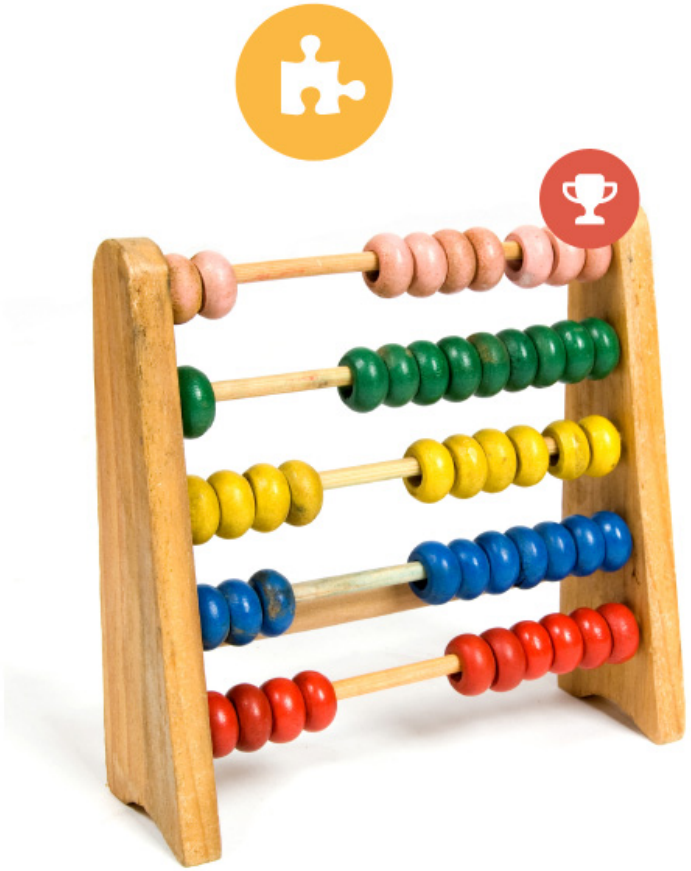
Programación en Lenguaje Ruby

Estructuras de control

Delio Tolivia

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



Índice

Expresiones

- Ruby utiliza como operadores para comparar objetos el **==** (igualdad), **>** (mayor que), **<** (menor que), **>=** (mayor igual que), **<=** (menor igual que).

```
2.0.0-p648 > name="Bob"
 => "Bob"
2.0.0-p648 > name=="Bob"
 => true
2.0.0-p648 > age = 30
 => 30
2.0.0-p648 > age<=35
 => true
```

- Todas las expresiones booleanas devuelven un objeto true o false

Combinando Expresiones

- Para combinar expresiones booleanas podemos utilizar el operador ***&&*** (AND) y el **||** (OR)

2.0.0-p648 :006 > (age >= 23) && ((name== 'Bob')||(name=='Jill'))
=> true

Negando Expresiones

- Para negar una expresión booleana utilizaremos el operador ***!***

2.0.0-p648 > name!="Bob"
=> false

If…else (I)

- En Ruby existe la construcción ***if….else*** como en otros lenguajes:

```
2.0.0-p648   > def check_sign(number)
2.0.0-p648 ?>     if number > 0
2.0.0-p648 ?>         "#{number} is positive"
2.0.0-p648 ?>     else
2.0.0-p648 ?>         "#{number} is negative"
2.0.0-p648 ?>     end
2.0.0-p648 ?> end
 => nil
2.0.0-p648 :015 > check_sign(-1)
 => "-1 is negative"
2.0.0-p648 :016 > check_sign(1)
 => "1 is positive"
```

If…else (II)

- Con el ejemplo anterior tenemos un problema con el 0:

```
2.0.0-p648 > check_sign(0)
 => "0 is negative"
```

- Podemos cambiar las condiciones o utilizar ***elseif***:

```
def check_sign(number)
    if number == 0
        "#{number} is zero"
    elsif number > 0
        "#{number} is positive"
    else
        "#{number} is negative"
    end
end
```

Unless

- Ruby presenta una estructura de control particular que es la que utiliza la palabra clave ***unless***. Es equivalente a comprobar una condición negativa:

```
2.0.0-p648 > age=10
 => 10
2.0.0-p648 > def check_age (age)
2.0.0-p648 ?>   unless age >=18
2.0.0-p648 ?>           puts "Sorry, you need to be at least eighteen to drive a car"
2.0.0-p648 ?>   end
2.0.0-p648 ?>end
 => nil
2.0.0-p648 > check_age(age)
Sorry, you need to be at least eighteen to drive a car
 => nil
```

Operador ternario

- Como en otros lenguajes disponemos del operador ternario (condición)?... : ....

```
2.0.0-p648 > def check_sign(number)
2.0.0-p648 ?>     number > 0 ? "#{number} is positive" : "#{number} is negative"
2.0.0-p648 ?>end
 => nil
2.0.0-p648 > check_sign(-1)
 => "-1 is negative"
```

Verdad y falsedad de los objetos

- En Ruby solamente false y nil evaluan a false. Cualquier otro objeto evalua a true por ejemplo:

```
2.0.0-p648 > def prueba
2.0.0-p648 ?>   if 0
2.0.0-p648 ?>     puts "0 is true!!!"
2.0.0-p648 ?>   end
2.0.0-p648 ?> end
 => nil
2.0.0-p648 > prueba
0 is true!!!
 => nil
```

Bucles loop do

- Esta estructura nos permiten repetir una zona de código de forma infinita. La forma de salir de estas repeticiones es el uso de ***break*** con una condición:

```
2.0.0-p648 > def loopdo (x)
2.0.0-p648 ?>     loop do
2.0.0-p648 ?>         x=x+1
2.0.0-p648 ?>         break if x>100
2.0.0-p648 ?>     end
2.0.0-p648 ?>     puts "x = #{x}"
2.0.0-p648 ?> end
 => nil
2.0.0-p648 > loopdo(50)
x = 101
 => nil
```

Bucles n.times do

- Otra forma de realizar los bucles “n” veces es utilizar la siguiente estructura:

```
2.0.0-p648 > def ring_bell(n)
2.0.0-p648 ?>    n.times do
2.0.0-p648 ?>            puts "RING!!!!"
2.0.0-p648 ?>    end
2.0.0-p648 ?> end
 => nil
2.0.0-p648 > ring_bell(5)
RING!!!!
RING!!!!
RING!!!!
RING!!!!
RING!!!!
 => 5
```