Elementos básicos de Ruby

¡Comencemos!

¿Qué aprenderás?

- Definir comentarios, variables y constantes en Ruby.
- Conocer y utilizar operaciones aritméticas en Ruby.
- Manipular strings utilizando concatenación en Ruby.
- Manipular strings utilizando interpolación en Ruby.
- Manipular entrada y salida de datos utilizando puts, print y gets en Ruby.
- Manipular entrada de datos por línea de comandos.
- Diferenciar un error sintáctico de uno semántico.

¡Vamos con todo!

Comentarios

Los comentarios son líneas ignoradas en la ejecución del código, sirven para dar indicaciones y documentar nuestros programas.

```
# Esta línea es un comentario
# Los comentarios son ignorados
# Pueden ir en una línea sin código.
puts 2 + 2  # O puede acompañar una línea de código existente
# puts 4 + 3 Si comentamos al principio todo la línea será ignorada
```

Comentarios en múltiples líneas

Existe otra forma de hacer comentarios de largo mayor a una línea.

Para lograrlo, envolveremos todo el comentario entre un `=begin` y un `=end`

```
=begin
Comentario multilínea:
Ruby
lo
ignorará
=end
puts "hola"
```

Importante

A lo largo de este módulo utilizaremos comentarios para mostrar el resultado de una instrucción.

```
puts 2 + 2 # 4
```

Si el resultado de la instrucción no se muestra en pantalla lo mostraremos de la misma forma que IRB, con: =>

```
2 + 2 # => 4
```

Introducción a variables

- Podemos entender las variables como contenedores que pueden almacenar valores.
- Los valores que hay dentro de estos contenedores pueden variar y por eso reciben el nombre de variables.

Partes de una variable

Una variable se compone de:

- Un nombre.
- Un valor.

Asignando un valor a una variable

En Ruby podemos asignar un valor a una variable de la siguiente forma:

```
a = 27
```

En este ejemplo estamos asignando el valor numérico `27` a la variable llamada `a`

Introducción a tipos de datos

En Ruby -y en muchos otros lenguajes de programación- existen diversos tipos de dato. Por ejemplo, existen los números enteros, los decimales, fechas, horas, arrays, hashes y muchos otros tipos más.

En este capítulo nos enfocaremos en enteros y strings.

Integers

Los números enteros son números sin decimales que pueden ser positivos o negativos. En inglés se les denomina Integers.

```
a = 27
a = -31
```

Floats

Los números de punto flotante pueden tener decimales.

```
a = 2.5
b = 3.1
```

Strings

A una palabra o frase se le conoce como cadena de caracteres o String. Un String puede estar formado por:

```
a = 'Hola Mundo!' # Una o varias palabras
b = 'x' # Simplemente un caracter.
```

¿Por qué los Strings se escriben entre comillas?

Ruby necesita algún mecanismo para diferenciar si el programador está haciendo referencia a una variable o a un string.

```
a = 'Esto es un string'
b = a
puts b # 'Esto es un string'
```

En la instrucción:

```
b = a
```

Ruby entiende que nos estamos refiriendo a la variable a y no a un carácter 'a'.

¿Comillas simples o dobles?

Para trabajar con Strings podemos utilizar comillas simples (') o dobles (""):

```
a = 'Esto es un String'
b = "Esto también es un String"
```

Se recomienda utilizar comillas simples, con excepción de algunos casos que estudiaremos a lo largo de esta unidad.

El salto de línea

El salto de línea es un carácter especial que nos permite separar una línea de texto de la siguiente, dentro de un string.

Este carácter es \n, indicador de 'nueva línea'.

```
a = "hola\n a\n todos"
print a
# hola
# a
# todos

\n cuando está entre comillas dobles se lee como si fuera un solo
caracter.

puts "\n".length
puts '\n'.length
```

Esto es porque al usar comillas simples el string es entendido tal cual. Al usar comillas dobles se interpretan los metacaracteres, o indicadores de símbolos que no se pueden usar normalmente. El \ en Ruby se ocupa como carácter de escape, esto quiere decir que permite una interpretación alternativa del texto. Existen otras combinaciones interesantes como "\t" que permite tabular un texto.

Operando con variables

Hasta ahora hemos aprendido que una variable puede contener números enteros Integers, con decimales Floats o palabras Strings. Utilizando estos valores podemos realizar operaciones.

Sumas, restas, multiplicaciones y divisiones

Podemos operar con variables de tipo entero tal como si estuviéramos trabajando en una calculadora.

```
a = 5
b = 2
```

```ruby
puts a + 2

# 7

puts b - a
# 3

puts a * b
#20

puts a / b
# 2
```

De seguro les llama la atención el resultado de esta última operación. Esto es porque Ruby entiende que al dividir enteros el resultado debe ser entero. Por ejemplo, si tenemos dos nidos y cinco pollitos, no podemos dejar 'dos pollitos y medio' en uno de los nidos.

Para indicar que queremos el resultado como float, debemos expresar alguno de los operandos como tal, de la siguiente forma:

```ruby
a = 5.0
b = 2

puts a / b
# 2.5
```

Modificando una variable

Una variable puede cambiar de valor por medio de una nueva asignación.

```ruby
a = 'HOLA!'
a = 100

puts a
# 100
```

También podemos modificar el valor de una variable operando sobre ella misma, este tipo de operación es muy utilizada:

```ruby
a = 2
a = a + 1 # => 3
puts a # 3
```

Operando con variables de tipo String

¿Qué obtendremos al "sumar" dos Strings?

```ruby
a = 'HOLA'
b = ' MUNDO'
puts a + b
# "HOLA MUNDO"
```

Concatenando números

Observemos el siguiente ejemplo:

```ruby
a = '7'
b = '3'

puts a + b
#"73"
```

Al asignar los valores utilizando comillas simples, Ruby los interpreta como String.

La acción de 'sumar', o sea unir dos o más Strings se conoce como concatenación, porque enlaza dos cadenas de caracteres.

Interpolación

Otra acción muy importante y ampliamente utilizada al momento de trabajar con Strings es la interpolación.

La interpolación es un mecanismo que nos permite introducir una variable (o un dato) dentro un String sin necesidad de concatenarlo. Para interpolar simplemente tenemos que introducir la variable (o dato) utilizando la siguiente notación:

```ruby
edad = 30
texto = "tienes #{edad} años"
puts texto
# "tienes 30 años"
```

La interpolación sólo funciona sobre comillas dobles

```ruby
edad = 30
texto = 'tienes #{edad} años'
puts texto
# "tienes #{edad} años"
```

¿Recuerdas que los Strings se pueden declarar entre comillas simples o dobles?

La interpolación sólo funciona sobre comillas dobles. Si intentamos interpolar utilizando comillas simples obtendremos el contenido literal.

La interpolación es más fácil de usar que la concatenación

```ruby
nombre = 'Carlos'
apellido = 'Santana'

# Concatenación
puts "Mi nombre es " + nombre + " " + apellido
# "Mi nombre es Carlos Santana"

# Interpolación
puts "Mi nombre es #{nombre} #{apellido}"
# "Mi nombre es Carlos Santana"
```

Podemos obtener los mismos resultados utilizando concatenación e interpolación, sin embargo, se prefiere la interpolación debido a que es más rápida y presenta una sintaxis más amigable para el desarrollador.

Constantes

- Existe un tipo especial de variable llamado constantes.
- Una constante sirve para almacenar un valor que no cambiará a lo largo de nuestro programa.

En Ruby, la única regla para definir una constante es que su nombre debe comenzar con una letra mayúscula.

Intentando modificar una constante

```
Foo = 2
# => 2

Foo = 3
# (irb):2: warning: already initialized constant Foo
# (irb):1: warning: previous definition of Foo was here
# => 3

Foo
# => 3
```

Las constantes pueden modificarse, pero bajo alerta

En este caso vemos que es posible modificar una constante, pero obtendremos un aviso de que estamos haciendo algo mal.

Es convención de que las constantes se escriben completamente con mayúsculas, de esta forma si vemos algo como:

```
NO_ME_CAMBIES = 1
```

Sabremos que es un error cambiarlo

Salida de datos

Hay varias formas de mostrar datos en pantalla, las dos más utilizadas son puts y print. Ambas son muy similares:

- **puts** agrega un salto de línea.
- **print** no lo agrega.

```
puts "con puts:"
puts "hola"
```

```
puts "a"
puts "todos"

#con puts:
#hola
#a
#todos

print "con print:"
print "hola"
print "a"
print "todos"

#con print:holaatodos
```

A veces a mostrar le diremos imprimir. Esto tiene razones históricas, de cuando no existían las pantallas y la única forma de mostrar el resultado era imprimirlo. Por eso a veces se dice 'imprimir en pantalla'.

Entrada de datos

Con frecuencia nuestro script necesitará interactuar con el usuario, ya sea para seleccionar una opción de un menú o para simplemente ingresar un valor sobre el cual nuestro script va a operar.

Podemos capturar datos introducidos por un usuario utilizando la instrucción gets, y de la siguiente forma:

Ingresando datos con gets

Antes de mostrar el valor, la consola quedará bloqueada hasta que ingresemos una secuencia de caracteres y presionemos la tecla enter.

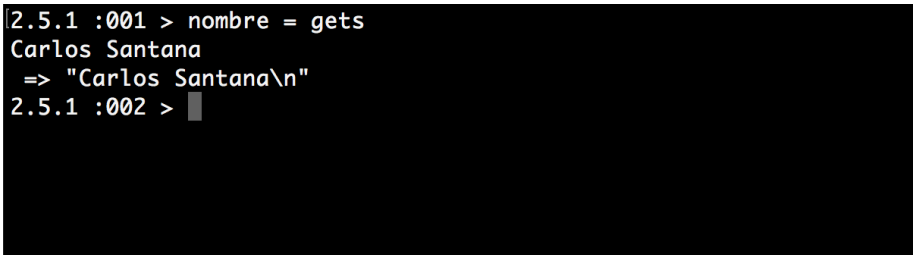

Imagen 1. Gets
Fuente: Desafío Latam

El problema del método gets, como podemos observar en la imagen, es que también captura el salto de línea que se genera al presionar la tecla enter. En Ruby (y en muchos otros lenguajes) el salto de línea se representa como **\n**.

Removiendo el salto de línea con chomp

Para solucionar este problema podemos utilizar .chomp:

```
nombre = gets.chomp
puts nombre # "Carlos Santana"
```

Ingresando datos por consola

A veces, cuando nuestro programa no está pensado para ser usado por humanos, sino por otros programas, es más conveniente que los datos puedan ser entregados como argumentos de línea de comandos. Esto quiere decir que en vez de escribir ruby **programa.rb**, lo invocamos como `ruby programa.rb dato`.

Estos argumentos se almacenan en un arreglo (una lista, una colección) llamada ARGV.

Para obtener estos datos basta con extraer de este arreglo los datos que nos interesan, contando desde el cero en adelante.

Para ponerlo de forma más concreta, hagamos un programa llamado "**suma3.rb**", que sume tres números ingresados como argumento de línea de comandos:

```
primero = ARGV[0].to_i
segundo = ARGV[1].to_i
tercero = ARGV[2].to_i
puts primero + segundo + tercero
```

Esto nos permitirá llamar al programa y entregarle todos los datos al mismo tiempo, escribiendo `ruby suma3.rb 5 8 12`.

Debería mostrarnos el número 25.

Creando un programa con lo aprendido

Con esta simple entrada de datos ya podemos crear programas sencillos. Por ejemplo, un programa que solicite tu nombre y luego te salude:

```
nombre = gets.chomp
puts "Hola #{nombre} !"
```

Concatenando números por error

Otro detalle interesante es que gets siempre entrega un String, por lo tanto, si aplicamos operaciones de suma (+) a números ingresados por teclado, estos serán concatenados.

```
num1 = gets.chomp # 4
num2 = gets.chomp # 6
puts num1 + num2 # 46 :(
```

Transformando los datos

Este comportamiento se puede modificar aplicando transformaciones a los tipos de datos.

```
num1 = gets.chomp.to_i # 4
num2 = gets.chomp.to_i # 6
puts num1 + num2 # 10 :(
```

La transformación de tipos de datos la estudiaremos en profundidad en un próximo capítulo

chomp no siempre es necesario

```
num1 = gets.to_i # 4
num2 = gets.to_i # 6
puts num1 + num2 # 10 :(
```

Si estamos transformando Strings a números, el método chomp no es necesario. Prueba el código anterior removiendo el .chomp.
De todas formas incorporarlo no afecta en el resultado.

Ingresando un dato al azar

Existe un método para generar un número al azar dentro de un rango.

```
puts dado = rand(1..6) # Genera un valor entre 1 y 6
```

Errores sintácticos vs errores semánticos

Los errores podemos clasificarlos en dos tipos: sintácticos y semánticos.

- Los errores sintácticos son muy similares a los errores gramaticales, donde escribimos algo que contraviene las reglas de cómo debe ser escrito.

Por ejemplo: Asignar una variable es de izquierda a derecha, a = 5 mientras lo contrario sería un error sintáctico 5 = a.

- Los errores semánticos son muy distintos, el programa funcionará de forma normal pero el resultado será distinto al esperado. Los errores semánticos corresponden a errores de implementación o de diseño.

Errores sintácticos

Un error sintáctico será detectado automáticamente cuando Ruby intente leer esa línea, al momento de detectar el error lo reportará y dejará de leer.

Resumen del capítulo

En este capítulo revisamos varios elementos claves para construir un programa en Ruby, estos son:

- **Variables:** Contenedores de datos.
- **Operadores:** Nos permiten sumar, restar y hasta concatenar datos.
- **Entrada y salida de datos:** Con gets y puts.
- **Tipos de datos:** Influyen en la forma en que el computador interpreta la información.
- **Interpolación:** Forma de insertar una variable dentro de un string.
- **Comentarios:** Forma de anotar sin que modifique el comportamiento del código.
- **Salto de línea:** "\n" permite indicar múltiples líneas en un string.
- **"" vs '':** Las comillas dobles permiten interpolación y caracteres especiales; las comillas simples, no.