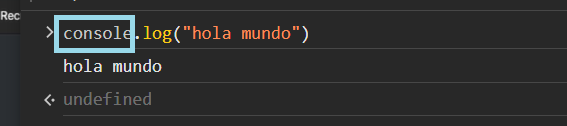
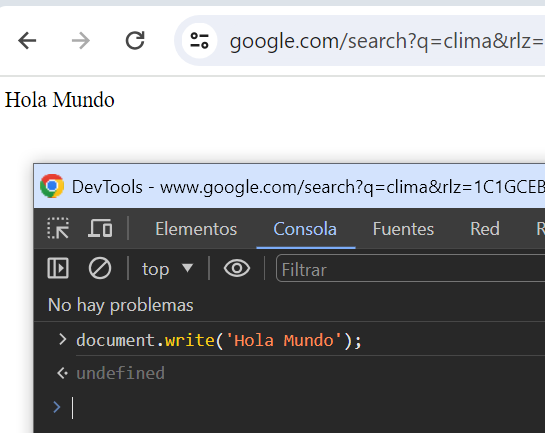
**JAVASCRIPT**

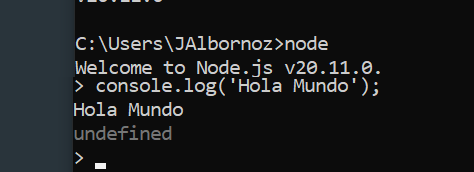
Todas las funciones en javascript tienen un return, por eso devuelve el **undefined**.  Console es un **objeto** y cuando ponemos un punto significa que es un **método**. No es más que una **función** dentro de un **objeto**.

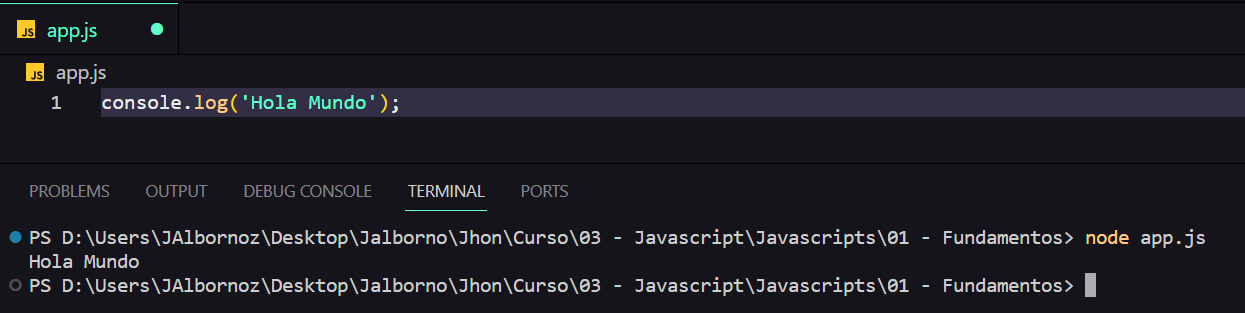
Ahora lo hacemos de otra manera, agarramos el **objeto** document y usamos el **método** write:

 De esta manera escribimos en el documento y se muestra.

Uno lo escribe en consola y el otro en el html.

**Node** nos permite ejecutar **javascripts** fuera del navegador web.

Con el comando **node** ya entra en modo edición y puedo ejecutar javascripts.

 Desde la terminal de visual estudio code, también se puede ejecutar, mediante el comando

**node app.js.**

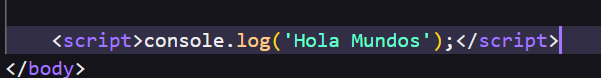
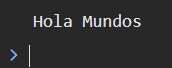
Ahora bien únicamente imprimio el Hola Mundo y no devolvió el undefined, que sale porque la funciones siempre devuelven algo tienen un return, al no tener nada especificado sale el undefined.

Para poder ver el undefined tengo que hacer un console.log de toda la ejecución como lo hace la terminal cmd o el navegador sería así:

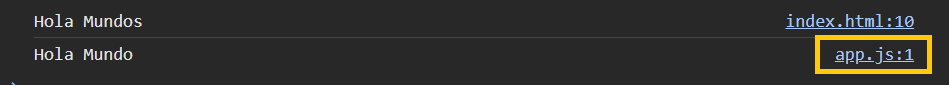


**Si solo pongo node me entra en modo edición para ejecutar javascripts.**

Como lo ejecutamos dentro de un html, bueno con las etiquetas scripts:

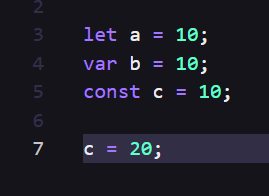
 Como es un console log, cuando abrimos el archivo html que creamos  mediante visual estudio code.  nos sale en la consola. El  ya no se utiliza por defecto el script es javascripts.

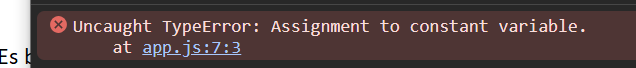
Para importar el archivo js hago un src :  Si el archivo js no esta en la misma carpeta, hago el path correspondiente.



Es buena práctica colocar los **tags scripts** al final de todo del HTML, porque si hay algún error hasta que no termine de ejecutar todos los scripts no va a terminar de renderizar la pagina.

Y también es recomendable colocar el script en un archivo a parte.



****

**No se le puede asignar un valor nuevo a una constante.**

**** Podemos ver mediante consola. ESC para sacar el mensaje.

**EC5**

Es la versión que tiene compatibilidad con todos los sitios web, el EC6 (Ecma script 6) tiene mayor compatibilidad con los sitios web modernos.

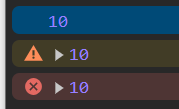
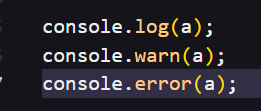
**polyfill**

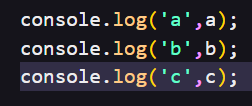
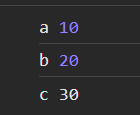
Es un código que provee el funcionamiento de una nueva característica del EC6 a una versión del EC5.

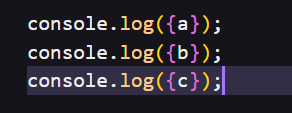
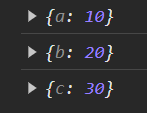
Let o var es lo mismo, var ya quedo obsoleto, casi todos los navegadores ya soportan let como asignación de variable.

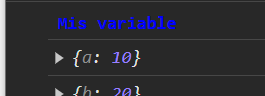
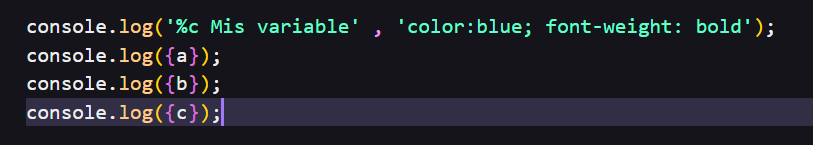
**Javascripts es un lenguaje interpretado, lo que quiere decir es que se va a ejecutar línea por línea el archivo JS. (secuencial)**

Maneras de mostrar los **console.**



  Los morados son números, los grises letras.

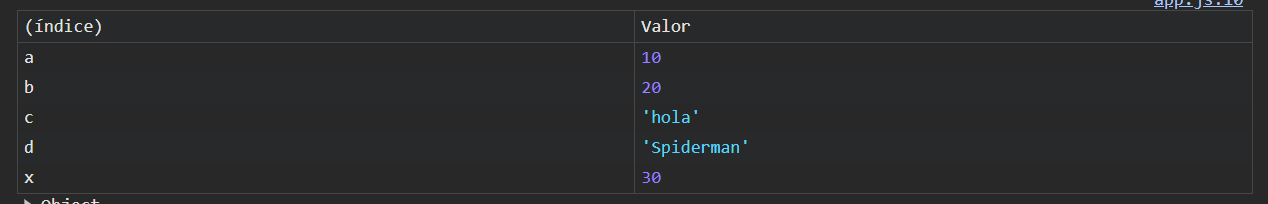
  **{}** Las llaves hacen automáticamente la adición de la variable, hace referencia al objeto.

 %c es el comando clave para agregar un estilo .

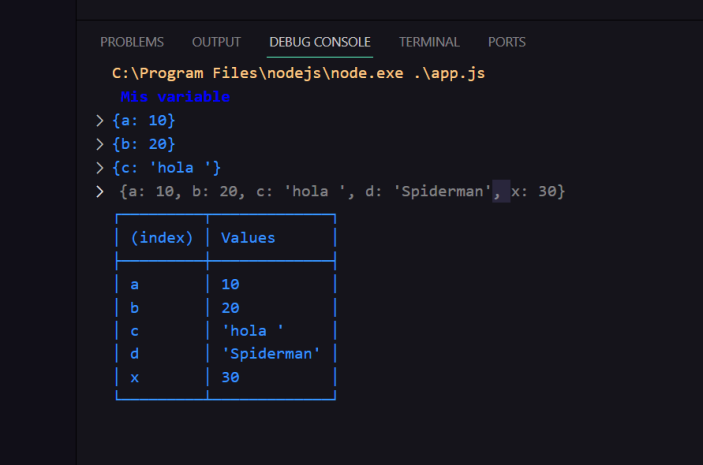
Console.table puede recibir un arreglo, es bueno para cuando tengo muchas variables y tienen un orden lógico.

Se puede agregar con un array []  o con {} para los objetos.

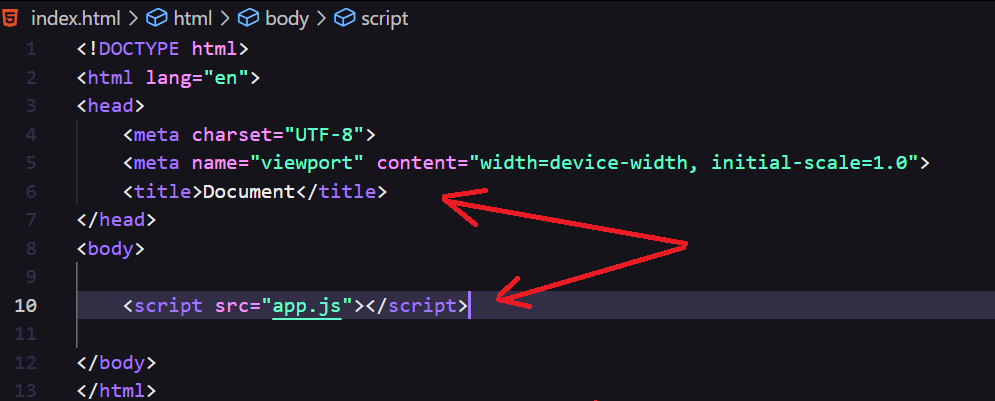




Debug console en la terminal:



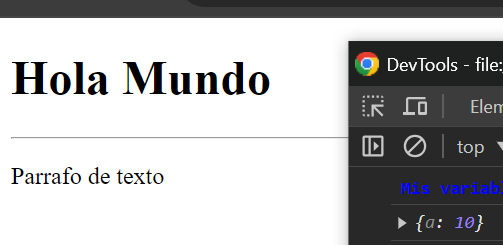
**Orden y lugar de las importaciones**

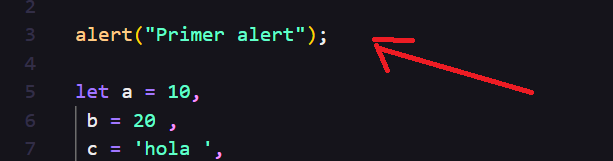
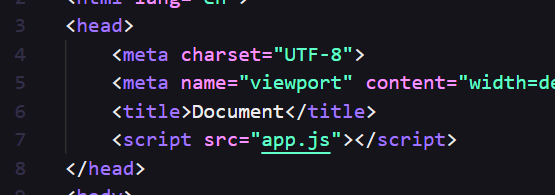


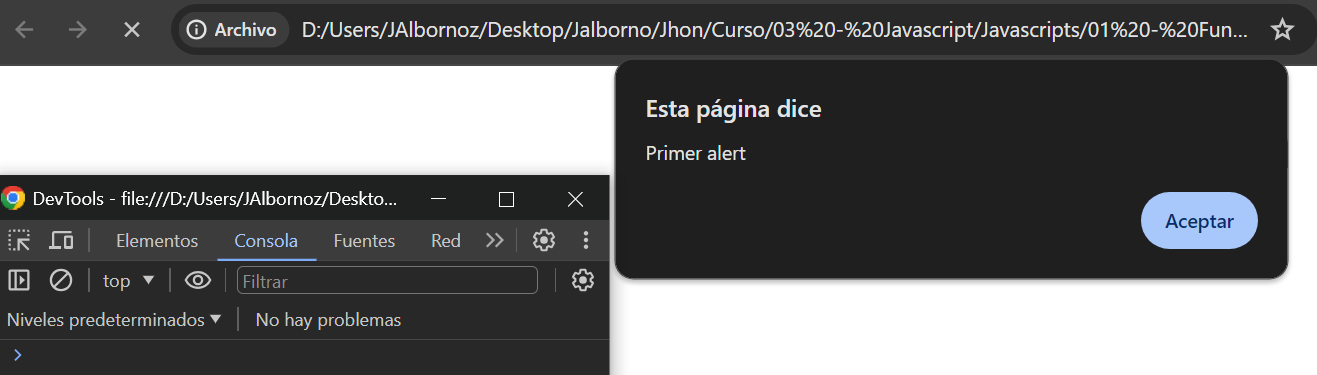
Se pueden poner en el HEAD o antes del cierre del BODY.

**HEAD**

Le agregarmos un titulo y algo para mostrar

 Ahora bien vamos a ver que pasa si coloco un **ALERT** en el .js primera instrucción.

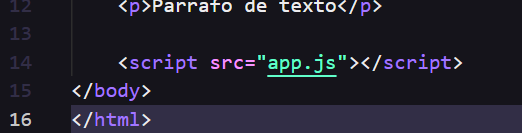




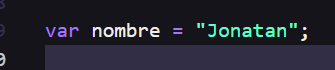
No se ejecuta nada como la función ALERT es **bloqueante de código** y hasta que no le de aceptar no para y el scripts estaba en el head no muestra nada de la pagina web.

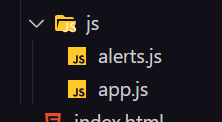
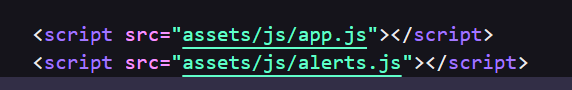
**ALERT** es una función de **WINDOWS** por eso se declara sin un objeto detrás.

**BODY**

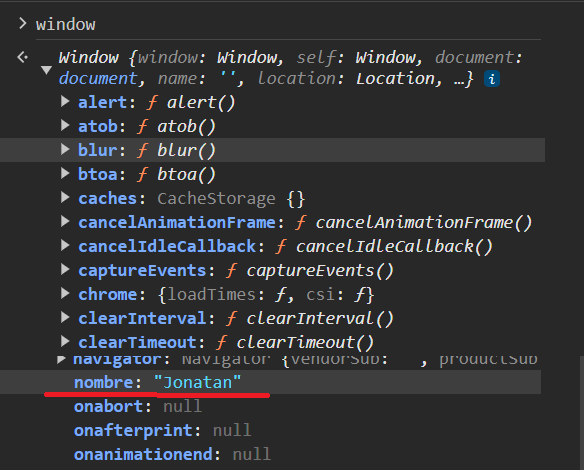
 Si lo dejamos al final del body, entonces se va a ejecutar mi html y luego quedará en stand by el .js .Pero se carga la pagina.

Ahora creamos una carpeta assets recursos donde hay una carpeta js, en el código agregue una variable declarada como **var** en **app.js**

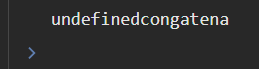
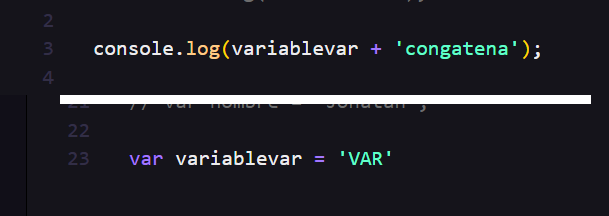


 Cree otro archivo **alerts.js** y ambos los llamos en el html

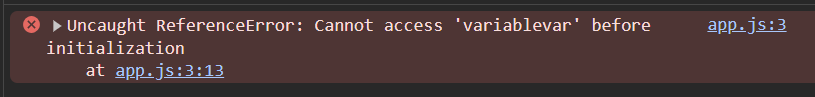
Al declararlo como **var** lo esta colocando dentro de un objeto GLOBAL llamado WINDOWS

 si buscamos el objeto **window** esta nombre adentró.

Otro problema viene con los errores que puede dar, si declaro la variable var a lo último, la útilizo al comienzo y la concateno con algo va a funcionar, porque funciona, javascript siempre hace un índice de todos los obetos y variables declaradas, entonces lo encuentra, dice hay una variable declarada en el archivo con ese nombre:



Es muy difícil encontrar el verdadero problema. Si cambiamos a let o una constante el error cambia:

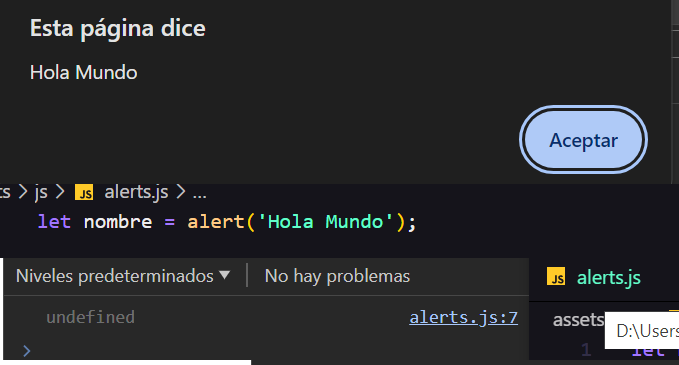


No puedo acceder a la variable tal, antes de su iniciación.

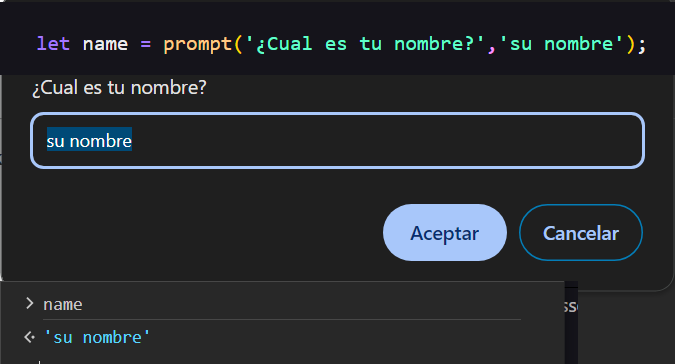
**TRES TIPO INGRESO DE INFORMACION DEL USUARIO**

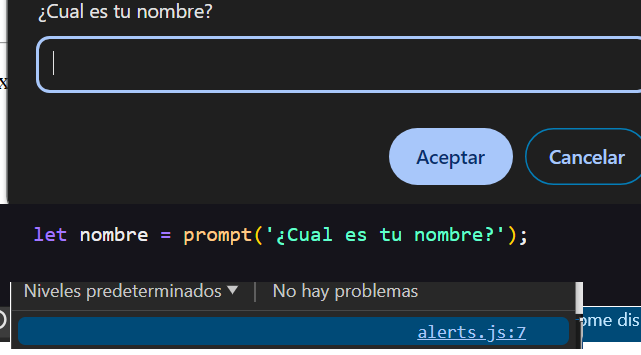
No son muy utilizadas mucho hoy en día, porque existen librerías que hacen alert mucho mas dinámicas.

**Alert(); 🡪** Es bloqueante , devuelve un undefined.

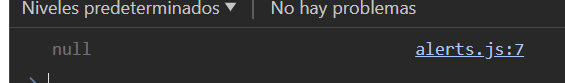


**Prompt() 🡪** Es bloqueante, ingreso de string (El segundo parámetro es lo que ingresa el usuario)



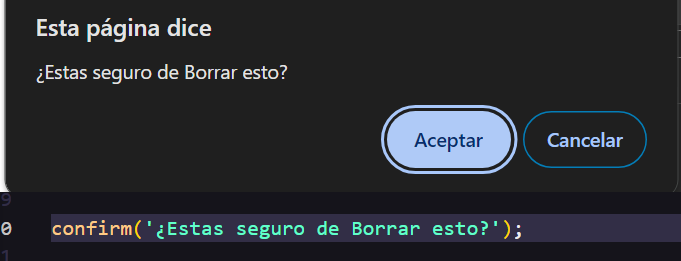
**Devuelve un string vacio ‘’**

Si apretó CANCELAR me devuelve

**NULL <> Undefined**

Null no contiene valor, el string vacio se considera un valor.

**CONFIRM() 🡪** Es bloqueante, devuelve un true o false. **(Boleano)**

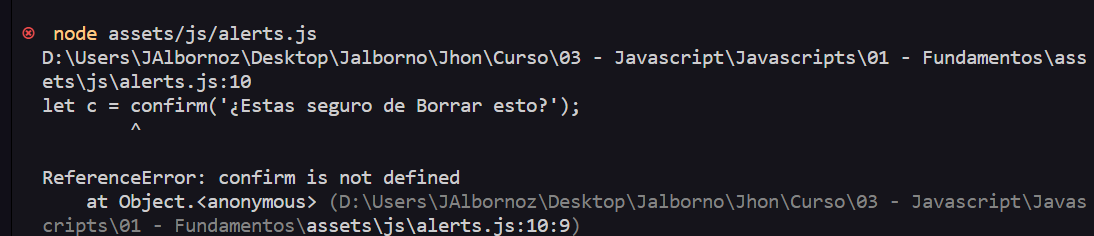
**True, false**

**Alert() confirm() prompt()**

**Window**

Son ***funciones*** dentro del Objeto **Window**. Si corro en otro navegador web que no tenga la instancia de Window, no lo va a ejecutar.

Ejemplo: Lo ejecutamos por **Node**.



Da error porque en **Node** no existe el objeto **window**, existe el objeto **global** que es parecido, pero solo para **Node**.

**TIPADO Y VARIABLES**

Javascrip es case sensitive, es un lenguaje débilmente tipado, javascript interpreta que la **variable** va a contener solo string o solo números.

¿Qué son los primitivos?

Es una información que **no es un objeto** y son **inmutables**:

BOOLEANOS (true.false)

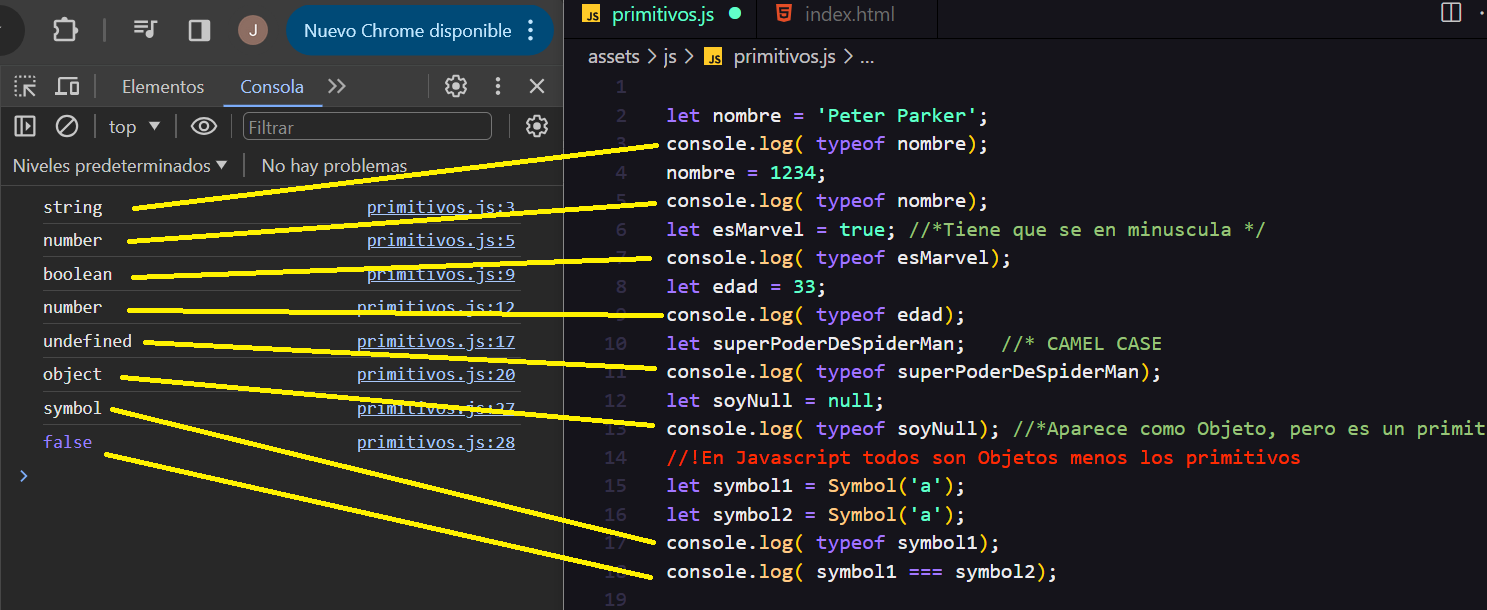
NULL (sin valor en absoluto)

UNDEFINED (sin asignación de valor)

NUMBER (No hay diferencia con los decimales 1 = 1.00000001)

STRING

SYMBOL (Es un valor único que no es igual a ningún otro valor, identificador único)



**Palabras Reservadas**

Son palabras que tienen un uso específico.

lowerCamelCase 🡪 Para declaración de variables

UperCamelCase 🡪 Para la declaración de Clases

Para ir con Node a una ruta en particular en una carpeta y esta contiene espacios o caracteres particulares que pueden invalidar la búsqueda de la ruta con cd:

Entonces tienen que agregarle comillas para que lo tome correctamente:  
