



Flujo

Sesión Conceptual 2



- Conceptualización
- Ejercitación



- Reflexión



Desarrollo

{desafío}
latam_



100 minutos

/* Condiciones de borde */

Analizando una condición de borde

```
puts "¿Qué edad tienes?"
edad = gets.chomp.to_i

if edad >= 18
  puts "Eres mayor de edad"
else
  puts "Eres menor de edad"
end
```

En este ejemplo, el punto que define una rama de la otra es el 18. A este punto se le suele llamar punto crítico, mientras que los bordes son los números que "bordean" el 18, es decir, analizar los bordes corresponde a probar con los valores 17, 18 y 19.



Quiz

{desafío}
latam_



/* Operadores lógicos */

Operadores lógicos

Operador	Nombre	Ejemplo	Resultado
&&	y (and)	false && true	Devuelve true si ambos operandos son true, en este ejemplo se devuelve false.
||	o (or)	false || true	Devuelve true si al menos una de los operando es true, en este ejemplo devuelve true.
!	no (not)	!false	Devuelve lo opuesto al resultado de la evaluación, en este ejemplo devuelve true.

Identities

'Igual' es lo mismo que 'no distinto': Negar algo dos veces es afirmarlo (en español, no siempre es así; en programación, sí).

Mayor y no menor igual: Un caso similar es la comparación $a > 18$. Decir que a no es mayor a 18, es decir que es menor o igual a 18, (debemos incluir el 18 al negar).

Unless: Para ayudarnos a escribir las condiciones siempre en positivo, existe una instrucción que es el antónimo del if, está se llama unless.

/* Simplificando IFs anidados */

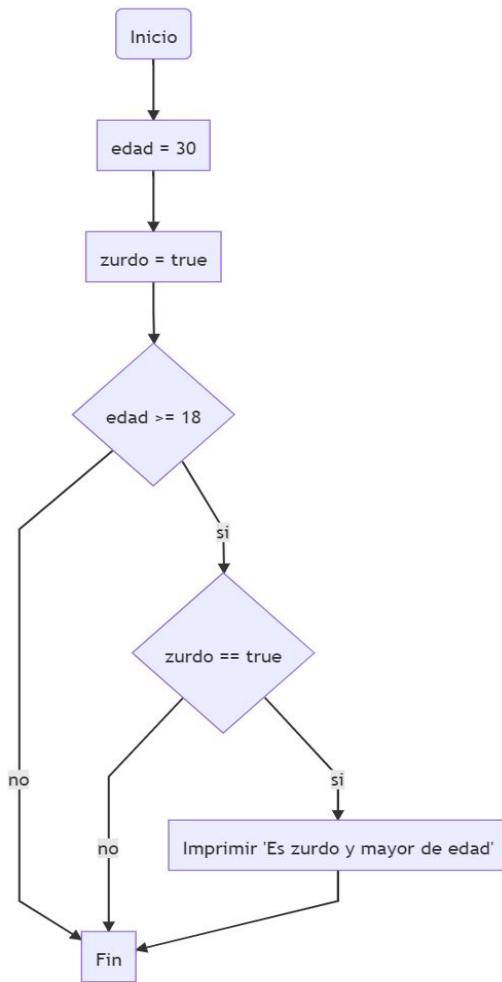
Identificando ifs anidados

```
edad = 30
zurdo = true

if edad >= 18
  if zurdo == true
    puts 'Es zurdo y mayor de edad'
  end
end
```

En el ejemplo anterior vemos un if dentro de otro if: a esto se le llama tener if anidados.

El código escrito muestra el texto sólo si se cumplen las dos condiciones.



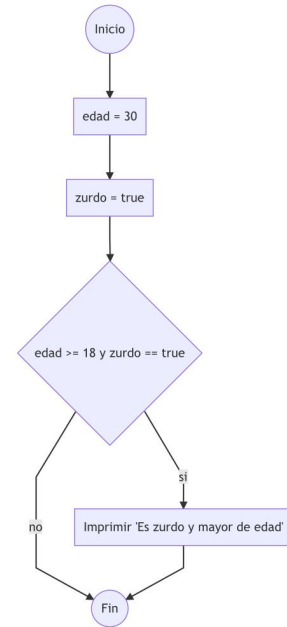
Operadores lógicos

Podemos simplificar el código para evaluar ambas condiciones en una misma instrucción if.

- ¿Qué operador podemos utilizar si necesitamos que ambas condiciones sean verdaderas?

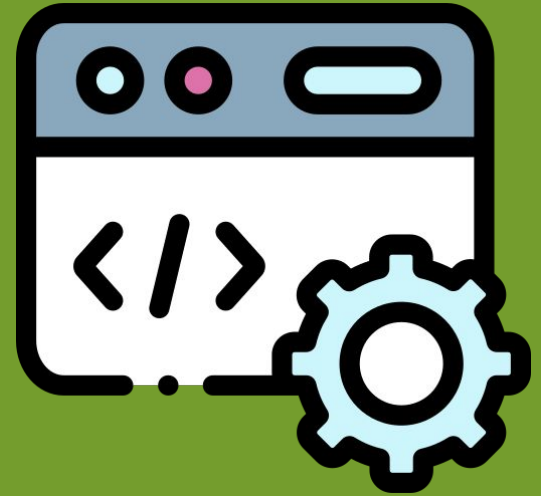
```
edad = 30
zurdo = true

if edad >= 18 && zurdo == true
  puts 'Es mayor de edad y zurdo'
end
```



Ejercicio de integración

Se busca crear un programa que solicite al usuario ingresar tres números. El programa debe determinar el mayor de ellos. Se asume que los números ingresados serán distintos.



/* Simplificando el flujo */

Variantes de IF

If en una línea (inline)

En Ruby, también es posible utilizar versiones cortas de la instrucción if y unless de la siguiente forma: action if condition.

Operador ternario

El operador ternario es una variante de if que permite operar en base a dos caminos en condiciones simples.

La lógica es la siguiente:

```
si_es_verdadero    ?    entonces_esto    :  
sino_esto
```

Refactorizar

Podemos refactorizar las comparaciones en los condicionales que son innecesarias.

¿Cómo?: Recordemos que la instrucción `if` espera que el resultado se evalúe como `true` o `false`. Por lo tanto:

```
mayor_de_edad = true

if mayor_de_edad == true
  puts "Mayor de edad"
end
```

Es igual:

```
mayor_de_edad = true

if mayor_de_edad
  puts "Mayor de edad"
end
```


Análisis léxico

Existe un programa encargado de traducir cada una de nuestras instrucciones a un lenguaje que nuestro computador pueda entender.

Conocer los procesos que realiza Ruby al leer un programa nos permitirá entender el por qué son necesarias estas reglas y nos ayudará a memorizarlas.

- **¿Qué sucede cuando ingresamos al terminal y escribimos ruby mi_programa.rb?**
 - Ruby lee y procesa nuestro código.

El proceso

Etapa 1: Tokenización

Ruby, al leer un código, lo primero que realiza es una tokenización. Esto consiste en separar cada palabra o símbolo en unidades llamadas tokens.

Etapa 2: Lexing

A través de un proceso llamado lexing se clasifican los tokens. Cada token es identificado según reglas.

Etapa 3: Parsing

Consiste en generar un árbol llamado árbol de sintaxis abstracta. Esto corresponde a una forma de representar el código que le permite -a Ruby- saber en qué orden ejecutar las operaciones, o descubrir si todo paréntesis abierto se cierra.

Reglas Básicas

Ahora estudiaremos las reglas básicas de Ruby a partir de los tipos de tokens.

Mencionamos anteriormente que en una etapa los tokens son clasificados. Esta clasificación los separa en:

- Identificadores.
- Literales.
- Comentarios.
- Palabras reservadas.

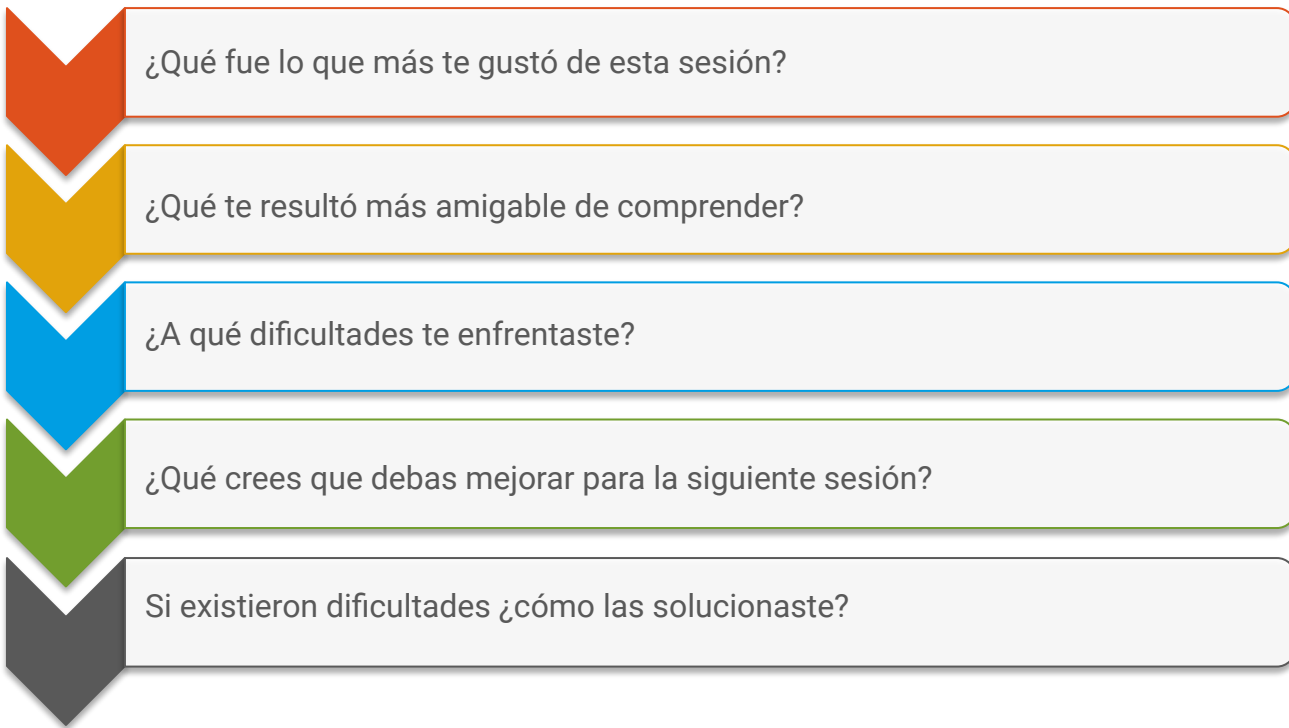


Cierre

{desafío}
latam_



20 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