**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.

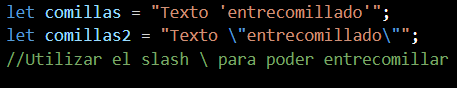
Las variables declaradas con var tienen un alcance de función (en princio, porque luego si se puede). Las de let y const tienen un alcance de bloque, si omitimos la palabra let o var la convertimos en global.

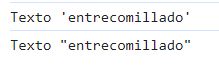
Javascript no exige punto y coma pero es buena practica ponerlo, porque asi se delimitan instrucciones.

let uno = 1; let dos = 2;

**Strings**

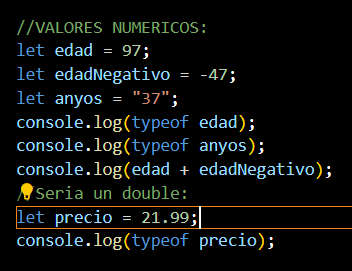
Se pueden declarar utilizando comillas simples o comillas dobles. Para entrecomillar:

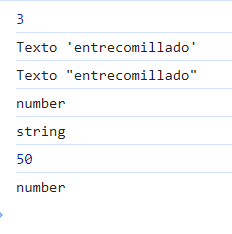




**NUMEROS**

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**

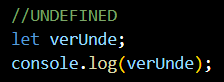
Es un valor existente, que indica algo que sí está definido, pero su valor está vacío o directamente se le ha dado un valor nulo. Puede ser por ejemplo un array vacio o directame a ente:



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

**UNDEFINED**

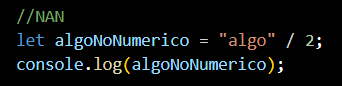
Se refiere a algo que ni tan solo ha sido definido. Caso mas obvio declarar una variable pero no asignarle un valor:



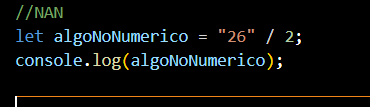


**NAN🡪NOT A NUMBER**

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

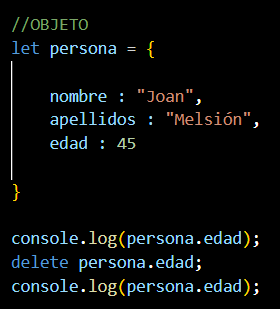
 

Así sí lo coge, porque hace una conversion:

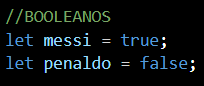


**DELETE**

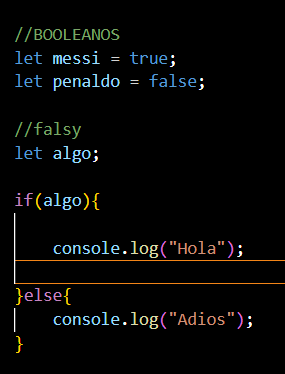
Se utiliza para eliminar características de objetos:



**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;

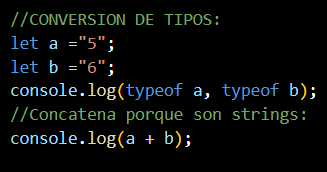
Y asi sucesivamente también para división y para el módulo.

**CAMBIO DE SIGNO**

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

**CONVERSIONES DE TIPOS**

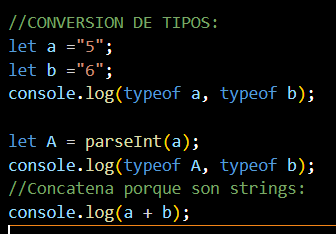
Hay varias maneras de hacerlo:

 Esto da resultado de string.

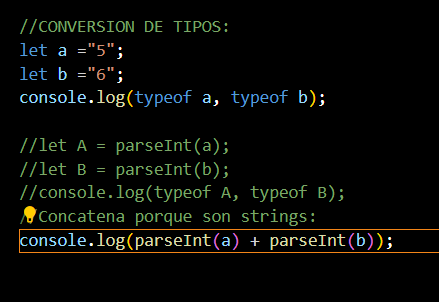
Cuando se recupera valores de formulario siempre se recupera el valor como un string. De los inputs salen strings, que sumados se concatenarían.

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

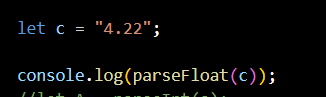
**PARA CONVERSION PODEMOS UTILIZAR🡪parseInt asignando el valor parseado a una nueva variable.**

****

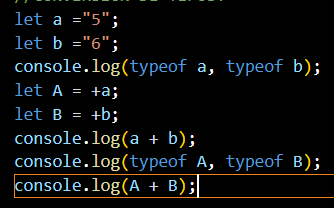
Esto pasaría el tipo de dato string a un entero. Si solo se parsea la a b seeguiria siendo un string y por tanto concatenaria las variables, no las sumaria.

****

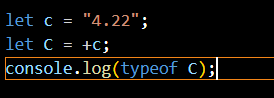
**En el mismo contexto existe el parseFloat:**

****

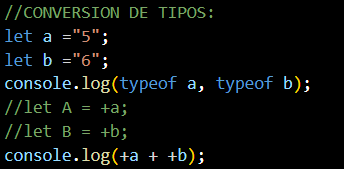
**Otro tipo de conversión de numero entero seria el símbolo +**

****

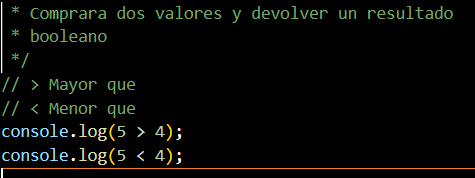
**Funciona tanto en integer como en float:**

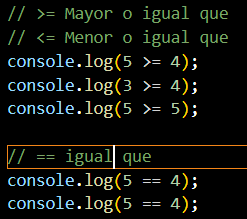
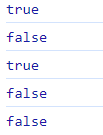
****

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

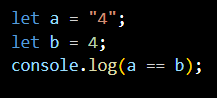
****

**OPERADORES DE COMPARACION**

Siempre devolverá true or false;

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

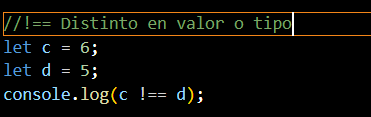
 Aunque uno es string y otro number al ser los dos 4 da **true.**

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

**DINSTIGUIR ENTRE NUMBER Y CADENA**

****

** Esto da false.**



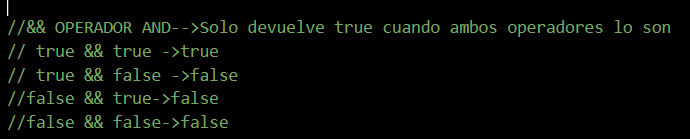
**EXPRESIONES BOOLEANAS:**

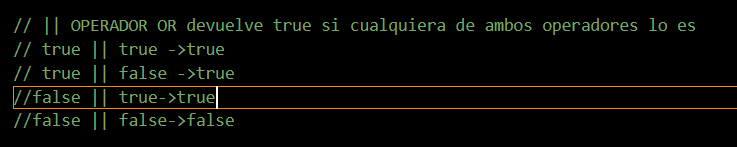
Permiten comparar expresiones booleanas con las que se construyen condiciones

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

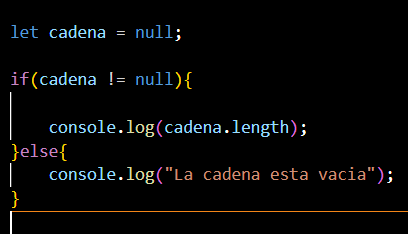
//&& OPERADOR AND-->Solo devuelve true cuando ambos operadores lo son

Se tienen que cumplir ambas condiciones.

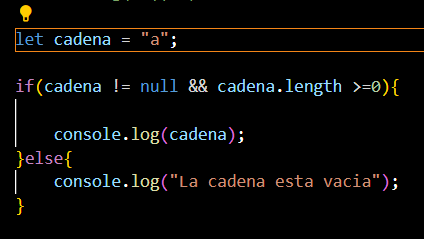
****

****

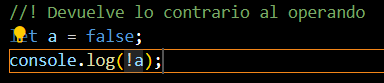
**COMPARACIONES CONTRA NULL, PARA COMPROBAR ELEMENTOS VACIOS:**

****

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.

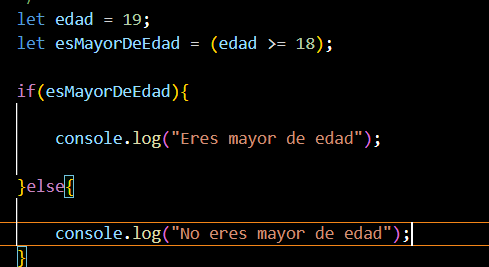


**NEGATIVO:**

****

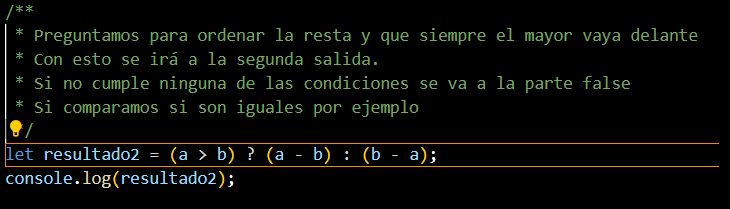
**ESTRUCTURAS DE CONTROL:**

**Condicionales IF**

****

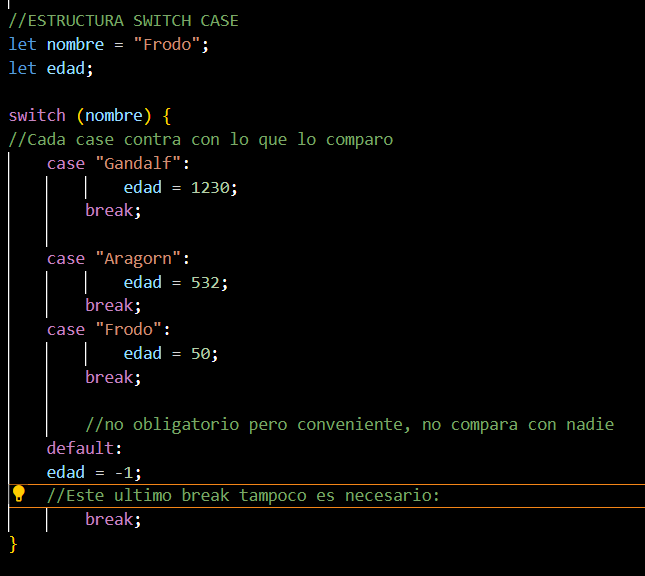
OPERADOR TERNARIO🡪Sustituyen en algunos casos a los if

?-->pregunta, :🡪resultado



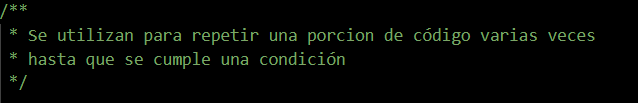
**SWITCH CASE:**

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.



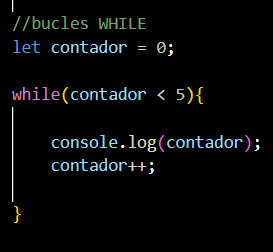
Sale Frodo al darle el valor por defecto.

**ITERACIONES🡪BUCLES**

****

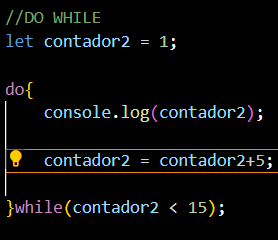
**Bucle While**

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

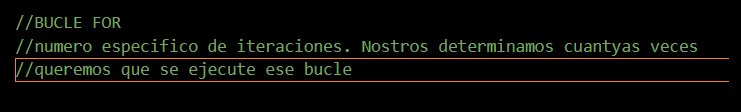


**Do while**

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



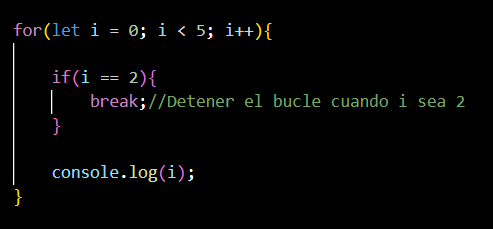
**Bucle For**

****

**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:**

