



# ¡Que el último apague la luz!

- 

Empecemos por algo sencillo, ¿te acordás del operador `!`? Se lo denomina negación, not o complemento lógico y sirve para negar un valor booleano.

Si tengo el booleano representado por `tieneHambre`, el complemento será `!tieneHambre`.

¿Y esto para qué sirve? Por ejemplo, para modelar casos de alternancia como prender y apagar una luz:

```
let lamparaPrendida = true;

function apretarInterruptor() {
  lamparaPrendida = !lamparaPrendida;
}
```

¡Ahora te toca a vos!

Definí el procedimiento `usarCierre` para que podamos abrir y cerrar el cierre de una mochila.

 Solución  Consola

```
1 let mochilaAbierta = true;
2
3 function usarCierre() {
4   mochilaAbierta = !mochilaAbierta;
5 }
```

 Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Jessica Saavedra bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# Negar no cuesta nada

Por el momento no parece una idea muy interesante, pero nos puede servir para reutilizar la lógica de una función que ya tenemos definida.

Por ejemplo, si contamos con una función `esPar`, basta con negarla para saber si un número es impar.

```
function esImpar(numero) {  
  return !esPar(numero);  
}
```

¡Ahora te toca a vos! Definí `esMayorDeEdad`, que recibe una edad, y luego `esMenorDeEdad` a partir de ella.

Solución Consola

```
1  
2 function esMayorDeEdad(edad){  
3   return edad >= 18;  
4 }  
5 function esMenorDeEdad(edad){  
6   return !esMayorDeEdad(edad);  
7 }
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Cada una de las funciones representa **un estado de dos posibles**: ser mayor o ser menor de edad. No se puede ser ambos al mismo tiempo y tampoco se puede evitar pertenecer a alguno de los dos grupos. Es decir, ¡siempre sos uno u otro!

Por eso decimos que son complementarios y que juntos forman el *conjunto universal*.

---

Esta guía fue desarrollada por Jessica Saavedra bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# Los peripatéticos

•

Otro de los operadores con el que ya te encontraste es la **conjunción lógica** (también llamada **y lógico**, o *and* por su nombre en inglés), que sólo **retorna verdadero** cuando todas las expresiones que opera son verdaderas.

Podemos encadenar varias de ellas mediante el operador `&&` y alcanza con que sólo una de ellas sea falsa para que toda la expresión resulte falsa.

Por ejemplo, si cuento con la función:

```
function esCantanteProlifico (cdsEditados, recitalesRealizados, graboAlgunDVD) {  
  return cdsEditados >= 10 && recitalesRealizados > 250 && graboAlgunDVD;  
}
```

y tenemos un cantante que no grabó un DVD, entonces no se lo considera **prolífico**, incluso aunque haya editado más de 10 CDs y dado más de 250 recitales.

Definé una función `esPeripatetico` que tome la profesión de una persona, su nacionalidad y la cantidad de kilómetros que camina por día. Alguien es peripatético cuando es un filósofo griego y le gusta pasear (camina más de 2 kilómetros por día). Ejemplo:

```
> esPeripatetico("filósofo", "griego", 5)  
true  
> esPeripatetico("profesor", "uruguayo", 1)  
false
```

 Solución  Consola

```
1 function esPeripatetico (profesion, nacionalidad, km_caminados) {  
2   return profesion === "filósofo" && nacionalidad === "griego" &&  
   km_caminados > 2;  
3 }
```

 Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Jessica Saavedra bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# La verdad detrás de la conjunción

En la lógica booleana, se puede definir el comportamiento de un operador con una *tabla de verdad* donde **A** y **B** son las expresiones o valores de verdad a ser operados y el símbolo  $\wedge$  representa la conjunción. Cada celda tiene una V si representa verdadero o F si representa falso.

Por ejemplo, supongamos que una casa consume poca energía si se usa el aire acondicionado a 24 grados y tiene al menos 5 lamparitas bajo consumo. Podemos representar las expresiones de la siguiente forma:

- **A**: En la casa se usa el aire acondicionado a 24 grados
- **B**: La casa tiene al menos 5 lamparitas bajo consumo
- **A  $\wedge$  B**: La casa consume poca energía

Como indicamos, la casa consume poca energía (**A  $\wedge$  B**) cuando tanto **A** como **B** son verdaderos. Esto se puede representar mediante la siguiente tabla de verdad:

A	B	A $\wedge$ B
V	V	V
V	F	F
F	V	F
F	F	F

En el mundo de la lógica estas expresiones se llaman *proposiciones*. Pero... ¿qué cosas pueden ser una proposición? Sólo hace falta que porten un valor de verdad, es decir, cualquier expresión booleana puede ser una proposición.

¿No nos creés? Probá en la consola la función `consumePocaEnergia`, que recibe una temperatura y una cantidad de lamparitas, y comprobá si se comporta como en la tabla:

- > consumePocaEnergia(24, 5)
- > consumePocaEnergia(24, 0)
- > consumePocaEnergia(21, 7)
- > consumePocaEnergia(18, 1)

&gt;



Esta guía fue desarrollada por Jessica Saavedra bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)







# ¡Juguemos al T.E.G.!

¿Y si basta con que una de varias condiciones se cumpla para afirmar que una expresión es verdadera? Podemos utilizar otro de los operadores que ya conocés, ¡la disyunción lógica!

Recordá que se lo representa con el símbolo `||` y también se lo conoce como el operador `or`.

En el famoso juego `T.E.G.`, un jugador puede ganar de dos formas: cumpliendo su objetivo secreto o alcanzando el objetivo general de conquistar 30 países.

```
function gano(cumplioObjetivoSecreto, cantidadDePaisesConquistados) {  
  return cumplimientoObjetivoSecreto || cantidadDePaisesConquistados >= 30;  
}
```

Probá en la consola las siguientes expresiones:

- `> gano(true, 25)`
- `> gano(false, 30)`
- `> gano(false, 20)`
- `> gano(true, 31)`

¿Te animás a construir la tabla de verdad de la disyunción lógica?

`>`



Igual, 4.0.

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# Y ahora... ¿quién podrá ayudarnos?

•

Nuestra amiga Dory necesitaba hacer algunos trámites en el banco, pero cuando llegó notó que estaba cerrado.

Para evitar que le ocurra nuevamente, vamos a definir una función que ayude a la gente despistada como ella.

Sabemos que el banco está cerrado cuando:

- Es feriado, o
- Es fin de semana, o
- No estamos dentro del horario bancario.

La función `dentroDeHorarioBancario` ya la definimos por vos: recibe un horario (una hora en punto que puede ir desde las 0 hasta las 23) y nos dice si está comprendido en la franja de atención del banco.

Definí las funciones `esFinDeSemana` y `estaCerrado`. Tené en cuenta que los días se reciben en minúscula:

```
> estaCerrado(false, "sábado", 10)
true //Porque es fin de semana

> estaCerrado(true, "lunes", 10)
true //Porque es feriado

> estaCerrado(false, "martes", 20)
true //Porque no está dentro del horario bancario

> estaCerrado(false, "jueves", 11)
false
```

 Solución

 Biblioteca

 Consola

```
1 function esFinDeSemana (dia) {
```



```
2   return dia === "sabado" || dia === "domingo";
3 }
4
5 function estaCerrado(esFeriado, dia, horario) {
6   return esFeriado || esFinDeSemana (dia) ||
7   !dentroDeHorarioBancario(horario) ;
8 }
```



 Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Jessica Saavedra bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# Claroscuro

Oli está diseñando una página web y se está centrando en el contraste de sus componentes. Para lograrlo, nos pidió que definamos la función `tieneContraste` que recibe como argumentos el color de la letra y el color del fondo de la página y retorna si la página tiene contraste.

Para empezar ya contamos con la función `esTonoClaro` que toma un color por parámetro y retorna si es claro.

```
> esTonoClaro('rojo')
false

> esTonoClaro('blanco')
true
```



¡Ahora te toca a vos! Definí la función `tieneContraste`. Para que la página tenga contraste tiene que tener el fondo claro y la letra no o bien tener la letra clara y el fondo no.

Solución Consola

```
1 function tieneContraste(color_fondo, color_letra) {
2   return (esTonoClaro(color_fondo) && !esTonoClaro(color_letra)) ||
3   (!esTonoClaro(color_fondo) && esTonoClaro(color_letra)) ;
}
```



Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Jessica Saavedra bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# La verdad es que no hay una verdad

Ahora pensemos cómo sería la tabla de verdad que representa el comportamiento de la función que acabás de hacer.

8. La verdad es que no hay una

La proposición es `esTonoClaro`, y el valor de verdad que porte dependerá de cada color que esté evaluando.

El booleano final resultará de operar estos colores mediante `tieneContraste`:

la letra tiene tono claro	el fondo tiene tono claro	tiene contraste
true	true	false
true	false	true
false	true	true
false	false	false

Probá tu función `tieneContraste` con los siguientes valores y comprobá si se comporta como la tabla:

- > `tieneContraste("amarillo", "beige")`
- > `tieneContraste("azul", "violeta")`
- > `tieneContraste("blanco", "negro")`

>



# ¡Hola! Mi nombre es Xor

•

Ahora cambiemos las proposiciones `la letra tiene tono claro` y `el fondo tiene tono claro` por proposiciones genéricas **A** y **B**. Además, representemos la operación que realiza `tiene contraste` con el símbolo  $\vee$ . Lo que obtenemos es... ¡una nueva tabla!

A	B	$A \vee B$
V	V	F
V	F	V
F	V	V
F	F	F

Este comportamiento existe como un operador dentro de la lógica y se lo denomina `xor` o disyunción lógica excluyente.

A diferencia del `and`, `or` y `not`, el `xor` no suele estar definido en los lenguajes. Sin embargo, ahora que sabés cómo funciona, si alguna vez lo necesitás podés definirlo a mano.

Veamos si se entiende: definí la función genérica `xor`, que tome dos booleanos y retorne el valor de verdad correspondiente.

 Solución  Consola

```
1 function xor (AAA, BBB) {  
2   return (AAA && !BBB) || (!AAA && BBB) ;  
3 }  
4
```





 Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Jessica Saavedra bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# Precedencia

Cuando una expresión matemática tiene varios operadores, sabemos que las multiplicaciones y divisiones se efectuarán antes que las sumas y las restas:

```
5 * 3 + 8 / 4 - 3 = 14
```

Al igual que en matemática, cuando usamos operadores lógicos las expresiones se evalúan en un orden determinado llamado *precedencia*.

¿Cuál es ese orden? ¡Hagamos la prueba!

Teniendo definida la siguiente función, según la cual las tarjetas de débito ofrecen una única cuota, y las de crédito, seis:

```
function pagaConTarjeta(seCobraInteres, tarjeta, efectivoDisponible) {  
  return !seCobraInteres && cuotas(tarjeta) >= 3 || efectivoDisponible < 100;  
}
```

Probala en la consola con los valores:

- > pagaConTarjeta(true, "crédito", 320)
- > pagaConTarjeta(false, "crédito", 80)
- > pagaConTarjeta(true, "débito", 215)
- > pagaConTarjeta(true, "débito", 32)

>



---

Esta guía fue desarrollada por Jessica Saavedra bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# Un ejercicio sin precedentes

Si prestaste atención a la función anterior, habrás notado que la operación con mayor precedencia es la negación `!`, seguida de la conjunción `&&` y por último la disyunción `||`. ¿Pero qué pasa si quiero alterar el orden en que se resuelven?

Al igual que en matemática, podemos usar paréntesis para agrupar las operaciones que queremos que se realicen primero.

Delfi se puede concentrar cuando programa y toma infusiones, pero no cualquier infusión. Tiene que ser mate a exactamente 80°C o té que esté a por lo menos 95°C.

Definé la función `sePuedeConcentrar` que recibe una bebida, su temperatura y un booleano que nos dice si Delfi está programando:

```
> sePuedeConcentrar('té', 100, true)
true

> sePuedeConcentrar('mate', 70, true)
false

> sePuedeConcentrar('té', 95, false)
false
```

¡Intentá resolverlo en una única función! Después vamos a ver cómo quedaría si delegamos.

💡 ¡Dame una pista!

 Solución  Consola

```
1 function sePuedeConcentrar(bebida, temp, esta_prog) {
2   return (bebida=="mate" && temp === 80 && esta_prog) ||
   (bebida=="té" && temp >= 95 && esta_prog);
3 }
4
```

 Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

¿Y si delegamos? Podríamos separar la lógica de la siguiente manera:

```
function sePuedeConcentrar(infucion, temperatura, estaProgramando) {  
  return infucionATemperaturaCorrecta(infucion, temperatura) && estaProgramando;  
}
```



**Al delegar correctamente**, hay veces en las que no es necesario alterar el orden de precedencia, ¡otro punto a favor de la delegación!

Esta guía fue desarrollada por Jessica Saavedra bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# ¿Puedo subir?

En un parque de diversiones de la ciudad instalaron una nueva montaña rusa y nos pidieron ayuda para que le digamos a las personas si pueden subirse o no antes de hacer la fila. Los requisitos para subir a la atracción son:

- Alcanzar la altura mínima de 1.5m (o 1.2m si está acompañada por una persona adulta)
- No tener ninguna afección cardíaca

Definé la función de 3 parámetros `puedeSubirse` que recibe una altura de una persona en metros, si está acompañada y si tiene alguna afección cardíaca. Ejemplo:

```
> puedeSubirse(1.7, false, true)
false // no puede subirse
      // porque aunque tiene mas de 1.5m,
      // tiene una afección cardíaca
```

Solución Consola

```
1 function puedeSubirse(altura, acompa, afec_card) {
2   return (altura>=1.50 && !afec_card) || (altura>=1.20 && acompa &&
3   !afec_card);
4 }
```

Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Jessica Saavedra bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)

