



`/* Ciclos y métodos */`

Sesión conceptual 1





Inicio

{desafío}
latam_



15 minutos

- Conocer los ciclos y sus posibles aplicaciones en la programación.
- Leer y transcribir diagramas de flujo con iteraciones a código Ruby.
- Contar la cantidad de veces que un programa está dentro de un ciclo.
- Realizar programas donde el usuario ingrese múltiples datos hasta que decida detenerse.
- Ciclos y sumatorias.
- Escribir en ruby el código de una sumatoria.
- Diferenciar contadores de acumuladores.
- Utilizar bloques.
- Reconocer patrones de repetición en un ciclo.

Objetivo



Desarrollo

{desafío}
latam_



130 minutos

/* Ciclos */

Introducción a ciclos

- Los ciclos son instrucciones que nos permiten repetir la ejecución de una o más instrucciones.
 - Repetir instrucciones es la clave para crear programas avanzados.

Mientras se cumple una condición:

Instrucción 1

Instrucción 2

Instrucción 3

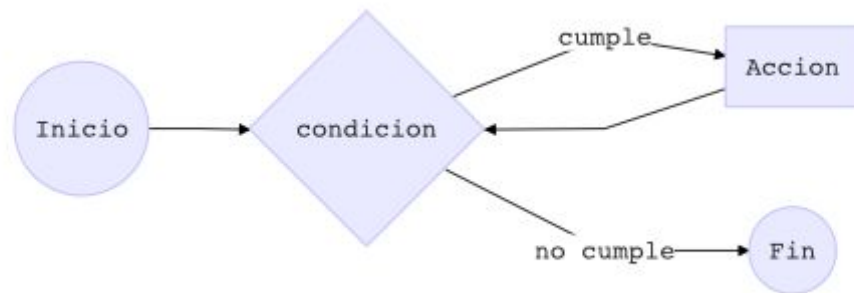
Usos de ciclos

- Los posibles usos de ciclos en algoritmos son infinitos, nos permiten recorrer colecciones de datos o espacios de búsqueda.
- Parte importante de aprender a programar es entender cómo utilizarlos correctamente y desarrollar las habilidades lógicas para entender cuándo utilizarlos.

WHILE

La instrucción while nos permite ejecutar una o más operaciones mientras se cumpla una condición.

While paso a paso

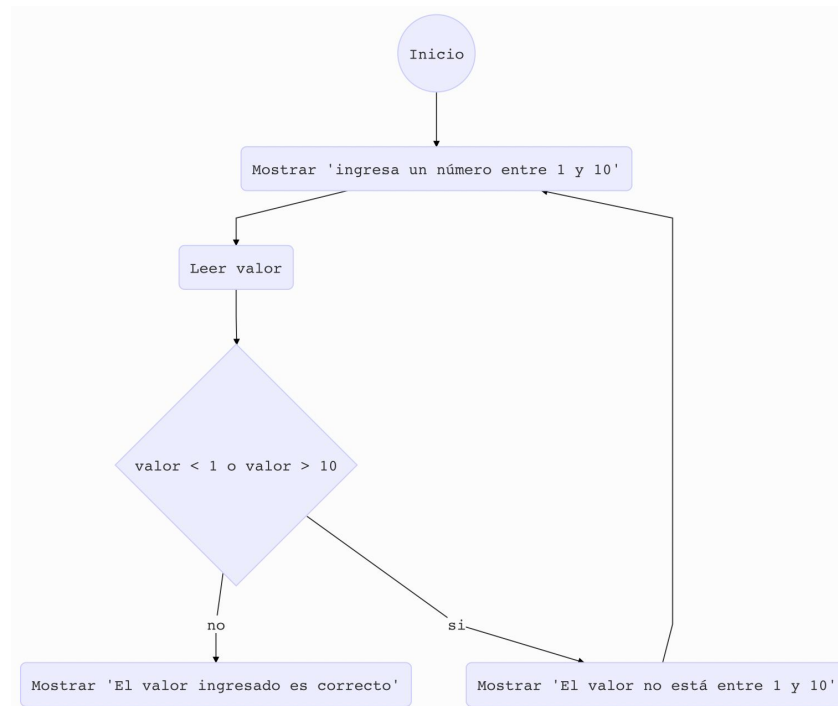


Salida del ciclo

Un algoritmo es una secuencia FINITA de pasos para resolver un problema.

Validación de entrada de datos utilizando while

Un ejemplo común para comenzar a estudiar los ciclos es validar la entrada de un dato, es decir, que este dato cumpla un criterio.



Existe una instrucción que es la inversa de while

En decir, `while i < 0` es lo mismo que `until i >= 0`.

`while` es mucho más utilizado que `until`, pero es bueno conocerlo por si lo encontramos en algún ejemplo.

Ejercicio: Validación de password

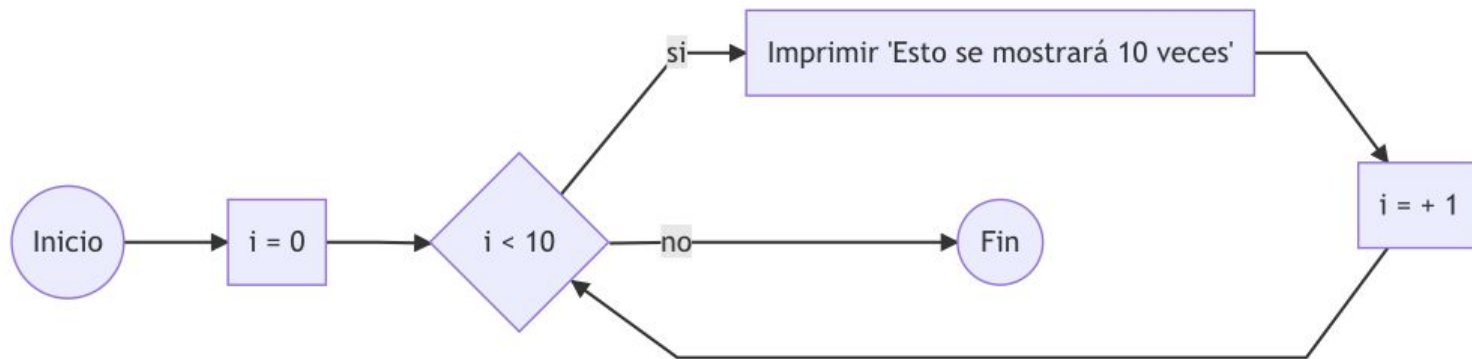
- Crear un diagrama de flujo con ciclos.
- Crear un programa llamado `password_validation.rb` que valide la contraseña de un usuario.

Ejercicio menú de opciones

- Podemos implementar de forma sencilla un menú de opciones para el usuario.
- La lógica es similar a la de la validación de entrada.
- Desarrolla el diagrama de flujo de la solución y luego implementa el algoritmo. En el próximo capítulo revisaremos la solución.

/* Ciclos y contadores */

- Iterar es dar una vuelta al ciclo. Hay muchos problemas que se pueden resolver iterando. Por ejemplo repetir un mensaje 10 veces o contar desde cero hasta 10.



Contando con while

```
i = 0
while i < 10
  puts "Esto se mostrará 10 veces" # Código que queremos repetir.
  i += 1 # IMPORTANTE
end
```

- La instrucción puts "Esto se mostrará 10 veces" se repetirá hasta que la variable i alcance el valor 10.
- Para entonces, la comparación de la instrucción while se evaluará como false y saldremos del ciclo.

Operadores de asignación

Operador	Nombre	Ejemplo	Resultado
=	Asignación	a = 2	a toma el valor 2
+=	Incremento asignación	a += 2	a es incrementado en dos y asignado el valor resultante
-=	Decremento asignación	a -= 2	a es reducido en dos y asignado el valor resultante
*=	Multiplicación asignación	a *= 3	a es multiplicado por tres y asignado el valor resultante
/=	División asignación	a /= 3	a es dividido por tres y asignado el valor resultante

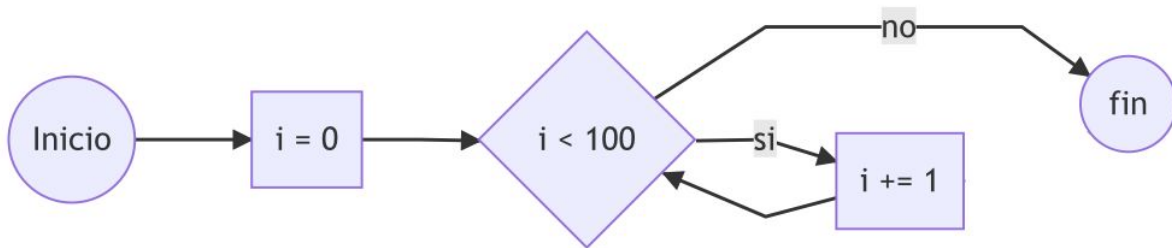
Ejercicio: La bomba de tiempo

- Crearemos un algoritmo sencillo que realice una cuenta regresiva de 5 segundos.
- Contar de forma regresiva es muy similar, solo debemos comenzar desde el valor correspondiente e ir disminuyendo su valor de uno en uno.

/* Ciclos y sumatorias */

Sumando números

La sumatoria consiste en sumar todos los números de una secuencia. Por ejemplo: sumar todos los números entre 1 y 100. Esto no solo sirve para resolver ecuaciones matemáticas, sino también para que generemos las habilidades de abstracción necesarias para resolver diversos problemas.



Ejercicio: Generador de listas en HTML

- Se pide crear el programa `generador_li.rb` donde el usuario ingrese un número como argumento y se genere una lista de HTML con esa cantidad de ítems.

`/* Otras instrucciones para ciclos */`

Ciclos con la instrucción For

La instrucción for nos permite iterar en un rango. Podemos, por ejemplo, iterar de 1 a 10 con la siguiente sintaxis:

Recordemos que es convención utilizar i como variable de iteración.

```
for i in 1..10  
  puts "Iteración #{i}"  
end
```

Ventaja de for

La principal ventaja de for es que nos podemos olvidar de implementar un contador, ya que la variable de iteración aumenta en cada iteración de forma automática.



{desafío}
latam_

Desventaja de for

La desventaja es que es menos flexible, porque no podemos manipular el incremento de manera personalizada.



El ciclo times

- El método times que podemos traducir como a veces nos permite iterar de una forma muy sencilla.

```
5.times do  
  puts "repitiendo"  
end
```

Introducción a bloques

- Tanto times como muchos otros métodos pueden recibir bloques. Profundizaremos en él más adelante, sin embargo, en este punto necesitamos entender su sintaxis.
- Un bloque se puede escribir de dos formas:

Con **DO** y **END**

```
10.times do  
  puts 'Repitiendo 10 veces'  
end
```

Entre llaves {}.

```
10.times { puts 'repitiendo 10 veces' }.
```

Los bloques son closures

- Todo lo que declaremos dentro de un bloque queda definido solo dentro del bloque.
- Ejemplo de declaración dentro de un bloque:

```
10.times do |i|
```

```
  z = 0
```

```
end
```

```
puts z # undefined local variable or method `z' for main:Object
```

Iterando con times y el índice

De la primera forma:

```
10.times do  
  puts 'Repitiendo 10 veces'  
end
```

De forma inline

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

/* Sumatorias y productorias */

Productorias

- Existe una expresión similar a las sumatorias, llamadas productorias. Son exactamente lo mismo, pero los números se multiplican en lugar de sumarse.

```
producto = 1 # Es importante no inicializar el producto en 0, porque  
cualquier multiplicación por cero dará como resultado cero.  
for i in (1..10)  
  producto *= i  
end  
print producto
```

/* Introducción a patrones con ciclos */

Ejercicio: Dibujando puntos

- Crear el programa solo_puntos.rb que dibuje n puntos. Donde n es un valor ingresado por el usuario al momento de ejecutar el script.

Ejercicio: Puntos y números

- Crear el programa puntos_y_numeros.rb que dibuje N números intercalados por puntos. Donde N es un valor ingresado por el usuario al momento de ejecutar el script.

¿Cuál es el patrón?

La primera pregunta es cuál es el patrón, en este caso vemos que los números impares son los que son reemplazados por puntos y los números pares se muestra el mismo número.

- Si número es par => número.
- Si número es impar => punto.

¿Par o impar?

Hay dos formas de saber si un elementos es par.

Utilizar el método
.even.

Utilizar la operación
resto

Utilizando el método .even?

- En ruby existe un método muy sencillo llamado .even?.

```
2.even? # => true
```

```
3.even? # => false
```

Utilizando la operación resto

- Pero existe otra forma un poco más flexible y muy utilizada en general que es calcular el resto de la operación. O sea, si al dividir un número por dos, la división es exacta, es par, y si queda un resto, entonces es impar.

```
5 % 2 == 0 # => false
```

```
5 % 2 == 1 # => true
```

Ejercicio: Dibujando asteriscos y puntos

- Crear el programa `asteriscos_y_puntos.rb` que dibuje asteriscos y puntos intercalados hasta **n**. Donde **n** es un valor ingresado por el usuario al momento de ejecutar el script.

Ejercicio: dos por dos

- Crear el programa **dos_por_dos.rb** que dibuje el siguiente patrón de asteriscos y puntos intercalando hasta **n**.
- Donde **n** es un valor ingresado por el usuario al momento de ejecutar el script.

Ejercicio: patrón

- Escribir el programa patron3.rb que permita dibujar el siguiente patrón:

```
..**||..**||..**||
```



Quiz

{desafío}
latam_



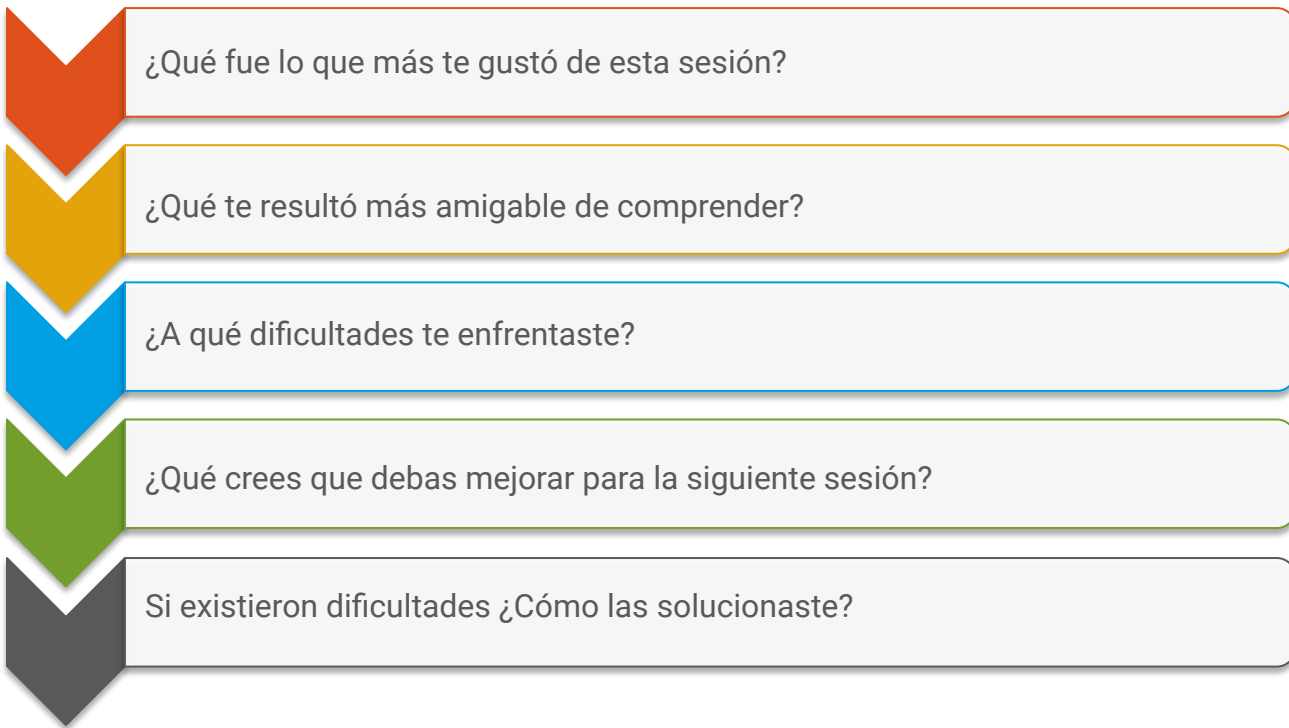


Cierre

{desafío}
latam_



15 minutos



¿Qué fue lo que más te gustó de esta sesión?

¿Qué te resultó más amigable de comprender?

¿A qué dificultades te enfrentaste?

¿Qué crees que debas mejorar para la siguiente sesión?

Si existieron dificultades ¿Cómo las solucionaste?



*Academia de
talentos digitales*

www.desafiolatam.com



/DesafioLatam



/DesafioLatam



/DesafioLatam



/DesafioLatam