Alcance de variables

¡Comencemos!

¿Qué aprenderás?

- Conocer los tipos de variables.
- Conocer el concepto de alcance.
- Diferenciar variables locales de variables globales.
- Entender qué sucede con las variables dentro de un método.

Introducción

Los tipos de variable y el alcance de estas son conceptos muy importantes, nos permiten entender desde dónde podemos acceder a una variable.

En este capítulo estudiaremos las reglas de alcance de las variables locales y globales.

¡Vamos con todo!

Tipos de variable

En Ruby existen 4 tipos de variable:

- Globales.
- Locales.
- De instancia.
- De clase.

Las últimas dos se ocupan dentro de la creación de objetos, por lo que no las abordaremos todavía.

Nos enfocaremos en las variables globales y locales.

El alcance

El alcance, o Scope en inglés, es desde dónde podemos acceder a una variable.

Alcance de una variable local.

Una variable **definida dentro de un método no puede ser accedida fuera del método**. A las reglas desde dónde puede ser accedida a una variable se le denomina alcance.

```
def aprobado?(nota1, nota2)
 promedio = (nota1 + nota2) / 2
 promedio >= 5 ? true : false
end

aprobado?(4, 5) # false
aprobado?(10, 5) # true
```

```
true
```

```
puts promedio # undefined local variable or method `promedio'
```

Los parámetros también cuentan como variables locales

```
def aprobado?(nota1, nota2)
 promedio = (nota1 + nota2) / 2
 promedio >= 5 ? true : false
end

aprobado?(4, 5) # false
puts nota1 # undefined local variable or method `nota1' for main:Object
```

No importa el orden de declaración

Es un tema de espacio de trabajo, no de orden.

```
nombre = 'Montgomery Burns'

def saludar
 puts "Hola #{nombre}!"
end
# undefined local variable or method `nombre'
```

Main

El espacio principal de trabajo recibe el nombre de **main**.

```
# Esto está siendo definido en el ambiente 'main'
nombre = 'Homero Simpson'
edad = 40

def cualquier_metodo
 # Esto está siendo definido en un ambiente nuevo: el del método
 # Aquí no existen las variables nombre ni edad
 palabra = 'diez'
 numero = 10
end

# Esto vuelve a ser el ambiente 'main'
# Aquí no existe las variables palabra ni número
puts nombre
puts edad
```

Cuando ingresamos en un método, estamos trabajando en un ambiente nuevo que, aunque posea variables con el mismo nombre, **no afecta las variables "externas"**.

```
# El alcance de estas variables es 'main'
name = 'Homero Simpson'
age = 40
address = 'Springfield'
human = false

def presentar
 # El alcance de las variables definidas aquí es el método
 # Estas instrucciones NO afectan las variables de 'main'
 name = 'Milhouse Van Houten'
 age = 10
 occupation = 'student'

 puts "#{name} tiene #{age} años!"
end

presentar

puts "#{name} tiene #{age} años!"
```

Este tipo de variable se conoce como **variables locales**, por consiguiente, podemos afirmar que:

- `address` es una variable local, cuyo alcance es `'main'`.
- `occupation` es una variable local, cuyo alcance es el método `presentar`.
- `address` no existe dentro del método `presentar`.
- `occupation` no existe fuera del método `presentar`.

Existe un método para saber si una variable es local.

```
defined? edad
# => "local-variable"
```

Variables globales

Las variables globales, como su nombre lo indica, pueden ser accedidas desde todos los espacios.

En Ruby las variables globales se definen comenzando por un `$`. Ingresemos a IRB y hagamos la siguiente prueba:

```
$continente = 'Sudamérica'
# => "Sudamérica"

defined? $continente
#=> "global-variable"
```

Al estar definida como una variable global, podemos operar con ellas dentro y fuera de nuestros métodos.

Veámoslo en la práctica:

```
# Definición de la variable global
$continente = 'Sudamérica'

def modificar_continente(nombre_continente)
 # La variable global si existe dentro del método
 $continente = nombre_continente
end

# Llamado al método que modifica el nombre de la variable global
modificar_continente('Europa')

puts $continente
# "Europa"
```

El problema de las variables globales

Las variables globales son consideradas una mala práctica ya que hacen muy fácil romper un programa por error.

¿Cómo se rompe un programa con variables globales?

Un programa puede tener miles de líneas de código e incorporar varias bibliotecas (programas de tercero). En este aspecto es sencillo que alguien llame por error a una variable de la misma forma que otra persona la llamó y cualquier cambio puede romper todo el código.

Existen muchas otras razones por las que estas no se recomiendan, pero que las dejaremos, por ahora, fuera de discusión.

Cierre

En esta unidad aprendimos sobre:

- Ciclos.
- Métodos.

Ciclos

Los ciclos nos ayudan a resolver problemas en base a iteraciones y son muy esenciales para la resolución de problemas.

- Algunos tipos de ciclos como `.times` pueden recibir bloques.
- Las variables (locales) definidas dentro de los bloques no pueden ser accedidas desde fuera del bloque.
- Al utilizar `while` tenemos que tener cuidado de cambiar el índice para que haya un punto de salida del ciclo.

Métodos

Los métodos nos permiten reutilizar código. Sobre métodos aprendimos:

- Pueden recibir parámetros.
- El retorno de la última línea es implícito.
- Las variables definidas dentro de los métodos no pueden ser accedidas desde fuera.