**TEMARI DE: ARQUITECTURES I EINES DE PROGRAMACIÓ**

Estábasado en cinco puntos:

1.- Mecanismos de ejecución de código en un navegador web.

2.- Capacitats i limitacions d’execució.

3.- llenguatges de programación en entorn client.

4.-Tecnologies i llenguatges associats.

5.- Integració del codig amb etiquetes HTML.

**MECANISMES**

JS nace en 1995, creado por Brendan Eich, en aquel momento trajaba para Netscape. Javascript tuvo mucho éxito, Microsoft sacó una versión propia llamada Jscript ( para pillar tajada del éxito de Javascript). Mucha gente va trabajando en lenguajes tipo Javascript, para llegar a un entendimiento común y estandarizar los lenguajes de script se llega a ECMA Script.

ECMA Internacional hacen ECMAScript y la primera versión publicada en 1997 es ECMAScript1. Ahora hemos llegado a la versión ECMAScript ES6(del 2015), ES2015.

Para poder interpretar el lenguaje los navegadores incluyen un motor de JS.

Los más conocidos son:

* SpiderMonkey(Firefox)
* V8(Chrome), también lo utiliza Edge
* JavascriptCore (Safari)
* Carakan (Ópera)
* Chackra (Intérprete de JS para configurar Internet explorer).

JS es un lenguaje con un solo hilo de ejecución (singles thread). Esto quiere decir que las tareas se van ejecutando secuencialmente.

En 2008 aparece el motor V8 de Google, supone una gran mejora en los motores de JS. El primero en ser mezcla de interprete y compilador.

**JIT🡪**Técnica que utiliza Google para crear este tipo de motor, el resto de navegadores lo siguen.

**Como funciona JIT?(JUST IN TIME)**

El **compilador** traduce el código fuente a bytecode. Esta traducción la hace lo más rápida posible. El bytecode lo ejecuta el **intérprete**. Este sistema de intérprete y compilador cuenta con algo llamado **profiler**, (que es un mecanismo para optimizar el código con el fin de mejorar el rendimiento).

**Elementos fundamentales para comprensión interna de JS están en amarillo**

**CALL STACK HEAP🡪**

**EVENT LOOP y CALLBACK QUEUE(ES LO MISMO AMBOS CONCEPTOS)🡪**Son funciones que nos informan. Event loop mira a ver si queda algo. Si hay tareas pendientes en **CALLBACK QUEUE** las va acumulando en **CALL STACK HEAP.**

**WEB APIS🡪**Hace referencia a las apis que vienen integradas con el navegador y que están creadas con JS. Estas APIS facilitan la implementación de algunas funcionalidades.

**Alguna de las más conocidas:**

**-DOM: Document OBject Model🡪**Permite que JS acceda a los elementos de una página web, de los elementos HTML. Coge valores y los puede modificar.

**-XMLHttpRequest🡪**Permite hacer peticiones HTTP y comunicarnos con un servidor con otra pagina web, intercambiar información. Por ejemplo: el gadget que está colgado en muchas páginas web donde se puede ver el tiempo, esta conectado a esto.

**-GEOLOCALIZACION🡪**Gps, GoogleMaps. Son apis que nos posicionan.

**-FUNCIONES DE TIMER🡪**Con esto se puede crear cuentas atrás, cronómetros, intervalos de tiempo. Permite manejar tiempo.

**HOISTING:**

Se refiere a un comportamiento que tiene JS por el cual las declaraciones de variables si utilizamos **VAR** para declararlas. **Podemos llegar a utilizar la variables antes de declararla.** Esto asigna una determinada memoria en la fase de compilación de algo que no existe, que es lo malo. Esto se soluciona con **let** , que es lo malo. Esto se soluciona con **let y const.**

Incluso se puede hacer referencia a una función que no existe.

**29/09/2023**

**PUNT NUMERO 2: CAPCITATS I LIMITACIONS D’EXECUCIÓ**

Javascript es un lenguaje que es interpretado por el cliente. No se ejecuta ni se interpreta en el servidor. JS es un lengauje cliente. Existen multiplies clientes, en nuestro caso navegadores que soportan JS; Firefox, GoogleChrome, Safari, Opera, Edge.

Hemos de estar seguros de que nuestro código se ejecute en todos loas navegadores.

**Limitacions🡪**No todos los navegadores soportan lenguajes de script en el lado del cliente. Los lenguajes de lado servidor no suelen dar este tipo de problemas. Algunos dispositivos móviles tampoco pueden ejecutar JS.

Se puede desactivar las opciones de JS en las opciones del navegador.

**PUNT 3: LLENGUATGES DE PROGRAMACIÓ EN L’ENTORN CLIENT**

Cuando hablamos de lenguajes de programación en clientes web, podemos distinguir dos lenguajes; aquellos que nos permiten dar forma y estilo a una página web como son **HTML y CSS(formato y estilo),** y por otro los que le aportan dinamismo **(lenguajes de scripting) por ejemplo Typescript, Javascript, etc.**

**Lenguajes de programación web del lado servidor;**

**1.- CGI**

**2.- PERL**

**3.- ASP**

**4.- PHP**

**5.- JSP**

**Lenguajes de programación web del lado cliente;**

**1.- HTML**

**2.- JS**

**3.- APPLETS DE JAVA**

**4.- VBS(Visual Basic Script)**

**5.- FLASH**

**6.- CSS**

**Carácterísticas de JS**

Está orientado a dar soluciones a conseguir que nuestra página web interactue con el usuario, esto lo hará con elementos por ejemplo como los **formularios** o los **links.** Nos permitirá trabajar con grupos de datos y proporcionarnos una interfície para estos datos. También permite controlar múltiples ventanas, pluggins, marcos de navegación, y preprocesar datos en el lado cliente, antes de enviarlos al lado servidor. También podemos modificar estilos y contenido en los navegadores, mediante código JS podemos ir añadiendo elementos como cajas de texto o imágenes (podemos manipular tb estas imágenes). También podemos solicitar ficheros al servidor, y enviar peticiones de lectura y escritura a los lenguajes de servidor.

**Compatibilidades de JS**

**Seguridad**

Javascript no podrá realizar ninguna de las siguientes tareas;

No puede modificar o acceder a las preferencias del navegador del cliente. Es decir, desactivar por ejemplo o activar JS no se puede hacer por código.

Tampoco puede cambiar la apariencia de la ventana principal. No se puede imprimir desde JS, ni las capacidades de impresión.

No se puede acceder tampoco a los botones de acciones que tiene el propio navegador. No se puede abrir por ejemplo el menú del navegador ni cosas así.

No podemos ejecutar una aplicación del ordenador del cliente, no puede ejecutar un programa de por ejemplo la calculadora de Windows.

No podemos leer ni escribir ficheros o directorios en el ordenador del cliente, con una excepción, la de las cookies.

No puede JS escribir ficheros que estén en el servidor, podrá solicitarlos, pero no puede escribir en ellos si están en el servidor.

No puede capturar datos procedentes de una transmisión en streaming de un servidor para su retransmisión.

No puede enviar emails a nosotros mismos de forma invisible sobre los visitantes a nuestra pagina web.(Captura de datos cuando el usuario introduce sus datos) Aunque si se podría enviar los datos al servidor y el servidor enviar esos correos.

Javascript es incapaz de proteger el origen de las imágenes de nuestra página.

No puede implementar multiproceso o mutitarea.

**PUNT 4: TECNOLOGIES Y LLENGUATGES ASSOCIATS:**

Cosas a tener en cuenta para elegir un editor; codificación por colores. Verificación de sintaxis. Si genera parte del código genial.

**MANERAS DE LINKEAR UN HTML CON UN JS:**

En el head, en el body, debajo de todo el código html que generemos o incluso en el body con la tiqueta **script** picando directamente el código JS en el documento HTML, aunque esta forma es la menos recomendable.

La mayoría de las veces el script debe de ir justo antes de cerrar el body. Cuando se anda con evenetos tiene que estar debajo para que se lean antes las etiwuetas.

**3ª MANERA: CODIGO EN LINEA**

        <a href="#" onclick="console.log('Boquepasa!');">Haz click aquí!</a>

Se puede insertar código Javascript en línea, es decir, directamente en el **onclick=”” Lo que se ponga dentro debe de ir con comillas simples para que no genere problemas.**

En el href la almohadilla significa que se queda ahí mismo, así no da problemas.

**VARIABLES**

Las variables se utilizan para almacenar datos. Un dato es un registro de algo que podemos percibir.

Se pueden definir de tres maneras, **var, let** y **const.**

**A const sí hay que darle un valor.** A var y let no hay porque asignarle un valor, simplemente se pueden declarar.

Porque **const** es un valor constante, así que ya de entrada hay que darle un valor. Si se define un **array const SÍ** se puede cambiar lo que hay dentro.

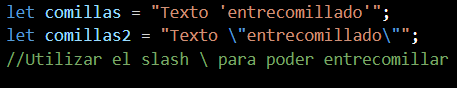
Las variables declaradas con var tienen un alcance de función (en princio, porque luego si se puede). Las de let y const tienen un alcance de bloque, si omitimos la palabra let o var la convertimos en global.

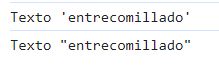
Javascript no exige punto y coma pero es buena practica ponerlo, porque asi se delimitan instrucciones.

let uno = 1; let dos = 2;

**Strings**

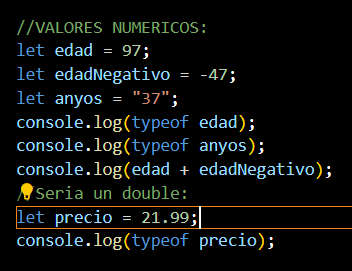
Se pueden declarar utilizando comillas simples o comillas dobles. Para entrecomillar:

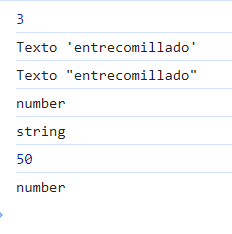




**NUMEROS**

Nunca llevan comillas. Si llevan comillas lo lee como si fuera un string:





**Infinity** es un numero infinito que se representara con un **typeof** como **Number.**

**TIPO DE DATO NULL**

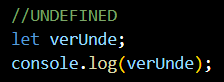
Es un valor existente, que indica algo que sí está definido, pero su valor está vacío o directamente se le ha dado un valor nulo. Puede ser por ejemplo un array vacio o directame a ente:



Si le hacemos un typeof a esto será un **object.**

**UNDEFINED**

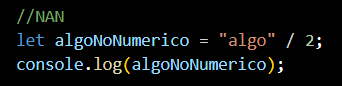
Se refiere a algo que ni tan solo ha sido definido. Caso mas obvio declarar una variable pero no asignarle un valor:



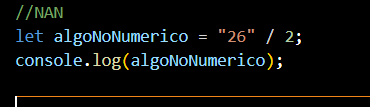


**NAN🡪NOT A NUMBER**

Cuando tratas de usar como numero algo que no lo es.

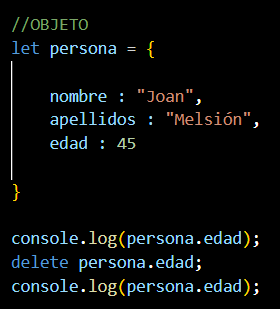
 

Así sí lo coge, porque hace una conversion:

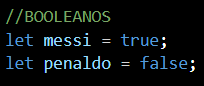


**DELETE**

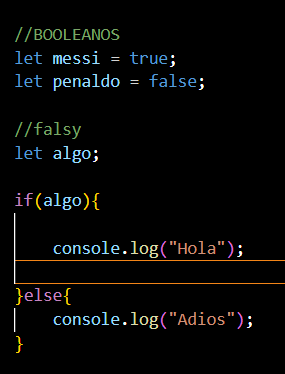
Se utiliza para eliminar características de objetos:



**BOOLEANOS**

Tambien denominados **falsy. ** Javascript tiene una colección de valores que también son **falsy**  y si se hacen comparaciones contra los valores **falsy** los va a tomar JS como false.

Javascript considera falsy una cadena vacía, NULL, undefined, el 0 y Nan.

 Esto es lo que escribirá.

**OPERADORES ARITMETICOS**

Operaciones matemáticos. Estan los de asignación:

+= a = a+ b;

-= a = a – b;

\*= a = a \* b;

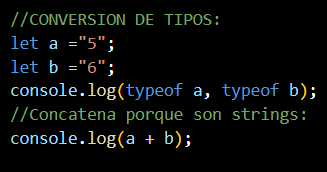
Y asi sucesivamente también para división y para el módulo.

**CAMBIO DE SIGNO**

Ejemplo: a = 5, b = -a, b= -5

**CONVERSIONES DE TIPOS**

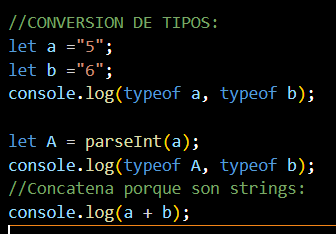
Hay varias maneras de hacerlo:

 Esto da resultado de string.

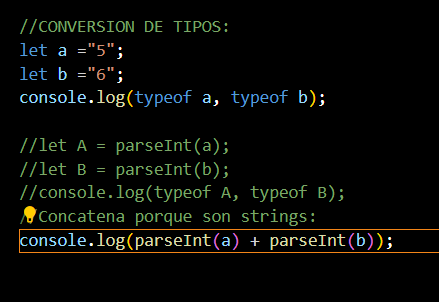
Cuando se recupera valores de formulario siempre se recupera el valor como un string. De los inputs salen strings, que sumados se concatenarían.

Para trabajar con eso hay que convertirlo a números. **Typescript sí que debe definir los tipos, si son numbers, etc ya desde el principio.**

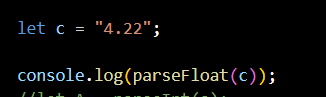
**PARA CONVERSION PODEMOS UTILIZAR🡪parseInt asignando el valor parseado a una nueva variable.**

****

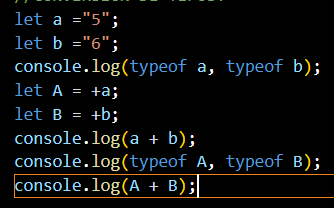
Esto pasaría el tipo de dato string a un entero. Si solo se parsea la a b seeguiria siendo un string y por tanto concatenaria las variables, no las sumaria.

****

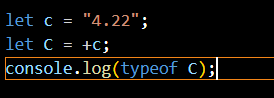
**En el mismo contexto existe el parseFloat:**

****

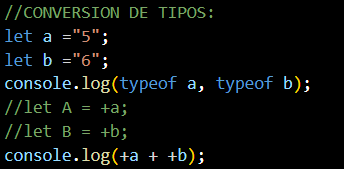
**Otro tipo de conversión de numero entero seria el símbolo +**

****

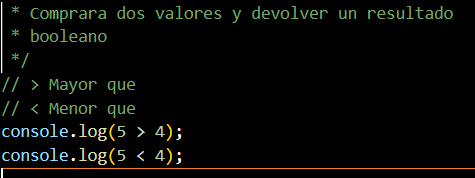
**Funciona tanto en integer como en float:**

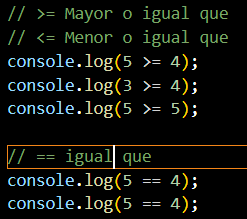
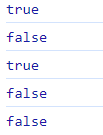
****

**Podriamos hacerlo directamente en el resultado del console log**

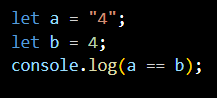
****

**OPERADORES DE COMPARACION**

Siempre devolverá true or false;

Esta comparación compara si dos valores son iguales. Es decir, por ejemplo:

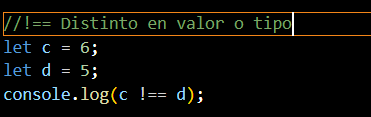
 Aunque uno es string y otro number al ser los dos 4 da **true.**

**Si uno fuera un cinco seria false.**

**DINSTIGUIR ENTRE NUMBER Y CADENA**

****

** Esto da false.**



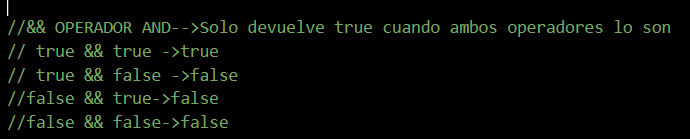
**EXPRESIONES BOOLEANAS:**

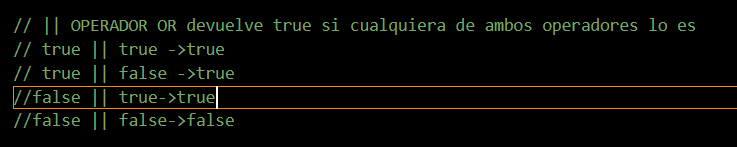
Permiten comparar expresiones booleanas con las que se construyen condiciones

 \* que se pueden aplicar en funciones, bucles, etc

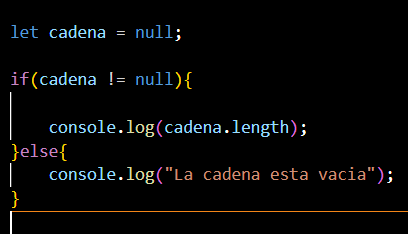
//&& OPERADOR AND-->Solo devuelve true cuando ambos operadores lo son

Se tienen que cumplir ambas condiciones.

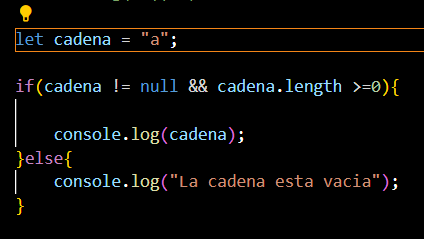
****

****

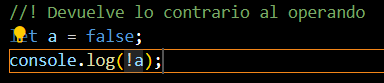
**COMPARACIONES CONTRA NULL, PARA COMPROBAR ELEMENTOS VACIOS:**

****

Siempre hay que mirar cuando llega info de un formulario o cuando npo sbe sabve si hay info eb una cadena si es null o esta vacia.

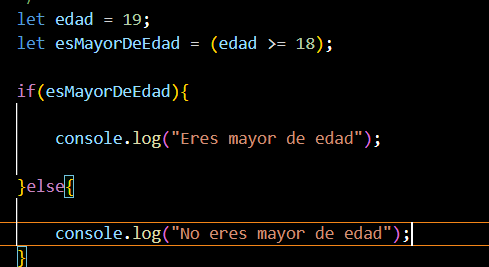


**NEGATIVO:**

****

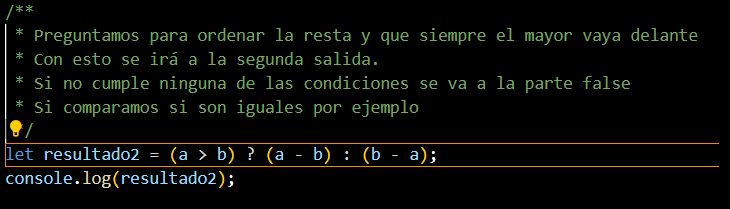
**ESTRUCTURAS DE CONTROL:**

**Condicionales IF**

****

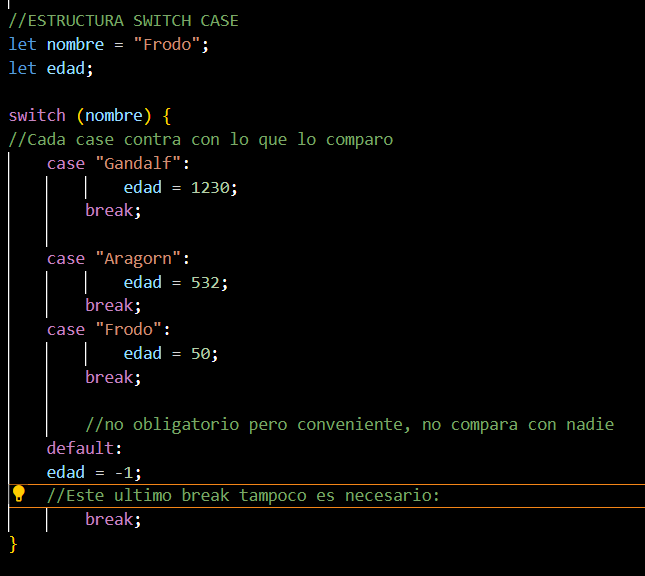
OPERADOR TERNARIO🡪Sustituyen en algunos casos a los if

?-->pregunta, :🡪resultado



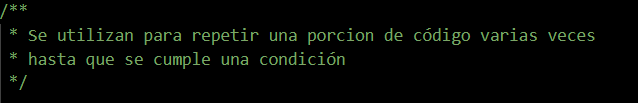
**SWITCH CASE:**

Cuando se quiera hacer una estructura condicional según el valor que tenga una variable o expresión. Nos permite utilizar más que números en los casos.



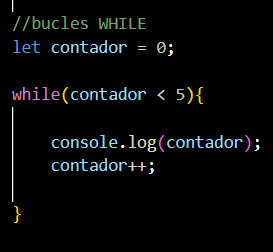
Sale Frodo al darle el valor por defecto.

**ITERACIONES🡪BUCLES**

****

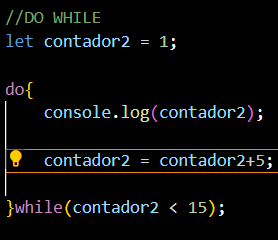
**Bucle While**

Repetir un bloque de código hasta que se cumpla una condición.

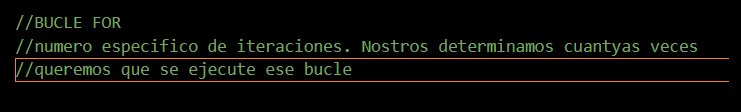


**Do while**

**S**e utiliza para repetir un bloque de código almenos una vez.



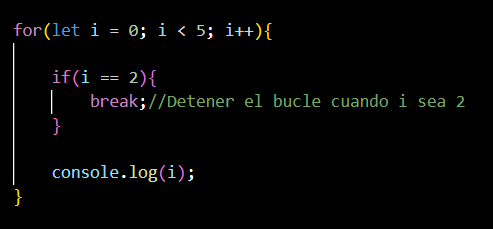
**Bucle For**

****

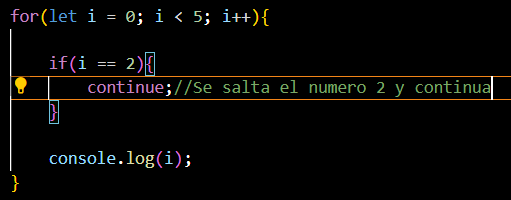
**Pasar saltar alguna iteración y luego continuat: break i continue**

Break se utiliza para detener la ejecución del bucle en cuanto entre en un break parara

El continue es un stop and go.El continue se utiliza para saltar a la siguiente iteración.



**CONTINUE:**



**09/octubre/2023**

**UNITAT 3**

**Como trabajar con fechas y tiempo**

**P#fecha-->**crea párrafo con id que se le indique

**p.fecha🡪**le da la clase.

//Acceder al parrafo

//getElement byusca el elemento que se le pasa por parametro

//El punto para accdere a las propiedades, con el innerHTML se accede al escrito en si

//La diferencia entre coger id y coger class es poner el . por parametro o no

let parrafo = document.getElementById('fecha');

//Accedemos a la propiedad innerHTML

/\*\*

 \* DATE es un constructor que devuelve un string.

 \*/

//parrafo.innerHTML = Date();-->Devuelve fecha actual

//Hay cuatro maneras diferentes de inicializar un objeto de tipo fecha:

// new Date();

//new Date(milisegundos);

//new Date(fecha formato string);

//let fecha = new Date(7778999);

//parrafo.innerHTML = fecha;

//Resultado de esto: Fri Jan 02 1970 02:41:40 GMT+0100 (hora estándar de Europa central)

//let fecha = new Date("09/09/2023");-->El primero el mes, dia y año

//let fecha = new Date("Aug 16 2023 GMT+0200");-->Formayo completo

//let fecha = new Date("2019-09-18");-->Formato iso, año, mes, dia

//new Date(año, mes, dia, horas, minutos, segundos, milisegundos);

//Introducir fecha

let fecha = new Date(71, 12, 29, 15, 15, 0, 0);

parrafo.innerHTML = fecha.getTime();

/\*\*

 \*

 \* METODOS DEL TIPO FECHA:

 \*/

//getDate();-->da el dia del mes, va del 1 al 31

//getDay()-->Dia de la semana en formato numerico 0 al 6

//getFullYear()-->Obtiene el año

//getHours()-->obtiene las horas

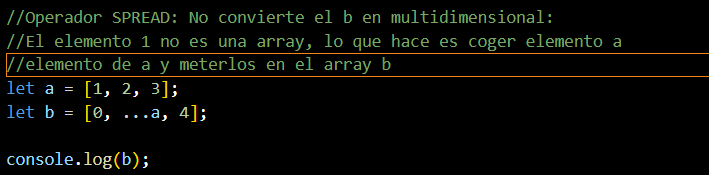
//getMilliseconds(), getMinutes() -->Se le pueden sumar cosas cuadno

//se opera como number cuando se introduce como number

//getMonth(), getSeconds(),

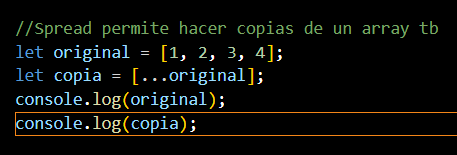
//getTime()-->da la hora en milisegundos, empieza por el dia 1 del año

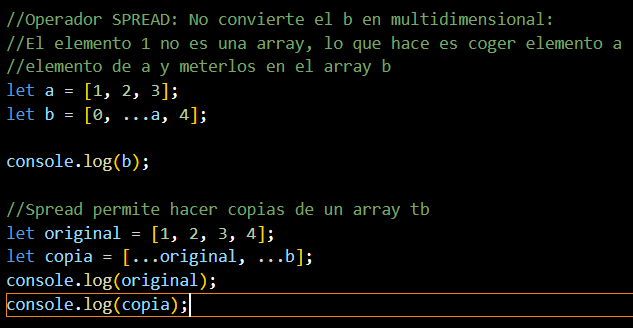
**UNIDAD 4**



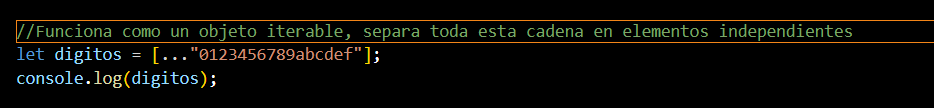
Se puede concatenar en la posición que sea.

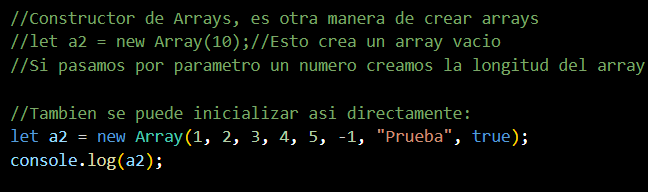
Tambien permite hacer copias:

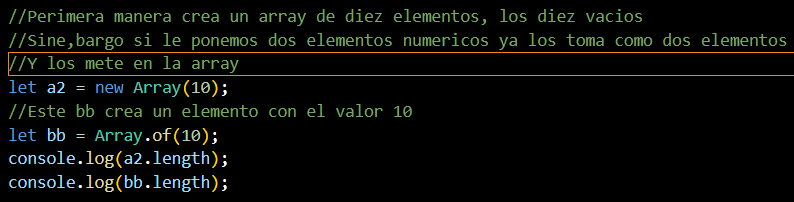




Aquí se ha hecho una copia, dentro de otra copia y otra copia.



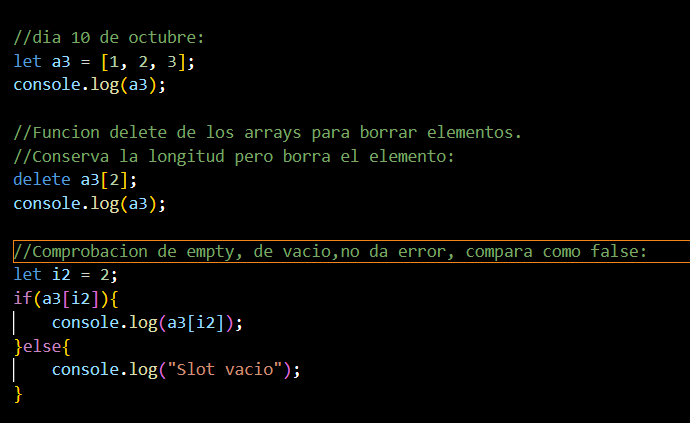




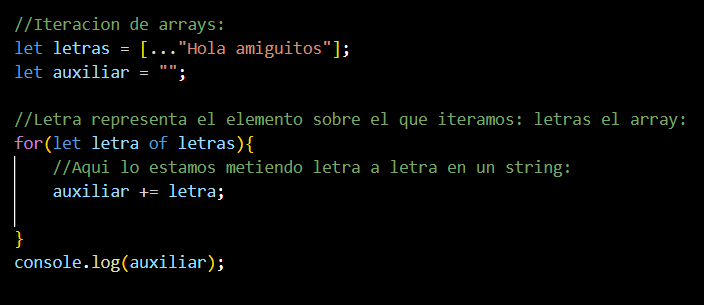
METODO PUSH🡪Metodo moderno para subir un elemento al array. Lo añade en la ultima posición. Como si hiciéramos🡪array[array.length]

10/10/2023

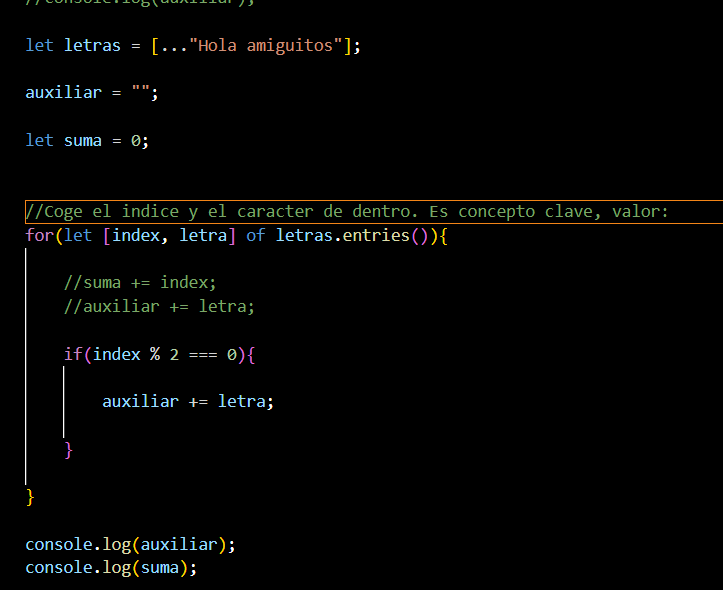
**METODO DELETE:**



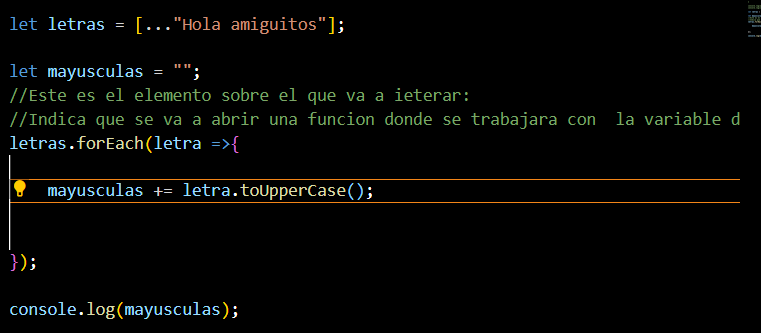
Iteracion:



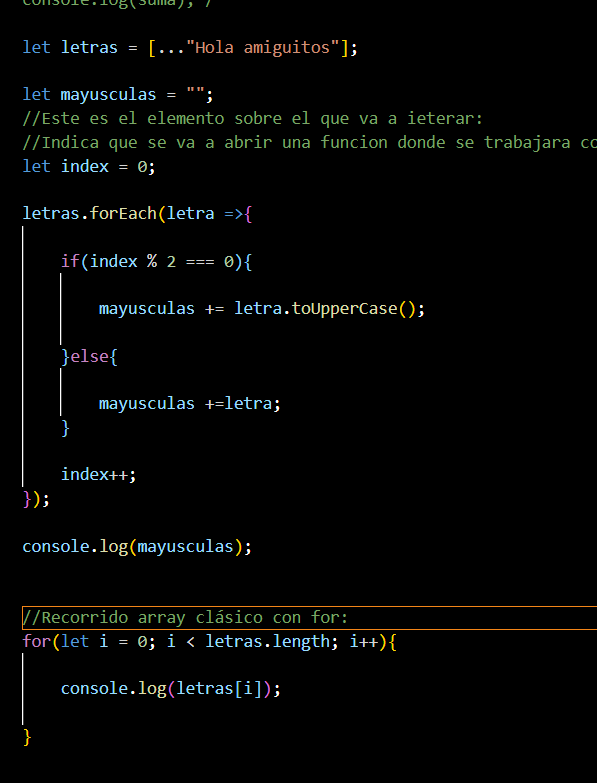
**Iterando con entries() e index y elemento guardado:**



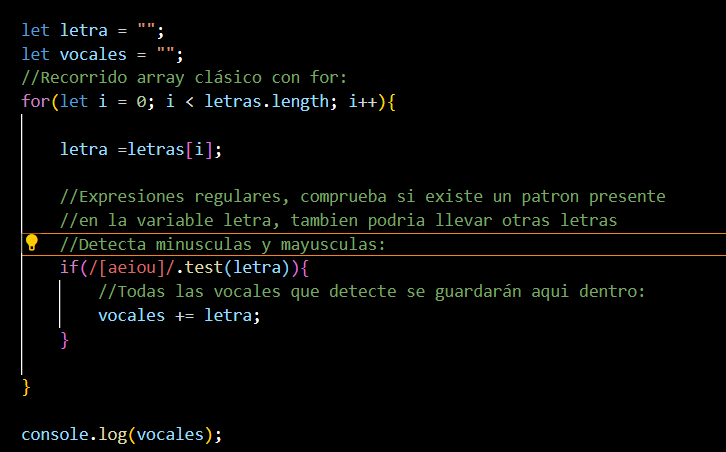
**Bucle forEach:**

****

**Bucle for clásico, y poner en mayúsculas los índices pares y en minúsculas los índices impares:**

****

**EXPRESIONES REGULARES: DETECCION DE PATRONES:**

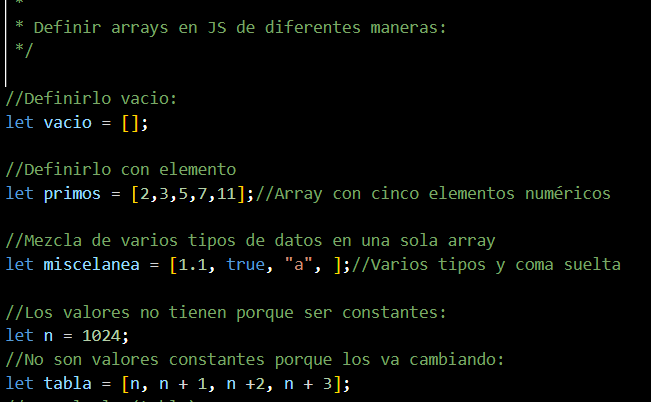


Webs de interés🡪https://medium.com/@jgratereaux/validar-correos-electr%C3%B3nicos-con-expresiones-regulares-7914751b6018

<https://es.stackoverflow.com/questions/453176/como-validar-correctamente-un-email-con-expresiones-regulares>

**ARRAYS JS**

**Definir arrays:**



**IMPORTANTE: Los arrays no petan aunque esten vacíos pero sí petan si son null:**

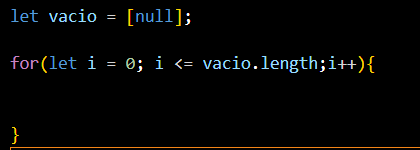
Si los definimos así: let vacio = []; y luego los recorremos con un bucle for o directamente los imprimimos con un console.log los arrays no petarán.

Sin embargo si los definimos como **null** no podemos imprimir su **length** porque se considera nulo (la impresión del array vacio daba 0), pero con null sí peta. Pero si podemos hacer:

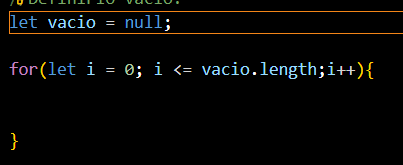


Por tanto lo primero que tenemos que hacer es ver si es null, porque si intentamos hacer un **.length** a un array vacio entonces petará.

**Diferencias a declarar null el array a declararlo con un elemento null:**

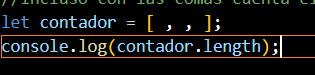
****

Así no peta porque considera que ya tiene un length, dado que hay dentro del array un elemento aunque sea null. Pero si hacemos esto:

****

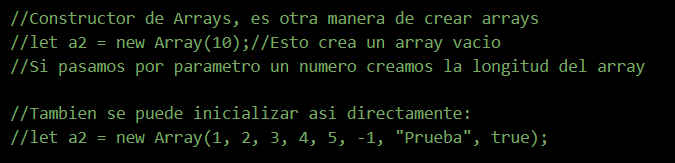
Entonces sí que peta, dado que el array es declarado null, no tiene length.

Incluso si hacemos esto:

****

Contaria los elementos, el espacio de detrás de la coma no lo cuenta.

**Constructor de arrays:**

****

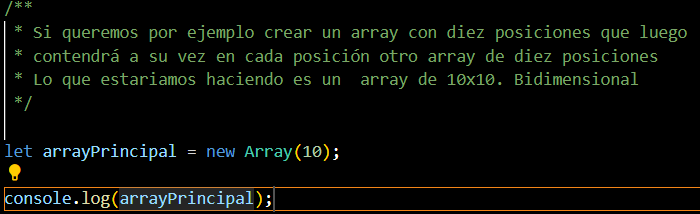
La priemra manera crea un array vacío que tendrá diez posiciones, pero también se puede inicializar directamente. Aunque se le diga que tiene diez, luego se puede ampliar tanto como queramos.

Se podría crear un array con un solo elemento, pero tendría que hacerse así:

 Para que interprete que le estamos diciendo que nos cree un array con un valor 10, en lugar del anterior que era crear diez posiciones vacías.

**Arrays multidimensionales:**

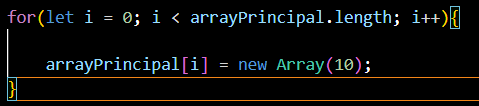
Los elementos del array por si mismos son otros arrays. Pasos a seguir para crear un multidimensional:



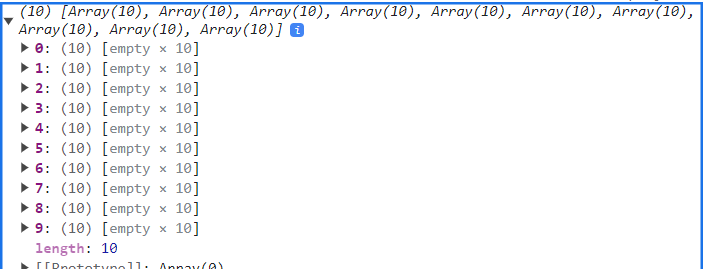
Esto devuelve:

Un array de diez posiciones. Todas ellas vacías.

Ahí será donde crearemos nuestras 10 arrays de 10 posiciones, que serán las columnas digamos de la fila principal.

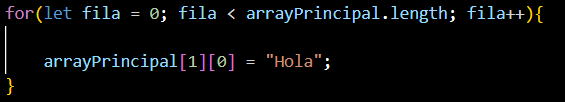


El resultado de esto es que hemos creado diez arrays vacías:

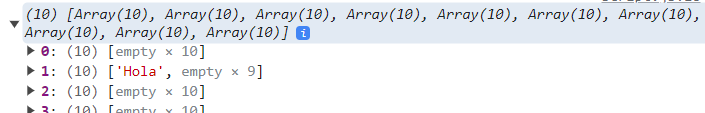
Ahora en cada posición del array principal hay otro array.

Por tanto acabamos de crear un array bidimensional, con filas y columnas.

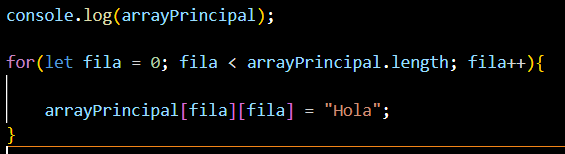
Para acceder a una posición en concreto de las columnas, que son los arrays que hemos creado. Pongamos queremos acceder a una posición del segundo array:

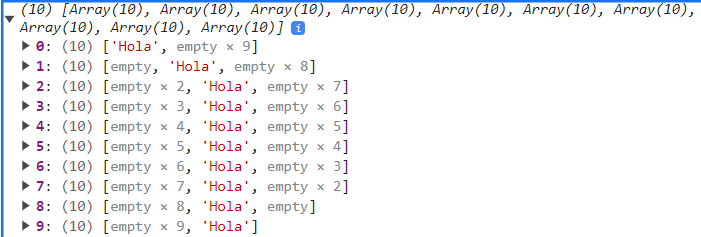


Esto nos daría:

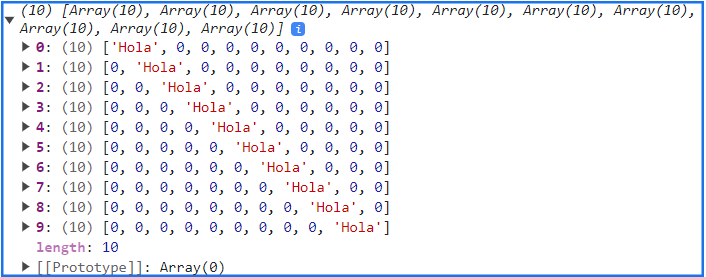
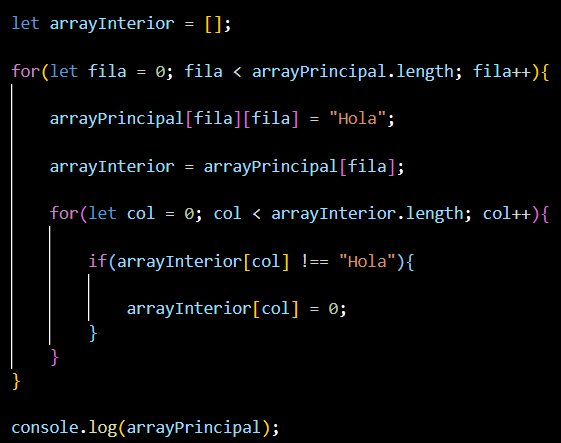


Para entender un poco mejor el comportamiento del array, ahora tiene filas y columnas, y esta es la manera en que se interpretan, si queremos meter u registro en todas las filas leera el numero de fila y hará lo siguiente:

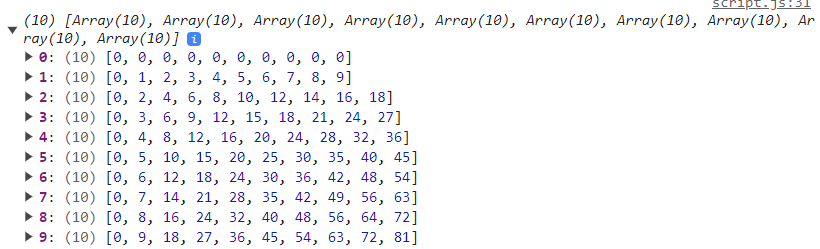




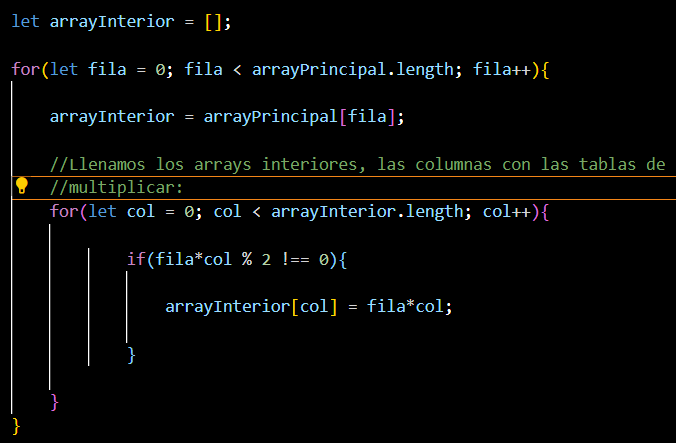
En realidad, la estructura de la fila, que es el array principal es la siguiente cuando llenamos las columnas con un registro diferente para ver realmente la forma que tiene el array:



Pongamos que queremos llenar todo el array con las tablas de multiplicar:



Lo único que hacemos es multiplicar fila \* col. Y si queremos saltar los pares por ejemplo:



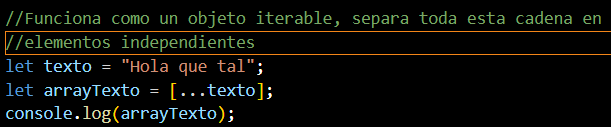
**OPERADORES, MÉTODOS DE LOS ARRAYS**

**Spread()🡪**son los tres puntitos **…** Sirve para:

* Puede coger elemento a elemento de un array y meterlo dentro de otra. **No mete el array, sino sus elementos de forma independiente:**
* let a = [1, 2, 3];
* let b = [0, ...a, 4];
* console.log(b);

Esto da como resultado del console🡪 [0, 1, 2, 3, 4];

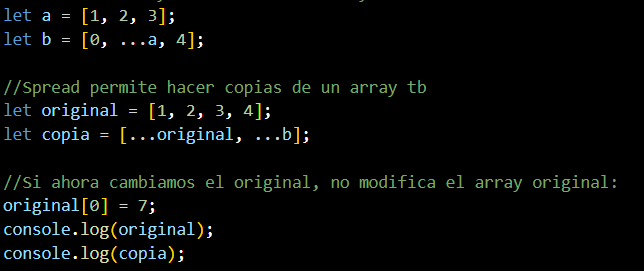
* Tambien puede servir para convertir una cadena de texto en un array de chars de dos maneras diferentes, o cogiendo un string que ya tenemos:



O metiendo un string directamente en el array:

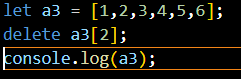


* Tambien sirve para hacer copias de arrays:

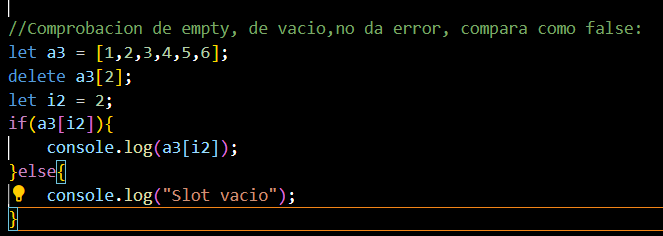


**Push()🡪**Se puede poner más de un elemento a la vez. Siempre lo pone al final.

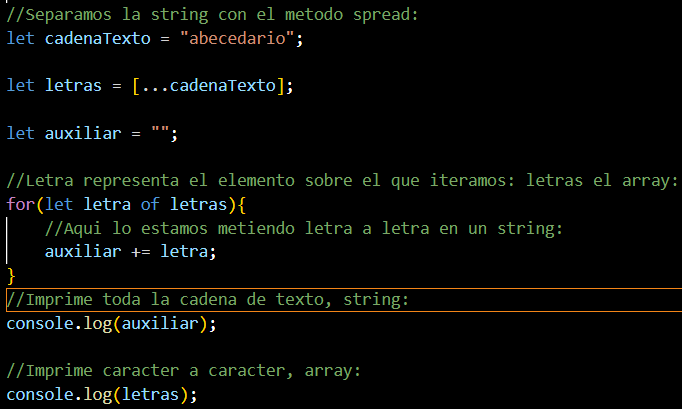
**Delete()🡪**Para borrar los elementos del array. Conserva la longitud pero borra los elementos.Ejemplo:

**Comprobacion de vacío, no da error, compara como false:**



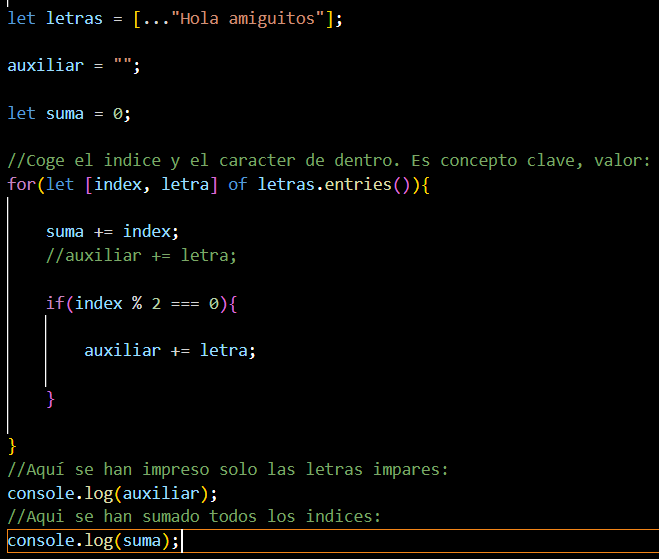
**Iteración de strings y concatenación:**

****

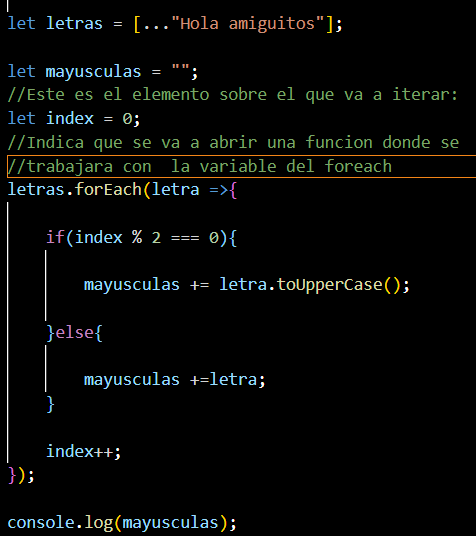
****

**Sacar índice y contenido a la vez con el bucle for(let [index, elemento] of arrayAIterar.entries():**

Es una desestructuración del bucle for donde por un lado obtenemos el índice(la posición en el array del elemento) y por otro el valor, el contenido de ese elemento. Concepto clave-valor que proporciona el método entries().

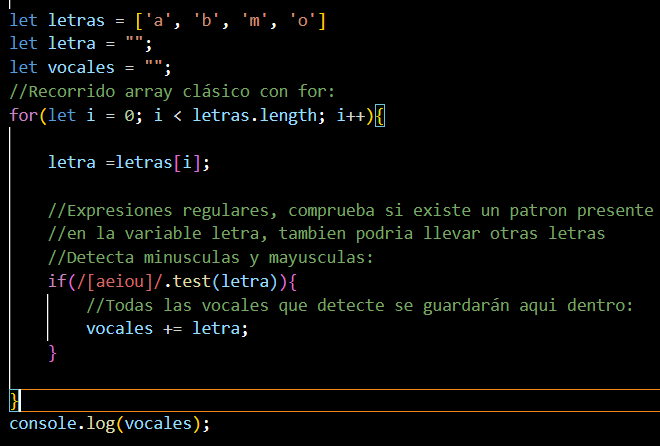


**Iteraciones con forEach, utilizando un índice:**

****

En cada iteración letra representará un elemento del array de letras.

**Iteraciones con bucle for clásico:**

****

**11/10/2023**

**Eliminar elementos de un array pero devolviéndolos(es decir, se pueden almacenar los destruidos en una variable)**

**Pop()🡪**Elimina el último elemento del array y LO DEVUELVE, además NO QUEDA EL HUECO EN EL ARRAY. Si tenemos un array con tres elementos pasa a tener dos.

**Shift()🡪**Elimina el primer elemento del array y LO DEVUELVE.

**Agregar elementos al final o al principio:**

**Push()🡪**Siempre agrega al final del array, se pueden agregar uno o mas.

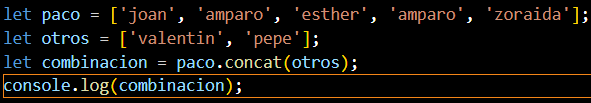
**Unshift()🡪**Agrega al principio del array. Se pueden agregar uno o más elementos.

**NOTA: El método splice() puede ser usado para agregar elementos a continuación de donde queramos. Recibe 3 parámetros, índice a partir del cual empieza a trabajar, el segundo número de elementos a eliminar (que si lo ponemos a cero es cuando no elimina pero agrega), y 3 los elementos que le queremos agregar al array. Para usar splice() solo para agregar sin eliminar:**

****

**Concatenar arrays:**

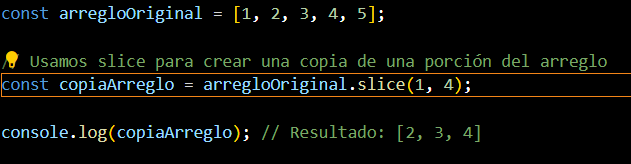
**Concat()🡪**Combina dos o más arrays y devuelve un nuevo array. Se puede concatenar como se quiera variando el orden. No modifica los originales.





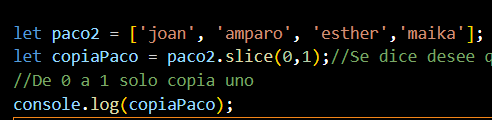
**Crear copias de arrays:**

A parte de poder hacerlo con spread se puede hacer con **slice()** que crea una copia del original, una copia superficial. Si modificamos la copia no modificamos el original. Se utiliza slice() normalmente para hacer una copia de una PORCION del array original.



El primer parámetro es INCLUSIVE, es decir, incluye el numero de ese índice, y el segundo es EXCLUSIVE, es decir, exluye el numero de ese índice.

Para copiar solo uno:



Solo copiaría Joan.

**Splice()🡪**Cambia el contenido de un array eliminando, reemplazando o agregando elementos.

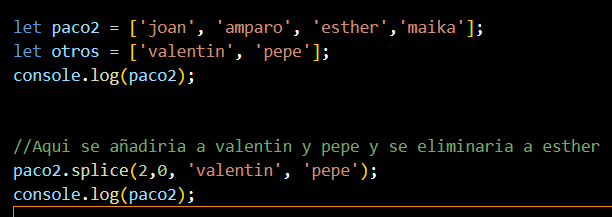
**Si solo se le proporcionan dos parámetros:**

Eliminará la cantidad de elementos que especifiquemos. El primer parámetro será el índice a partir del cual empezará (ese índice es INCLUSIVE, afectará al elemento que especifiquemos, si queremos eliminar el mismo elemento se hará poniendo un 1).

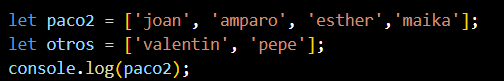
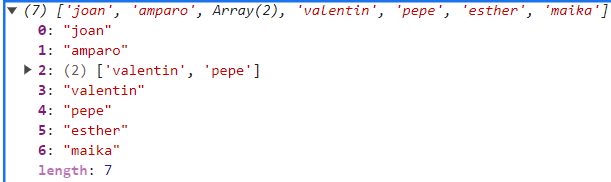
**Si se proporcionan 3 parámetros:**

Además de lo descrito arriba, el tercer parámetro agregará en la posición del índice a partir del cual le hemos indicado mediante el primer parámetro, los elementos que queramos meterle al array.

**Agregar sin eliminar, ponemos el índice en el cual queremos agregar los elementos y como segundo parámetro dejamos un 0:**



**Añadir arrays completos:**

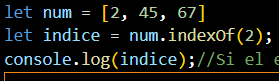
**** **** 

**Añadir arrays elemento a elemento mediante el método spread:**

****

**indexOf()🡪**Averigua si algo existe dentro de un string o un array. Devuelve el PRIMER ÍNDICE si encuentra el elemento que le pasamos por parámetro, sino lo encuentra devuelve -1. Esto se puede utilizar luego en condicionales, comparando en el condicional si el valor de indexOf es distinto a menos uno, y si lo es seguimos adelante. SOLO devuelve la primera aparición del paráemtro. Ejemplo:

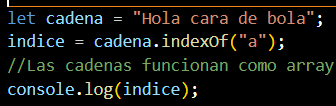
**Números:**

 Aquí nos devolverá 0, que es la posición que ocupa el elemento que estamos buscando.

**Strings con indexOf() tanto en strings como en arrays:**

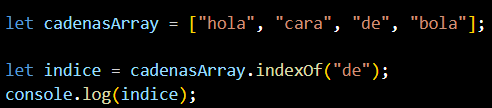
Si es una cadena de texto buscaremos por letra, si deseamos buscar una palabra completa debemos antes pasar la cadena de texto a array.

Las cadenas de texto en este método funcionan como arrays, por tanto, buscando una letra:



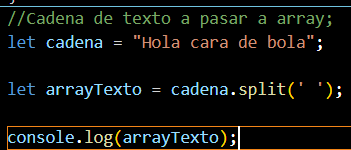
Aquí lo que se busca es por letra, no podemos buscar palabras de esta manera.

**Para que funcione buscando palabras debemos de tener las palabras en un array:**

****

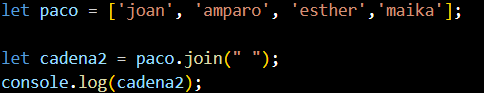
Así sí podemos buscar la palabra entera.

**Pasar cadena de texto a array:**

****

**lastIndexOf()🡪**Devuelve la útlima posición del elemento. Devuelve -1 si no lo encuentra. Es como indexOf() solo que funciona a la inversa.

**Join()🡪**Une todos los elementos de un array en una cadena de texto (utilizando un separador especifico). Podemos ponerle el separador que queramos, si lo dejamos vacio con un espacio simplemente espaciará la cadena:



Le podemos poner el que queramos de separador:



Se podría utilizar para generar contenido html dinámico. Para insertar texto en inputs por ejemplo. (Mirar esto).

**Sort()🡪**Ordena los elementos de un array alfabéticamente si son cadenas, numéricamente si son números.

**Nota importante sobre sort():** Numéricamente por defecto si utilizamos solo el nombre del array y le hacemos un .sort() como coge el primer numero si tenemos:

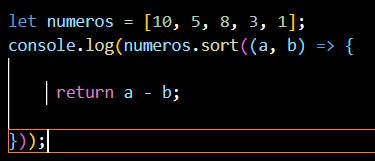


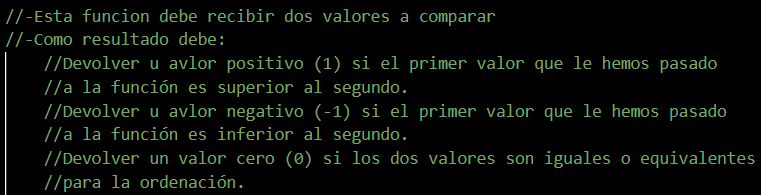
Lo que nos hace es:



Y si tenemos un array de cadenas de texto, lo que hace es que si hay alguna en mayuscula siempre la ponga antes, porque en ASCII las mayuscyulas tienen mas valor que las minúsculas.

**Por tanto para ORDENAR CORRECTAMENTE CON EL MÉTODO SORT, en caso de numéricos:**

**** hay que usarlo como una función que recibe dos parámetros, que representarán los elementos del array que queramos ordenar.



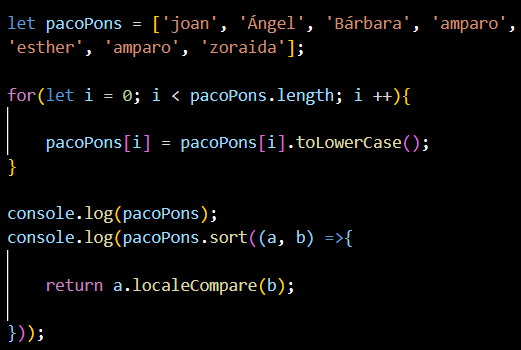
Al final es un booleano lo que hace, devuelve true si el primero es mayor que el segundo, false si es al revés. 0 si son iguales.

Para ordenar al revés simplemente invertir el orden de los parámetros: **b – a**

**Esto que hemos hecho aquí arriba puede resumirse en la siguiente función flecha:**

****

**ORDENAR EN ARRAYS DE STRINGS:**

****

**NOTA IMPORTANTE:** Pasar a minúscula siempre antes de comparar!

**Patrones:**

Muy necesarios y útiles para detectar ciertas cosas en cadenas de texto, emails, etc. Se utilizan con expresiones regulares y el método **test()** que recibe como parámetro la cadena donde queremos evaluar si existe esta expresión regular. Si queremos evaluar algo que esta dentro de un array podemos usar antes **toString() o join()** para pasarlo a cadena.

Estos son algunos de patrones para usar en expresiones regulares:

Detectar si hay **vocales 🡪** /[aeiou]/

**Detectar si hay espacios en blanco🡪 / /g**

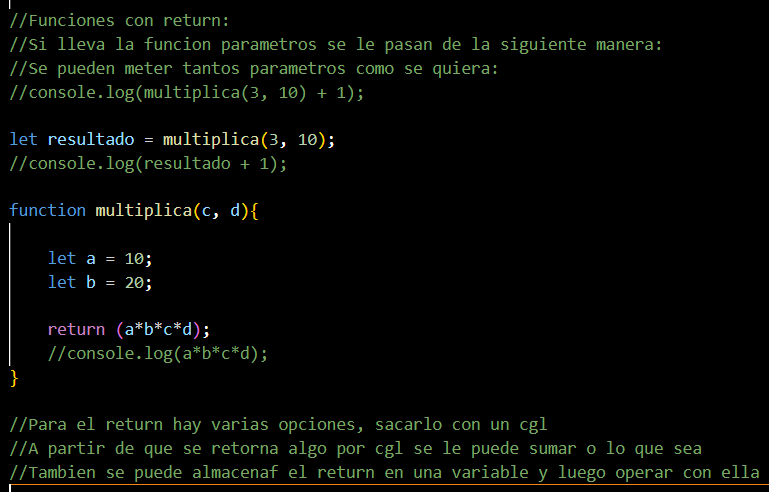
**Detectar si la cadena tiene solo texto, sea minúscula o mayuscula, sin signos de puntuación ni nada mas. No permite espacios en blanco!🡪/^[A-Za-z]+$/**

**Busca signos de puntuación como interrogantes, exclamaciones o comillas🡪/[\!\?\"]/g**

**Para buscar @-->/@/g**

**16/10/2023**

**FUNCIONES:**

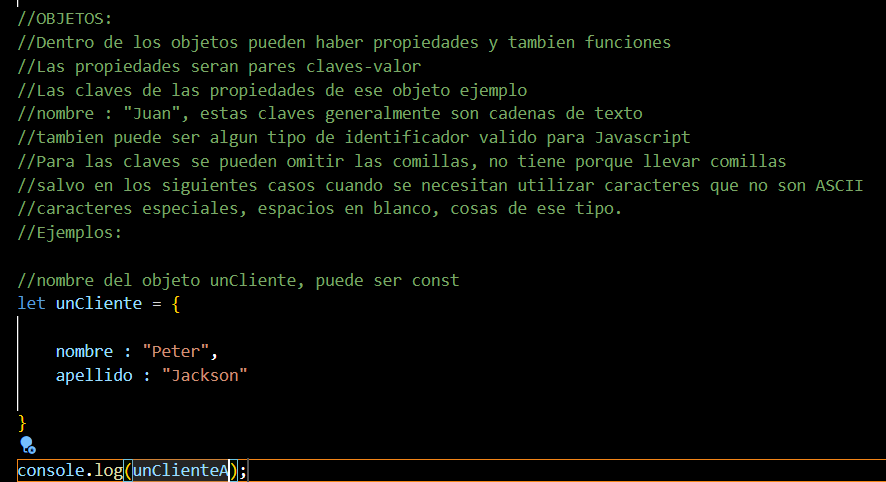
****

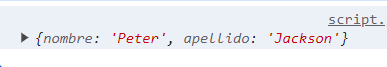
**OBJETOS**

**Nota: Los objetos pueden llevar distintos tipos de datos; desde arrays, a métodos del propio objeto(function()), a ints, strings, booleans, otros objetos…**

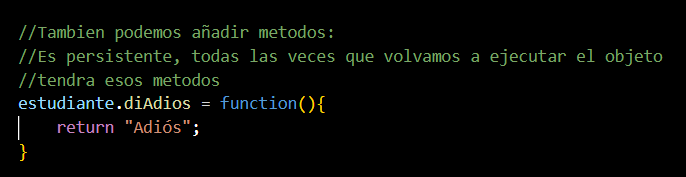
**Para agregarle una nueva característica a un objeto:**

Aunque aquí abajo no hayamos declarado que tiene una edad **unCliente** si se lo queremos agregar solo tendríamos que hacer **unCliente.edad = 23;** Y automáticamente el objeto ya incorporaría la edad.

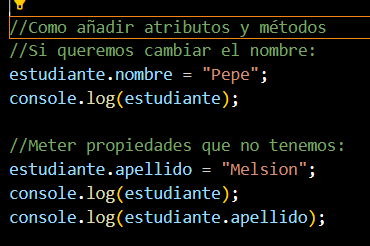




Tambien se pueden añadir métodos a posteriori accediendo de la misma manera:



Añadir atributos:

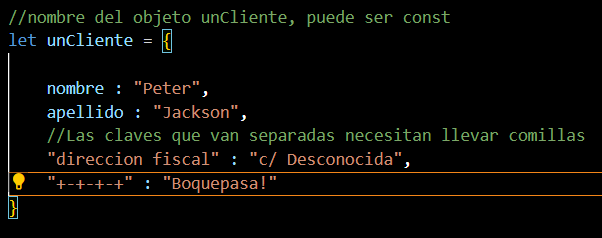


**Para cambiarle la característica a un objeto**

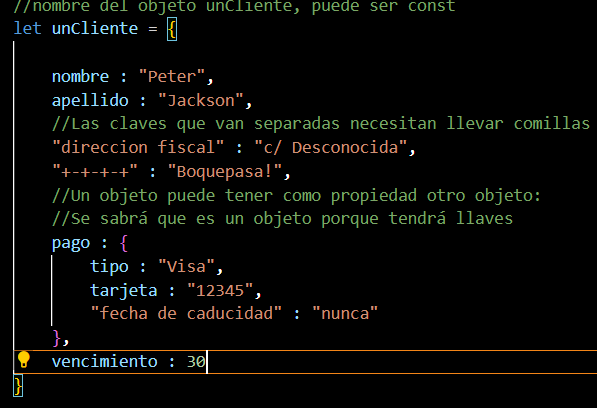
Si en **unCliente** quisiéramos cambiarle el nombre, lo único que tendríamos que hacer es acceder a la propiedad nombre; **unCliente.nombre =** “nuevo nombre unCliente” y ya está.

Si la clave estuviera entre **comillas** entonces se accedería asi; **unCliente[“nombre completo”]=”nuevo nombre del objeto unCliente”;**

Las **claves especiales** que sí que deben de llevar comillas son aquellas que contienen espacios en blanco o que están formadas por caracteres especiales:

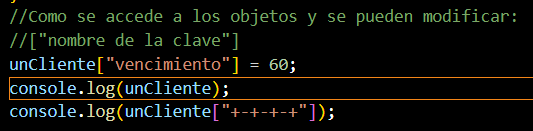


**Puede un objeto contener otro objeto:**

****

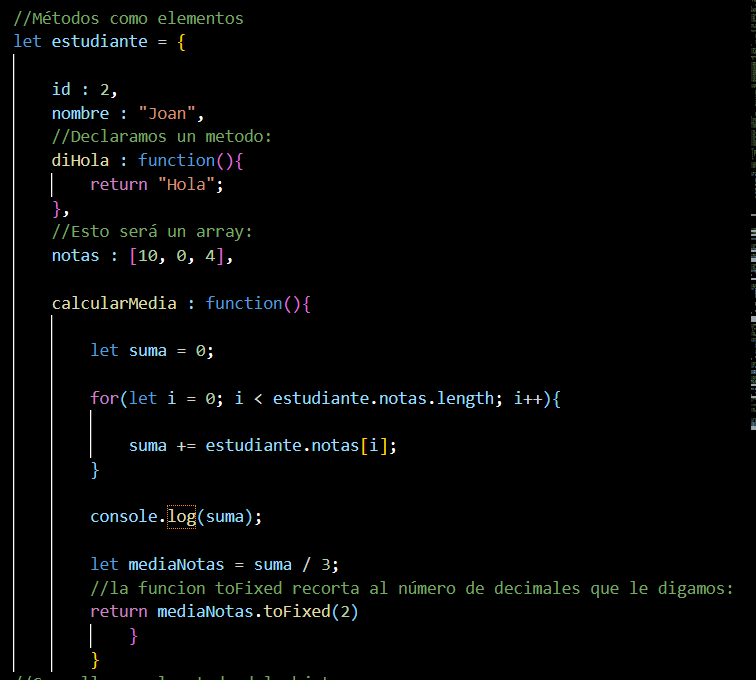
El objeto dentro del objeto se construye de la misma manera que el principal solo que ne lugar de usar el **=** usamos los **:** como con las propiedades.

**Acceder a objetios y modificarlos:**

****

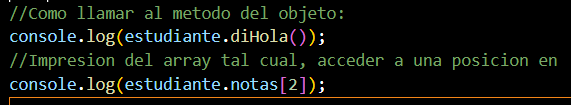
En caso de que la clave no haya necesitado comillas podemos acceder a ella para modificarla con **[]** o con **.** pero en el caso de que haya necesitado comillas siempre se necesitará **[]** (visual ya lo agrega automatico).

**Métodos de los objetos, métodos como elementos**



Se sabrá que es un método porque irá acompañado de la palabra **function(){** entre los corchetes pondremos el código que deseemos ejecutar en llamar a la función**}**

**Para llamar a la función (método) del objeto:**

****

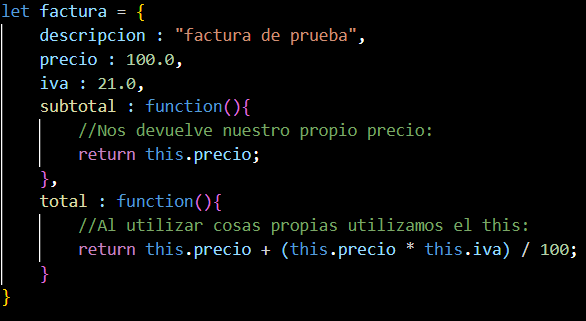
**Importante: Poner siempre en las llamadas los paréntesis!!**

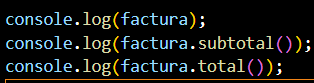
Así en el ejemplo de aquí arriba donde tenemos una función que retorna un parámetro si queremos imprimir el nombre del estudiante y su media lo llamaríamos asi desde fuera:

console.log("El estudiante", estudiante.nombre, " tiene de media", estudiante.calcularMedia() );

**Uso de la palabra reservada this para objetos**

Se refiere al objeto acrual, para referirnos a nuestras propias propiedades:

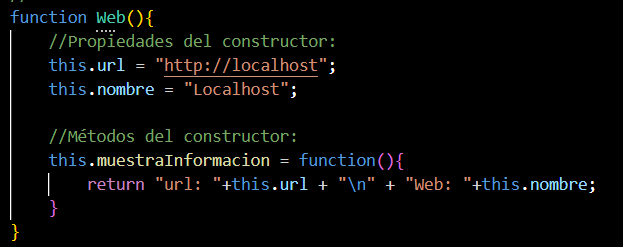




**Constructores**

Utilizamos la palabra clave **new. Se distingue** que estamos ante un **constructor** porque estamos **empezando con mayuscula.**

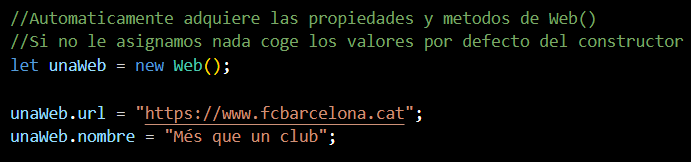
El constructor tiene propiedades(como el objeto) y también puede contener métodos propios(como el objeto).



Luego se pueden crear instancias de ese constructor (como pasaba con los objetos en Java, que “heredaban” las características de la clase, pues aquí pasa lo mismo entre las instancias que se crean a partir del constructor):

**Instancias del constructor:**

Si no le asignamos nada nuevo coge por defecto las propiedades y métodos que trae su constructor.



**Prototype; añadirle propiedades o métodos al constructor y que lo hereden las instancias**

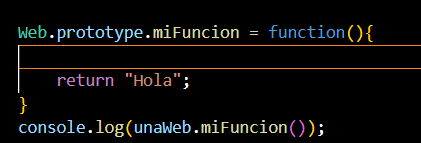
Prototype es una propiedad que tienen los objetos en Javascript.

Si en el ejemplo de Web queremos que las instancias hereden una nueva propiedad llamada **visitas** que no existía en el constructor original, hay que hacerlo así:



Si lo añadimos asi: Web.visitas solo se añadiría la nueva propiedad al constructor, pero no lo heredarían las instancias.

Lo mismo pasa con los métodos:



Aquí, unaWeb, que es una instancia de Web hereda la función que se le acaba de incorporar al constructor al llevar la palabra **prototype.**

**DIFERENCIAS ENTRE OBJETOS Y CONSTRUCTORES:**

* **Los objetos son instancias individuales con propiedades y métodos específicos, mientras que los constructores son funciones que se utilizan para crear objetos.**
* **Los objetos se crean directamente, mientras que los constructores se utilizan para crear múltiples instancias de objetos con una estructura común.**
* **Los objetos no tienen un prototipo definido de forma predeterminada, mientras que los constructores suelen definir métodos en su prototipo que pueden ser compartidos por todas las instancias.**
* **Para crear un objeto a partir de un constructor, se utiliza la palabra clave new, lo que asigna automáticamente el prototipo del constructor al objeto creado.**

**En resumen, los objetos representan instancias individuales con sus propias propiedades y métodos, mientras que los constructores son funciones que se utilizan para crear objetos con una estructura y comportamiento comunes.**

17/10/2023

**Ordenar objetos en arrays**

**Ordenar numérico:**

Seria hacer🡪nombreArray.sort((a,b)=>a.campoDelObjetoAComparar – b. campoDelObjetoAComparar);

**Ordenar por strings de objetos:**

 \* const ciudades = [

    {   //Elemento cero del array:

        municipio: "Zaragoza",

        provincia : "Zaragoza"

    },

    {   //Elemento cero del array:

        municipio: "Avila",

        provincia : "Avila"

    },

    {   //Elemento cero del array:

        municipio: "Madrid",

        provincia : "Madrid"

    },

    {   //Elemento cero del array:

        municipio: "Barcelona",

        provincia : "Barcelona"

    }

];

ciudades.sort((a, b)=>{

    return a.provincia.localeCompare(b.provincia);

});

**Const en Arrays:**

//Aun siendo constantes con las estructyruas se pueden añadir cosas:

const miArray = ["A", "b", "C"];

miArray.push("D");

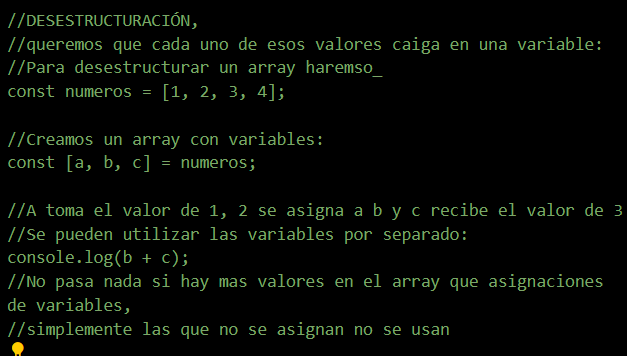
console.log(miArray);

Esto imprimiría tambien la D, que la añadiría al final.

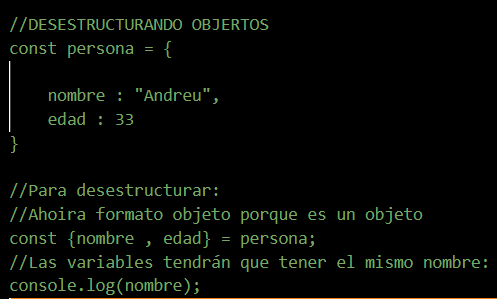
**Ojo porque en variables no lo permite!**

**Tambien en arrays se pueden modificar cosas, aun siendo const, cosa que en las variables no lo permite!**

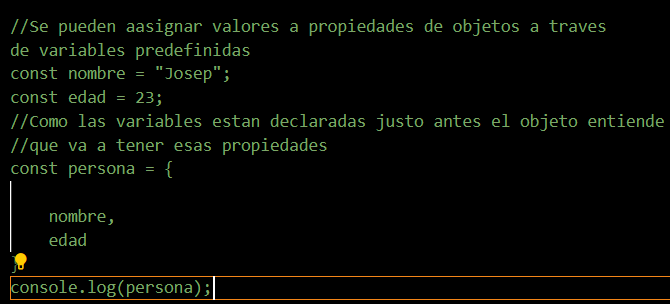
**DESESTRUCTURACION**

****

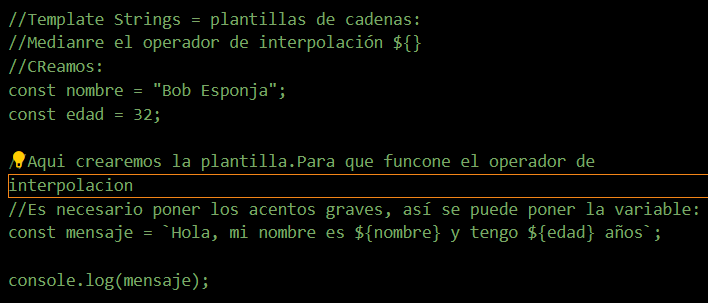
**DESESTRUCTURANDO OBJETOS**

****

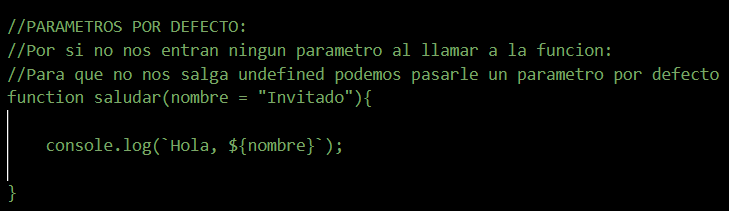
**ASIGNAR VALORES A PROPIEDADES DE OBJETOS A TRAVÉS DE VARIABLES PREDEFINIDAS**

****

**PLANTILLAS DE CADENAS**

****

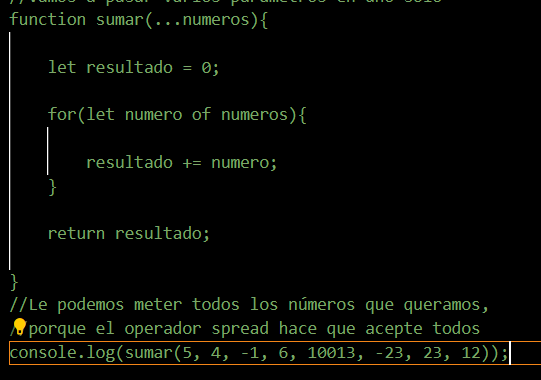
**PASAR VALORES POR DEFECTO (POR PARAMETRO)**

****

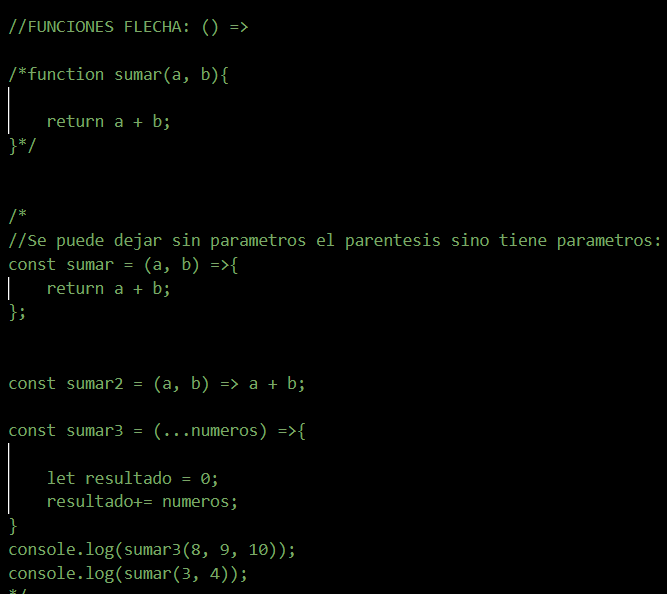
saludar();

**PARAMETROS REST (permite pasar x parámetro cuantos elementos queramos, numero variable)**

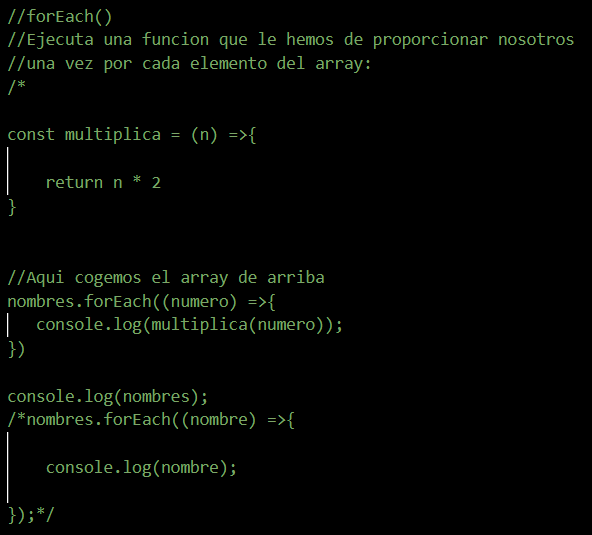
El parámetro rest visualmente es como el spread, son tres puntitos.

****

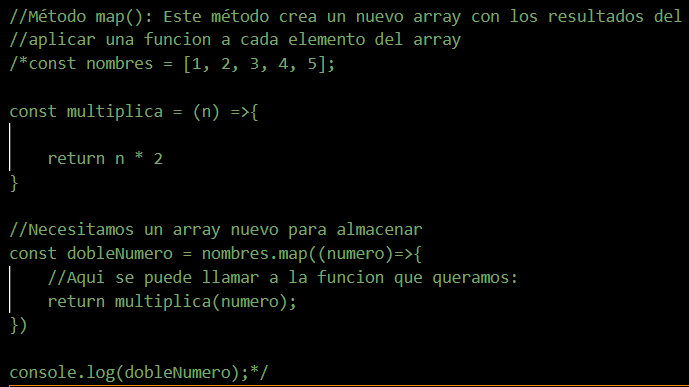
**FUNCIONES FLECHA**

****

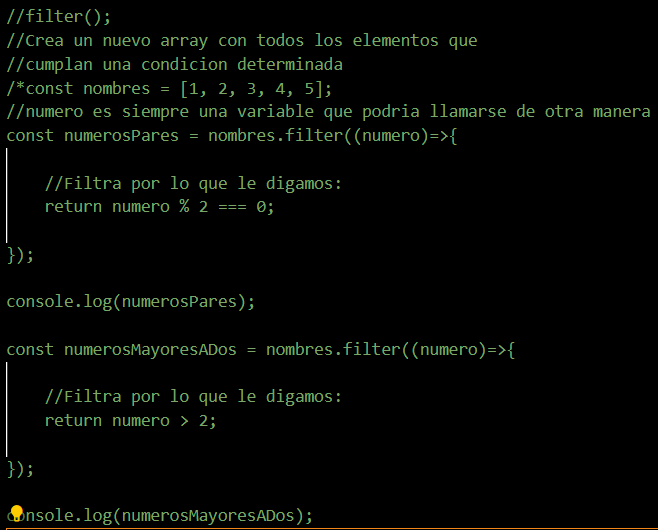
**BUCLE FOREACH (se puede llamar a una función dentro de el para ejecutar operaciones)**

****

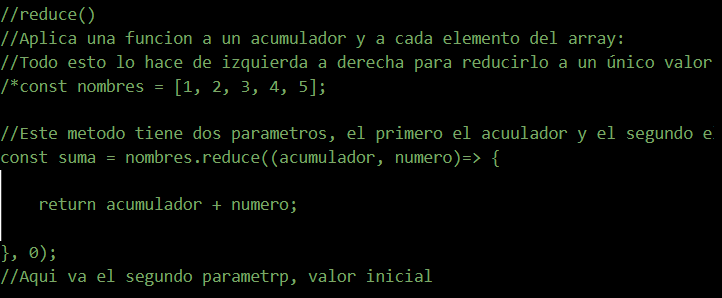
**METODO MAP()**

****

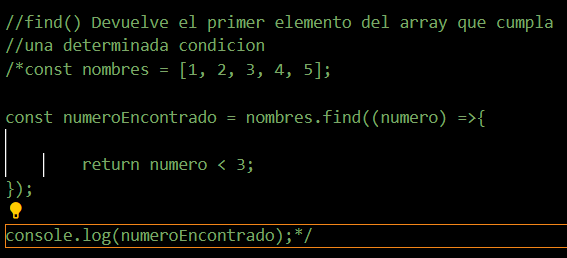
**FILTER()**

****

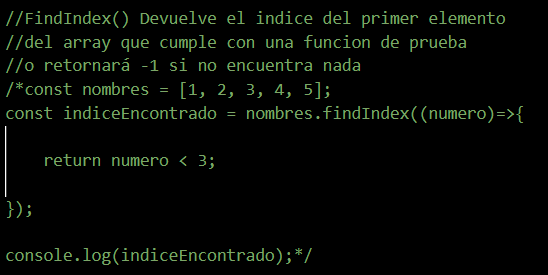
**REDUCE()**

****

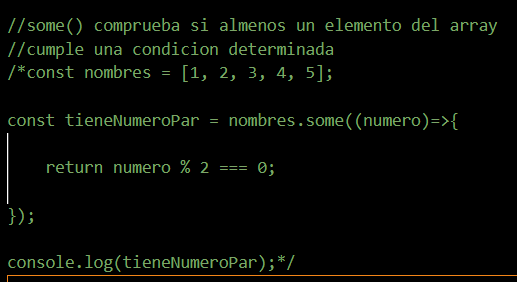
**FIND()**

****

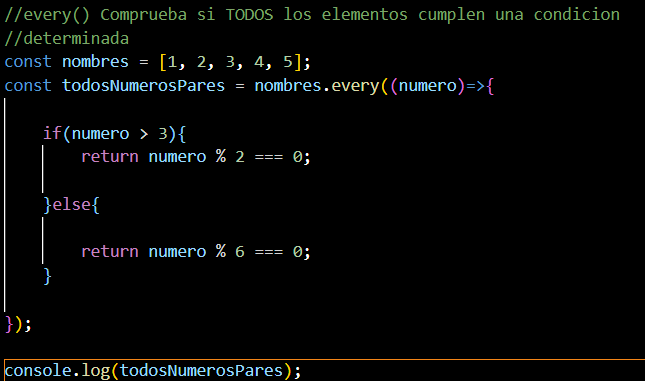
**FINDINDEX()**

****

**SOME()**

****

**EVERY()**

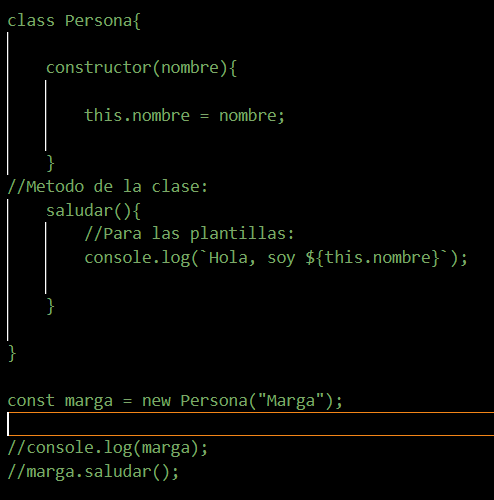
****

23/10/2023

**Arrays, funciones y clases, Colecciones iterables de Map y Set y crear objetos con CLASS:**

**Class🡪**palabra reservada en Javascript.

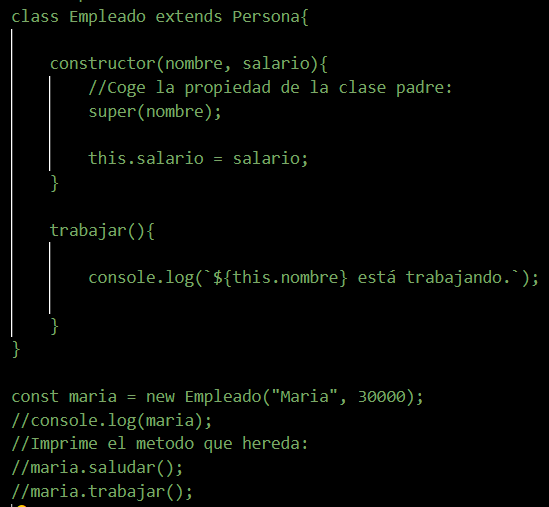
Como crear con clases, en ES6, que se crea igual o muy parecido a como se creaba en Java:



Dentro de la clase persona, tenemos el constructor como teníamos en Java donde vamos a darle propiedades al objeto que vamos a crear. Y también podemos darle métodos.

**Herencia**

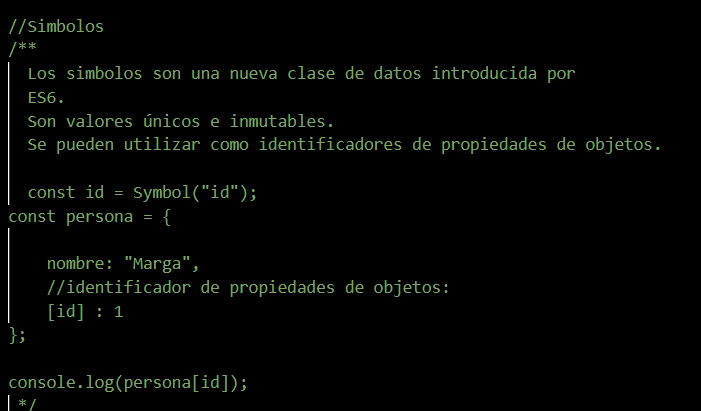
El concepto de herencia es igual que en Java, las clases que hereden de la clase padre, heredarán tanto las propiedades como los métodos. Igual que en java, se hace mediante la palabra **extends.**

****

Nota: Se utiliza la palabra reservada **super(**propiedad que queremos que herede**)**.

Así mismo también heredará sus métodos. Por tanto podemos llamar al método **saludar()** que pertenecían a Persona.

**SIMBOLOS**

****

**LO INTERESANTE DEL SIMBOLO ES QUE ES ÚNICO.**

**Iteradores**

/\*\*

 \* Son objetos que implementan el protocolo de iteracion en Javascript

 \* permiten recorrer y acceder a los elementos de una coleccion uno

 \* a uno.

 \* const numeros = [1, 2, 3];

//Creamos iterador utilizansdo simbolos:

const iterador = numeros[Symbol.iterator]();

//iterador queda ligado a la coleccion iterable (array) de numeros:

console.log(iterador.next());

//Se coloca en el primer elemento aqui y nos dice que no ha llegadso al final

console.log(iterador.next());

console.log(iterador.next());

//Se desborda, proque habia tres elementos y el no da true hasta el 4:

console.log(iterador.next());

 \*

 \*/

**CON ESTO LO INTERESANTE ES HACER UN BUCLE, DONDE PODEMOS COMPROBAR SI EL ITERADOR DA TRUE(OJO PORQUE SE DESBORDA, QUE ESTO ES CUANDO LLEGA AL FINAL DEL ARRAY QUE YA NO HAY ELEMENTOS DENTRO!!!) PODEMOS REALIZAR UNA SERIE DE ACCIONES.**

**SET🡪Colección iterable como había en java, NO PERMITE DUPLICADOS!!**

//Permite almacenar valores ÚNICOS de cualquier tipo

//No permite duplicados y ofrece métodos para agregar,

//eliminar y verificar la existencia de elementos.

 \*

 \*

 \* const numeros = new Set();

numeros.add(1);

numeros.add(2);

numeros.add(3);

//has para verificar si existe un elemento dentro del set:

console.log(numeros.has(2));

//Delete para el borrado de los elementos:

numeros.delete(2);

console.log(numeros.has(2));

//El tamaño del array:

console.log(numeros.size);

console.log(numeros);

//Se puede ityerar con iterator:

const iterador = numeros[Symbol.iterator]();

//Te va sacando cada valor y da treu cuando llega a donde no hay valor:

console.log(iterador.next());

console.log(iterador.next());

console.log(iterador.next());

 \*

 \*/

**MAP🡪Ojo!! Que no es el método!! QUE ES LA COLECCIÓN ITERABLE TIPO COMO UN ARRAY**

//Map permite almacenar PARES clave-valor donde cada clave es ÚNICA.

//Permite operaciones de búsqueda, inserción y eliminación.

const mapaNombres = new Map();

//Aqui estamos instanciando clases

//Será clave-valor

mapaNombres.set("nombre", "Ana");

mapaNombres.set("edad", 33);

mapaNombres.set("profesion","Desarrollador");

//el get treu el valor de la clave que vulguem:

console.log(mapaNombres.get("nombre"));

//has devuelve true or false, para comprobar si tiene una clave

//para evitar errores:

console.log(mapaNombres.has("edad"));

mapaNombres.delete("edad");

console.log(mapaNombres.has("edad"));

//Tambien tiene un size que nos dirá cuantos elementos tiene:

console.log(mapaNombres.size);

console.log(mapaNombres);

**24/10/2023**

**FECHA con objeto Date() Y FUNCIONES DE TEMPORIZADORES setInterval() y setTimeOut()**

Javascript tiene un objeto interno que es el Date();

**Date()🡪es un constructor que devuelve un string**

Hay cuatro maneras diferentes de inicializar un objeto de tipo fecha:

// new Date();

//new Date(milisegundos);

//new Date(fecha formato string);

//let fecha = new Date(7778999);

//parrafo.innerHTML = fecha;

//Resultado de esto: Fri Jan 02 1970 02:41:40 GMT+0100 (hora estándar de Europa central)

Aquí lo inicializamos en milisegundos.

**Formatos diferentes que ofrece Date();**

//let fecha = new Date("09/09/2023");-->El primero el mes, dia y año

//let fecha = new Date("Aug 16 2023 GMT+0200");-->Formato completo

//let fecha = new Date("2019-09-18");-->Formato iso, año, mes, dia

//new Date(año, mes, dia, horas, minutos, segundos, milisegundos);

//Introducir fecha

let fecha = new Date(71, 12, 29, 15, 15, 0, 0);

parrafo.innerHTML = fecha.getTime();

**Ojo porque hay que mirarse el porqué, al meter mes 12 en realidad sale January y por tanto el año 71 se convierte en año 1972!**

**Métodos del objeto Date()**

//getDate();-->da el dia del mes, va del 1 al 31

//getDay()-->Dia de la semana en formato numerico 0 al 6

//getFullYear()-->Obtiene el año

//getHours()-->obtiene las horas

//getMilliseconds(), getMinutes() -->Se le pueden sumar cosas cuadno

//se opera como number cuando se introduce como number

//getMonth(), getSeconds(),

//getTime()-->da la hora en milisegundos, empieza por el dia 1 del año

**FUNCIONES setInterval()**

//setInterval(funcionALlamar, tiempo dado en milisegundos)

//Ejecuta la funcionALlamar de manera periódica, según el tiempo del segundo

//parámetro.

//clearInterval()

//esta funcion detiene la ejecución iniciada con setInterval()

**FUNCIONES setTimeOut()**

//setTimeOut(funcionALlamar (o lo que queramos llamar), tiempo dado en milisegundos)

//tiene dos parametros, la funcion a la que llamamos

//setTimeOut() ejecuta la funcionALlamar cuando ha transcurrido el tiempo

//indicado en el segundo parámetro

//El nombre de la funcion a llamar va sin paréntesis

//si le pusieramos paréntesis ejecutaria la fución al instante, por eso no se ponen

**Diferencia entre setInterval() y setTimeOut()🡪**setTimeOut ejecuta una función una sola vez mientras que setInterval la ejecuta repetidamente en un cierto intervalo de tiempo hasta qye detengamos ese intervalo con **clearInterval** pasandole pro parámetro la variable donde previamente hayamos almacenado el intervalo.

Ejemplo de uso combinando ambas:

function mostrarMensaje() {

console.log("Ha pasado un segundo");

}

// Crear un temporizador con setInterval y guardar el identificador

let id = setInterval(mostrarMensaje, 1000);

// Cancelar el temporizador después de 5 segundos

setTimeout(() => clearInterval(id), 5000);

UD2🡪Ejercicio Metropolis y linkear un script.js al documento HTML.

UD3🡪Estructuras de Control (Bucles & condicionales).

Apuntes Fechas (tiempo, horas, minutos, segundos).

UD4🡪Arrays Multidimensionales, declaración de arrays y length.

Métodos de Arrays.

Objetos y Constructores.

Ordenar objetos y desestructurar objetos.

Herencia y clases.

Set y Map.