{desafío} latam_

Lo que debes saber antes de comenzar la unidad *Arreglos y archivos*

Ciclos

Los ciclos son instrucciones que nos permiten repetir la ejecución de una o más instrucciones.

Existen diversas formas de implementar estos ciclos en ruby, particularmente revisamos:

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

Los problemas de iteración suelen utilizar contadores y acumuladores en las soluciones.

- Los contadores son una variable que va sumando de uno en uno, muy útiles cuando una operación tiene
  que ocurrir n veces.
- Los acumuladores guardan el valor de la iteración anterior y lo juntan con el de la iteración actual, ejemplos de estos son las sumatorias y productorias.

Ciclos y Condiciones de borde

Las condiciones de borde son muy importante en los ciclos, debemos analizar el número de iteraciones para evitar contar (o acumular) de más o de menos.

Por ejemplo para que el siguiente código se repita 5 veces debemos tener cuidado de ocupar > y no mayor o igual.

```
i = 5
while(i > 0)
   puts i
   i -= 1
end
```

Bloques

Los bloques nos permiten pasar un grupo de instrucciones para ejecutarlas dentro de un método. Un bloque es todo lo que está entre `do` y `end` o entre llaves {}.

Bloques y ciclos

Algunos métodos que realizan ciclos utilizan bloques, como por ejemplo `times`.

Creando bloques

Existen dos formas de declarar un bloque:

con `do` y `end`

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

Y de forma inline

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

Métodos

Los métodos nos permiten:

- Reutilizar código.
- Ordenarlo.
- Evitar repeticiones innecesarias (DRY).
- Abstraernos del problema.

Para crear un método ocuparemos la instrucción `def`

```
def metodo
end
```

Los métodos pueden recibir parámetros obligatorios u opcionales.

```
def metodo(x, y z = 5)
    puts z # 5
end
```

Los métodos retornan la evaluación de la última línea.

```
def metodo(x, y z = 5)
    puts z # puts z devuelve nil
end
resultado = metodo(2,3,1) # Se muestra 5 en pantalla, pero resultado será nil
puts resultado # nil
```

El alcance o **scope** es la asociación entre el nombre de una variable y una zona del programa. Hemos estudiado dos tipos de alcances.

- locales (solo se pueden acceder desde el método).
- globales (se pueden acceder desde cualquier parte del programa).

El alcance de las variables locales tiene límite entre el `def` y `end`:

```
def foo
    bar = 5
end

foo
puts bar # error :)
```

Las variables globales son peligrosas, especialmente cuando trabajamos con más personas o código de otros, puesto que es muy probable que sobreescribamos variables.