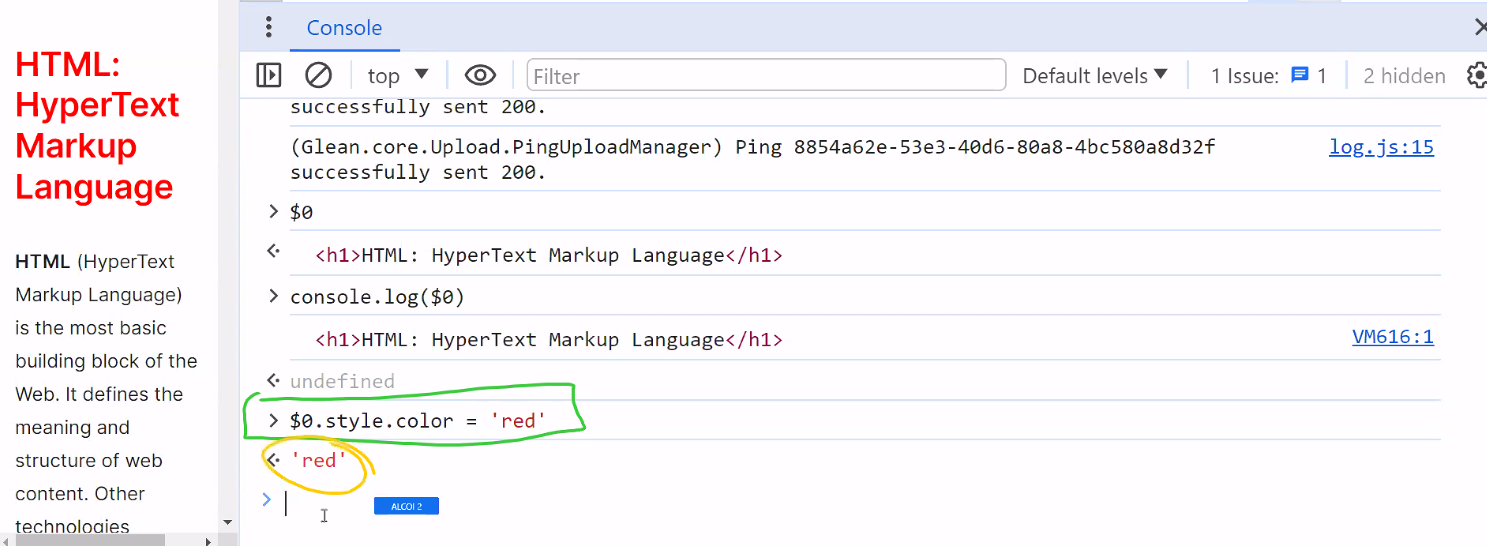
Marcando $0 cogiendo un elemento de una web podemos acceder a el.

El primero te devuelve el elementos en si al poner solo $0 y al poner console.log($0) te dice lo que contiene el elemento pero te devuelve indefinido.

Con esto podemos cambiar los estilos del elemento por ejemplo.

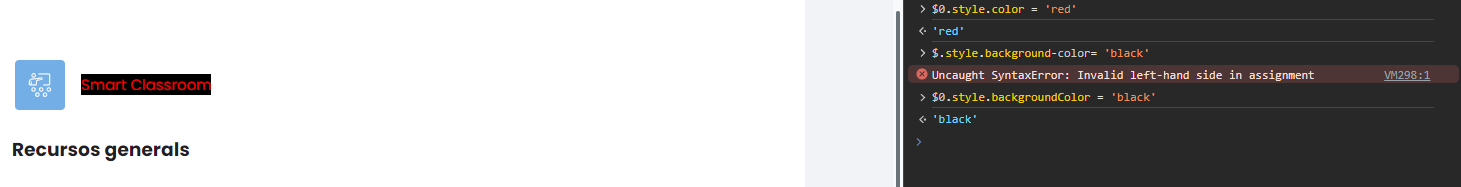


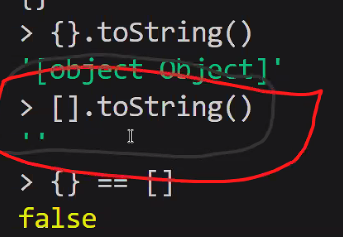
$0.style.color = ‘red’ y nos devuelve el color rojo y pone el titulo de color rojo.

Vamos a aplicar esto a **la pagina del campus** =>



Aplicamos al enlace Smart classroom el style.color = ‘red’ y nos cambia de color.

Ahora aplicamos un background color para hacerlo mas visible…



En este caso {} == [] va a dar Falso porque al pasarlo a string son cosas distintas.

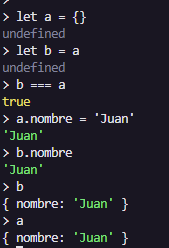
En cambio por ejemplo :

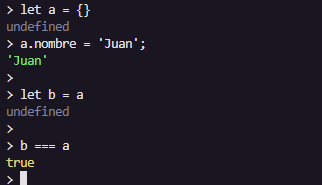


Eso va a dar igual porque al pasar el primer 3 a string va a ser igual al segundo ‘3’ == ‘3’.

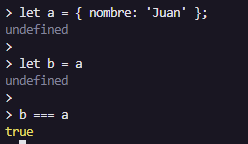
Pero con el triple igual..

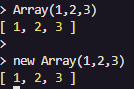
 Aquí da false porque uno es un número y otro es un string.



Al poner una variable igual a otra que tiene un objeto vacío va a ser === .

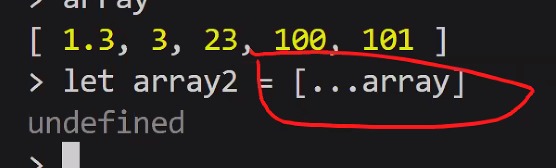
Pasa igual si es un objeto con un valor dentro.



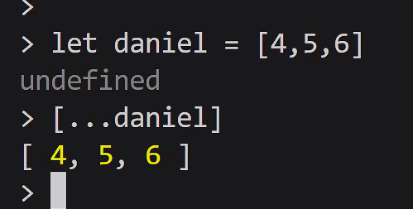
Formas de crear un **array**:  


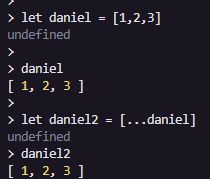
A la función Array o new Array podemos pasarle los parámetros con los elementos que queremos meter dentro del array. Por el contrario también podemos escribirlo con el numero de elementos que queremos pero con el array vacío si ponemos un solo parámetro:



Podemos copiar un array de un array anterior con esto:  


Ahora aquí array2 es igual a array poniendo […array]

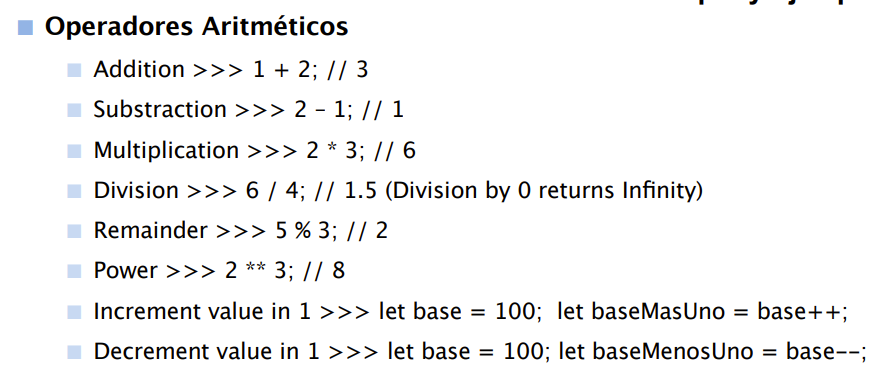
aquí también se hace una duplica del array con el nombre de la variable donde tenemos el array.



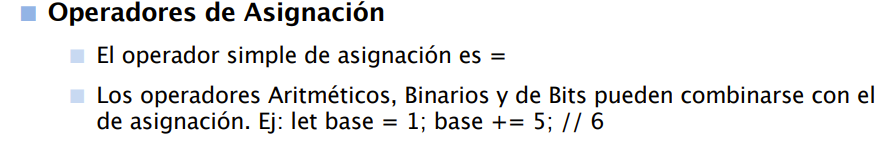
Aquí la copia se la asignamos a otra variable.

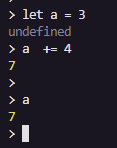
Hay varios **OPERADORES** en JS …

**Aritmeticos**

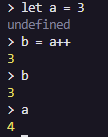


**Asignación.**

****

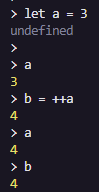
**El += lo suma y luego le da el valor a la variable.**

**Ahora los de incremento:**

****

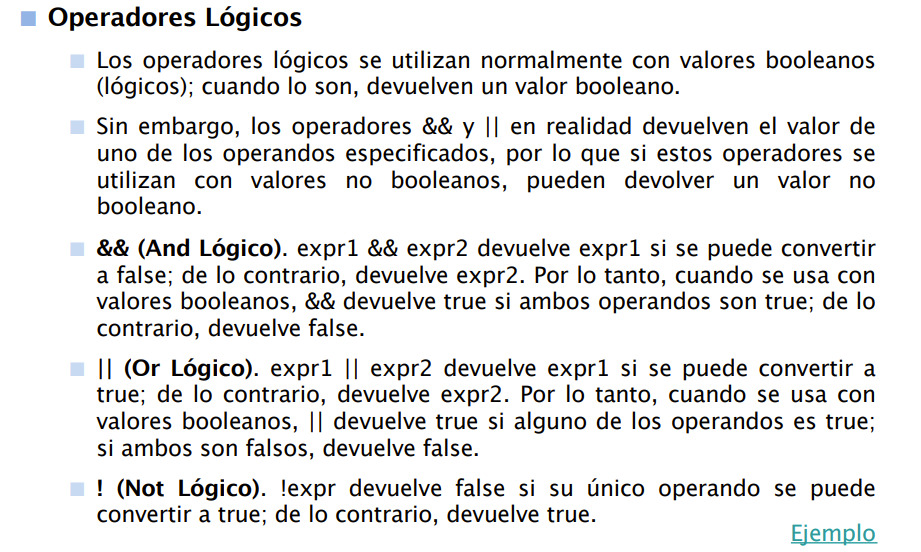
**A vale 3 y al declarar b como a ++ primero evalua que b = a y luego le añade 1 a la variable a por lo que b vale 3 y a vale 4.**

**Por el contrario…**

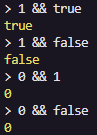
****

**Si primero hacemos el incremento y luego ponemos la variable, el incremento se hace antes de la evaluación por lo que a será 4 y b también.**

**Operadores lógicos: Se utilizan para hacer comparaciones.**

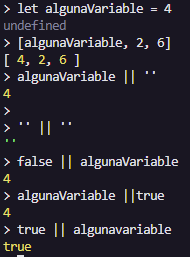


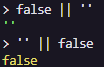
**En el operador && evalua los dos valores a ambos lados, si ambos son verdaderos devuelve el de la derecha, en cambio si uno es falso, nos devuelve el falso**

**En el operador & si ambos son verdaderos como el primer caso, nos devuelve el de la derecha.**

**Si uno es verdadero y el otro falso, nos devuelve el falso (2º y 3r caso)**

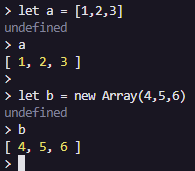
**Si ambos son falsos nos devuelve el valor de la izquierda.**

**el operador || devuelve siempre el valor verdadero, si los dos son verdaderos da el valor de la izquierda.**

**Si los dos son falsos…**

**…En este caso devuelve el valor de la izquierda, el primero que lee .**

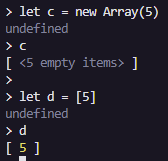
**Para crear un array hay dos formas declarar la variable con [] o el método new Array().**

****

**Ambas son formas sencillas pero tienen casos en los que son necesarios el uso de una u otra.**

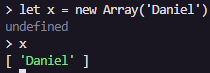
**Por ejemplo si queremos iniciar un array con un solo numero es obligatorio usar el primer método…**

**Y esto se hace porque…**

****

**Si creamos el array con un valor entre paréntesis solamente, nos crea un array con ese valor de elementos vacíos en cambio con el método de aplicar [] a la variable si se consigue hacer.**

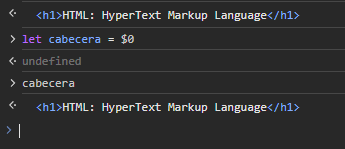
**-----ESTO SOLO APLICA A NUMEROS !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!**

****

**Vamos a trabajar con el navegador…**

**Inspeccionamos el h1 de la pagina** [**https://developer.mozilla.org/en-US/docs/Web/HTML**](https://developer.mozilla.org/en-US/docs/Web/HTML) **y al hacer $0 nos coge el h1**

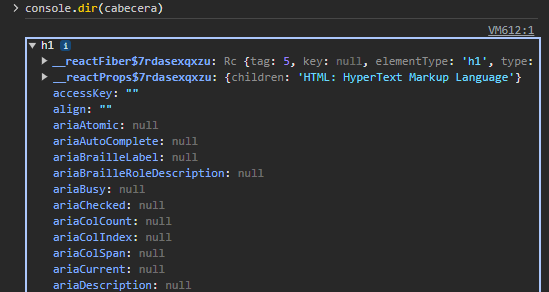
**Lo guardamos en una variable cabecera…**

****

****

**Al intentar convertirla a string nos dice que es un objeto**

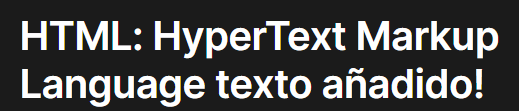
**Para ver los detalles de un objeto console.dir… nos da TODOS los atributos de ese elemento incluso los que no están definidos.**

****

Podemos añadir texto…

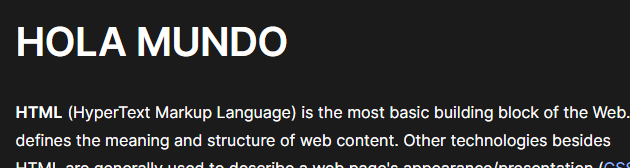


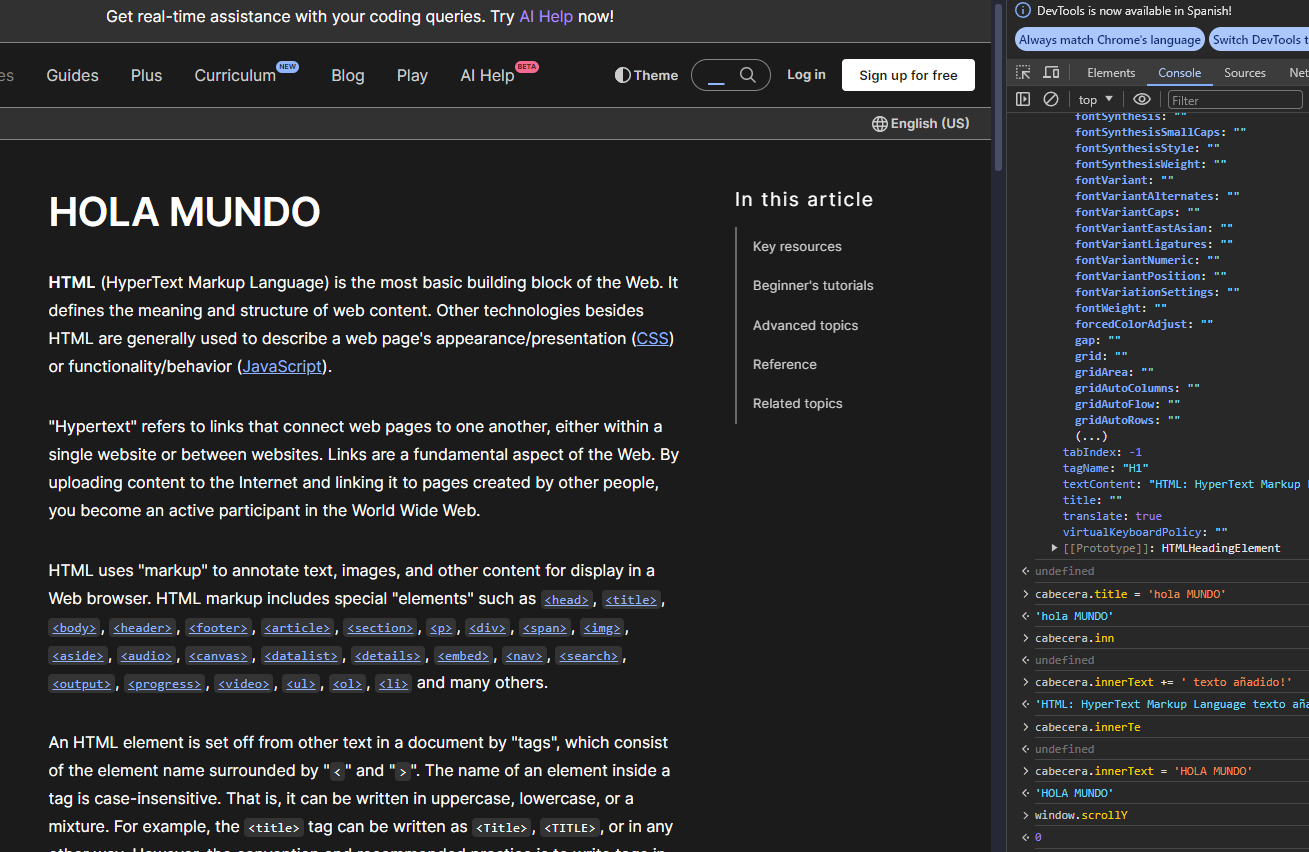
El += añade texto a continuación del elemento

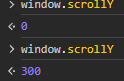


Si por el contrario solo escribimos = , cambiara todo el contenido por el nuevo que le asignemos…





Tambien podemos mirar con scrollY la posición del scroll en la que nos encontramos en la página. En este caso en el inicio el scrollY es = 0 y en cuanto bajamos un poco…

**…cambia a 300**

**Tambien podemos medir la distancia entre un elemento y el principio de la pagina…**

** con offsetTop**

**Con classList.add le podemos añadir una clase y con classList.remove se la podemos quitar.**

****

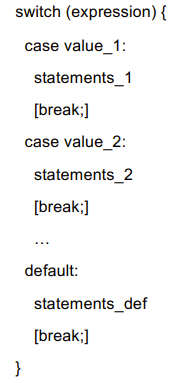
****





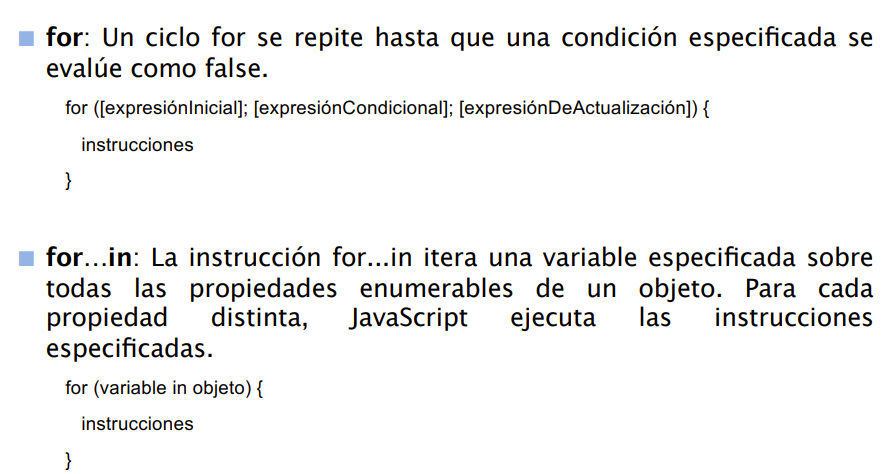
En ambos casos hay que especificar el nombre de la clase para añadir obviamente pero también para removerla.

ESTRUCTURAS DE CONTROL DE FLUJO (diapo 54+)

**SWITCH:**

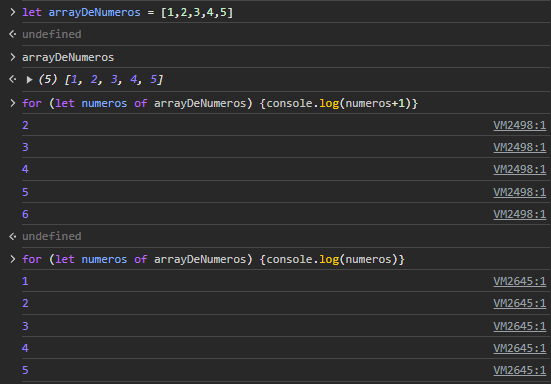
**BUCLES**

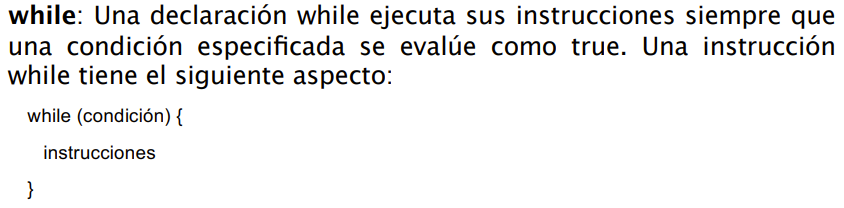
**FOR:**

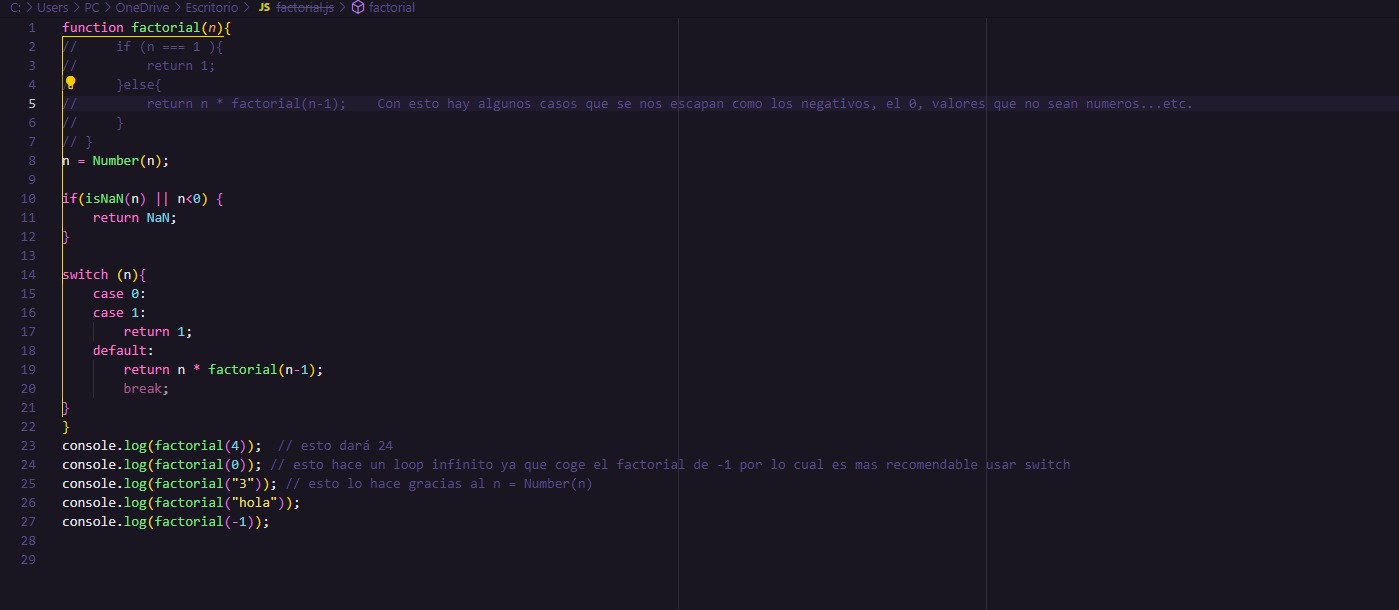
****

con **FOR OF**, podemos recorrer de esta manera cada item del array siendo la variable declarada dentro del paréntesis (**numeros** en este caso) cada uno de los elementos del array.

Para hacer un recorrido con una condición se deberá utilizar el FOR normal o preferiblemente WHILE, breaks… (diapositivas 60+)



**WHILE**

**FUNCION FACTORIAL**

En este caso hacemos la función factorial de un numero por ejemplo 4 (primer ejemplo). La primera línea lo transforma a numero por si lo que entrásemos es un array de un número.

A partir de ahí valorara si es un numero o no o si es un numero negativo para que devuelva NaN.

Si no entra en ese if, va al switch y valora cada caso, en este caso no es 0 ni 1 por lo que pasa a default y hace el cálculo. Retorna n multiplicado por la función de nuevo 4-1 que son 3 = 12. Y vuelve a empezar hasta que la n es 1 donde va por 1 y devuelve 1. Por lo que es 12\*2 = 24 , 2-1=1 y ahí para.

**Cálculo del factorial**:

Para otros valores de n (mayores que 1), se utiliza una estructura de control switch. En este caso, se utiliza un bucle de recursión. Cada vez que factorial n se llama recursivamente, se multiplica n por factorial n-1. Esto continúa hasta que n alcanza 1.

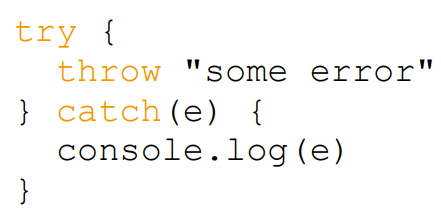
**GESTION DE ERRORES:**

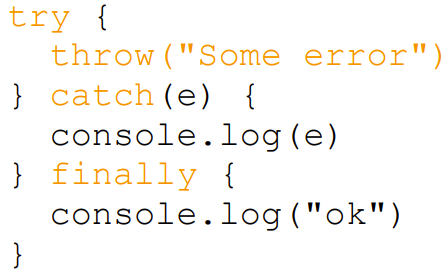
Cuando el intérprete de JavaScript no puede ejecutar una instrucción, se pondrá en un estado especial de error.

■ Eso permite capturar el error, y reaccionar de manera apropiada.

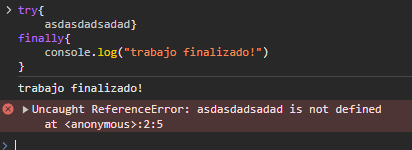
■ Toda ejecución normal se ignorará, hasta que se encuentre la siguiente instrucción de gestión de errores

■ Si un error no se gestiona, se pierde todo el contexto y el intérprete mostrará el error al usuario, por ej en consola.

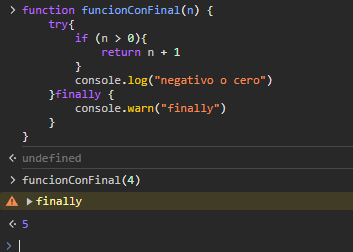
**  
Palabras clave: Try, catch, throw y FINALLY**

****

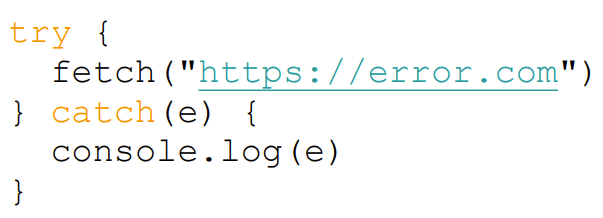
En este caso Finally es lo que tiene que hacer la aplicación aunque de el mensaje de error. Por ej. Una lavadora da un error pero aunq salte el mensaje de error tiene que acabar de hacer ciertos procesos como el vaciado de agua y la apertura del cierre de la puerta, etc. El bloque finally siempre se ejecutara en cualquier caso.



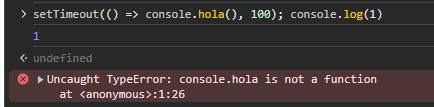
En este ejemplo vemos como primero sale el mensaje de finally (ahí podría ir cualquier tipo de función, proceso etc) y luego nos salta el error, pero el finally se ejecuta igual SI O SI.



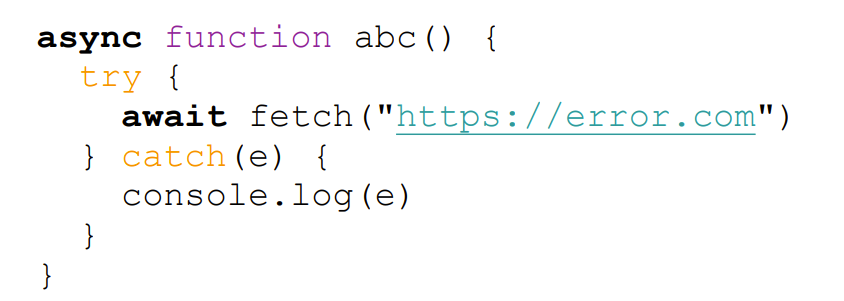
Pero finally no se utiliza simplemente para manejar errores sino como depuración. En este caso igualmente se ejecuta antes pese a que no da ningún error. Devuelve primero el finally y luego el return.



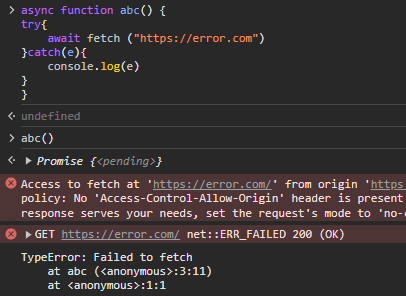
La cuestión es que las funciones de fetch y timeout son asíncronas, ocurren después de la ejecución por mucho que las escribamos antes, por ej:

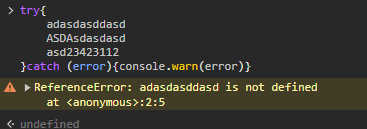


Sale primero el console.log y luego el error. Por lo que hay errores que se nos van a escapar por salir del ámbito de nuestro catch ya que solo captura lo inmediato. Para capturar ese tipo de errores lo podemos hacer con la función asíncrona y con el await.



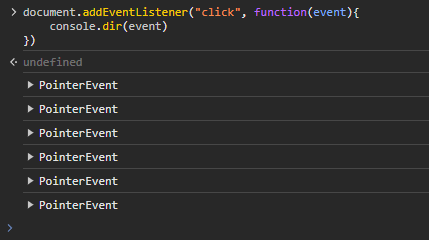
El **fetch** definitivamente es una función que tienen los navegadores para preguntar cosas de la aplicación.

En este caso al ejecutar la función abc() queda una promesa pendiente hasta que salta el error.

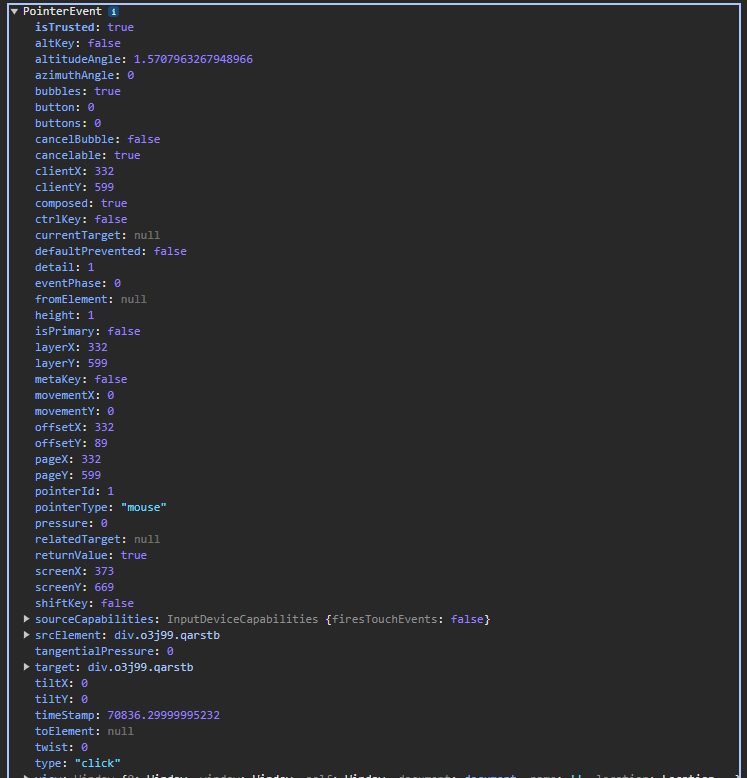
Hay que tener claro que esto solo se utiliza para situaciones excepcionales. Es decir solo capturará el primer error ya que al haber un error no se ejecutará el resto del código.

**GESTION DE EVENTOS:**

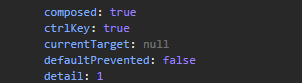
Un evento es un estado del navegador que se produce cuando el usuario interactua con la página.

Al escribir esta función el console.log nos dice el tipo de evento que es, que es un click con el puntero.

Cuando desplegamos el evento Pointer evento nos da todo tipo de información:



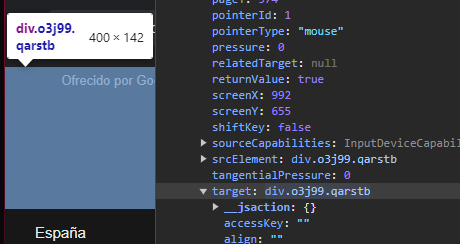
Como por ejemplo:



Si presionamos control mientras clickamos, etc.

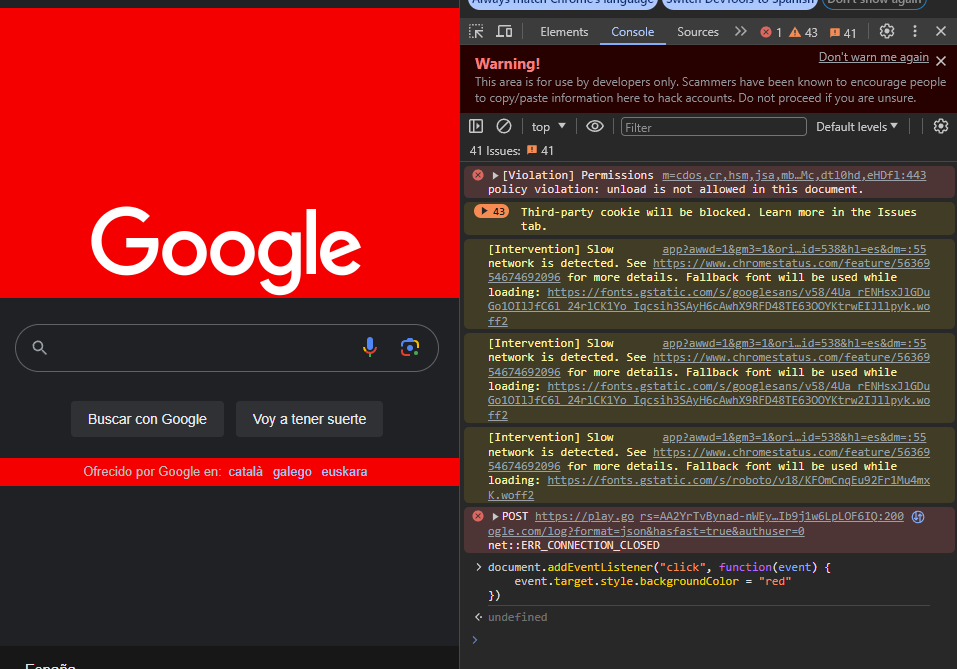
Una de las informaciones más interesantes es el target (donde nos dice DONDE hemos clickado).

Ademas al poner el cursor encima de ese div.o3… nos lo señala en el navegador…



Mediante el evento podemos modificar el navegador como por ejemplo:  


Esto hará que el elemento donde clickemos se ponga de color rojo.



Esto con la información de los target se puede modificar, de esta forma para hacerlo más complejo.

document.addEventListener("click", function(event) {

if (event.altKey){

event.target.style.backgroundColor = "green"

}else if (event.shiftKey){

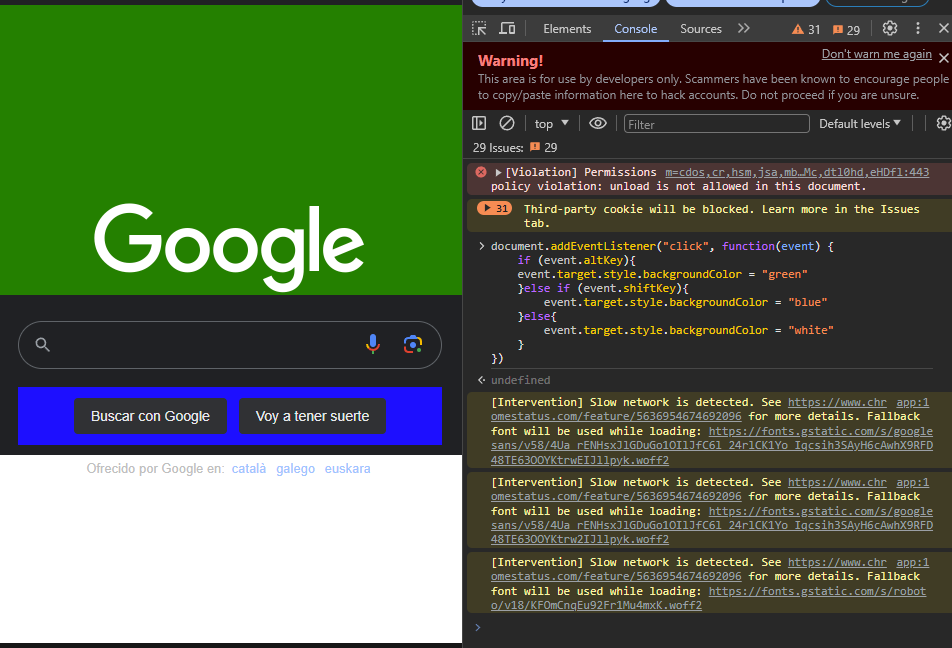
event.target.style.backgroundColor = "blue"

}else{

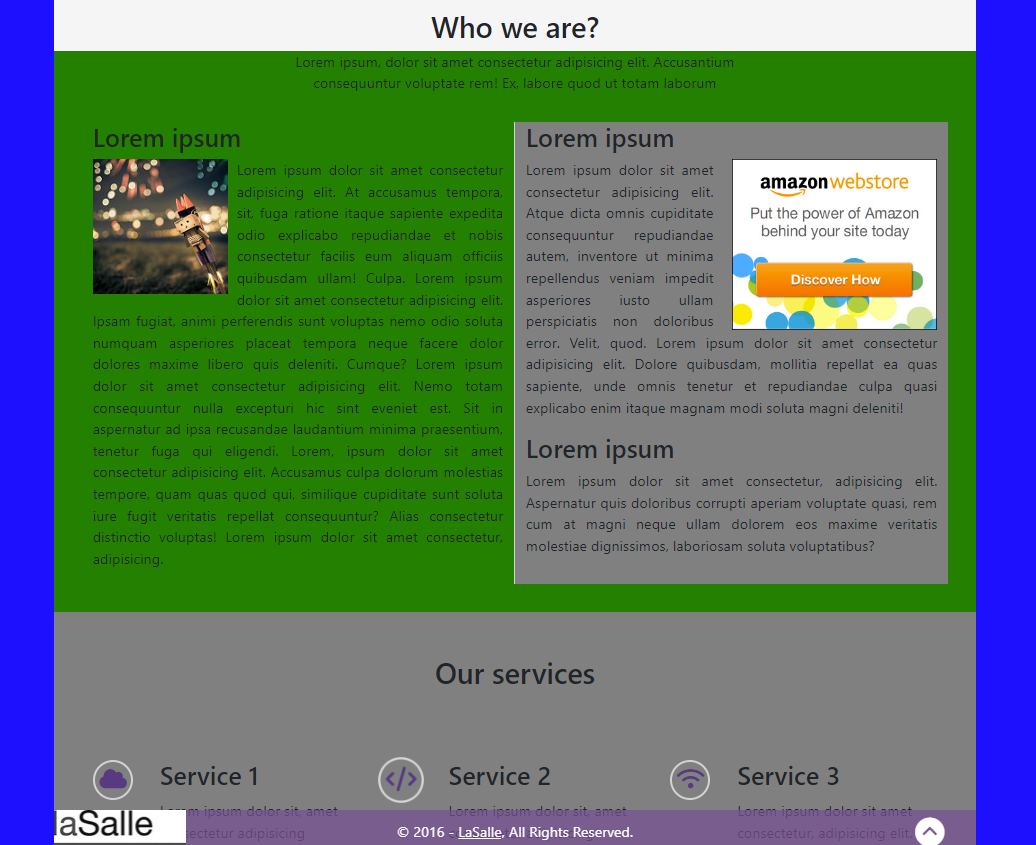
event.target.style.backgroundColor = "white"

}

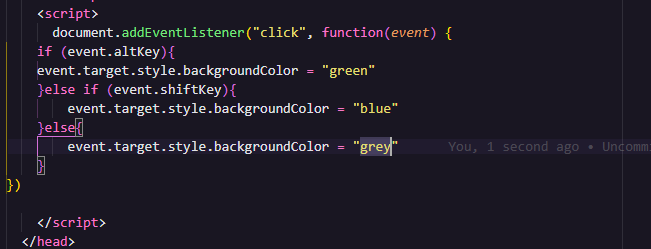
})



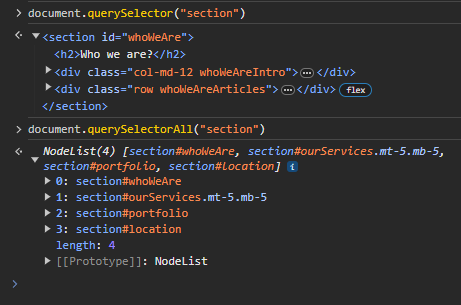
Con esto conseguimos que pinte de distintos colores dependiendo de lo que hagamos mientras clickamos.



Lo podemos implementar a nuestra pagina de bootsrap en nuestro propio código.



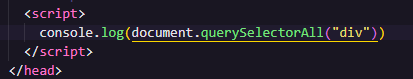
Document se refiere a todo el documento en general. Pero se puede buscar elementos concretos como por ejemplo querySelector / querySelectorAll:



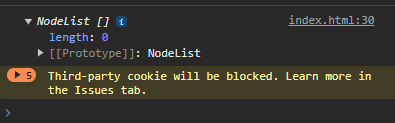
querySelector coge el primer elemento con ese nombre, queryselectorAll coge todos los elementos con ese nombre (en este caso la etiqueta section).

O con getElementById para buscar un ID concreto.

Que pasa si pasamos el código a nuestra aplicación:

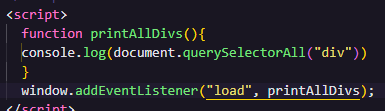


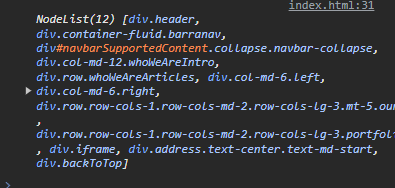
Al cargar la pagina…



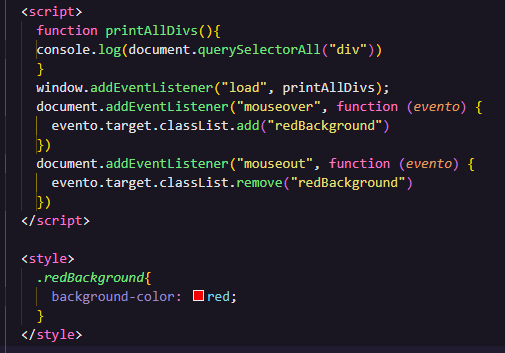
Nos sale el array vacio ya que hacemos el queryselector antes de que cargue la pagina.

Para arreglarlo se puede hacer el evento load…

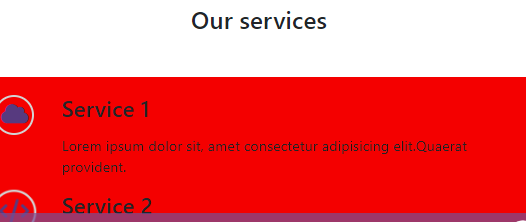




Con window.addEventListener, conseguimos que en el evento load (cuando esté la pagina cargada) que haga la función printAllDivs por lo que ahora si funcionará.



Haremos dos eventos para nuestro trabajo en Bootstrap que hace que cuando pongamos el mouse sobre algún elemento le añada la clase redBackground y definiremos esa clase en css poniendole un background rojo y un evento mouseout para que cuando saquemos el mouse del elemento elimine esa clase.

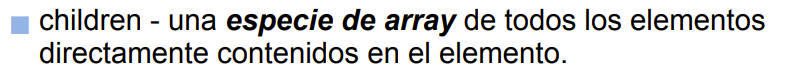


**IMPORTANTE**

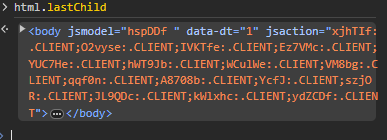
**document**. Se refiere a todo lo que está en el HTML es decir entre etiquetas HTML

**window**. Se refiere a TODA la ventana (o pestaña) del navegador.

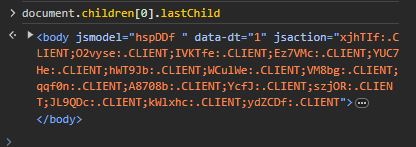
Cada elemento del DOM tiene ciertos atributos:







En este caso pusimos el html en un let html con document.children, pero también se puede hacer así :



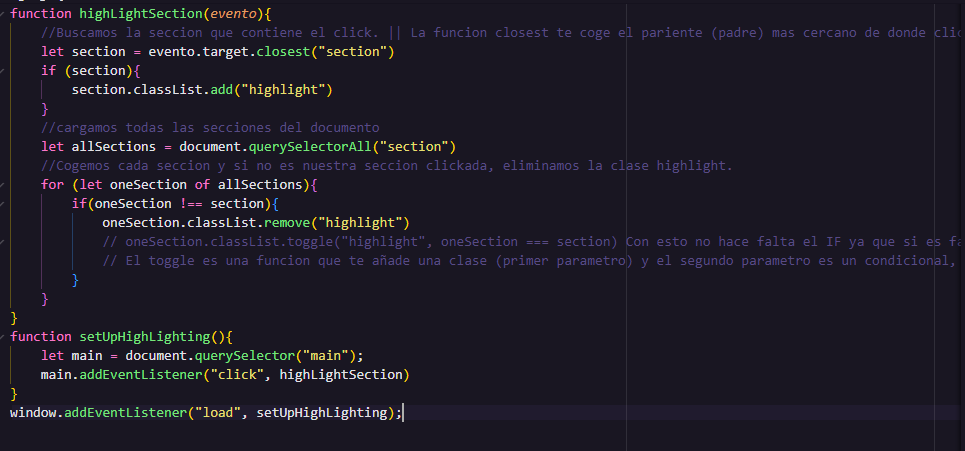
Para encontrar un elemento por su clase se utiliza el document.querySelector(“laClase”)

Eso te encuentra el primer elemento con esa clase, si hay varios y quieres todos document.querySelectorAll("laClase”);

Esto lo buscara de arriba abajo ordenados uno a uno .

Para buscarlos desde abajo hacia arriba (del ultimo al primero) se puede hacer con closest haciendo document.closest(“laClase”).

Closest() también nos devuelve un querySelector.



Con este código hacemos que al hacer click en cada una de las secciones de nuestra pagina de Bootstrap cambie el background del section en el que hacemos click mientras que si al clickar había otra ya con el background cambiado, lo removerá. Esto lo haremos gracias a la clase “highlight” creada en nuestro css:

