Parametrizando métodos

¡Comencemos!

¿Qué aprenderás?

- Crear métodos que reciben parámetros.
- Diferenciar parámetros de argumentos.
- Crear métodos con parámetros opcionales.

¡Vamos con todo!

Creando un método con parámetro

Crear un método que recibe un parámetro es sencillo. En la definición del método utilizaremos paréntesis para especificar los parámetros que debe recibir.

```
def incrementar(numero)
 total = numero + 1
 puts "El resultado es #{total}"
end
```

Llamando a un método que exige parámetros:

```
incrementar(5)
# => El resultado es 6

incrementar 10 # Los paréntesis son optativos
# => El resultado es 11

incrementar 51
# => El resultado es 52
```

Decimos que los parámetros se exigen porque si no los especificamos, obtendremos un error:

```
def incrementar(numero)
 total = numero + 1
```

```
 puts "El resultado es #{total}"
end

incrementar #Wrong number of arguments (given 0, expected 1)

# Tampoco podemos pasar argumentos de más
incrementar 2,4 #Wrong number of arguments (given 2, expected 1)
```

¿Qué es un argumento?

En el error leímos Wrong number of arguments pero ¿Qué es un argumento?

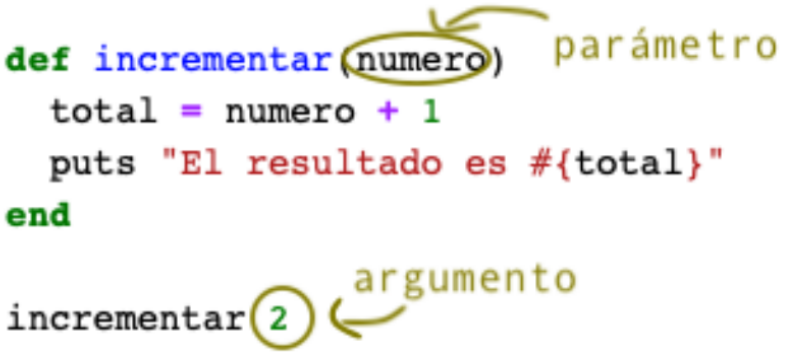

Imagen 1. Argumento.
Fuente: Desafío Latam.

- Las variables, en la definición de un método, se denominan **parámetro**.
- Los objetos que pasamos, al llamar al método, se denominan **argumento**.

Otra forma de verlo es decir que el parámetro es un valor genérico, no sabemos exactamente cuál es el valor que se va a pasar. En cambio, hablamos de argumento cuando especificamos el valor.

Cuidado con los paréntesis

Tenemos que tener cuidado al utilizar espacios y paréntesis.

```
incrementar 2  # ruby lo lee como incrementar(2).
incrementar (2) #ruby lo lee como incrementar((2)), esto no es un
problema.
```

Pero si el método no recibe parámetros.

```
def parentesis
   puts "x"
end


parentesis () # ruby lo lee como parentesis(())
```

- parentesis(()) lo que es lo mismo que parentesis(nil), o sea estamos pasando un parámetro y el método no recibe parámetros por lo que obtendremos el error:

```
# (wrong number of arguments (given 1, expected 0))
```

Una variable puede ser usada como argumento

```
def incrementar(numero)
 total = numero + 1
 puts "El resultado es #{total}"
end

a = 2
incrementar(a)
```

Esto lo utilizaremos con bastante frecuencia.

Un método puede recibir más de un parámetro

Podemos especificar todos los parámetros que queramos, simplemente separándolos por una coma.

```
def incrementar(numero, cantidad)
   total = numero + cantidad
   puts "El resultado es #{total}"
end

incrementar(2, 3)
incrementar 3, 3 # El uso de paréntesis es opcional
```

Si el método exige dos parámetros tenemos que pasarle 2 argumentos

```
incrementar 2 # ArgumentError: wrong number of arguments (given 1,
expected 2)
```

Métodos con parámetros opcionales

Para crear un método que recibe parámetros de forma opcional tenemos que especificar qué debe hacer Ruby cuando el parámetro no se especifica.

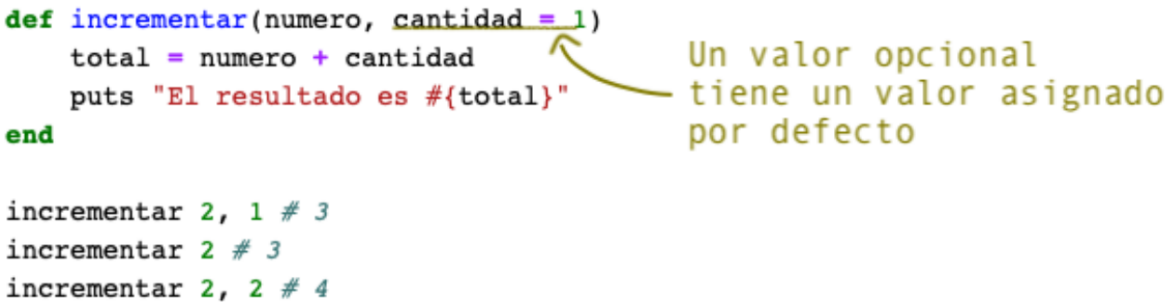

Imagen 2. Parámetros opcionales.
Fuente: Desafío Latam.

En este caso sólo uno de los valores es opcional (pero podríamos crear un método donde todos los parámetros fueran opcionales).

Si no le pasamos ningún argumento veremos el siguiente error:

```
def incrementar(numero, cantidad = 1)
   total = número + cantidad
   puts "El resultado es #{total}"
end

incrementar
# ArgumentError: wrong number of arguments (given 0, expected 1..2)
```

- 1..2 se lee como que espera mínimo 1 y máximo 2.

Los métodos deben ser reutilizables

Veamos el mismo código escrito de dos formas distintas.

```ruby
# Opción 1:
def fahrenheit()
 puts 'Ingrese la temperatura en fahrenheit'
 fahrenheit = gets.to_i
 celsius = (fahrenheit + 40) / 1.8 - 40
 puts "la temperatura es de #{celsius} celsius"
end
```

```ruby
# Opción 2:
def fahrenheit(f)
 celsius = (f + 40) / 1.8 - 40
 puts "la temperatura es de #{celsius} celsius"
end

puts 'Ingrese la temperatura en fahrenheit'
fahrenheit(gets.to_i)
```

¿Cuál es más flexible? Esto es lo que discutiremos a continuación.

Ejemplo de reutilización

¿Qué pasaría si ya no queremos que el usuario ingrese los valores, y queremos generar los valores al azar? El segundo es más flexible porque nos permite llamarlo independiente de cómo se ingresen los valores.

```ruby
# Opción 2:
def fahrenheit(f)
 celsius = (f + 40) / 1.8 - 40
 puts "la temperatura es de #{celsius} celsius"
end

puts 'Ingrese la temperatura en fahrenheit'
fahrenheit(rand(50))
```

Desafío

Crear el programa `validar_edad.rb` que contenga el siguiente código pero que cumpla las siguientes condiciones:

- Modificar el método para que reciba la edad.
- Llamar al método 3 veces, con edades generadas al azar.

```ruby
def validar_edad
 edad = gets.to_i
 if edad >= 18
   puts "es mayor"
 else
   puts "es menor"
 end
end
```

Resumen

- Los parámetros nos permiten hacer los métodos muy flexibles.
- Algunos parámetros pueden ser opcionales, para eso debemos asignarles un valor por defecto.

```ruby
def prueba(x = 2)
    puts x
end

prueba
prueba(3)
prueba 3
```

- Las variables pueden ser utilizadas como argumentos.

```ruby
def prueba(x = 2)
    puts x
end

a = 5
prueba(a)
```