Claves y Símbolos

Objetivos

- Conocer los símbolos en Ruby.
- Diferenciar una clave Símbolo con una clave String.
- Diferenciar la notación con : y con `=>.

Introducción

```
:hola_símbolo
```

Los símbolos son tipos de datos similares a los strings, pero están optimizados para ocupar menos memoria y son inmutables. Si en un programa tenemos declarado varias veces un string "Hola", cada vez será un objeto diferente (distinto espacio de memoria), pero si tenemos varias veces declarado el simbolo :hola, siempre se está referenciando al mismo objeto. En definitiva, son más eficientes en terminos de rendimiento computacional, pero son más limitados en sus métodos.

Los símbolos son muy utilizados como claves en los hashes.

Creando un símbolo

Los símbolos empiezan con ":"

```
a = :hola
puts a
```

¿Para qué sirven los símbolos?

Los símbolo son similares a los strings, pero son más simples. Soportan menos operaciones y sirven para representar estados o valores que no pueden cambiar como las llaves en los hashes.

```
a = {}
a[:foo] = 'foo'
a[:bar] = 'bar'
a[:foo] == 'foo' # => true
a[:bar] == 'bar' # => true
```

Los símbolos son más rápidos

Pero también tienen menos funcionalidades, o sea tienen menos métodos que los strings. Esto lo podemos probar ocupando el método `.methods`

```
print "prueba".methods - :prueba.methods
```

Los strings son mutables, los símbolos no

Los strings al ser mutables (poder cambiar de estado) son malos para representar cosas que no pueden cambiar. Para este tipo de situaciones son mejores los símbolos.

Por ejemplo supongamos que tenemos un semáforo que puede tener tres estados, y estos pueden ser `"rojo"`, `"amarillo"` o `"verde"`, en ese caso conviene utilizar símbolos y simplemente guardar el estado como símbolo, ejemplo `semáforo = :amarillo`

Arreglos y símbolos

Si el arreglo almacena posibles estados entonces también podemos aplicar la misma lógica.

```
estados_semaforo = [:verde, :amarillo, :rojo]
semaforo1 = estados_semaforo[0]
```

Hashes y símbolos

Los símbolos suelen ocuparse como clave de los hashes. Pero debemos ocupar un símbolo para acceder al elemento.

```
a = {:llave1 => 5, :llave2 => 10}
a[:llave1] # => 5
a['llave1'] # => nil
a[:'llave1'] # => 5
```

Hashes con símbolos y strings

Un hash puede tener simultaneamente claves que sean string y símbolos.

```
notas_alumnos = {'Camila' => 6, :David => 5}
```

Para acceder al respectivo valor tenemos que ocupar la clave correcto

```
notas_alumnos['Camila'] # => 6
notas_alumnos[:Camila] # => nil
notas_alumnos['David'] # => nil
notas_alumnos[:David] # => 5
```

Dos notaciones al momento de crear

`:` V.S. `=>`

Hasta al momento al crear un hash hemos agregado los pares de clave valor utilizando el operador rocket hash (o sea de la siguiente forma `clave => valor`). Existe otra forma de hacerlo, y es utilizando la notación `clave: valor`

Asociando elementos con `:`

La notación `'clave': valor` es muy interesante, convierte automáticamente un string en un símbolo. Creemos un diccionario utilizando ambas notaciones simultáneamente para compararlas.

```
a = {'forma1' => 5, 'forma2': 10}
```

Al utilizar esta forma vemos en el output ambos valores no se procesaron de la misma forma. Mientras que la clave `forma1` se mantuvo como string, la otra clave fue transformada en un símbolo de forma automática.

La notación con `:` es relativamente nueva

Esta forma de agregar elementos fue introducida en `Ruby 2.0`, Se aconseja utilizarla a menos que por algún motivo en especial necesitemos que la clave sea un string en lugar de un símbolo.

Otra ventaja de la notación de `:`

Las símbolos no tienen comillas, por lo mismo al ingresar claves lo podemos hacer sin ellas.

```
hash = { clave1: 5, clave2: 10, clave3: 15 }
```

Recordar esto será importante porque lo ocuparemos para muchos métodos que reciben hash.

Simple es mejor

Al momento de escoger alguna notación (si es que tienen el mismo resultado) debemos recordar que la forma más simple es mejor. Es es la filosofía que hace a Ruby un lenguaje fácil de leer.

Clave símbolo + valor símbolo

En la mayoría de los casos las claves serán símbolos, y en algunos casos los valores también serán símbolos.

```
hash = {clave1: :valor1, clave2: :valor2, clave3: :valor3}
```

Crearemos y veremos hashes de esta forma muy frecuentemente.

Haciendo memoria

Ya hemos ocupado esta notación, cuando abrimos un archivo CSV. Veamos el código:

```
CSV.open('data.csv', converters: :numeric)
```

Para Ruby esto es lo mismo que:

```
CSV.open('data.csv', {converters: :numeric})
```

Visitaremos nuevamente esta notación cuando veamos métodos que reciben hashes como argumentos

Reglas de sintaxis de un símbolo

Una de las reglas de símbolos es que no pueden ser un número o empezar con un número.

```
:hola2 # Válido
:2 # Inválido
:2hola # Inválido
```

Precaución con las claves que son números

Por la misma regla tenemos que tener cuidado cuando utilicemos la notación que convierte un string en un número.

```
colores = {'1' => 'verde', 2: 'amarillo' } #Error
```

El comportamiento también es raro cuando ocupamos un string que contiene solo un número

```
colores = {'1' => 'verde', '2': 'amarillo' }
=> #{"1"=>"verde", :"2"=>"amarillo"}
colores[:"2"] # => amarillo
```

Por lo mismo evitaremos ocupar números como clave.