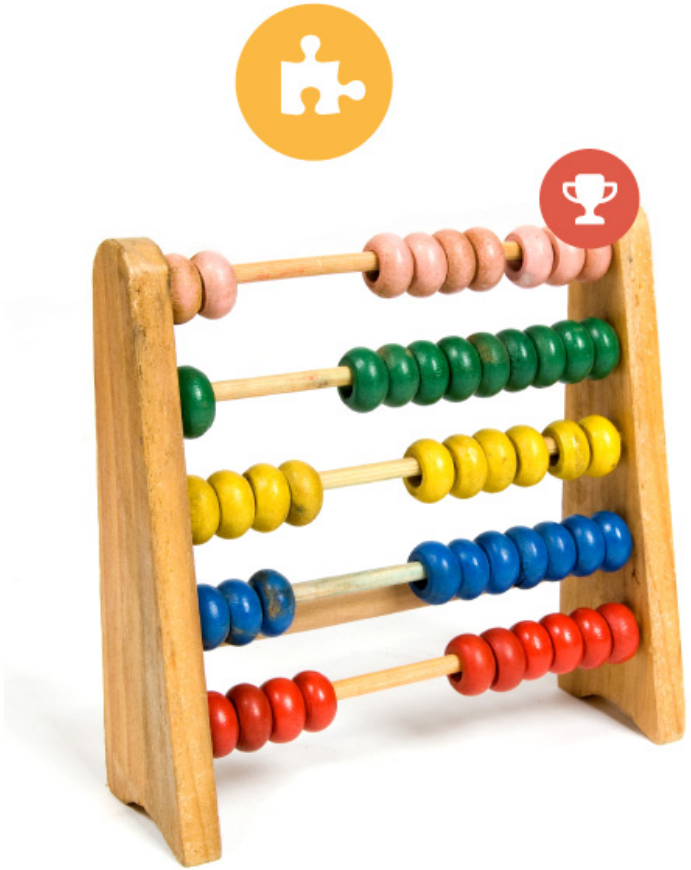
Programación en Lenguaje Ruby

Colecciones

Delio Tolivia

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



Índice

Enumerator y Enumerable

Iterar, filtrar y transformar

all?, any? Y none?

Unión, diferencia e intersección

Enumerator y Enumerable

- ***Enumerable*** es un módulo que se utiliza para hacer mixin con algunas colecciones como los arrays. Proporciona una serie de métodos como ***map***, ***select*** e ***inject*** (el método ***each*** es responsabilidad de la clase definirlo). Estos métodos retornan un objeto de tipo ***Enumerator***.

```
enumerator = [3, 7, 14].each
enumerator.each { |e| puts e + 1 }

enum = [0, -1, 3, 2, 1, 3].each_with_index
p enum.select { |element, index| element < index }
```

Iterar, filtrar y transformar (I)

- Con ***each*** podemos iterar a lo largo de un array. La variante ***each_with_index*** nos itera y permite definir un bloque con dos argumentos el valor y al índice del elemento en un array o en un hash:

```
{:locke => "4", :hugo => "8"}.each_with_index do |kv, i|
  puts "#{kv} -- #{i}”
end
```

```
[:locke, "4"] -- 0
[:hugo, "8"] -- 1
```

Iterar, filtrar y transformar (II)

- Con ***map*** haremos transformaciones en la colección (***each*** devuelve el array original y ***map*** devuelve el array resultante de su operación).

```
def map_value
    [3, 7, 14, 15, 22, 41].map { |e| e + 1 }
end

p map_value

[4, 8, 15, 16, 23, 42]
```

Iterar, filtrar y transformar (III)

- Con ***inject*** podemos iterar, acumular y transformar al mismo tiempo.

```
[4, 8, 15, 16, 23, 42].inject(0) do |accumulator, iterated|
  accumulator += iterated
  accumulator
end

108
```

- El argumento es opcional, le da el valor al primer argumento del bloque (si no lo tiene le da el valor del primer elemento de la colección).
- El segundo argumento del bloque es el elemento devuelto al iterar sobre la colección.

Ejercicio

- Hacer el ejercicio 8 del pdf de ejercicios

Iterar, filtrar y transformar (IV)

- Podemos usar inject para crear un hash por ejemplo

```
[4, 8, 15, 16, 23, 42].inject({}) { |a, i| a.update(i => i) }

{4=>4, 8=>8, 15=>15, 16=>16, 23=>23, 42=>42}
```

all?, any? y none?

- Con estos enumeradores podemos comprobar si alguno (any?), todos (all?) o ninguno (none?) de los elementos de una colección cumplen una condición

```
[4, 8, 15, 16, 23, "42"].any? { |e| e.class == String }

True

{:locke => 4, :hugo => 8}.any? { |candidate, number| number < 4 }

false
```

Ejercicio

- Hacer el ejercicio 10 del pdf de ejercicios

Unión, diferencia e intersección (I)

- El operador | permite realizar la unión de dos colecciones devolviendo una colección con los elementos de ambas colecciones eliminando los duplicados

```
union_example = ["a", "b", "a"] | ["c", "c"]
p union_example

["a", "b", "c"]
```

- El operador **&** realiza la intersección

```
array_interesection = [1,2,3, 1,2,3] & [1,2]
p array_interesection

[1,2]
```

Unión, diferencia e intersección (II)

- El operador **–** realiza la diferencia

array_difference = [1,2,3, 1,2,3] - [1]
p array_difference

[2, 3, 2, 3]

- Como podemos ver mantiene los elementos duplicados y elimina cualquier aparición del elemento indicado.

Ejercicio

- Hacer el ejercicio 9 del pdf de ejercicios
- Hacer el ejercicio 15 del pdf de ejercicios