Otras instrucciones para ciclos

¡Comencemos!

¿Qué aprenderás?

- Aprender la sintaxis de otros elementos que permiten iterar dentro de Ruby.
- Conocer la ventaja y desventaja de estas instrucciones.
- Conocer los rangos.
- Conocer los bloques.

Introducción

Además de while existen otras dos instrucciones muy útiles para iterar.

- for.
- times.

Algunas situaciones que estudiaremos serán más sencillas de resolver utilizando estas instrucciones.

¡Vamos con todo!

Ciclos con las instrucción For

La instrucción `for` nos permite iterar en un rango. Podemos, por ejemplo, iterar de 1 a 10 con la siguiente sintaxis:

```
for i in 1..10
 puts "Iteración #{i}"
end
```

Recordemos que es convención utilizar `i` como variable de iteración.

Rangos

`1..10` es un tipo de dato bien especial, llamado rango.

```
(1..10).class
# Range
```

Para crear un rango podemos utilizar caracteres o enteros.

```
for i in 'a'..'z'
 puts i
end
```

Disminuir es ligeramente más complejo.

```
for i in 10.downto(1)
 puts "hola #{i}"
end
```

Ventaja de for

La principal ventaja de for es que nos podemos olvidar de implementar un contador, ya que la variable de iteración aumenta en cada iteración de forma automática.

Desventaja de for

La desventaja es que es menos flexible, porque no podemos manipular el incremento de manera personalizada.

El ciclo Times

El método times que podemos traducir como a veces nos permite iterar de una forma muy sencilla.

```
5.times do
   puts "repitiendo"
end
```

Un comportamiento interesante: ¡podemos imprimir el número de iteración en el que estamos trabajando!

```
5.times do |i|
   puts "repitiendo: #{i}"
end
```

Recordemos que el nombre `i` para la variable es sólo una convención, podemos utilizar el nombre que más nos acomode.

Introducción a bloques

Tanto times como muchos otros métodos pueden recibir bloques. Este concepto **es muy importante** en Ruby. Profundizaremos en él más adelante, sin embargo, en este punto necesitamos entender su sintaxis.

Un bloque se puede escribir de dos formas:

1. Con `do` y `end`:

```
10.times do
   puts 'Repitiendo 10 veces'
end
```

2. Entre llaves `{}`.

```
10.times { puts 'repitiendo 10 veces' }.
```

Esta segunda forma recibe el nombre de inline porque es ideal para escribir bloques de una sola línea.

Los bloques son closures

El concepto de closure viene del mundo matemático y su definición puede ser bien compleja de entender, por ejemplo, si vemos la de wikipedia encontraremos que:
"Es una técnica para implementar ámbitos léxicos en un lenguaje de programación con funciones de primera clase".

En palabras sencillas

Todo lo que declaremos dentro de un bloque queda definido solo dentro del bloque.

Ejemplo de declaración dentro de un bloque:

```
10.times do |i|
   z = 0
end

puts z # undefined local variable or method `z' for main:Object
```

Esto solo aplica a las declaraciones y no a las asignaciones.

```
z = 0
10.times do |i|
   z += 1
end
```

```
puts z
10
```

Si este tema no te quedó completamente claro no te preocupes, lo revisaremos más adelante en esta unidad cuando discutamos el concepto de alcance de variables. En este momento lo único importante es recordar que las variables declaradas dentro de bloques solo pueden ser accedidas dentro del mismo bloque.

Iterando con times y el índice

De la primera forma:

```
10.times do |i|
   puts i
end
```

De forma inline

```
10.times { |i| puts i }
```

Al igual que las variables definidas dentro del bloque, las variables de iteración tampoco existen fuera del ciclo.

```
10.times do |i|
   puts "repitiendo: #{i}"
end

puts i

# repitiendo: 0
# ...
# ...
# repitiendo: 9
# undefined local variable or method `i' for main:Object (NameError)
```