Programming for Everybody

3. Loops e Iteradores

le wagon

@lewagonargentina

Porqué Looping?

Para repetir una acción en Ruby!

Ex: mostrar posts en tu página web

While loop

El número de veces que vas a realizar el loop es **desconocido**

Repite una acción en Ruby mientras cierta condición sea **verdadera (true)**

verifica si cierta condición es verdadera, y mientras lo sea, el loop continua; apenas la condición termina de ser verdadera, el loop se frena

Until loop

El número de veces que vas a realizar el loop también es **desconocido**

Repite una acción en Ruby mientras cierta condición sea **FALSA (false)**

verifica si cierta condición es falsa, y mientras lo sea, el loop continua; apenas la condición se transforme en verdadera, el loop se frena

Cuidado con los loops infinitos!

```
contador = 1

while contador < 11
    puts contador
    contador = contador + 1  (igual que counter +=1)
end
```

si nos olvidamos de **incrementar el counter** el loop va a ir verificando que si el counter es menor a 11

y como 1 es siempre menor a 11, el loop nunca terminará!

For loop

el número de veces que se repetirá la acción es **desconocido**

para repetir una acción de Ruby dentro de un rango de elementos

1..10 -> un rango que incluye los números de 1 a **10**

1…10 -> un rango que incluye los números de 1 a **9**

Next

usado para saltar algún paso dentro del loop

```
for number in 1..5
    next if number % 2 == 0
    print number
end
```

(saltea imprimiendo todos los números pares)

Iteradores

otra forma de hacer loops en Ruby!

un **iterador** es un método de Ruby que repetidamente invoca a un bloque de código

el bloque de código es la parte que contiene las instrucciones a repetir (y esas instrucciones pueden ser cualquier cosa que quieras!)

Iteradores

1. Loop

es el iterador más simple de todos

```
loop { print "Hola, mundo!" }
```

es lo mismo que

```
loop do
   print "Hola, mundo!"
end
```

Iteradores

1. Loop (cont.)

cuando usamos el método loop necesitamos usar un **“break”** para frenar el loop cuando cierta condición se cumpla

```
number = 0

loop do
   number += 1
   print number
   break if number > 5
end
```

(el loop frena luego de imprimir los números del 1 al 6)

Iteradores

2. Each

un operador más poderosos que puede aplicar una expresión a **cada elemento** de una **colección**, uno a la vez

```
collection_name.each do | item |
    #do something to each item
end
```

el nombre entre | | puede ser cualquier cosa -> es solo un placeholder para cada elemento de la colección que se esta aplicando .each

Iteradores

3. Times

hace algo una **cantidad de veces** que le indiques

```
10.times do
    #do something
end
```

Gracias! :)

le wagon

@lewagonargentina