siempre redondea hacia abajo (Por supuesto, 4 es la respuesta correcta en aritmética con enteros para 9 / 2; simplemente tal vez no sea el resultado que esperaba).

Tal vez te preguntes para qué es útil la división entera. Bueno, digamos que va al cine, pero sólo tiene $9. Aquí en Portland, puede ver una película en el Bagdad por $2. ¿Cuántas películas puede ver allí?9 / 2... 4 películas. 4.5 definitivamente no es la respuesta correcta ya que no le dejarán ver la mitad de una película o dejar que una mitad suya vea la película entera... algunas cosas simplemente no son divisibles.

¡Así que ahora experimenta con algunos programas por tu cuenta! Si quiere escribir expresiones más complejas, puede usar paréntesis. Por ejemplo:

puts 5 \* (12 - 8) + -15  
puts 98 + (59872 / (13 \* 8)) \* -52

Eso resulta en:

5  
-29802

## Textos

Así que hemos aprendido todo acerca de [números](http://rubysur.org/aprende.a.programar/capitulos/numeros.html), ¿pero qué hay acerca de letras?, ¿palabras? o ¿textos?

Nos referimos a grupos de letras en un programa como textos. Tú puedes pensar en letras impresas siendo insertadas en un cartel. Aquí hay algunos textos:

"Hola."  
"Ruby la rompe."  
"5 es mi número favorito... ¿Cuál es el tuyo?"  
"Snoopy dice #%^?&\*@! cuando le aplastan el dedo del pie."  
" "  
""

Como puedes ver, los textos pueden tener puntuación, dígitos, símbolos y espacios dentro ... más que solo palabras. Ese último texto no tiene nada y podemos llamarlo un texto vacío.

Hemos estado usando puts para imprimir números, intentémoslo con textos:

puts "Hola, mundo!"  
puts ""  
puts "Adiós."

El resultado es el siguiente:

Hola, Mundo!  
  
Adiós.

Eso funcionó muy bien. Ahora inténtalo con algún texto tuyo.

Aritmética de textos

Al igual que hacer aritmética con números, ¡tú también puedes hacer aritmética con textos! Bueno, algo así... digamos que puedes sumar textos. Intentemos sumar dos textos y ver que hace puts con eso.

Código:

puts "Me gusta" + "el pastel de manzana."

Resultado:

Me gusta el pastel de manzana.

Whoops! Me olvidé de poner un espacio entre "Me gusta" y "el pastel de manzana.". Los espacios generalmente no importan salvo si se escriben dentro de los textos (es verdad lo que dicen: las computadoras no hacen lo que tú quieres que hagan, sólo lo que tú les dices que hagan). Intentémoslo nuevamente:

Código:

puts "Me gusta " + "el pastel de manzana."  
puts "Me gusta" + " el pastel de manzana."

Resultado:

Me gusta el pastel de manzana.  
Me gusta el pastel de manzana.

Como puedes ver, no importó a que texto le agregué el espacio.

Así que puedes sumar textos, pero ... ¡también puedes multiplicarlos! (por un número...).

Observa esto:

Código:

puts "parpadeo " \* 4

Resultado:

batea tus párpados

Solo bromeaba, en realidad muestra esto:

parpadeo parpadeo parpadeo parpadeo

Si lo piensas, tiene mucho sentido. Después de todo, 7 \* 3 realmente sólo significa 7 + 7 + 7, así que "moo" \* 3 sólo significa "moo" + "moo" + "moo".

12 vs "12"

Antes de ir más allá, debemos asegurarnos de entender la diferencia entre números y dígitos. 12 es un número, pero "12" es un texto de dos dígitos.

Juguemos un poco con esto:

Código:

puts 12 + 12  
puts "12" + "12"  
puts "12 + 12"

Resultado:

24  
1212  
12 + 12

Y qué sucede con esto?:

Código:

puts 2 \* 5  
puts "2" \* 5  
puts "2 \* 5"

Resultado:

10  
22222  
2 \* 5

Estos ejemplos fueron bastante directos. De cualquier modo, si no tienes cuidado en cómo mezclas textos y números podrías encontrarte con...

Problemas

En este punto podrías haber probado algunas cosas que no funcionaron. Si no lo has hecho, aquí hay algunas:

Código:

puts "12" + 12  
puts "2" \* "5"

Resultado:

in `+': no implicit conversion of Fixnum into String (TypeError)

Hmmm... un mensaje de error. El problema es que en realidad no puedes sumar un número a un texto o multiplicar un texto por otro texto. Tiene tan poco sentido como esto:

puts "Betty" + 12  
puts "Fred" \* "John"

Algo más para tener cuidado: Puedes escribir "pig" \* 5 en un programa, dado que sólo significa 5 veces el texto "pig" todo junto. De cualquier modo, tú no puedes escribir 5 \* "pig", ya que eso significa "pig" veces el número 5, lo cual es simplemente tonto.

Finalmente, por ejemplo, si quisieras que tu programa imprimiera ¡Mi sobrenombre es

"Pepe"!? Podrías intentar:

puts "¡Mi sobrenombre es "Pepe"!"  


Bueno, eso no va a funcionar. La computadora no va a poder ejecutarlo porque piensa que el texto termina en el segundo " (antes de Pepe) y que Pepe es un método de Ruby, lo cual no es para nada cierto.

Entonces, ¿cómo hacemos para que la computadora entienda el texto que incluye doble comillas (")? Olvidé mencionarte que también puedes usar comillas simples (') para escribir texto en Ruby.

Entonces, podrías combinar comillas simples (') o dobles (") según lo requieras. Es más fácil de entender, si miramos los siguientes ejemplos:

Código:

puts "Mi nombre es Jo'C."  
puts '¡Mi sobrenombre es "Pepe"!

## Variables

Hasta ahora, cuando usamos 'puts' para un texto o número, esto desaparece. A lo que me refiero es que, si queremos imprimir algo dos veces, necesitamos escribirlo dos veces:

Código:

puts '...puedes decir eso de nuevo...'  
puts '...puedes decir eso de nuevo...'

Resultado:

...puedes decir eso de nuevo...  
...puedes decir eso de nuevo...

Sería bueno si pudiésemos escribir solo una vez y quedárnoslo... guardarlo en algún lado. Bueno, sí podemos, por supuesto; de otra manera, ¡no lo hubiese mencionado!

Para guardar el texto en la memoria de tu computador, necesitamos darle un nombre al texto. Los programadores se refieren frecuentemente a este proceso como asignación y llaman a los nombres variables. Esta variable puede ser cualquier secuencia de letras o números, pero el primer caracter necesita ser minúscula. Probemos nuestro programa de nuevo, pero esta vez voy a darle el nombre 'myString' al texto (aunque podría haber usado cualquier nombre, por ejemplo 'str' o 'myOwnLittleString' o 'enriqueOctavo').

Código:

myString = '...puedes decir eso de nuevo...'  
puts myString  
puts myString

Resultado:

...puedes decir eso de nuevo...  
...puedes decir eso de nuevo...

En todo momento cuando haces referencia a 'myString', el programa usa en su reemplazo "'...puedes decir eso de nuevo...'". Puedes pensar en la variable 'myString' como "apuntando" al texto "'...puedes decir eso de nuevo...'". Este es un ejemplo un poco más interesante.

Código:

name = 'Patricia Rosanna Jessica Mildred Oppenheimer'  
puts 'Me llamo ' + name + '.'  
puts 'Wow! "' + name + '" es un nombre realmente largo!'

Resultado:

Me llamo Patricia Rosanna Jessica Mildred Oppenheimer  
Wow! Patricia Rosanna Jessica Mildred Oppenheimer es un nombre realmente largo!

También, así como podemos asignar un objeto a una variable, podemos reasignar un objeto diferente a esa variable (Por eso es que las llamamos variables: porque varían.)

Código:

composer = 'Mozart'  
puts composer + ' fue "el amo", en su día.'  
  
composer = 'Beethoven'  
puts 'Pero yo prefiero a ' + composer + ', personalmente.'

Resultado:

Mozart fue "el amo", en su día.  
Pero yo prefiero a Beethoven, personalmente.

Por supuesto, las variables pueden apuntar a cualquier tipo de objeto, no sólo texto:

Código:

var = 'solo otro ' + 'texto'  
puts var  
  
var = 5 \* (1+2)  
puts var

Resultado:

solo otro texto  
15

De hecho, las variables pueden apuntar a casi cualquier cosa... excepto otras variables. ¿Pero qué pasa si lo intentamos?

Código:

var1 = 8  
var2 = var1  
puts var1  
puts var2  
  
puts ''  
  
var1 = 'ocho'  
puts var1  
puts var2

Resultado:

8  
8  
  
ocho  
8

Primero, cuando tratamos de apuntar var2 a var1 en realidad apuntamos a 8 (que es el valor al cual apuntaba var1). Luego cuando apuntamos var1 a 'ocho', el valor de var2 no cambia ya que en realidad no estaba apuntando a var1 sino a 8. Ahora que tenemos variables, números y textos, vamos a ver cómo mezclarlos.

Herramientas

Usar Ruby - CLI - Editor de Texto

Ejercicios

Generar arquitectura de nuestro proyecto en backend en Ruby