**Condicionales en Línea**

Los condicionales en Ruby son similares a otros lenguajes. Digamos que somos guardianes en un club de baile. Nuestro trabajo es dejar entrar personas con edad mayor o igual a 21 años.

age = 22

if age >= 21

puts "Bienvenido a la fiesta."

else

puts "Todavía no."

end

Si algo es verdadero, la primera parte de la sentencia se ejecutará, de lo contrario la sentencia **else**se activará. Podemos tener múltiples sentencias **if - else**. Un par de diferencias que Ruby utiliza es el termino **elsif**en lugar de **else if** o **elif**en Python y tenemos que agregar la palabra clave **end**después de cada bloque de condiciones.

number = 15

if number % 3 == 0 && number % 5 == 0

puts "FizzBuzz"

elsif number % 3 == 0

puts "Fizz"

elsif number % 5 == 0

puts "Buzz"

end

Algunas veces queremos hacer algo si algo no es verdadero. Por ejemplo, podemos tomar los guardianes del ejemplo anterior y reescribirlo de la siguiente manera:

if !(age < 21)

puts "Bienvenido a la fiesta."

else

puts "Todavía no."

end

Esto es bastante difícil de leer. No te preocupes, Ruby hace que estas sentencias sean más fáciles con **unless**:

unless age < 21

puts "Bienvenido a la fiesta."

else

puts "Todavía no."

end

Las sentencias **if**ejecutan el comando una vez que la condición sea verdadera. La sentencia **unless**es la negación de la sentencia *if*. Es lo mismo que *if not.*Intenta leer tu sentencia **unless** como:

A menos que la edad del cliente sea menor de 21 déjelo entrar, si no, rechácelos en la puerta.

**NIL**

Combinamos sentencias **if**y **else**con expresiones que evalúan verdadero o falso. En Ruby, solo 2 cosas son falsas: **nil**y**false**.

¿Qué significa esto? Esto significa que una cadena vacía devuelve un valor verdadero (true).

**Energía Positiva**

if ""

puts "Soy positivo."

end

if 0

puts "Soy positivo."

end

**Energía Negativa**

unless nil

puts "Soy negativo."

end

unless false

puts "Soy negativo."

end

**Condicionales en Línea**

Podemos escribir una sentencia **if/unless**en una línea. ¡Que hermoso!

puts "Soy positivo." if "hello"

puts "Soy positivo." if 24

puts "Soy negativo" unless nil

puts "Soy negativo" unless false

## Bucle While

i = 0

num = 5

while i < num do

puts "Dentro del bucle i = #{i}"

i +=1

end

Si quieres salir del bucle while antes que llegue al final, puedes utilizar **break:**

i = 0

num = 5

while i < num do

puts "Dentro del bucle i = #{i}"

i += 1

break if i == 2

end

Intente esto en un archivo diferente de ruby, cuando i = 2, el bucle while se detendrá.

## Bucle For

for i in 0..5

puts "El valor de la variable local es #{i}"

end

Podemos utilizar **break** para detener el bucle for antes que llegue al final, de la misma manera que lo haríamos con el bucle while.

for i in 0..5

puts "El valor de la variable local es #{i}"

break if i == 2

end

Otro método útil es **next**. Utilizaremos next, si en lugar de terminal de bucle for, queremos saltar al siguiente índice.

for i in 0..5

next if i == 2

puts "El valor de la variable local es #{i}"

end

A medida que conozca más acerca de Ruby, aprenderá que raramente utiliza el bucle **for**tradicional para hacer lo que desea.

**Iteradores**

* **.any? { |obj| block }** => true or false
  + e.g. ["ant", "bear", "cat"].any? { |word| word.length >= 3 } # => true
* **.each** => invoca el *bloque*una vez para cada elemento en ruby self, enviándo este elemento como parámetro al *bloque*.
  + e.g. ["ant", "bear", "cat"].each { |word| print word, "--" } # => ant--bear--cat--
* **.collect { |obj| block }** => devuelve un nuevo arreglo con los resultados después de haber ejecutado el *bloque* una vez para cada elemento en enum
  + e.g. (1..4).collect { |i| i\*i } # => [1, 4, 9, 16]
  + e.g. (1..4).collect { "cat" } # => ["cat", "cat", "cat", "cat"]
* **.detect/.find** => devuelve el primer valor para el que el *bloque* sea verdadero.
  + e.g. (1..10).detect { |i| i %5 == 0 and i % 7 == 0 } # => nil
  + e.g. (1..100).detect { |i| i %5 == 0 and i % 7 == 0 } # => 35
* **.find\_all { |obj| block }** ó **.select { |obj| block }** => devuelve un arreglo que contiene los elementos de enum para los que el *bloque* sea verdadero.
  + e.g. (1..10).find\_all { |i| i % 3 == 0 } # => [3, 6, 9]
* **.reject { |obj| block }** => lo opuesto de find\_all
  + e.g. (1..10).reject { |i| i % 3 == 0 } # => [1, 2, 4, 5, 7, 8, 10]
* **.upto(limit)** => recorre el bloque hasta alcanzar el número límite
  + e.g. 5.upto(10) { |i| print i, " " } # => 5 6 7 8 9 10