Programación en Lenguaje Ruby

Objetos

Delio Tolivia

Gijón crece 2016 / 2019
AYTO. DE GIJÓN / CCOO / FADE / UGT
Ayuntamiento de Gijón
Gijón impulsa Empleo
dicampus TALENTO FORMACIÓN

Índice

Introducción a objetos

- En Ruby todo son objetos. Para hacer cosas hay que "conversar" con otros objetos diciéndoles que cosas hay que hacer.

- Para saber que objeto se es en cada momento se puede usar la palabra ***self***

```
$ irb
2.0.0-p648 :001 > self
 => main
```

- Si no especificamos un objeto automáticamente Ruby nos pone en el papel de ***main*** por defecto.

Hablando con objetos (I)

- Para interactuar con los objetos se utilizan los métodos. Más específicamente un objeto “llama o invoca” los métodos de otro objeto.
- Por ejemplo podemos invocar el método ***even?*** En el objeto **2** (es un numero, tipo de dato entero que también es un objeto). Para poder llamar al método utilizaremos el **.** después del objeto seguido del nombre del método a invocar:

```
2.0.0-p648 > 2.even?
 => true
2.0.0-p648 > 3.even?
 => false
```

Hablando con objetos (II)

- La llamada a un método siempre tiene como respuesta otro objeto.
- Por ejemplo la llamada ***1.next*** (el método ***next*** nos da como resultado el siguiente número) nos devolverá el objeto ***2***.
- Se pueden concatenar llamadas a métodos añadiendo más puntos y nombres de métodos. Por ejemplo para obtener el ***3*** a partir del ***1***:

```
2.0.0-p648 > 1.next.next
 => 3
```

Hablando con objetos (III)

- Para conocer los métodos de un objeto tenemos el método ***methods***. El resultado serán todos los nombres de los métodos que dispone ese objeto (cada método aparece con **:** delante de su nombre). Podemos ordenarlos alfabéticamente concatenando la llamada al método ***sort***

```
2.0.0-p648 :006 > 1.methods.sort
 => [:!, :!=, :!~, :%, :&, :*, :**, :+, :+@, :-, :-@, :/, :<, :<<,
:<=, :<=>, :==, :===, :=~, :>, :>=, :>>, :
  [], :^, :__id__, :__send__, :abs,
:abs2, :angle, :arg, :between?, :ceil, :chr, :class, :clone,
  ..............
```

Hablando con objetos (IV)

- Cuando se llama a un método es posible darle información adicional, a esta información se la denomina “argumentos del método”. Por ejemplo utilizaremos el método ***index*** de un ***array*** que nos encuentra la posición del objeto indicado por el argumento:

```
2.0.0-p648 > ['rock','paper','scissors'].index('paper')
 => 1
```

- Como podemos ver los argumentos se colocan entre paréntesis. En Ruby podemos realizar la llamada sin ellos aunque por claridad del código es recomendable utilizarlos.

```
2.0.0-p648 > ['rock','paper','scissors'].index 'paper'
 => 1
```

- Si un método necesita más de un argumento se concatenan con comas:

```
2.0.0-p648> 1.between?(1,3)
 => true
```

Hablando con objetos (V)

- Como pudimos ver en el listado de los métodos del objeto 1 aparecían los operadores matemáticos como + y -. Como tal podríamos llamarlos de la siguiente manera:

```
2.0.0-p648 > 4.+(3)
 => 7
```

- Para ser más amigable Ruby hace una excepción sintáctica con los operadores más utilizados para no tener que utilizar los .

```
2.0.0-p648 > 4+3
 => 7
```

- Los operadores más comunes en los que ocurre esto son: **+ - * / = == != > < >= <= []**

- El último es más especial pues se utiliza para acceder a un elemento de un array e incluye el argumento del método en su interior (Ejercicio: Probar a hacer la misma llamada con la sintaxis del .):

```
2.0.0-p648 > words = ["foo", "bar", "baz"]
 => ["foo", "bar", "baz"]
2.0.0-p648 > words[1]
 => "bar"
```

Literales

- Se conoce como literales cuando definimos un valor sin asignarlo a una variable. Por ejemplo un numero o una cadena.

- Hay sintaxis especial para los números:

  - Podemos utilizar notación científica: ***1.2e-10***

  - Podemos usar la barra baja _ para facilitar la lectura de números grandes: ***1_234_234_234***

- Podemos definir un rango de números con el operador ..

```
start = 101
finish = 201
ranger_rick = start..finish
```

Ámbito

- Es la zona de código donde una variable que hayamos definido tiene visibilidad. Cada vez que Ruby encuentra un ***def*** entra en un nuevo ámbito. Por lo tanto las variables definidas dentro de un método no podrán ser accedidas desde fuera del mismo.

- Con la palabra clave ***defined?*** podremos comprobar si una variable está definida en un ámbito en particular

```
def scope_the_scope
 on_the_inside = "oh. hi, friends."
 puts "from the inside: #{defined?(on_the_inside).inspect}" # from the inside: "local-variable"
end

scope_the_scope
puts "from the outside: #{defined?(on_the_inside).inspect}" # from the outside: nil
```

- Se pueden definir variables con ámbito global con el operador ***$***. Ruby ya define algunas (***$!*** Último error, ***$@*** localización del error, ***$_*** última cadena leída por gets …..)

Constantes

- Las constantes en Ruby se definen al ponerles la primera letra mayúscula en su nombre.

- La definición es un poco particular pues puede cambiárseles su valor si estamos usando el interprete. Pero cuando intentamos cambiar su valor de forma dinámica dentro de un programa nos genera un error.

- Por ejemplo el nombre de las clases son constantes

clone(I)

- Las variables en Ruby guardan referencias a los objetos por lo tanto

```
a = [1,2,3]
b = a
b << 4

p a # [1, 2, 3, 4]
p b # [1, 2, 3, 4]
```

- Hay que tener cuidado al hacer una asignación simplemente hacemos que nuestra nueva variable tenga una referencia al mismo objeto

clone(II)

- Si queremos tener una nueva variable con una referencia a un nuevo objeto que sea una copia de otro anterior tenemos que usar el método ***clone***.

```
a = [1,2,3]
b = a.clone
b << 4

p a # [1,2,3]
p b # [1,2,3,4]
```

freeze

- Si no queremos que se pueda modificar un objeto podemos utilizar el método freeze

```
a = "test"
a.freeze
a << "change”

RuntimeError: can't modify frozen String
```

- Hay que tener en cuenta que al hacerlo e intentar modificarlo nos salta un error.