

EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: EL OPERADOR TERNARIO Y SU USO EN JAVASCRIPT.
- EXERCISE 2: EVALUANDO CONDICIONES USANDO AND Y OR.

EXERCISE 1: EL OPERADOR TERNARIO Y SU USO EN JAVASCRIPT

El operador ternario nos permite tener una funcionalidad que no podemos lograr con ninguna otra estructura en JavaScript. Ésta se puede asemejar con un ciclo **if**, pero dicho operador alcanza un resultado distinto, con menos líneas de código. En primer lugar, debemos conocer lo siguiente: ¿cuál es la estructura de un operador ternario? A modo de comparación, mostraremos un ciclo **if**, y a continuación veremos un operador ternario para notar cómo discrepan en cuanto a sus sintaxis:

```
1 var variable;  
2  
3 //Ciclo if  
4 if (condicion) {  
5     //Si la condición se cumple se ejecuta esto:  
6     variable = accionVerdadero;  
7 } else {  
8     //Si la condición NO se cumple se ejecuta esto:  
9     variable = accionFalso;  
10 }  
11  
12 //Operador ternario  
13 variable = condicion ? accionVerdadero : accionFalso;
```

“Ternario” quiere decir “tres partes”; y las podemos definir así:

Una **condición** seguida de (?), luego una **expresión para ejecutar si la condición es verdadera** (accionVerdadero) seguida de dos puntos (:), y finalmente la **expresión para ejecutar si la condición es falsa** (accionFalso).

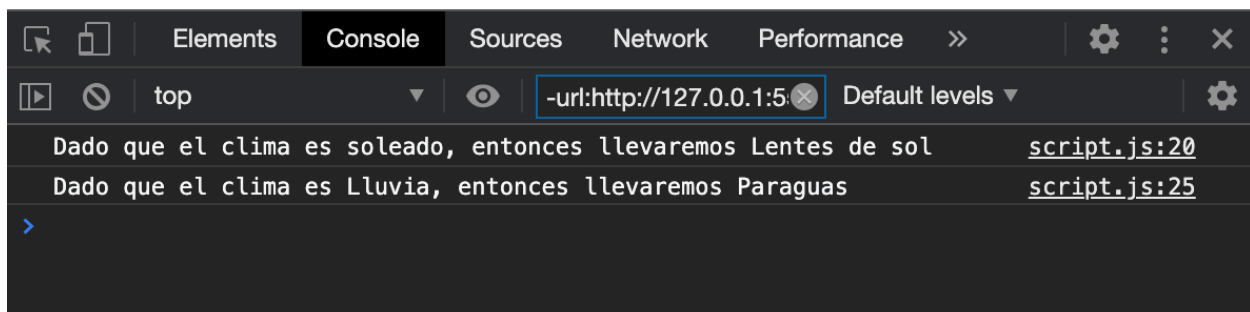
Por ejemplo, supongamos que una aplicación del clima tiene la funcionalidad de sugerirnos usar ciertos accesorios, como sombreros o paraguas, basándose en la condición que se presente. Mediante un

operador ternario podemos evaluar la condición de “*qué accesorio usar*”, dependiendo del “*clima afuera*”.
Establecemos nuestras variables accesorios y clima:

```
1 var accesorio;  
2 var clima = 'soleado';  
3  
4 // Cuando si se cumple la condición:  
5 accesorio = (clima === 'soleado') ? 'Lentes de sol' : 'Paraguas';  
6 console.log(`Dado que el clima es ${clima}, entonces llevaremos  
7 ${accesorio}`);  
8  
9 // Cuando no se cumple la condición:  
10 clima = 'Lluvia';  
11 accesorio = (clima === 'soleado') ? 'Lentes de sol' : 'Paraguas';  
12 console.log(`Dado que el clima es ${clima}, entonces llevaremos  
13 ${accesorio}`);
```

Entonces, ya con nuestro primer operador ternario, evaluamos lo siguiente: si el clima es soleado, esta aplicación nos recomendará utilizar lentes de sol; y de no ser así, usar un paraguas. Como en este caso la variable clima es soleado, se efectuará la condición verdadera, que retornará el accesorio de lentes de sol. En cuanto al segundo operador ternario, la variable clima toma el valor de lluvia, por lo que no cumple la condición evaluada; y por esta razón, la respuesta del operador ternario es que sugiera usar un paraguas, que si es el valor falso dentro de la expresión.

El resultado por consola, cuando se cumple la condición y no, es el siguiente:



La primera línea de la consola muestra el resultado cuándo es verdadero, y éste evalúa que la condición si se cumple; mientras que la segunda línea, permite ver cuando no se cumple la condición, haciendo que la variable “accesorio” tome el valor de “Paraguas”.

Revisado esto, ¿se considera que el operador ternario es similar a un `if`, o no?: independientemente de si existe una similitud, dicho operador ternario nos permite evaluar condiciones que pueden ampliar la funcionalidad de nuestros programas en JavaScript.

EXERCISE 2: EVALUANDO CONDICIONES USANDO AND Y OR

En el ejemplo anterior, se consideró el uso de un operador ternario al momento de evaluar condiciones. Aunado a ello, existen otras estructuras que pueden ayudar a efectuar esa labor con más precisión, al momento de evaluar las condiciones dentro de un bucle o ciclo; y son los operadores lógicos AND y OR.

¿Qué son operadores lógicos?: palabras o símbolos que establecen una relación entre dos variables, para evaluar si esta relación establecida retorna un valor booleano verdadero o falso. Si se piensa en pseudo código, **AND** y **OR** cobran la siguiente funcionalidad:

Si “a” y (**AND**) “b” cumplen sus requisitos, retornará verdadero. Aquí ambos deben cumplir para retornar un **true**.

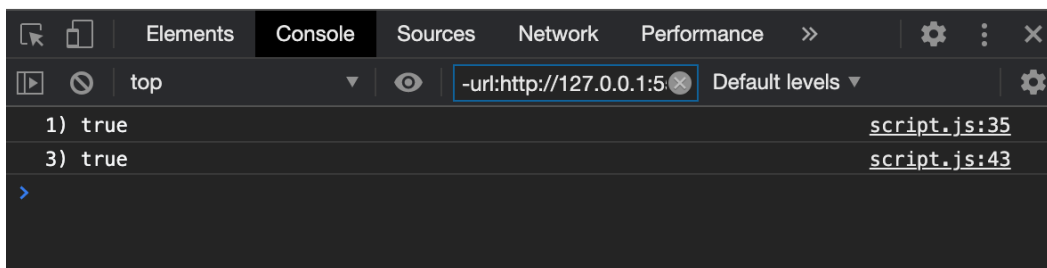
Si “a” o (**OR**) “b” cumplen sus requisitos, retornará verdadero. En este caso, al menos uno debe cumplir para retornar un **true**.

El operador lógico **OR** establece que una expresión siempre resultará ser verdadera, cuando uno de los dos argumentos lo es. Se puede ver en el siguiente ejemplo:

```
1 var verdadero = 5 < 10;
2 var falso = 5 < -10;
3
4 // Se cumple al menos una condición:
5 if (falso || verdadero) {
6     console.log("1" +(falso || verdadero));
7 }
8 // No se cumple ninguna condición (Ningún argumento es verdadero):
9 if (falso || falso) {
10     console.log("2" +(falso || falso));
11 }
```

```
12
13 // Se cumplen ambas condiciones:
14 if (verdadero || verdadero) {
15     console.log("3) " + (verdadero || verdadero));
16 }
17
18 //La condición siempre se cumplirá si UNO de los dos argumentos es
19 verdadero.
```

Como muestran estos 3 ciclos ir, el operador lógico **OR** siempre se cumplirá si solamente uno de los argumentos es verdadero. El resultado de este código por consola es el siguiente:



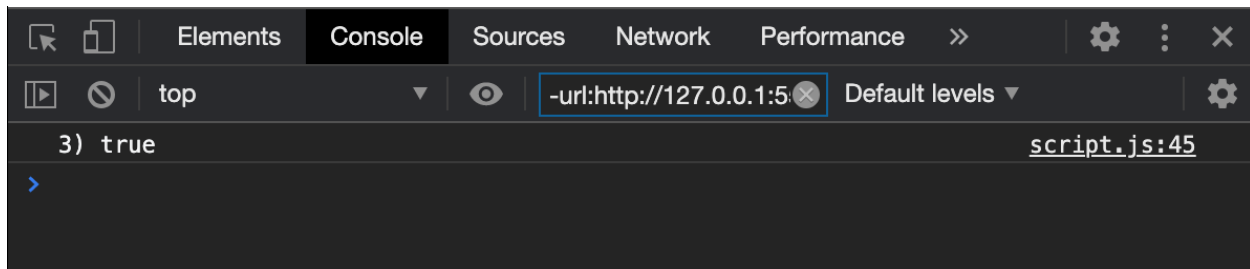
Se puede notar, que la consola solo mostró los mensajes de las condiciones que se cumplían con el operador lógico OR, los cuales fueron el primer y tercer condicional.

El operador lógico **AND**, establece que una expresión siempre resultará ser verdadera cuando los dos argumentos cumplen con la condición, retornando verdadero. Se puede apreciar este comportamiento reemplazando el operador **OR** del ejemplo anterior, por el operador **AND**:

```
1 var verdadero = 5 < 10;
2 var falso = 5 < -10;
3
4 // Se cumple al menos una condición:
5 if (falso && verdadero) {
6     console.log("1) " + (falso && verdadero));
7 }
8 // No se cumple ninguna condición (Ningún argumento es verdadero):
9 if (falso && falso) {
10     console.log("2) " + (falso && falso));
11 }
```

```
12
13 // Se cumple al menos una condición:
14 if (verdadero && verdadero) {
15     console.log("3) " + (verdadero && verdadero));
16 }
17
18 //La condición solo se cumplirá si LOS DOS argumentos son verdaderos.
```

Como este operador demanda que ambas variables cumplan con la condición, la consola solo mostró el mensaje del condicional que tenía ambas variables verdaderas:

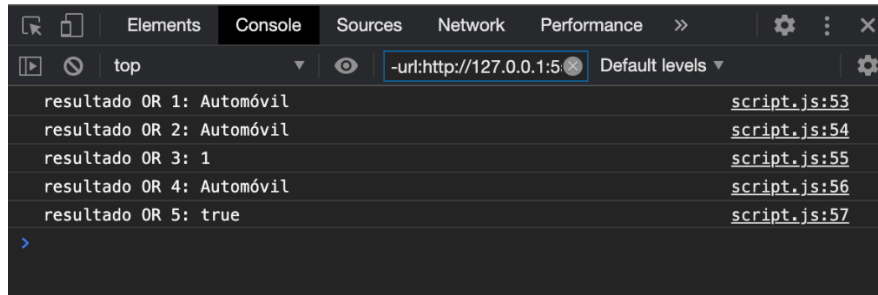


Tal como se ve, el operador lógico **AND** establece que la condición sólo se va a cumplir si los dos argumentos son verdaderos.

Hasta el momento, hemos revisado el uso de estos operadores con variables de tipo booleano, pero ¿qué sucede si introducimos como argumentos, valores que no son necesariamente de este tipo? Tomemos por ejemplo el siguiente caso:

```
1 // OR: Siempre retornará valor Truthy
2 console.log("resultado OR 1: " + (false || 'Automóvil'))
3 console.log("resultado OR 2: " + (0 || 'Automóvil'))
4 console.log("resultado OR 3: " + (1 || 'Automóvil'))
5 console.log("resultado OR 4: " + ('Automóvil' || 1))
6 console.log("resultado OR 5: " + (true || 'Automóvil'))
```

En consola, el resultado de todos estos **console.log()** es:



```

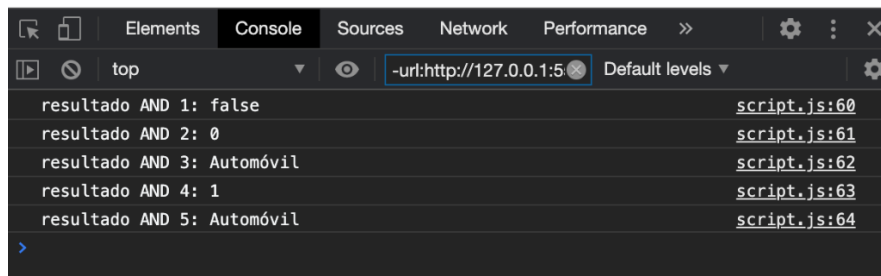
Elements  Console  Sources  Network  Performance  >>
top  -url:http://127.0.0.1:5... Default levels
resultado OR 1: Automóvil script.js:53
resultado OR 2: Automóvil script.js:54
resultado OR 3: 1 script.js:55
resultado OR 4: Automóvil script.js:56
resultado OR 5: true script.js:57
>
  
```

Y si reemplazamos el operador **OR** con **AND**:

```

1 // AND: Siempre retornará valor falsy
2 console.log("resultado AND 1: " + (false && 'Automóvil'))
3 console.log("resultado AND 2: " + (0 && 'Automóvil'))
4 console.log("resultado AND 3: " + (1 && 'Automóvil'))
5 console.log("resultado AND 4: " + ('Automóvil' && 1))
6 console.log("resultado AND 5: " + (true && 'Automóvil'))
  
```

En consola vemos lo siguiente:



```

Elements  Console  Sources  Network  Performance  >>
top  -url:http://127.0.0.1:5... Default levels
resultado AND 1: false script.js:60
resultado AND 2: 0 script.js:61
resultado AND 3: Automóvil script.js:62
resultado AND 4: 1 script.js:63
resultado AND 5: Automóvil script.js:64
>
  
```

Como se puede notar, en las anotaciones del código definimos el comportamiento de estos dos operadores lógicos, al decir que **OR** siempre retornará un valor Truthy, y **AND** un valor Falsy.

¿A qué se refieren Truthy y Falsy? Un valor “verdadero” (Truthy), es aquel que se considera verdadero en un contexto booleano. En cambio, los valores “falsos” (Falsy) son el inverso: aquel que se considera falso en un contexto booleano. Los valores Falsy que existen son: **false**, **0**, **-0**, **0n**, **""**, **null**, **undefined** y **NaN**.



Tanto estos operadores lógicos, como el operador ternario que analizamos previamente, ayudan a precisar la evaluación de condiciones, y permite seguir incrementando las funcionalidades de nuestro código JavaScript.