Lo que necesitas saber antes de comenzar la unidad *Hashes y APIs*

Para entender esta unidad es necesario que tengas dominio de los siguientes temas vistos al comienzo del módulo:

- Crear métodos o funciones.
- Alcance de Variables.
- Ciclos y estructuras de control.
- Bloques.
- Arreglos (Arrays) e iteradores.

Ciclos

Existen diversas formas de repetir una o más instrucciones varias veces.

- `while` y `until`
- `for`
- `.times`

Además, para iterar los arreglos (recorrer los elementos uno a la vez) existen métodos incluidos en los mismos arreglos para recorrerse a sí mismos.

- `.each`
- `.map` o `.collect`
- `.select`
- `.reject`
- `.inject`

Estos últimos métodos más el .times van acompañados de bloques para indicar que se debe hacer en cada iteración.

Creando bloques

Existen dos formas de declarar un bloque; entre `do` y `end`

```
10.times do |i|
   puts i
end
```

Y de forma inline con las llaves.

```
10.times { |i| puts i }
```

Alcance de las variables locales

También es importante que recordemos que las variables tienen alcance y si las definimos dentro del bloque solo existirán dentro del bloque.

```
%w[1 2 3 4].each do |letter|
   money_symbol = '$'
   puts "#{money_symbol + letter}"
end

puts money_symbor # NameError: undefined local variable or method `money_symbol'
```

En el caso de los métodos, el alcance de las variables locales tiene límite entre el `def` y `end`:

```
def foo
   bar = 5
end

foo
puts bar # NameError: undefined local variable or method `money_symbol'
```

No ocuparemos variables globales.

Índices de los arreglos

- Los elementos en el array tienen una posición, a esta posición se le llamada índice.
- Los índices van de cero hasta n - 1, donde n es la cantidad de elementos del arreglo.
- Si un array contiene 5 elementos, el primer elemento estará en la posición cero, y el último en la cuarta posición.

```
a = [1, 2, 'hola', 'a', 'todos']
a[0] # => 1
a[1] # => 2
a[4] # => todos
```

Índices fuera de borde

En caso de que el índice sea mayor o igual a la cantidad de elementos obtendremos nil, sin ninguna otra consecuencia.

```
a[8] # => nil
```

Los índices también se pueden utilizar con números negativos y de esta forma referirse a los elementos desde el último al primero.

```
a = [1, 2, 3, 4, 5]
a[-1] # => 5
```

Membresía

Podemos saber si un elemento se encuentra dentro de un array utilizando el método `.include?`

```
a.include? 4 # => true
```

Remover un elemento

El método .delete entrega el elemento removido o nil si no lo encuentra.

```
a = [1, 2, 3, 4, 5, 6, 7]
a.delete(2) # => 2

a = ['Do', 'Re', 'Mi', 'Fa', 'Sol', 'La', 'Si']
a.delete(2) # => nil ¿Por qué?
```

Ejemplos de iteración con índice

El iterador each_with_index entrega además de cada elemento el índice (posición) que ocupa en el array partiendo desde el cero.

```
a = ['Do', 'Re', 'Mi', 'Fa', 'Sol', 'La', 'Si']
a.each_with_index do |value,index|
   puts "#{index + 1})#{value}"
end
```

Ejemplos de iteración con `.each`

```
array = ['Ruby','Javascript', 'Git', 'CSS']
array.each do |technology|
   puts "En el BootCamp aprenderé #{technology}"
end
```

Transformación con `.each`

```
array = [1, 2, 6, 1, 7, 2, 3]
new_array = []
array.each do |ele|
   new_array.push ele + 1
end
print new_array
```

Filtrando con `.each`

```
array = [8, 2, 5.3, 7, 2, 9, 9, 6]
new_array = []
array.each do |ele|
  if ele > 5
    new_array.push(ele)
  end
end
print new_array
```

Creando un nuevo array a partir de otro con `.map`

El método .map entrega un nuevo array con el resultado del bloque aplicado a cada elemento. No modifica el array original.

```
a = [1, 2, 3, 4, 5, 6, 7]

b = a.map do |e|
    e * 2
  end

# O con bloque inline
b = a.map { |e| e * 2 }
```

Filtrando con `.select`

```
a = [1, 2, 3, 4, 5, 6, 7]
b = a.select{ |x| x % 2 == 0} # seleccionamos todos los pares
# => [2,4,6]
```

Filtrando con `.reject`

```
a = [1, 2, 3, 4, 5, 6, 7]
b = a.reject{ |x| x.even? } # lo mismo que { |x| x % 2 == 0}, pero más legible (syntactic sugar)
```

Reduciendo con `.inject`

El método inject entrega un solo elemento concatenando sucesivamente el resultado de cada iteración.

```
a = [1, 2, 3, 4, 5, 6, 7]
b = a.inject(0){ |sum, x| sum += x } # => 28
```

Abrir y leer un archivo

Leer todo un archivo y guardarlo en un string

```
content = File.read 'file.txt' # Si el archivo no es muy grande
```

Transformar datos un archivo

```
data = File.open('data').read.chomp.split(',')
```

Leer un archivo de múltiples líneas

```
data = File.open('archivo2').readlines
```

Guardar los resultados

```
File.write('/path/to/file', 'datos')
```