

HINTS

DIFERENCIAS ENTRE INTERPRETADO O COMPLICADO

De forma general, la diferencia entre los **lenguajes compilados** e **interpretados** es que los primeros usan un compilador para poder traducirlo y ejecutar el programa, mientras que los segundos requieren de un intérprete que traduzca el código al momento de la ejecución.

LEER CÓDIGO

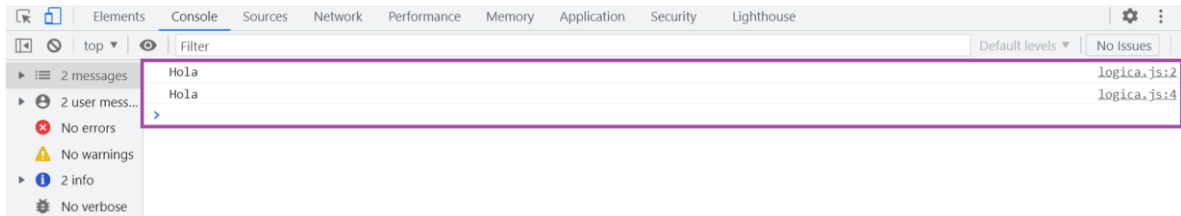
Leer código es muy bueno para aprender a comprenderlos. Cuando comenzamos a escribir código se puede hacer fácil leer nuestro propio código, pero muy difícil leer el código escrito por otra persona. Mientras más código leamos más fácil será ir comprendiéndolo.

JAVASCRIPT CON Y SIN PUNTO Y COMA

Cuando escribimos código **JavaScript** terminamos las instrucciones colocando un punto y coma (;), pero, en este lenguaje, si no colocamos el punto y coma, el código compilará de la misma forma:

```
1 //CON PUNTO Y COMA
2 console.log("Hola");
3 //SIN PUNTO Y COMA
4 console.log("Hola")
```

Cuando revisamos la consola obtenemos como resultado:



OPERADORES DE ASIGNACIÓN COMPUESTOS

Hasta el momento hemos trabajado con el operador de asignación igual (=) con el que asignamos un valor a una variable.

```
1 var numero = 1;
```

Y si queremos sumarle un valor (o restarle, dividirlo o multiplicarle un valor), realizamos la siguiente acción:

```
1 numero = 1 + 2;
```

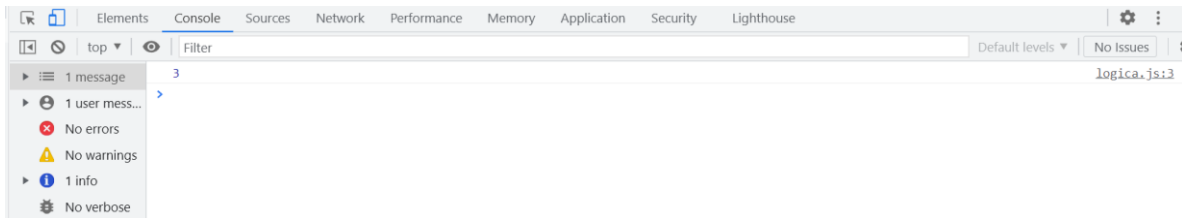
Sin embargo, existen los operadores de asignación compuestos que utiliza el operador de asignación combinado con un operador aritmético y de esta forma abrevia o reduce ciertas expresiones, así podemos, al mismo tiempo, realizar la operación y la asignación al mismo tiempo.

```
1 numero += 2;
```

Imprimamos ambas posibilidades.

```
1 var numero = 1;  
2 numero = 1 + 2;  
3 console.log(numero);
```

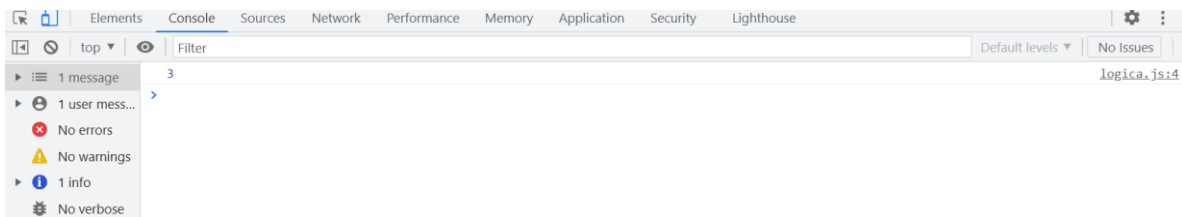
Nos muestra:



Imprimamos la segunda posibilidad:

```
1 var numero = 1;  
2 //numero = 1 + 2;  
3 numero += 2;  
4 console.log(numero);
```

Dando como resultado:



Los operadores de asignación compuestos son:

Operador	Ejemplo	Expresión equivalente
<code>+=</code>	<code>a += b;</code>	<code>a = a + b</code>
<code>-=</code>	<code>a -= b;</code>	<code>a = a - b</code>
<code>*=</code>	<code>a *= b;</code>	<code>a = a * b</code>
<code>/=</code>	<code>a /= b;</code>	<code>a = a / b</code>
<code>%=</code>	<code>a %= b;</code>	<code>a = a % b</code>

OPERADORES LÓGICOS

Cuando trabajamos evaluando condicionales, podemos, en ocasiones, tener la necesidad de evaluar más de una condición al mismo tiempo. Los operadores lógicos son:

OPERADOR	SIGNIFICADO
&&	Operador lógico and o y. Las dos o más condiciones deben ser true para que la evaluación sea true.
 	Operador lógico or u o. Una de las condiciones evaluadas debe ser true para que la evaluación sea true.
!	Operador lógico not o no. Evalúa la condición y niega el resultado. Si es true, es false.

Un ejemplo de su uso será el siguiente:

```
1 var numero = 5;
2
3 if( numero > 2 && numero < 6){
4     console.log("el número es correcto");
5 }
```

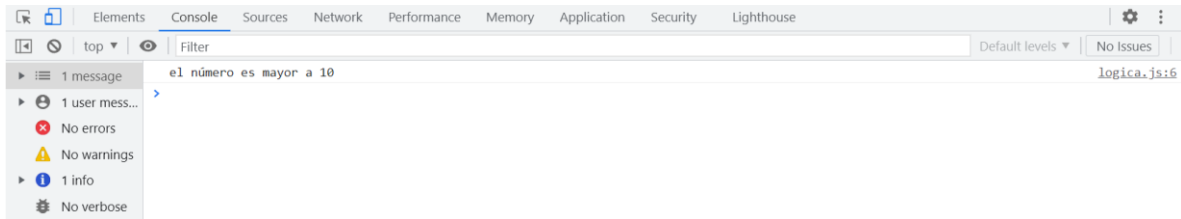
Para este caso, el número evaluado debe ser mayor a 2 y menor a 6 para que se considere *true*.

ELSE IF

Al trabajar con la condicional **if**, en ocasiones, posterior a la evaluación realizada, requeriremos realizar otra evaluación, para eso existe la condición **else if** (entonces sí), la cual se utiliza de la siguiente manera:

```
1 var numero = 15;
2
3 if (numero < 2) {
4     console.log("el número menor a dos");
5 } else if (numero > 10) {
6     console.log("el número es mayor a 10");
7 } else {
8     console.log("el número es distinto");
9 }
```

De esta forma, el flujo ingresa al **if** y como no cumple la condición, pasa a la siguiente evaluación **else if**. Al cumplirla ingresa ahí. Si no la cumpliera ingresaría al **else**.



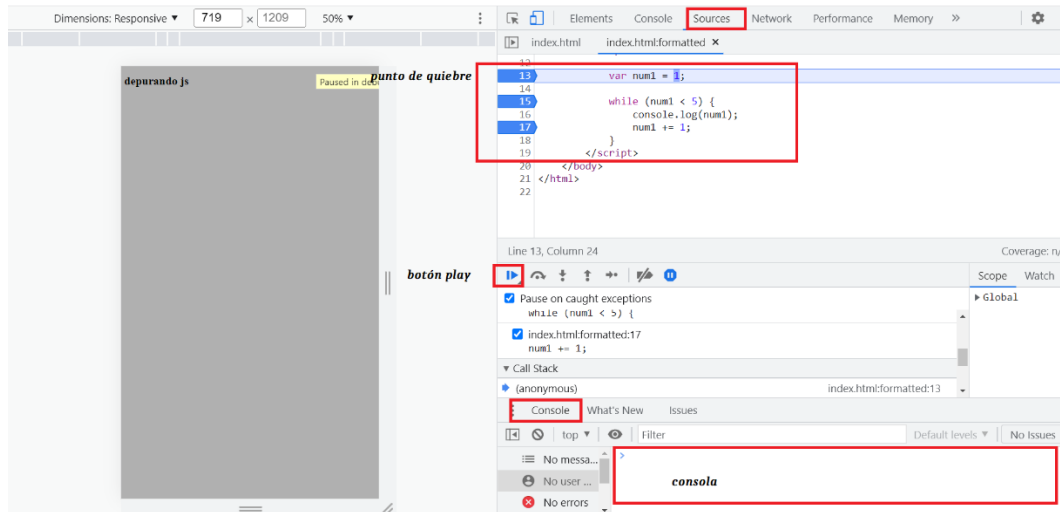
DEPURACIÓN EN CONSOLA

En ocasiones necesitaremos ver el flujo de trabajo de nuestro proyecto, excepciones u otras cosas. Una herramienta muy practica para esto es el uso de depuración en la consola. Supongamos que no tenemos claridad en como funciona un ciclo While. Podemos verlo en acción con la depuración.

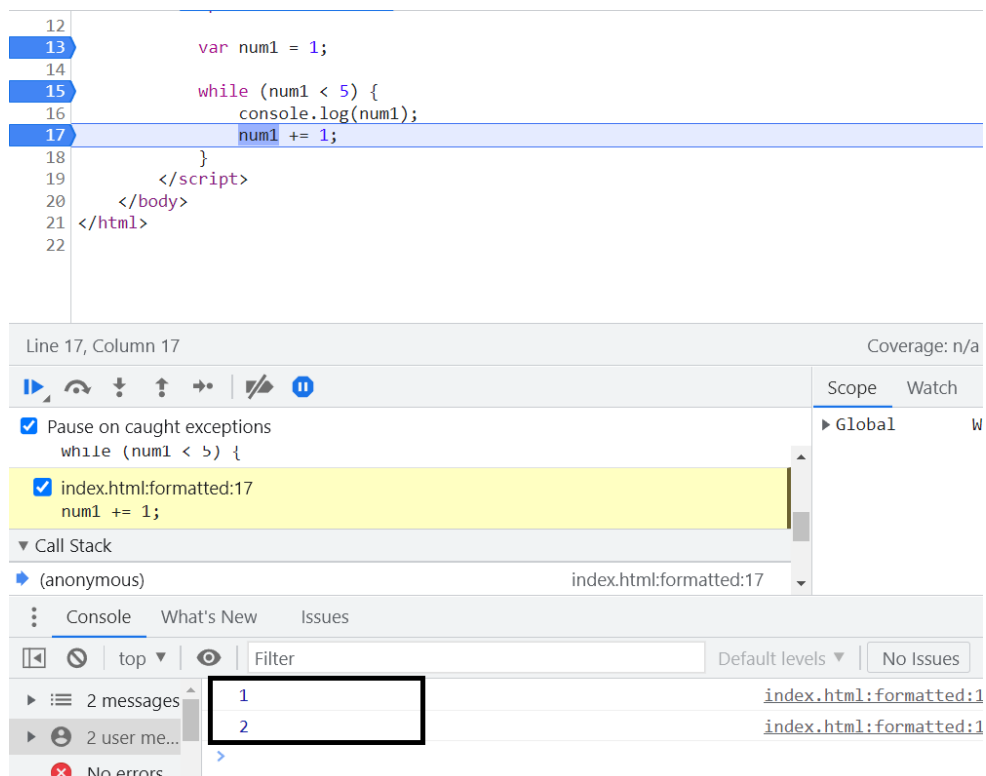
```
1 var num1 = 1;
2
3     while(num1 < 5) {
4         console.log(num1);
5         num1 +=1;
6     }
```

En la consola del navegador, ingresaremos al apartado source y veremos el código, en el cual podemos incluir puntos de quiebre, es decir, puntos en los cuales la ejecución se detendrá para que la veamos en detalle.

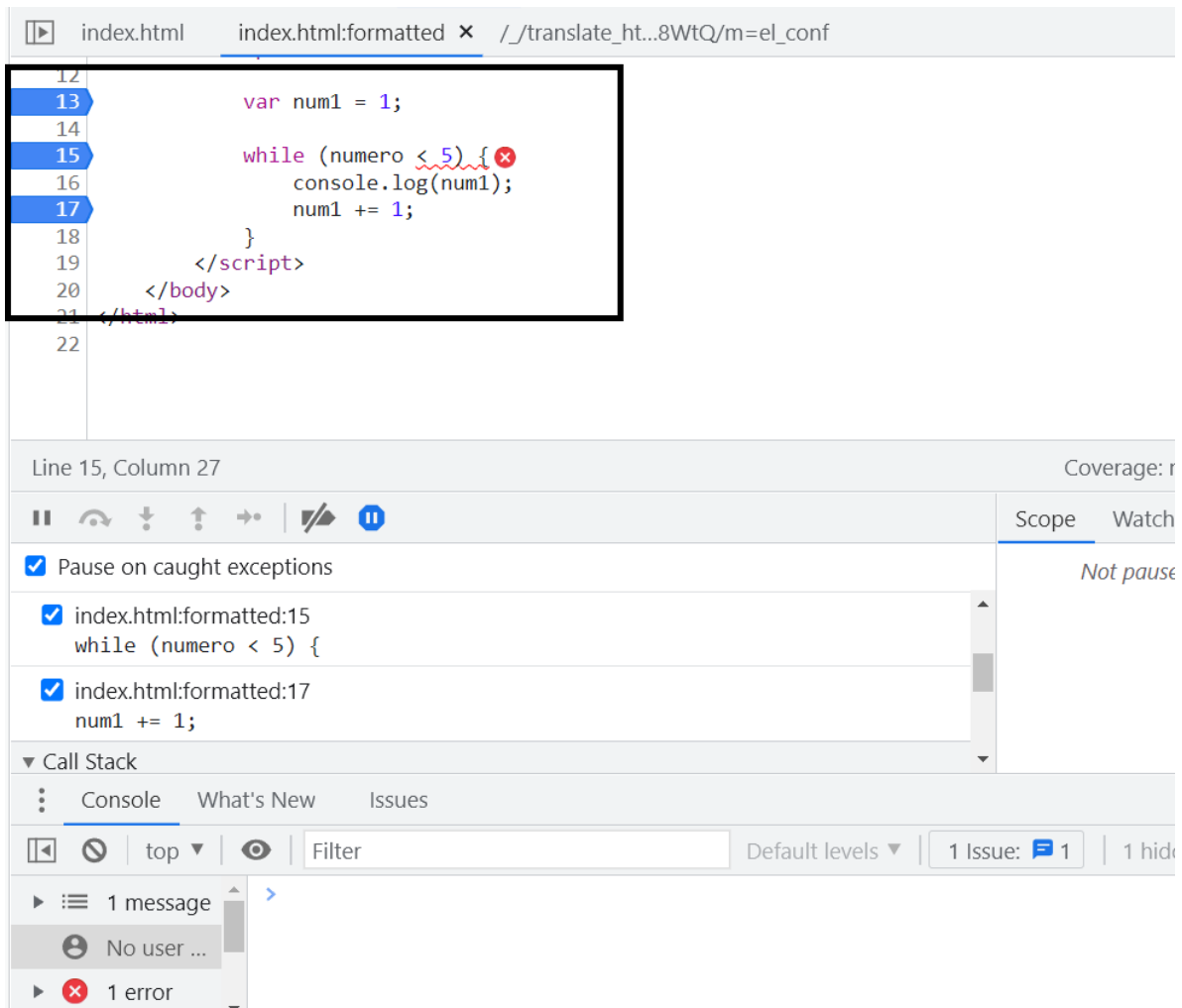
Utilizando el botón de Play, podremos avanzar paso a paso.



Si movemos este ejercicio, con el botón play, iremos viendo el flujo de trabajo.



Mientras que, si nuestro programa tuviera un error que puede provocar su caída, podemos verlo en este mismo lugar para encontrarlo y solucionarlo.



The screenshot shows a web browser's developer console with the following components:

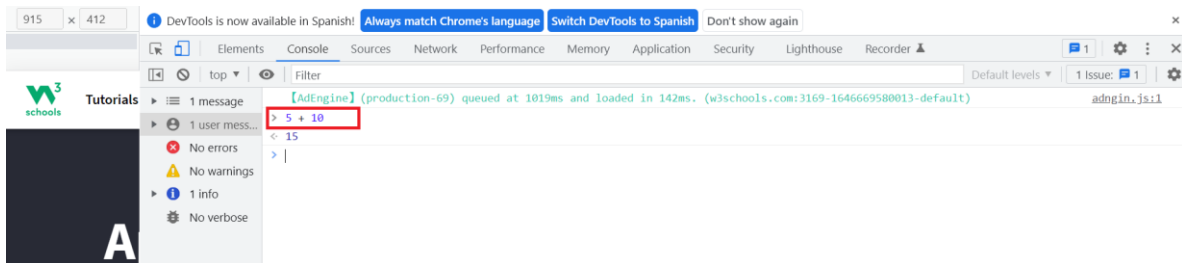
- Code Editor:** Displays the following JavaScript code:

```
12  
13     var num1 = 1;  
14  
15     while (numero < 5) {  
16         console.log(num1);  
17         num1 += 1;  
18     }  
19 </script>  
20 </body>  
21 </html>  
22
```

Line 15, column 27 is highlighted with a red squiggly line and a red 'x' icon, indicating an error.
- Breakpoints:** A list of breakpoints is shown, including:
 - ☒ Pause on caught exceptions
 - ☒ index.html:formatted:15 while (numero < 5) {
 - ☒ index.html:formatted:17 num1 += 1;
- Call Stack:** A section titled "Call Stack" is visible.
- Console:** A section titled "Console" is visible, showing a list of messages and errors. The "1 error" section is expanded, showing a message from "No user ...".

ESCRIBIR CODIGO EN LA CONSOLA

JavaScript es un lenguaje que nos permite practicar directamente en la consola sin aun desarrollar algoritmos en un archivo de extensión JS. Para eso, debemos dirigirnos a la consola del navegador y escribir, por ejemplo, una suma:



Esta suma la escribimos directamente en la consola y al hacer Enter, obtuvimos el resultado, sin declarar variables, sin usar un editor de texto, sin utilizar un archivo JS.

Esto nos puede servir para practicar código.