Objetos y sus tipos

Objetivos

- Identificar un entero en Ruby
- Identificar un flotante en Ruby
- Identificar un string en Ruby
- Identificar un boolean en Ruby
- Conocer el objeto `nil`.
- Diferenciar el comportamiento del método `+` en Integer vs String
- Transformar Strings a Integers.

Clases y objetos

En Ruby, existen distintos tipos de dato. Ya sabemos que estos tipos de datos son **clases** y los elementos de un tipo en específico reciben el nombre de **objetos**

Clases más frecuentes

- Integer: Corresponde a un número entero.
- String: Corresponde a un caracter o una cadena de caracteres.
- Float: Corresponde a un número que puede tener decimales.
- Time: Corresponde a una fecha y hora.
- Boolean: Corresponde a verdadero (`true`) o falso (`false`). Son el resultado de la evaluación de una proposición lógica.
- Nil: corresponde al objeto nulo, la ausencia de un valor.

A medida que avancemos, profundizaremos más en estas clases.

¿Cómo saber de qué clase es un objeto?

Podemos saber el tipo de dato utilizando el método `.class`

Por ejemplo si dentro de **IRB** escribimos `2.class` obtendremos como resultado **Integer**, o si probamos con `'hola'.class` obtendremos como resultado **String**.

```
suma = 5 + 2
# => 7

suma.class
# => Integer

otra_suma = 2.3 + 0.1
otra_suma.class
# => Float

hora_actual = Time.now
# => 2018-09-10 14:44:24 -0300

hora_actual.class
# => Time
```

¿Por qué es importante el tipo de objeto?

Existen distintas reglas para operar entre estos distintos tipos de objetos. Estas reglas las conoceremos consultando la documentación oficial.

Por ejemplo: al sumar dos números obtenemos el resultado de la suma, pero al 'sumar' dos palabras obtenemos la concatenación de estas.

En algunas situaciones, cuando faltemos a estas reglas, las operaciones no serán válidas.

Concatenando strings

Observemos el siguiente ejemplo: el método `+` del objeto String recibe como parámetro otro String a concatenar.


```
"Hello from " + self.to_s    #=> "Hello from main"
```

Concatenando un string con otro tipo de dato

¿Qué sucede si intentamos concatenar un Integer a un String?

```
"HOLA" + 2
# TypeError: no implicit conversion of Integer into String
```

¿Y si queremos sumar dos números ingresados por teclado?

```
numero_uno = gets.chomp
# => "10"

numero_dos = gets.chomp
# => "20"

puts numero_uno + numero_dos
# "1020"
```

Para solucionar este problema y, dependiendo de nuestro objetivo, podemos aplicar transformaciones a los tipos de objetos.

Trasformando tipos de objetos

Existen distintos métodos que nos permiten transformar un objeto de un tipo a otro. Dentro de estos métodos podemos destacar:

- El método `to_i` (To Integer) nos permite convertir un String en un Integer.
- El método `to_s` (To String) nos permite convertir un Integer en un String.

```
2018.to_s
# "2018"

"365".to_i
# 365
```

¡Ahora sí podemos sumar dos números ingresados por teclado!

```
numero_uno = gets.chomp
# => "10"

numero_dos = gets.chomp
# => "20"

puts numero_uno.to_i + numero_dos.to_i
# 30
```

¿Qué se obtiene como resultado en el siguiente ejemplo?

```
numero_uno = gets.chomp
# => "10"

numero_dos = gets.chomp
# => "20"

puts (numero_uno + numero_dos).to_i
# ??
```

Juntando métodos

Veremos de forma frecuente código como el siguiente:

```
numero_uno = gets
numero_uno = numero_uno.chomp
numero_uno = numero_uno.to_i
```

Lo anterior se puede reducir a la siguiente expresión:

```
numero_uno = gets.chomp.to_i
```

La expresión se lee de izquierda a derecha

- **gets** nos devuelve un string con un salto de línea al final
- **chomp** transforma el string en un nuevo string sin el salto de línea
- **to_i** transforma el string en un número entero
- El resultado es guardado en la variable `numero_uno`

`.chomp` no siempre es necesario

Recordemos además que el `.to_i` remueve el salto de línea, por lo que nuestra expresión se puede reducir a:

```
numero_uno = gets.to_i
```

Nota

Más adelante estudiaremos el concepto de *precedencia* y aprenderemos que no toda expresión se lee de izquierda a derecha.

Interpolación vs transformación vs concatenación

Observemos el siguiente comportamiento:

```
nombre =  'Carlos Santana'
edad = 71

# Concatenación
puts "Hola! Soy " + nombre + " y tengo " + edad + " años!"
# TypeError (no implicit conversion of Integer into String)

# Concatenación + Transformación
puts "Hola! Soy " + nombre + " y tengo " + edad.to_s
# Hola! Soy Carlos Santana y tengo 71 años!

# Interpolación
puts "Hola! Soy #{nombre} y tengo #{edad} años!"
# Hola! Soy Carlos Santana y tengo 71 años!
```

Al utilizar interpolación, no necesitaremos aplicar transformación al objeto `edad`. El método `to_s` será aplicado de forma automática al objeto que escribamos entre las llaves.