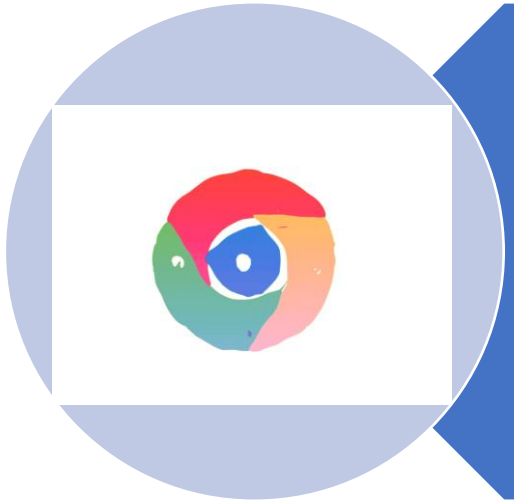
Jorge García Flores

Desarrollo Web en Entorno Cliente

Tema 2



Ejecución y Depuración en el navegador


Ubicación del código
Acceso a la consola
Content
Consola
Breakpoints

Ubicación de código

El código JS se puede ubicar directamente en una página html o en un fichero externo.

Normalmente el fichero externo tendrá extensión `.js`, pero no influye en su ejecución y podría no tenerla, aunque se considera buena práctica de programación ponerla.

Se deben usar rutas relativas, teniendo cuidado con los nombres de carpetas y ficheros porque existen muchos de los servidores web corren sobre Linux.

El código JS, dentro del fichero html, se puede incluir dentro de las etiquetas `head` o `body`.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <!-- AQUI SE PUEDE PONER CODIGO JS -->
    <script src="js/ejemplo.js"></script>
    <script>console.log('ejemplo')</script>
</head>
<body>
    <!-- AQUI SE PUEDE PONER CODIGO JS -->
    <script src="js/ejemplo.js"></script>
    <script>console.log('ejemplo')</script>
</body>
</html>
```

Ubicación de código

En cualquier caso, se recomienda ubicar el código JS en un fichero aparte, con extensión `.js` de tal forma que se pueda reutilizar, y llamarlo justo al final del bloque body.

El motivo de ubicarlo al "final" y no al "principio" de la página web, se debe a que, si se hace al contario, colocándolo al principio, y es un código pesado que tarda en ejecutarse, el usuario no "verá nada" porque el navegador web detendrá el código html hasta que se haya ejecutado el código.

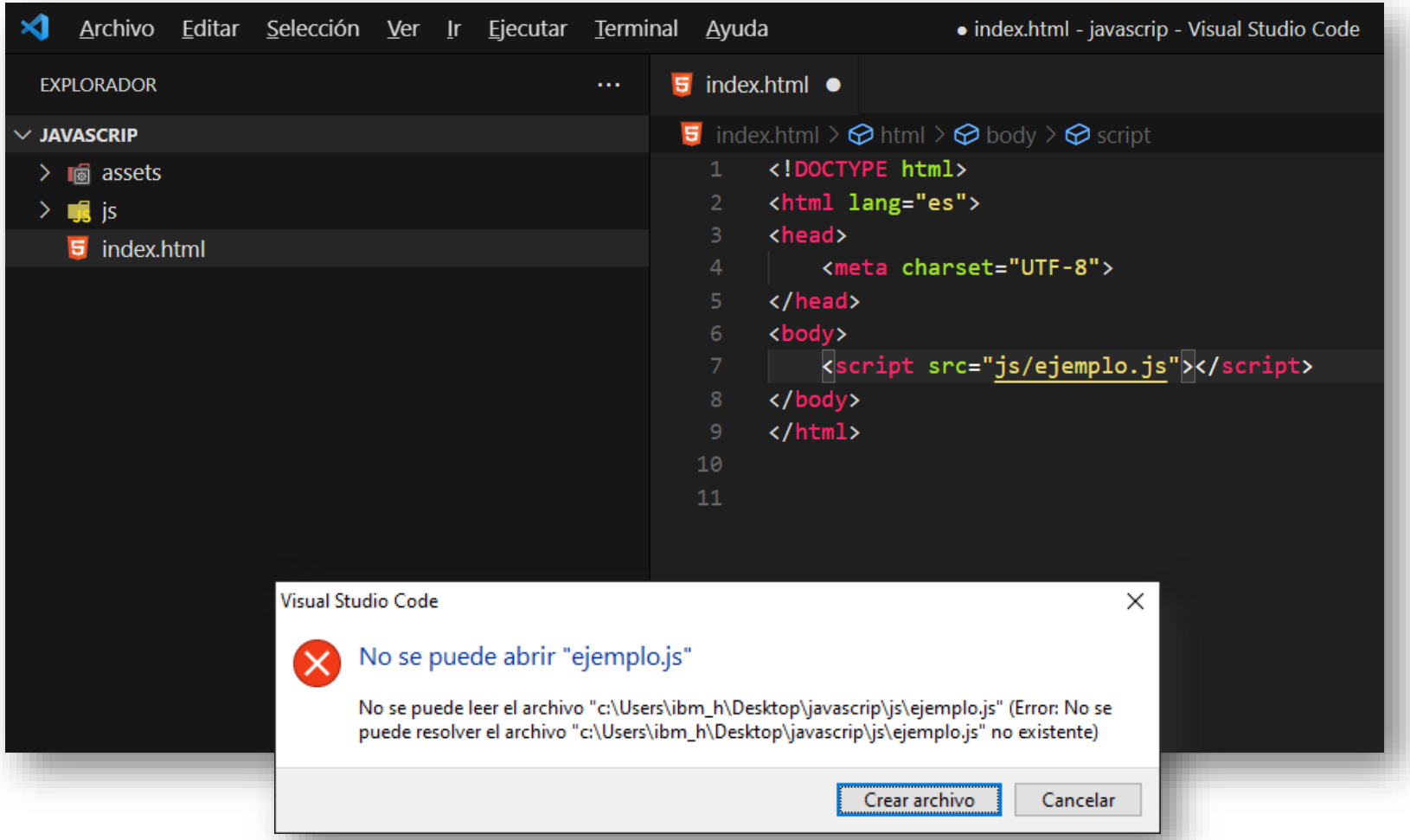
Resultando en una mala experiencia de usuario (lentitud en la carga de nuestra página web).

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <!-- AQUI SE PUEDE PONER CODIGO JS -->
    <script src="js/ejemplo.js"></script>
    <script>console.log('ejemplo')</script>
</head>
<body>
    <!-- AQUI SE PUEDE PONER CODIGO JS -->
    <script src="js/ejemplo.js"></script>
    <script>console.log('ejemplo')</script>
</body>
</html>
```

Ubicación de código

VS Code permite crear rápidamente los ficheros JS que se incluyen en la página html en el caso de que no exista.

Para ello solo se debe hacer *Ctrl+clic* sobre el nombre del fichero y responder "*Crear archivo*" en el cuadro de dialogo.



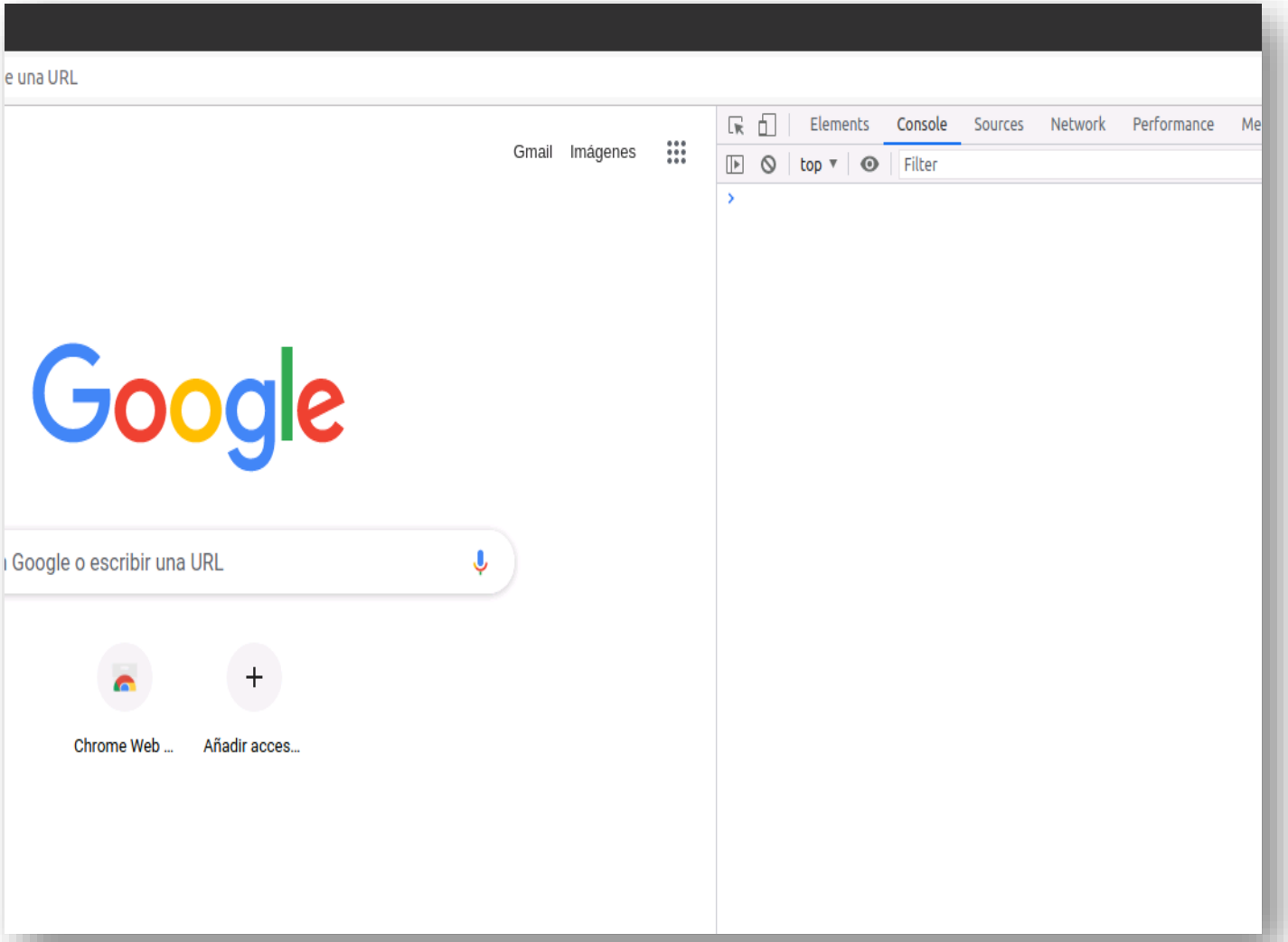
Acceso a la consola

Google Chrome

Atajo de teclado:
Windows/Linux: *Ctrl+Shift+J*
MacOS: *Comando+Opción+J*

Método del menú:
Clic en los tres puntos de la esquina superior derecha:
Más Herramientas-> Herramientas para desarrollador

Cuando la nueva pantalla se abra, seleccionar la pestaña "Consola" en la parte superior.



Acceso a la consola

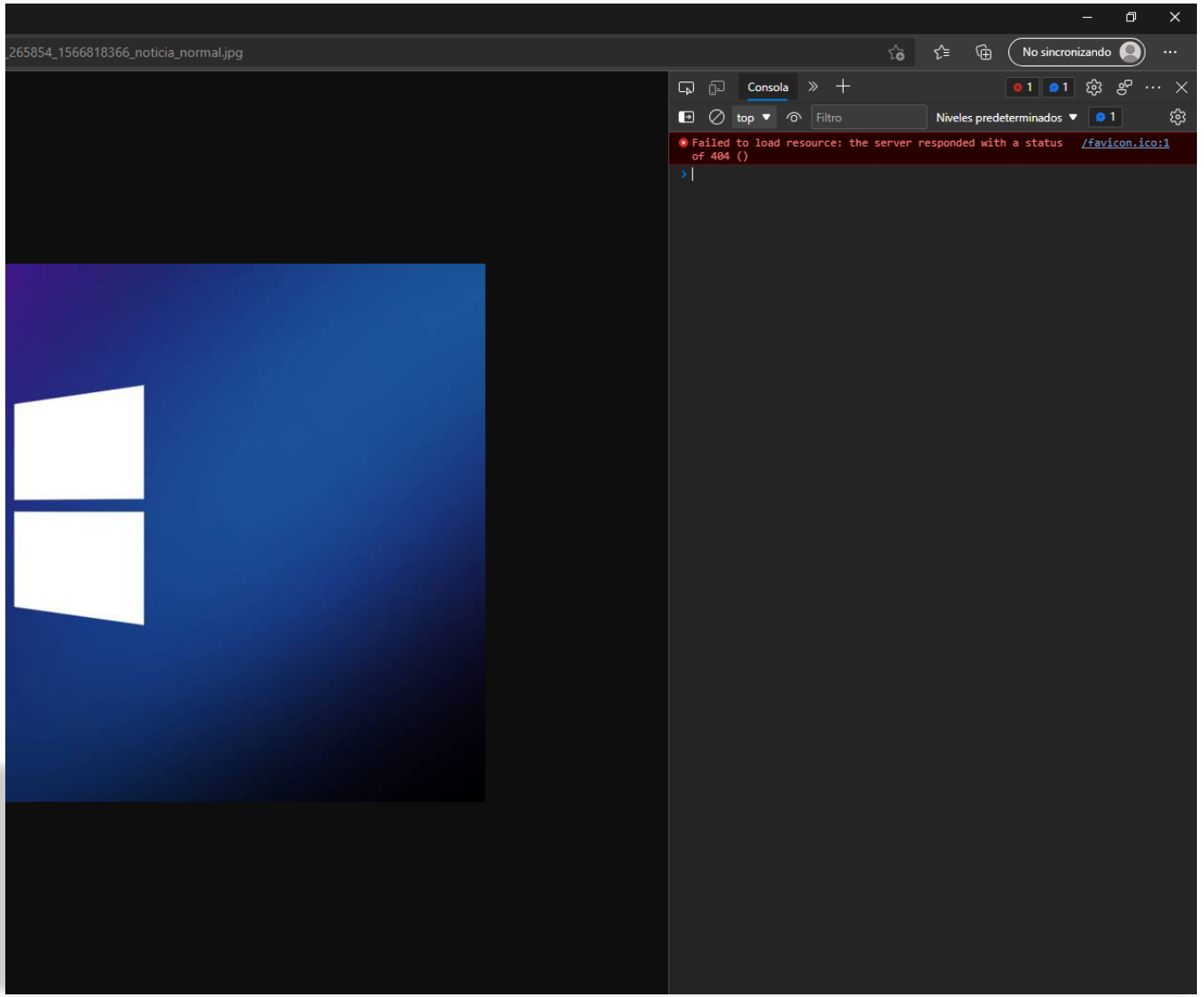
Internet Explorer y Edge

Atajo de teclado:
Tecla *F12* y luego seleccionar la pestaña *Consola*

Método del menú:
Clic en el botón en la esquina superior derecha del navegador:
Más Herramientas-> Herramientas de desarrollo

Safari

Atajo de teclado:
Comando+Opción+C

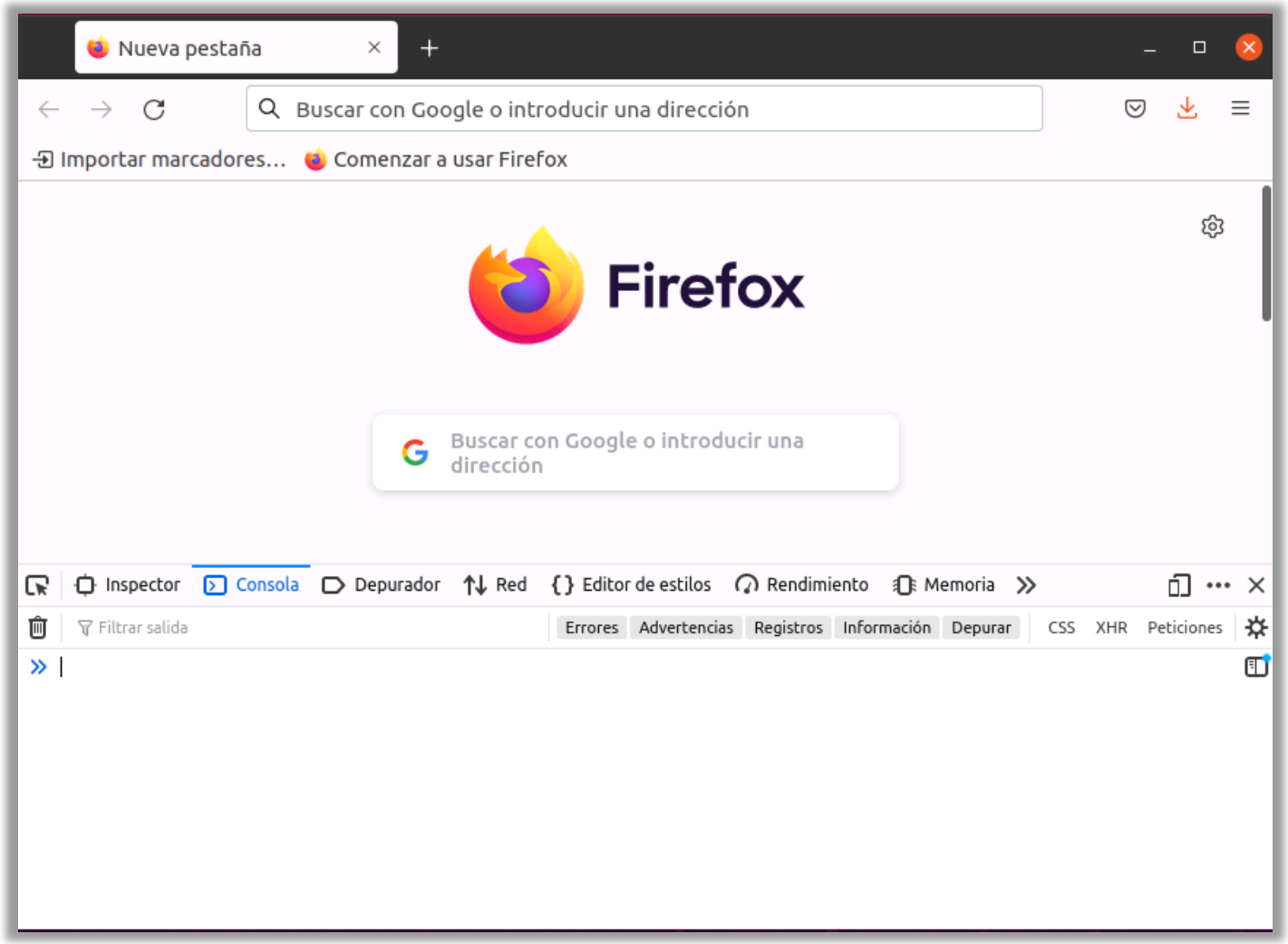


Acceso a la consola

Mozilla Firefox

Atajo de teclado:
Windows/Linux: *Control+Shift+K*
MacOS: *Comando+Opción+K*

Método del menú:
Desde el botón en la esquina superior derecha del navegador:
Más Herramientas-> Consola del desarrollador



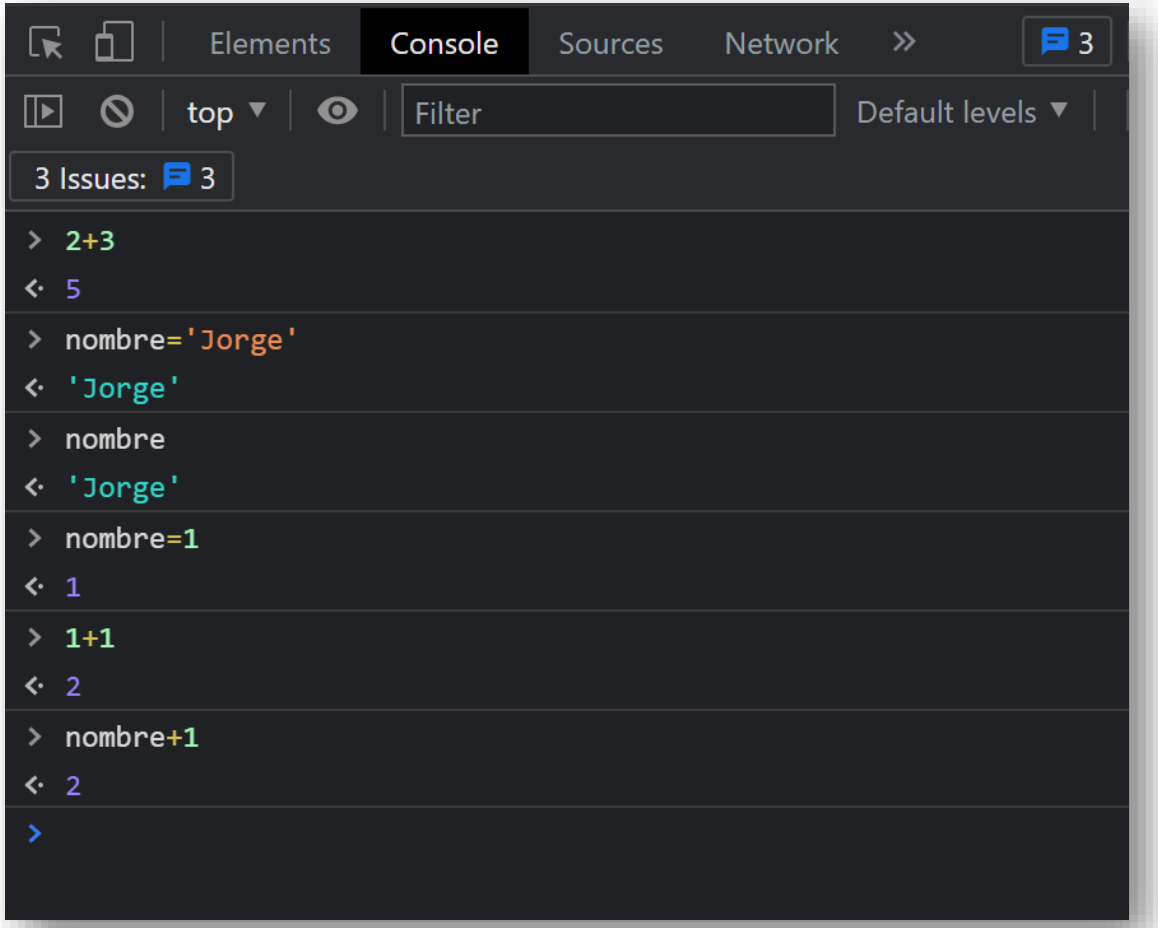
Consola

La consola de desarrollador de los navegadores puede ayudar a encontrar errores en JavaScript o inspeccionar los elementos del HTML.

Normalmente, la ejecución de un código JS se hace secuencialmente, ejecutando instrucción a instrucción (aunque puede haber bucles y otras características del lenguaje, como llamadas funciones).

La consola del navegador puede resultar útil para mostrar valores de variables o mensajes que indiquen si se ha ejecutado o no una parte del código.

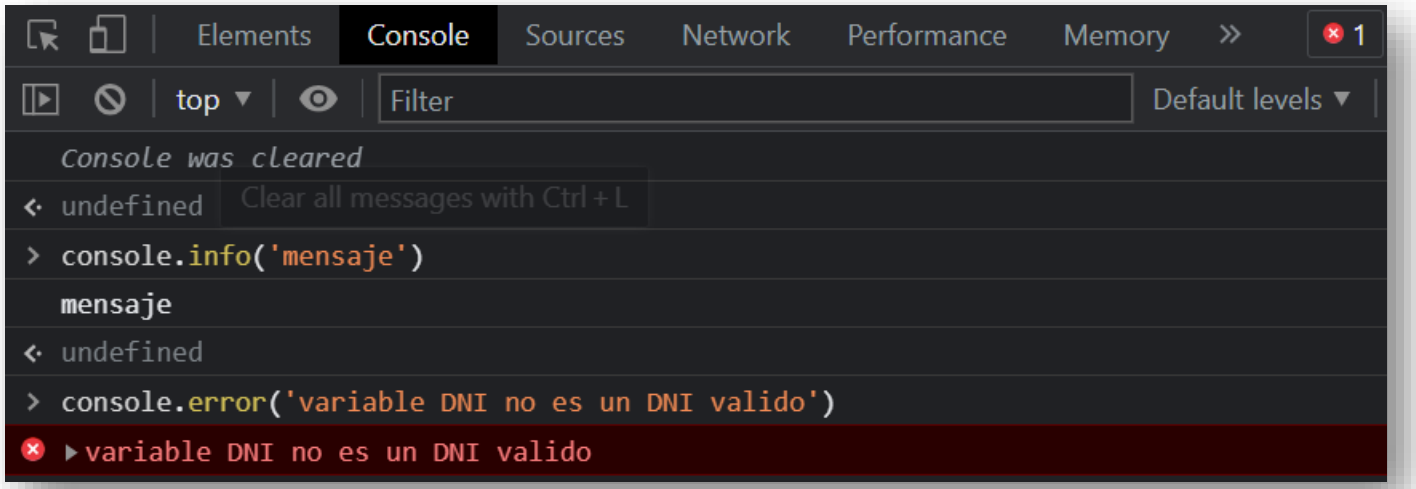
Los mensajes en consola son utilizados para no interferir con el flujo normal del script (*script: programa o conjunto de instrucciones*).



Consola

La consola del navegador también permite ejecutar código JS y mostrar los resultados en la misma.

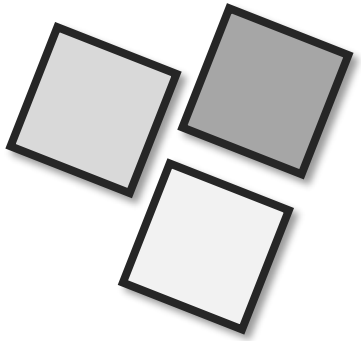
También se puede utilizar para alertar de un mal funcionamiento en un script. El usuario normal no se percatará, pero el desarrollador podrá revisarlo.



`console.clear()` este método limpia la salida de la consola, **pero no limpia la memoria**.

La consola del navegador dispone de varios espacios que se pueden consultar:

Errores: accesible a través del método `console.error('')`
Avisos: accesible a través del método `console.warn('')`
Informes: accesible a través del método `console.log('')`
Información: accesible a través del método `console.info('')`



Consola

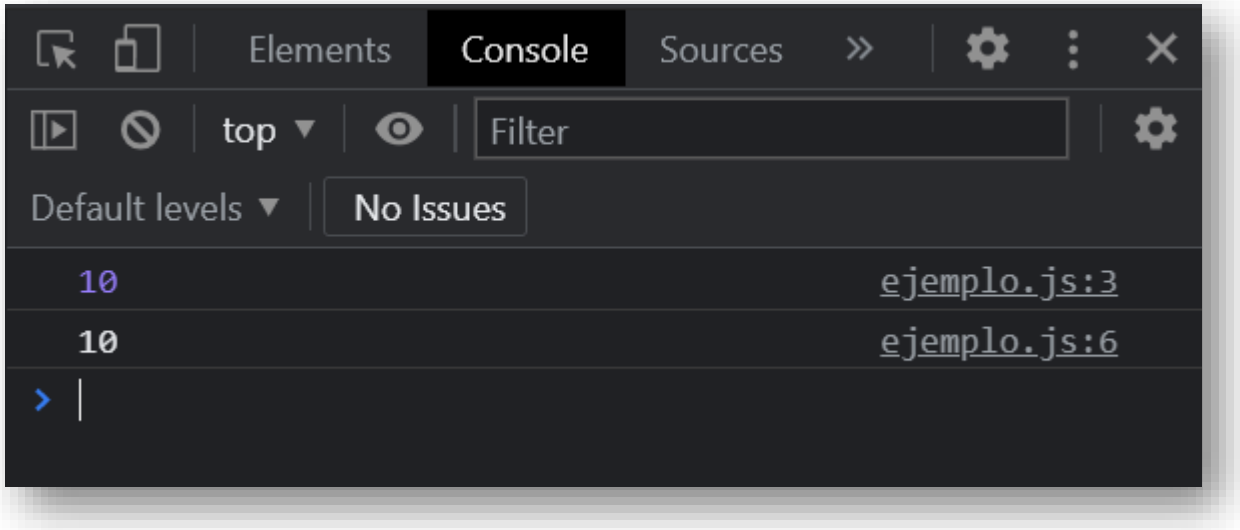
La consola también da información acerca del tipo datos que se muestran.

En el ejemplo de la derecha, se aprecia que, aunque los dos muestran el número 10, tienen distinto color, el azul representa a un dato de tipo numérico y el blanco a uno de tipo texto.

El color azul también se utiliza para los tipos booleanos que se verán más adelante.

```
let numero=10;
console.log(numero);

numero="10";
console.log(numero);
```



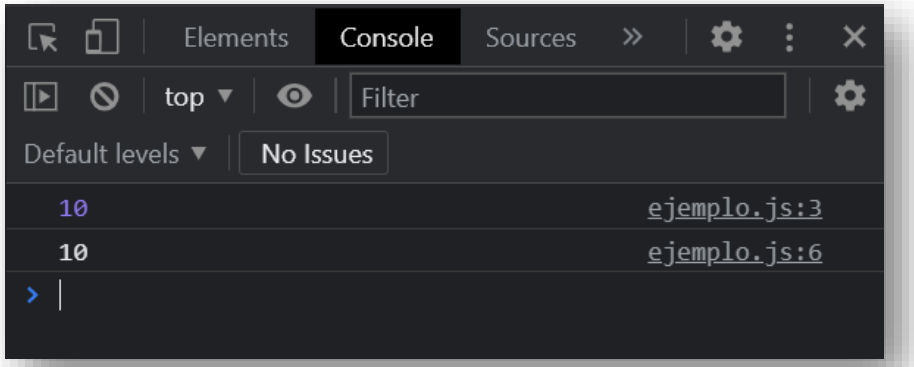
Consola

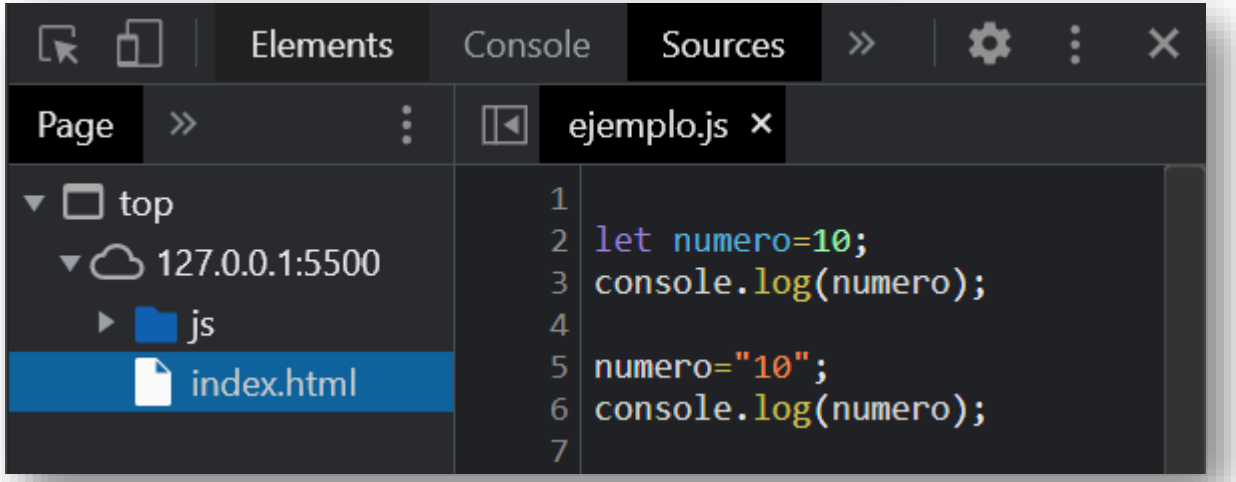
```
1
2   let numero=10;
3   console.log(numero);
4
5   numero="10";
6   console.log(numero);
```

La consola además da información sobre la línea del código JS que genera la salida.

En el ejemplo de la derecha, se aprecia que el primer `10` hace referencia al fichero `ejemplo.js:3` que es el fichero y la línea de código del mismo que generó la salida en la consola.

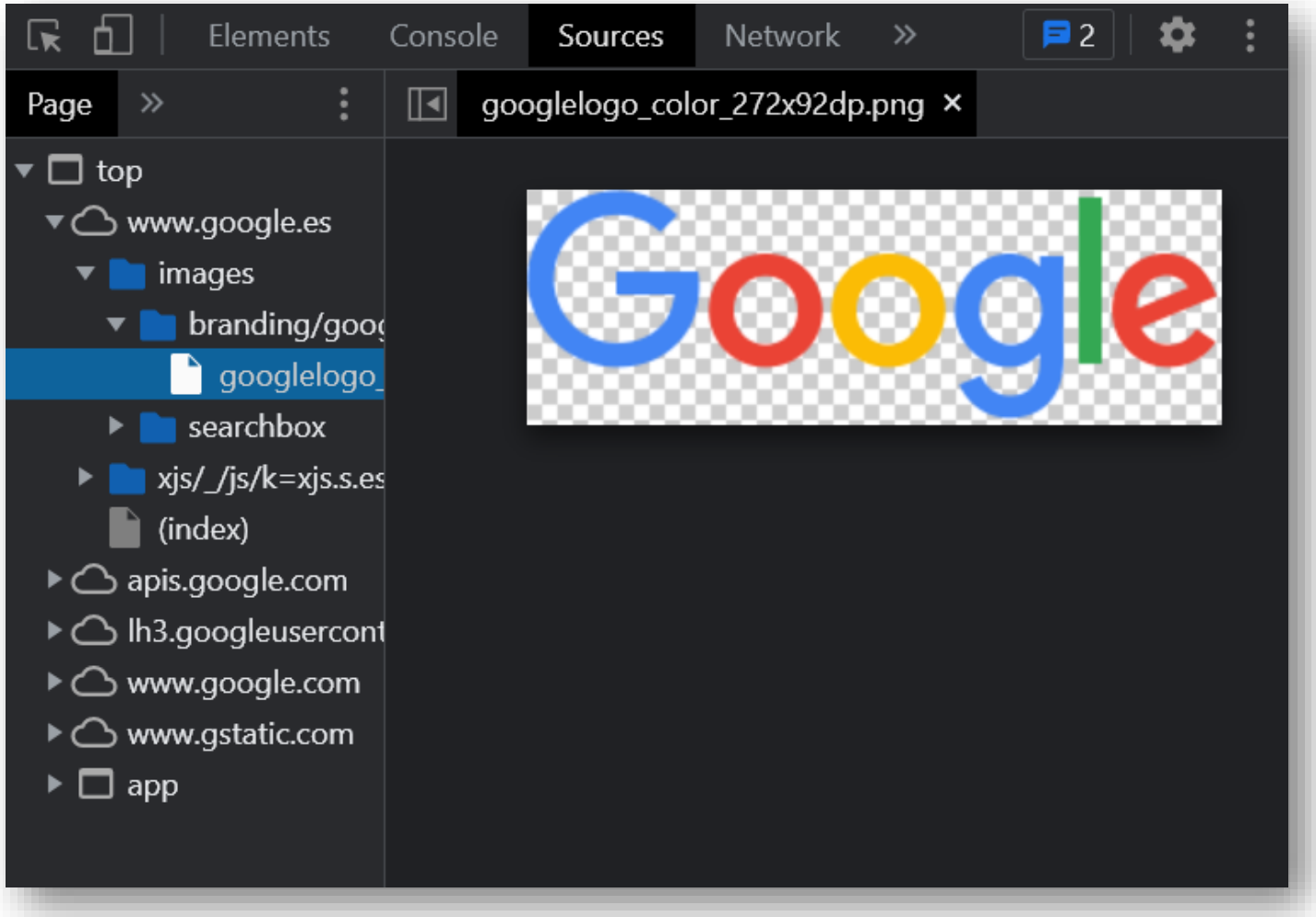
Si se hace clic sobre `ejemplo.js:3` el depurador del navegador web mostrará dicho fichero y la línea en cuestión.





Consola

Además, el depurador del navegador web permite abrir, a través de la pestaña `sources`, los distintos ficheros que influyen en la página web que se encuentra cargada, entre los que se incluyen ficheros JS, CSS, imágenes...

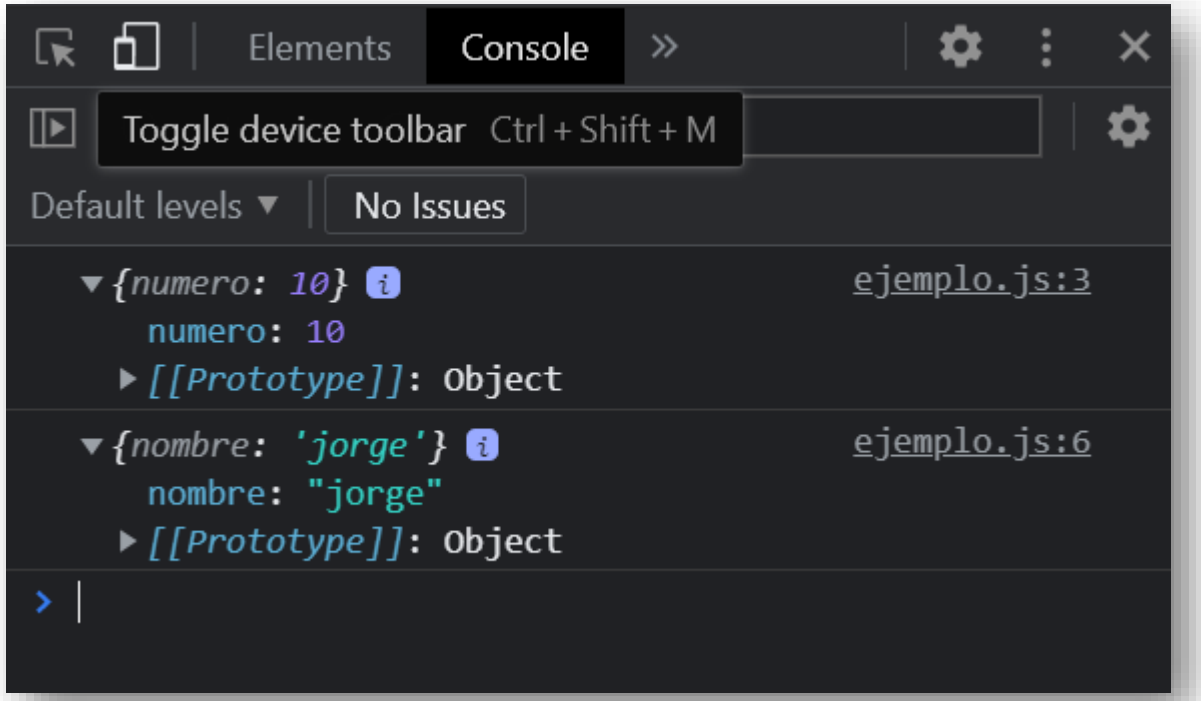


Consola

```
1
2   let numero=10;
3   console.log({numero});
4
5   nombre="jorge";
6   console.log({nombre});
```

Volviendo al ejemplo anterior, se estaba mostrando un mensaje en la consola, un 10, pero sin saber a qué hace referencia.

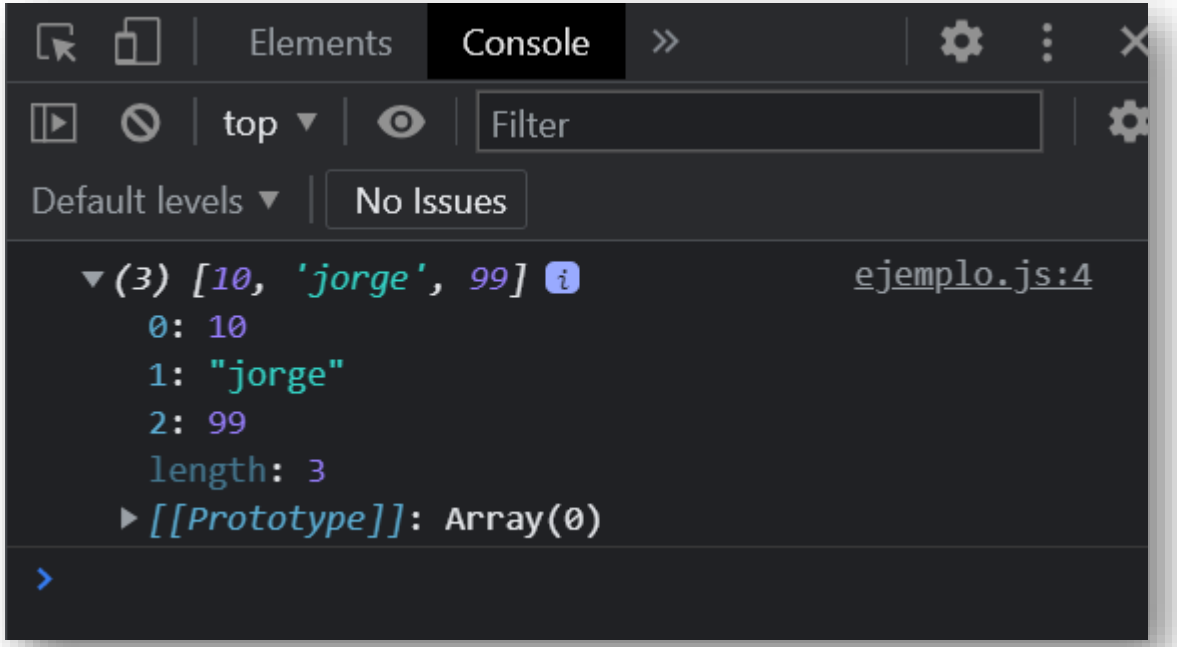
Se pueden usar llaves: { } para convertir las “variables” en objetos (se verá en próximos temas) y así tener más información en la consola.



Consola

```
1
2  let numero=10, nombre="jorge", edad=99;
3
4  console.log([numero, nombre, edad]);
5
6
```

Otra opción que está disponible es usar corchetes: [ ] para convertirlo en un array (se verá en próximos temas) y así agrupar el nombre junto a su valor.



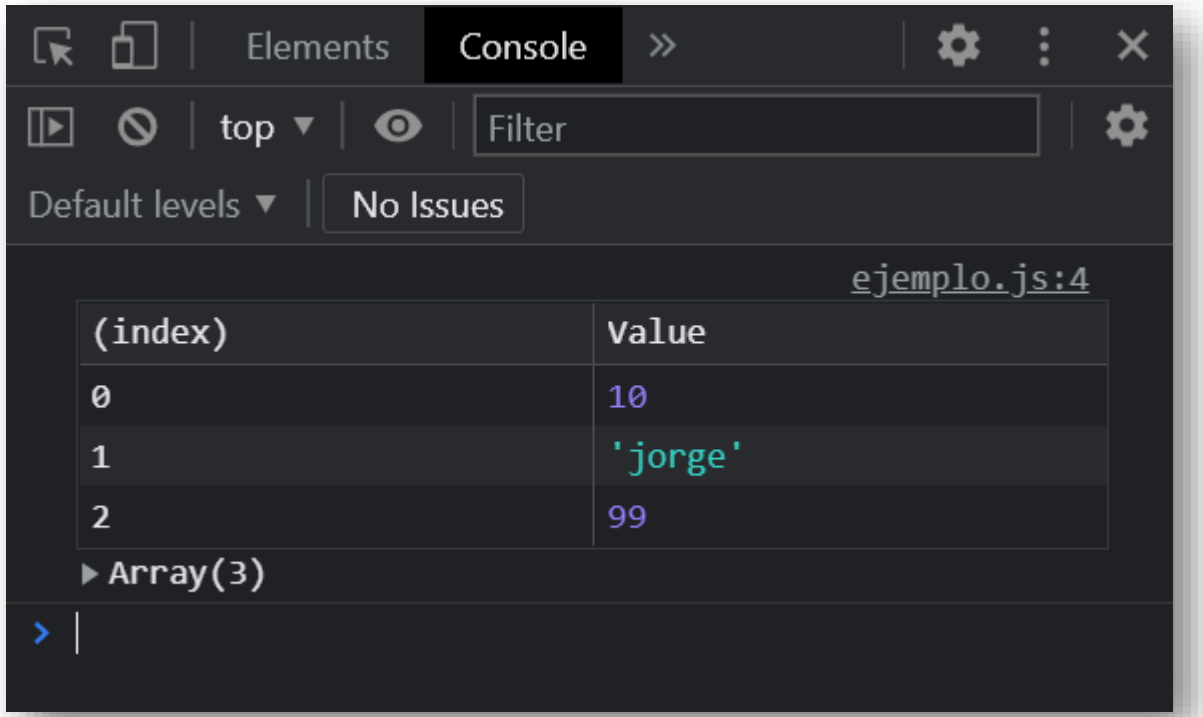
Consola

```
1
2  let numero=10, nombre="jorge", edad=99;
3
4  console.table([numero, nombre, edad]);
5
6
```

Por último, se puede usar el método `console.table()` que recibe un array, por lo que hay que poner lo que se quiera mostrar entre corchetes: [ ].

La salida aparecerá en formato tabla, con la posibilidad de poder ordenarlo.

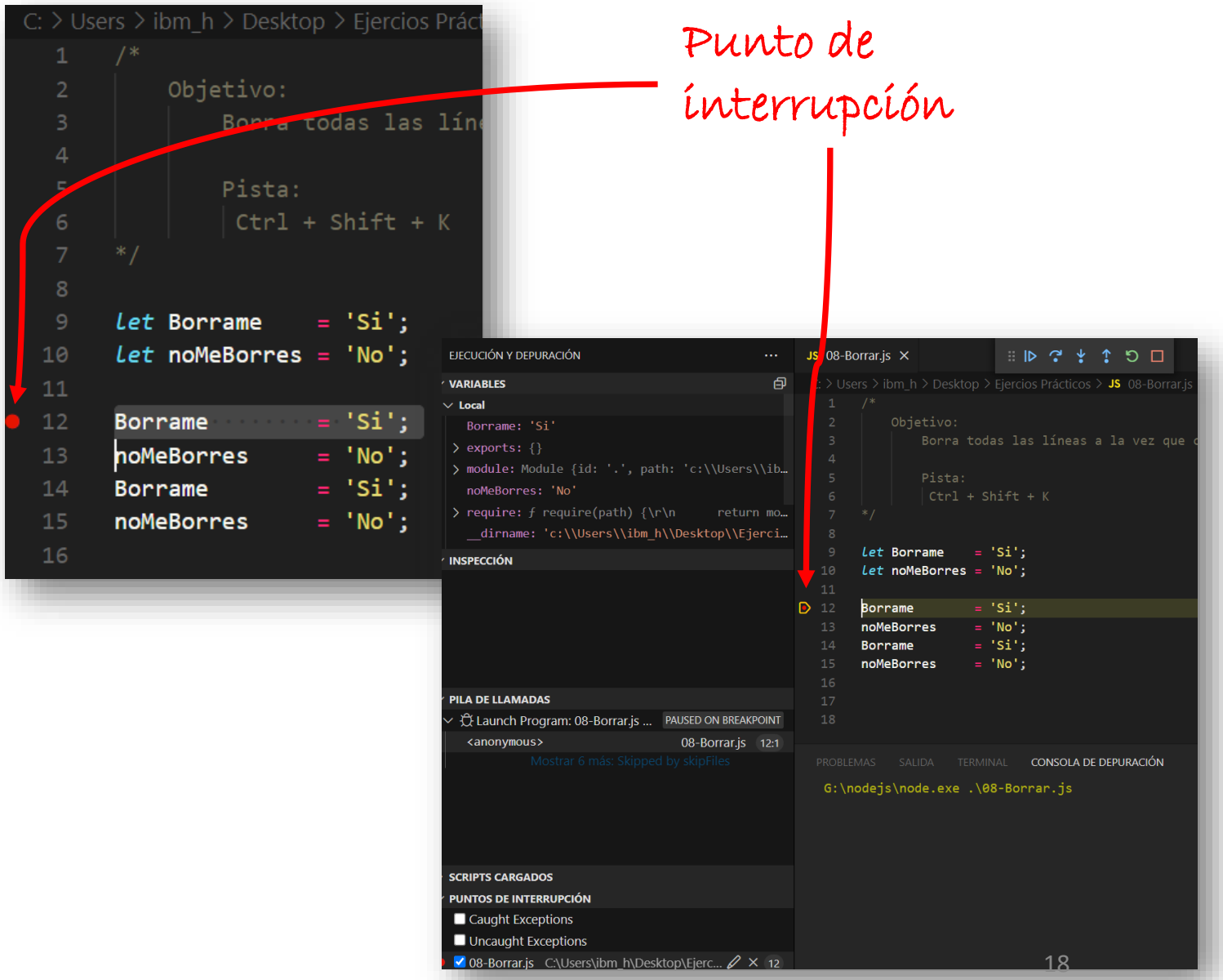
Este método es muy útil cuando se tiene mucha información que mostrar por consola.



Breakpoints

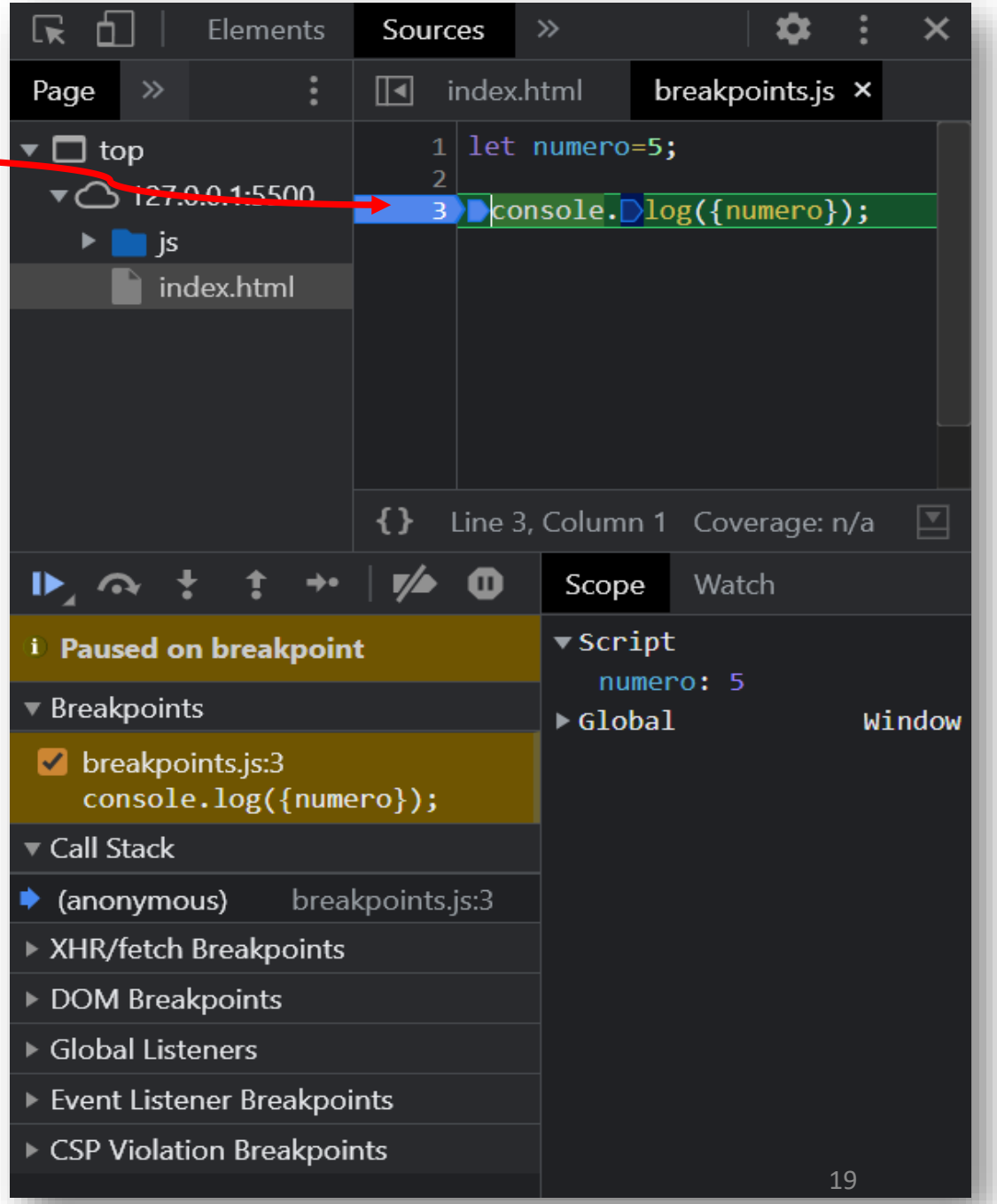
Un *breakpoint* (punto de interrupción) es una marca que se pone en una línea de del código fuente, de tal forma que cuando la ejecución llegue a ese punto el proceso se detendrá y se podrá analizar su estado en ese momento, justo antes de que se ejecute esa línea de código.

VS Code permite poner puntos de interrupción y hacer uso de ellos si está instalado Node.js.



Breakpoints

Punto de interrupción



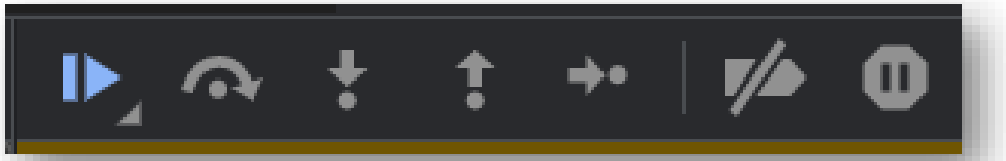
El navegador también permite insertar *breakpoints* para depurar el código.

Para ello, hay que hacer clic en la pestaña sources de la ventana inspeccionar y seleccionar el fichero JS, dentro de él, delante de la línea de se quiera colocar el *breakpoint* se hace clic, aparece una marca.

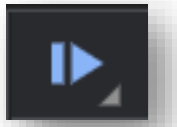
Al recargar la página (*F5*) el código se detendrá en el primer *breakpoint* que se haya colocado.

Para borrar un breakpoint se hace clic sobre el mismo.

Breakpoints



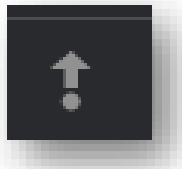
Se le puede indicar cómo se desea proceder a partir de este momento:



Continua la ejecución hasta el siguiente *breakpoint* o hasta que se acabe el script.

Continua la ejecución paso a paso sin detenerse en cada función.

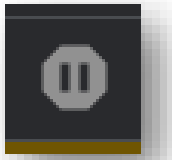
Continua la ejecución hasta una llamada de función.



Continua la ejecución hasta la salida de una función.

Continua la ejecución paso a paso a través del script.

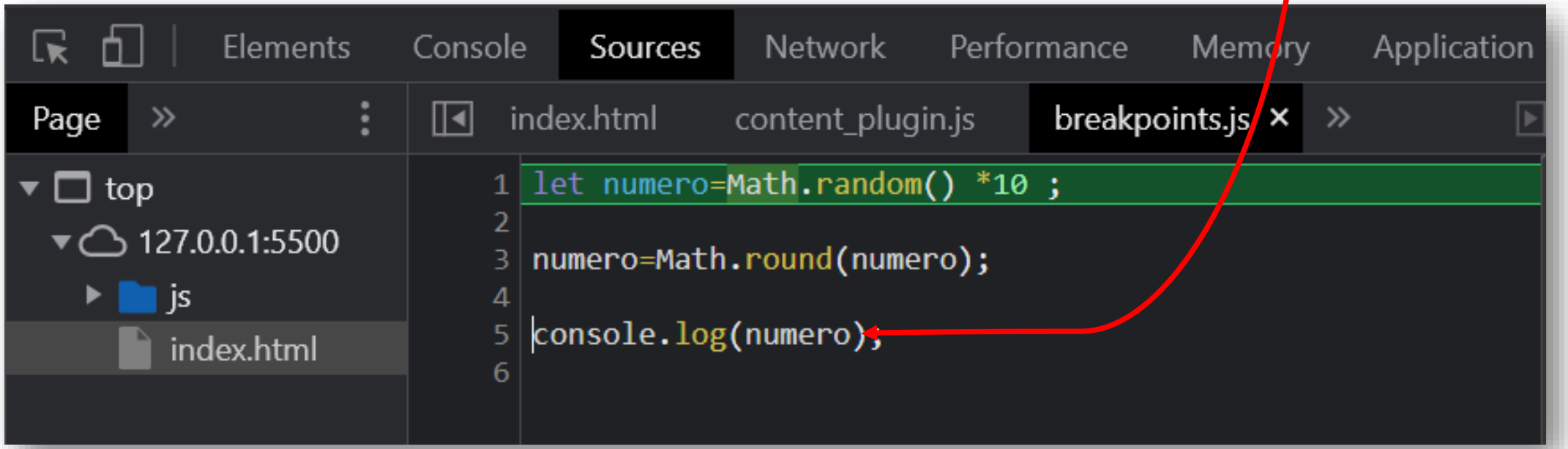
Activa o desactiva los *breakpoints*, no los borra.



El código no se detiene salvo que haya una excepción.

Breakpoints

Cómo averiguamos qué valor tiene número antes de salir por la consola



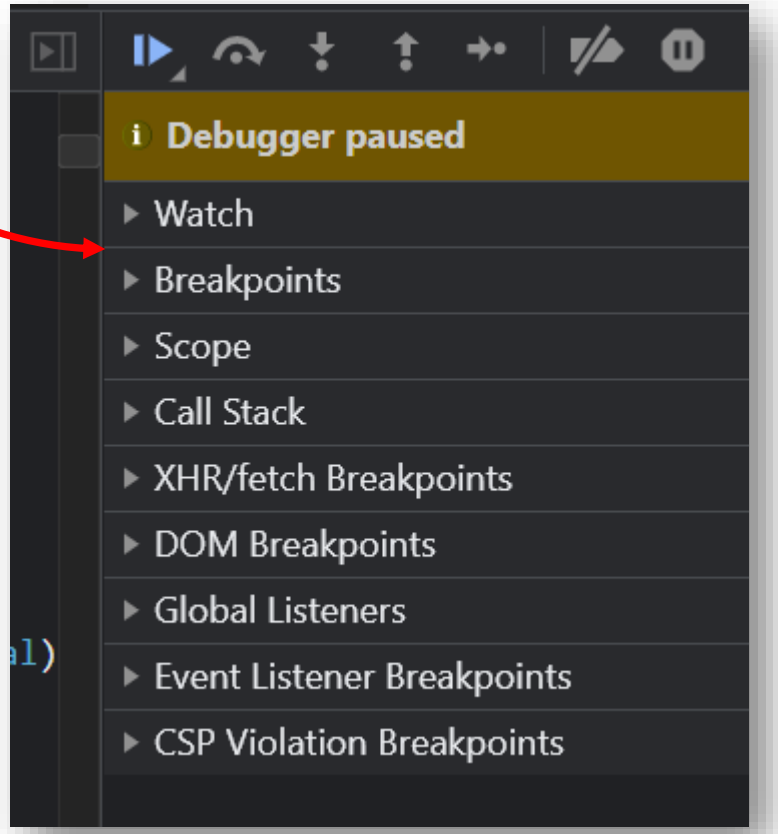
Breakpoints

Ventana debugger

El navegador también dispone de una ventana debuger para depurar el código.

En esta ventana, entre otras funciones, podemos:

- Centrarnos sobre alguna variable del código que queremos inspeccionar (*watch*).
- Acceder a los *breakpoints* que están colocados en el código.
- Las variables y sus valores definidas en los distintos ámbitos de existencia (*scope*).
- Ver la pila de llamadas.



Bibliografía

- https://developer.chrome.com/docs/devtools/console/
- https://www.google.com/intl/es_es/chrome/dev/