Clase 3: Ruby 3

Clase 3: Contenido

- Recapitular clase anterior: variables, métodos, if
- Más de Ruby
  - Else
  - Loops
  - Array y Hash
- Tarea

Launchpad

Recapitulando la Clase 2

- Variables
- Métodos
- If
- Mini tarea – revisión

Launchpad

Else

- El if es perfecto cuando quiero evaluar una sola condición
- Si tengo más de una, es más eficiente complementar el if con un else
- El programa evalúa las condiciones del else si es que las del if no se cumplen

```
if
  #algo
else
  #algo
end
```

Ejercicio #1

Instrucciones:

- Ajusta la última versión del ejemplo de going_hiking con un caso en el que **no** se cumple que la temperatura es mayor a 50 grados

```
puts “I’m staying in and coding!”
```

Launchpad

Elsif

- El último ejercicio nos permite definir si la temperatura es mayor a 50 o menor-igual a 50
- ¿Qué pasa si queremos agregar más condiciones? (ej.: la temperatura es 23)
- En este caso, usamos elsif
- Podemos usar todos los elsif que queramos y Ruby los va a recorrer en orden hasta llegar al else

Más operadores

- En Ruby y en muchos otros lenguajes de programación, existen los operadores OR y AND
  - OR (**||**): si el valor que va antes de || **o** el valor que va después de || es verdadero, todo es verdadero
  - AND (**&&**): si el valor que va antes de && **y** el valor que va después de && es verdadero, todo es verdadero

Ejercicio #2

Instrucciones:

- Ajusta el código de going_hiking para que retorne “It is 23 degrees” cuando los hayan y que retorne *"#{temp} is too cold--Brrrr!!”* si la temperatura es cualquier otro número menor a 50. HINT: usa && y !=

Ejercicio #3

```
def going_hiking(temp)

    if temp > 105 || temp < -5

        puts "#{temp} degrees is not a valid temperature for Santiago."

    elsif temp >= 50

        puts "#{temp} degrees is perfect for hiking!"

    else temp < 50

        puts "#{temp} degrees is WAY too cold!"

    end

end

puts going_hiking(temp)
```

Instrucciones:

- Ajusta este programa para chequear tanto la temperatura como el hecho de si está lloviendo o no
- El programa debe retornar “perfect por hiking” sólo si la temperatura está ok y no está lloviendo (más de una manera de hacerlo)

Launchpad

Tipo de variable: Boolean

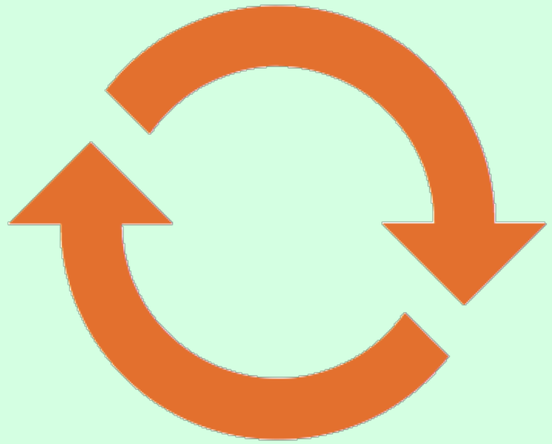
- Los boolean retornan verdadero o falso. Podemos usar la lógica booleana con números y strings

```
is_hungry = true

if is_hungry

    puts 'I am hungry.'

else

    puts 'I am not hungry.'

end
```

Launchpad

Loops

- En programación es muy común repetir x veces la ejecución de cierto pedazo de código, para lo que necesitamos usar loops
- En ruby hay diversas maneras de realizar loops



For loop (1/2)

- El **for** no se utiliza mucho en Ruby, pero es un buen primer ejemplo de lo que un loop puede hacer

```
puts "1! Bird on a wire.  Ha-ha-ha."

puts "2! Birds on a wire. Ha-ha-ha."

puts "3! Birds on a wire. Ha-ha-ha."

puts "4! Birds on a wire.  Ha-ha-ha."
```

¿Qué pasa si quisiéramos repetir esta frase 200 veces?

For loop (2/2)

- Un **for** bastará en este caso para no tener que escribir de más

```
puts "1! Bird on a wire.  Ha-ha-ha."

 #We can't get around typing this one out, since 'bird' is singular.  But now we can loop!

for n in 2..100
    puts "#{n}! Birds on a wire. Ha-ha-ha."

end

 #2..100 represents a Range in Ruby.  The loop will go through the range of numbers from low
#to high.  If you put three "..." instead of two "..", the range will exclude the high value.  Try it!
```

Ojo que acá es se comienza con el "1" por separado porque en este caso "bird" es singular

Launchpad

While loop (1/2)

- El **while** es bastante más común y se ejecutará continuamente siempre y cuando se mantenga **true** la condición impuesta

```
x = 0

while x <= 10

    puts "#{x} is the loneliest number."

end
```

Ojo que creamos un infinite loop! ¿Por qué? Córtalo con Ctrl+C

While loop (2/2)

- Ajustando el ejemplo anterior, lo podemos convertir en un loop no infinito

```
x = 0

while x <= 10

  puts "#{x} is the loneliest number."

   x = x + 1

end
```

x+=1 es lo mismo que x=x+1

Until loop

- El **until** es similar al **while**, sólo que al revés: el loop se ejecuta hasta que las condiciones se cumplan

```
x = 0

until x == 10

    puts "#{x} isn't 10 yet!"
    x += 1

end
```

Ejercicio #4

Instrucciones:

- Escribe un loop until que imprima los números pares entre 1 y 100

Launchpad

Times do loop

- El **times do** es otro loop, el cual se ve de la siguiente manera

```
5.times do |i|
    puts "We are at number #{i+1} “
    # the times do loop will start counting at 0, so on the 10th iteration, i is equal to 9
end
```

- También funciona con strings

```
3.times do

  puts “She loves you, yeah yeah yeah!”

end
```

Launchpad

Tipo de variable: Array

- El array o arreglo almacena valores en una lista ordenada
- Un arreglo puede almacenar strings, enteros e incluso otros arreglos

```
my_array = [ ]

 #the square brackets contain the information in the array.  Let's put something in my_array

 my_array[0] = "Hello"

 my_array[1] = 45

 my_array[2] = [1,2,3]

 #information in an array is stored according to an index number, which starts at zero.

 puts my_array
```

Launchpad

Rellenando un arreglo

- El arreglo también se puede rellenar en una sola línea

```
my_array = [“Hello”, 45, [1,2,3]]

puts my_array
```

- O usando el método push

```
my_array = [ ]

my_array.push(“Hello”)

my_array.push(45)

my_array.push([1,2,3])


puts my_array
```

Launchpad

Accediendo a un arreglo

- Para acceder a los valores de un arreglo, usa el index correspondiente

```
puts my_array[0]

#this will return the value associated with this index, in this case, “Hello”.
```

Arreglos y loops

- Con un loop se puede recorrer el arreglo completo, revelando la información que contiene

```
scores = [100, 85, 30, 79]

counter = 0

sum = 0

while counter < scores.length

sum = sum + scores[counter]

counter += 1 # same as counter = counter + 1

end

puts "The total is #{sum}"
```

Each loop

- El loop .each do recorrer cada uno de los elementos del arreglo

```
scores = [100, 85, 30, 79]

sum = 0

scores.each do |score|

sum = sum + score

counter += 1 # same as counter = counter + 1

end

puts "The total is #{sum}"
```

Tipo de variable: Hash (1/2)

- El hash es como un arreglo pero no tiene sus elementos ordenados en una lista del 1 al n

- Cada elemento del hash (**value**) es asociados a una llave única (**key**) en lo que llamamos lista desordenada

Tipo de variable: Hash (2/2)

```
my_hash = { }

my_hash[“name”] = “Dana”

 my_hash[“age”] = 34

 my_hash[“eye_color”] = “green”

 #this will populate your hash item by item

 my_hash = {

   “name” => “Dana”,

   “age” => 34,

   “eye_color” => “green” }

 #this method uses the hash rocket: “=>”, to assign values to keys.

 my_hash = { :name => “Dana”, :age => 34, :eye_color => “green”}

 #you can use the colon in front of the key value to enter information into a hash as well

puts my_hash
```

Launchpad

Accediendo a un Hash

- Para acceder a los valores de un hash, debemos usar el key correspondiente para legar al valor

```
my_hash[“name”]

 #returns “Dana”


my_hash[:name]
 #returns “Dana”
```

Hashes y loops

- Con un loop se puede recorrer el hash completo, revelando la información que contiene en pares key-value

```
my_hash = { :name => “Dana”, :age => 34, :eye_color => “green”}

my_hash.each do |key, value|

puts “The key is #{key} and the value is #{value}.”

 end
```

Launchpad

Tarea #2 (1/3)

1. Usa el método each para iterar sobre [1,2,3,4,5,6,7,8,9,10] e imprime cada valor
2. Lo mismo que antes, pero imprime sólo valores superiores a 5
3. Agrega 11 al arreglo original usando push o “<<“
4. Supón que tienes el hash h = {a:1, b:2, c:3, d:4}
   - Imprime el valor de la key :b
   - Agregar a este hash la key:value {e:5}
   - Elimina todos los pares key:value cuyo value es menor a 3.5

Tarea #2 (2/3)

5. Escribe un programa que mueva la información desde el arreglo hasta el hash que aplique a la persona correcta:

```
contact_data = [["joe@email.com", "123 Main st.", "555-123-4567"], ["sally@email.com", "404 Not Found Dr.", "123-234-3454"]] contacts = {"Joe Smith" => {}, "Sally Johnson" => {}}
```

6. Usando el hash recién creado en #5, ¿cómo accederías al teléfono de Joe y Sally?

Tarea #2 (3/3)

7. BONUS: En el ejercicio #5, seteamos manualmente los valores del hash uno por uno. Ahora, crea un programa que itere sobre el hash de contactos del ejercicio #5 y lo rellene con la data asociada de contact_data. Tip: probablemente tendrás que iterar sobre [:email, :address, :phone] y algunos métodos útiles pueden ser **shift** y **first** (Google)

Clase 3: Ruby 3