**JavaScript**

Es un lenguaje de scripting y no se compacta, es decir, que si escribimos líneas de código y lo abrimos en un navegador funcionara perfectamente. Todos los navegadores lo soportan.

Las nuevas versiones de Ecmascript (las reglas del lenguaje) pueden no ser soportadas por algunos navegadores. Se puede transpilar el código, es decir el código nuevo de las versiones actualizadas los hace compatibles con las versiones anteriores, con herramientas como babel.

JavaScript utiliza variables, funciones, constantes, if, etc.

\*Las clases para combinar códigos usan un punto entre ellas y al final hay un espacio = espacio.

\*Los textos (cadenas de textos) que se muestran como front end están entre comillas simples: ‘’ o dobles: “”. En ingles si se suele utilizar una comilla en la cadena de texto, para que no se la tome como código se usa \ antes de tal.

\*El código siempre se va a imprimir de arriba hacia abajo.

\*El punto y comas finaliza la línea de código.

\*JavaScipt es **DINAMICO** es decir, que no tiene que especificarse el tipo de dato en la variable (string, number, boolean(true/false), object, undefined), sino que este ya se encuentra en el propio valor.

Códigos esenciales

**VarN = nombre de la variable.**

\*Se pueden realizar ecuaciones entre **varN** si son números. Suma con +, resta con -, división con /, multiplicación con \*, etc.

**#(**nombre del id**):** Crea un div con el id que pusiste entre los ()

**<script></script>:** Es en este lugar donde posicionamos el código de js en el archivo html. Se recomienda usar tu propio archivo js y se enlaza utilizando el script y un **src=’’** en éste, en el body de html.

**//** :Se crea un comentario en js.

Funciones Pre hechas de JS

**varN.nodeType:** Con esto podemos saber qué tipo de nodo es el elemento y de esta forma ver que clases podemos usar en él. Con estas clases de los nodos podemos cambiar su html como su css a través de JS.

**varN.innerHtml:** Inyecta el código de una clase en js en código html.

**varN.setAttribute(“**clase**”, “**nueva\_clase**”):** Con esto le podes cambiar la clase a un elemento del Html ( que hayamos transformado antes en una variable)

**varN.style.display=”none”;:** Elimina el estilo de un elemento del Html ( que hayamos transformado antes en una variable)

**varN.style.display=”block”;:** Esto nos puede servir para que un elemento del HTML solo se vea cuando tenga un valor en java. Si esta en blanco estaría en un **display none;**

**varN.innerText:** Nos retorna el texto que forma parte de un elemento.

**varN.appendChild(varN** de nuevo elemento**):** Creamos un hijo en un elemento ya existente del Html.

**varN.cloneNode(true):** Clona el nodo o elemento de un Html.

**varN.childNodes[**número de nodo**]:** Esto nos deja ver cómo está arreglado el elemento con sus nodos. Con **dir(varN)** es mucho más simple.

**varN.insertBefore(varN** a ingresar**, varN.childNodes[**número de nodo**]):** Con esto podemos insertar un elemento luego o antes de otro elemento del Html.

**varN.nodeValue:** Nos da el valor o texto de los nodos asi podemos copiarlo, eliminarlo, transformarlo, etc.

**varN.value:** Nos da el valor de una variable, bueno para usar con botones.

**varN.parentNode.removeChild(varN):** Removemos un elemento del Html. Se usa **parentNode** dado que asi java entiende que debe hacer.

**varN.parentNode.replaceChild(varN2, varN):** Remplaza un elemento del Html por otro.

**varN.idenxOf(“**caracter**”); :** Busca cierto carácter que le indiques en el string. Si no existe este nos devuelve el valor -1 (importante para **if**)

**prompt():** Crea una ventana en el navegador que exclama lo que está entre los paréntesis.

**console.log(typeof varN)**:Te dice que tipo de dato es la variable, como string, number, boolean(true/false), etc.

**let varN = new Date();** : Esto nos da la fecha de hoy. Al igual que **Math, Date,** es una clase específica para fechas.

**dir(varN):** Nos deja ver las especificaciones de un elemento Html (que hayamos hehco **varN**) para luego poder utilizarlas y transformarlas.

**varN.focus(); :** Se focusea automáticamente en tal variable del HTML.

**alert(“**Texto**”); :** Nos tira un alerta en la página web. Usar en IF.

**parseInt(varN, 10) || 0; :**Convierte (parsea) un argumento de tipo cadena y devuelve un entero de la base especificada.

Seleccionar contenido del DOM

**document.GetElementById(‘**nombre de la id’**):** Con esto seleccionamos la id de un div.

**document.GetElementsByClassName (‘**nombre de la clase**):** Selecciona todos los elementos de esa clase. Con un selector de array (**classN[**número de elemento que quiero**])** puedo seleccionar un solo elemento.

**document.GetElementsByTag(‘**nombre del tag**’):** Este nos deja seleccionar el tag de los elementos (div, header, section, etc). Con un selector de array (**tagN [**número de elemento que quiero**])** puedo seleccionar un solo elemento. Con un loop (**for**) puedo escoger todos los elementos de ese tag y cambiarles su atributos.

**document.querySelector(“**nombre de la id/clase/tag**”);** : Con esto podemos seleccionar la id (con un #), una clase (con un punto) o el tag de un elemento . Se pueden seleccionar los hijos del padre poniéndolos con un espacio, ejemplo (h2 p). Solo puede retornar un elemento

**document.querySelectorAll(“**nombre de la id/clase/tag**”)[**elegir uno en especifico**];** Con esto podemos seleccionar la id (con un #), una clase (con un punto) o el tag de muchos elementos de la misma clase. También puedes elegir dos elementos al mismo tiempo agregando una coma (h2, h3)

Crear contenido en el DOM

**document.createElement(‘**elemento**’):** Podemos crear un elemento en nuestro Html.

**document.createTextNode(‘**texto**’):** Podemos crear un texto en nuestro Html.

Template string

Un template string nos deja crear una concatenación de texto y variables:

**´**texto **${varN}** texto **${varN2}´;** con comillas inclinadas.

O tamb:

**“**texto **” + varN + “**texto**” + varN2;**

Consola

Utilizar la consola de chrome o Firefox developer así ver bien el código, ya que solo te toma el de javascript.

1. **Console.log():** Manda un mensaje a la consola y te aparece en que línea del archivo esta. Puede ser texto, un cálculo, código, una variable, etc.
2. **Console.table():** Manda un mensaje a la consola pero en forma de tabla.
3. **Console.error():**Manda un mensaje a la consola pero en forma de error.
4. **Console.warn():**Manda un mensaje a la consola pero en forma de advertencia.
5. **Console.clear():** Limpia la consola.
6. **Console.time() / console.timeEnd():** Te indica cuanto ha sido el tiempo que tardo la consola entre estos dos códigos en imprimir o ejecutar los códigos interiores.

Creación de clases de código.

**Variables**

**Var** / **let** / **const:** Crea distintos tipos de variables, son piezas de información que todavía no sabemos pero al ser introducidas son imprimidas de esa forma, es decir varían en cuanto al usuario.

Estas suelen tener un nombre y luego para meterles código se usa =. Para referirse a una variable en otra parte del código se usan las comillas **`${**nombre de la var**}`.**

1. **Var:** Es la más soportada, ya que es la más vieja, pero está siendo sustituida por let o const. A esta se le introduce primero el nombre y luego = con el código que utiliza. Se puede reescribir el valor poniendo el nombre de la var y un =.

Se pueden crear múltiples **var** solo poniendo una coma entre sus nombres asignados.

1. **Let:** Es parecida a **var** en cuanto a sintaxis, y que luego puede ser *rellenada* con código y no en la misma línea. La diferencia más importante es que no se puede volver a declarar la clase con un **let**, pero si con solo el nombre de tal clase.

**Ej:** **let nombre;**

**Let nombre = ‘hola’** NO

**Nombre = ‘hola’** SI

1. **Const:** Es parecida a **var** y **let**  en su sintaxis. La diferencia recae en que siempre debe ser asignado el valor en su misma línea de código. Tampoco puede reescribirse de ninguna forma.

\*Estas se suman o concatenan en un código con un signo de + por ejemplo en un **console.log(var1 + var2).** Para agregar un espacio entre ellas, se utiliza dos signos de más y entre ellos comillas simples con un espacio = **console.log(var1+ ‘ ’ + var2)** o también utilizando un string que las contenga con las comillas inclinadas = **consola.log( ${var1} ${var2}´)**

**Funciones**

Las funciones son creaciones para poder luego variar las **var.** Son personales para cada proyecto.

Para crear tu propia función se realizan de dos formas:

//Function declaration

**Function funN(prmN) {**

Contenido de la función**;**

**};**

y se manda a llamar:

**funN(prmN);**

//Function expression

**let funN = function(prmN){**

Contenido de la función**;**

**};**

y se manda a llamar de la misma forma que el anterior.

La diferencia entre las dos es que la primera se puede mandar a llamar antes del código de creación, mientras que la segunda el llamado siempre tiene que estar después de su creación.

Existe una última función que es la IIFE, que se expresa al momento y se imprime o manda a llamar también al momento. Se escribe:

**(function)( {**

Contenido de la función**;**

**})(prmN);**

-------------------------------------------------

Se pueden crear **PARAMETROS** dentro de las funciones para que tengan más contenido, utilizando variables dentro de sus paréntesis ejemplo:

**Function funN(prmN, prmN2) {**

**console.log(“texto” + prmN + “texto” + prmN2)**

**};**

**funN (“Juan”** = prmN**, “Trabajo”** = prmN2**);**

**funN(“Antonio”** =prmN**, “caminar”** =prmN2**);**

JavaScript toma Juan como **prmN** y trabajo como **varN2** lo introduce luego en la función. Esto nos deja copiar el nombre de la **prmN** y agregar otras, creando distintas copias de la misma pero con diferentes mensajes. Es decir Antonio ahora es **prmN** y caminar es **prmN2**

O también con fnc expression:

**let funN = function(prmN, prmN2){**

**contenido.log(prmN + prmN2)**

**};**

**funN(10** =prmN**, 20**=prmN2**);**

**funN(30**=prmN**, 40**=prm2**);**

Funciona de la misma forma que el anterior.

O también con IIFE:

**(function(prmN) {**

**contenido.log(“texto” + prmN)**

**;**

**})(“texto”);**

Funciona de la misma forma que el anterior.

SE PUEDE CREAR UN VALOR INICIAL QUE SI NO SE EXPRESA SE IMPRIMIRA EN DEFAULT:

**Function funN(varN, varN2=”texto”) {**

**console.log(“texto” + varN + “texto” + varN2)**

**};**

Return Valores en fnc

Introducir un valor a la función para que luego pueda ser ingresado en una variable. SIEMPRE QUE SE CREE UNA FUNCION QUE ME DE UN VALOR TIENE QUE CREARSE UNA **VAR** QUE CONTENGA EL VALOR.

Ejemplo con fnc expression:

**let suma = function ( a = 0, b = 0) {**

**return a + b;**

**};**

**let resultadoSuma = suma(20, 30)**

**let multiplicar = function(a){**

**return a \* 5;**

**};**

**let total = multiplicar(resultadoSuma)**

**console.log(total) = 250**

Se puede reemplazar el **function return** con una flecha **=>** luego del **prmN.** Si el **prmN** es un solo valor se pueden quitar los paréntesis, si no debe llevarlos. Ademas se puede quitar el return y la llave, te quedaría en un ejemplo:

**let funN = prmN => prmN\* 5;**

Métodos

Los métodos son funciones creadas dentro de un objeto que luego las utilizamos cuando llamamos al objeto. Se escriben:

**let varN = {**

**funN: function(prmN) {**

**console.log(“texto” + prmN)**

**}**

**} ;**

Se manda a llamar con:

**varN.funN(“texto”** = prmN**)**

O también se puede crear un método por fuera, pero el objeto tiene que ser creado siempre antes, para que la consola sepa a que te referis por **varN.**

**varN.funN = function(prmN) {**

**contenido.log(“texto” + prmN)**

**}**

Se manda a llamar con:

**varN.funN(“texto”** = prmN**)**

Codigos de funciones y metodos en JS de manera default

**varN2 = varN1.concat();** : Se le concatena string o texto a la variable.

**varN2 = varN1.toUpperCase();** : Se pasa todo el string a mayúsculas.

**varN2 = varN1.toLowerCase();**: Se pasa todo el string a minúsculas.

**varN2 = varN1.split ();**: Se separa el string en las partes que le indiques, si pones “ ” separa en sus espacios.

**varN2 = varN1.replace (“**que quieres remplazar**”, “**con que quieres remplazarlo**”);** : Reemplaza una línea de texto.

**varN2 = varN1.includes (“**parte del string**”);** : Te indica si el texo posee una parte del texto que incluiste en los prentesis.

**console.log(varN.lenght); :** Te indica cuantas letras posee el código, el string o cuantos elementos el arreglo.

**Console.log(varN.repeat(**número de veces**);** : repite el string tantas veces como le indiques.

**Arreglo**

Un arreglo son datos agrupados en una sola variable.

Cada arreglo tiene su propio valor en el índice empezando por 0.

Este se crea de la siguiente forma:

**let varN = [**arreglo**,** arreglo**]**

O también:

**let varN = new Array(**arreglo**,** arreglo**)**

Para accede a un elemento del arreglo se utiliza:

**console.log(varN[**número de arreglo**]); :**Solo imprime los números del índice de arreglos que le digas.

Codigos de arreglos

**console.log(varN[**número de arreglo**]); :**Solo imprime los números del índice de arreglos que le digas.

**console.log(Array.isArray(varN));** :Nos comunica la consola si la variable es un arreglo.

**varN.push(**arreglo**);** : Agrega una variable a un arreglo en la última posición.

**varN.unshift (**arreglo**);** :Agrega un arreglo a la variable en la primera posición.

**varN[**número índice de arreglo**] =** arreglo**;** : Agrega un arreglo en esa posición del índice.

**varN.pop();** :Elimina el último arreglo de la variable.

**varN.shift ();** :Elimina el primer arreglo de la variable.

**varN.splice(**número del índice desde donde va a eliminar**,** numero de cuantos va a eliminar**);** :Nos deja eliminar arreglos desde su rango en el índice.

**varN.reverse ();** :Da vuelta por completo el arreglo.

**varN.sort();** :Ordena alfabéticamente el arreglo.

Objetos

**Objeto Literal**

Tiene el mismo objetivo que el arreglo, pero en este se le puede dar nombre al número del índice con las llaves.

Pueden ingresar arreglos al objeto e incluso objetos al objeto

Se escribe de tal forma:

**let varN = {**

**llaveN:** arreglo**,**

**llaveN2:** arreglo2**,**

**}**

Para acceder a las llaves se realiza:

**console.log(varN.llaveN);**

O tamb:

**Console.log(varN[‘llaveN’]** (como en un arreglo)

**Object Constructor**

Este funciona como el literal pero de una forma más dinámica ya que al funcionar con una FUNCION se puede ir cambiando a la vez que se usa. Se debe crear una función con sus parámetros más un **this**(asi JS sabe que son objetos y no **prm**), y luego una **var** que los pase de **prm** a objetos o arreglos con un **new.** Se escribe:

**Function funN(prmN, prmN2) {**

**this.prmN = prmN;**

**this.prmN2 = prmN2;**

**}**

**let varN = new funN(“texto”**=prmN**, “texto”**=prmN2**)**

Class de object constructor

Este Object Constructor se puede pasar a una **Class:**

**class classN {**

**constructor(prmN, prmN2) {**

**this.prmN = prmN;**

**this.prmN2 = prmN2;**

**}**

**}**

**let varN = new classN(“texto”**=prmN**, “texto”**=prmN2**)**

Condicionales

**If**

Este se active si la condición es verdadera. Mientras más signos de igual tenga mayor será el nivel de rigurosidad se tendrá (en 2 == no se toma en cuenta si el valor es string o no, con 3 sí). Se escribe:

**If(varN ===** valor o texto**){**

realizar este código**;**

**}**

CON UN > < REVISAS SI EL VALOR ES MAYOR O MENOR. SE LE PUEDE AGREGAR UN = PARA HACERLO MAYOR E IGUAL, O MENOR E IGUAL.

Existe su contrario que se activara si la condición no es verdadera.

**If(var !===** valor o texto**){**

realizar este código**;**

**}**

**Else**

Se activa si el **if** no se activa. Se escribe:

**If(varN ===** valor o texto**){**

realizar este código**;**

**} else {**

realizar este código**;**

**}**

**Else If**

Este se utiliza luego del **if** y se activa si este no es verdadero. Se puede usar el **else** luego por si no se activa ninguna de los **else if** anteriores (en orden de cascada)

Se activa si el **if** no se activa. Se escribe:

**If(varN =** valor o texto**){**

realizar este código**;**

**} else if {**

realizar este código**;**

**} else {**

realizar este código**;**

**}**

**Or**

**Or** se utiliza para agregar otra validación al **if**. Aunque el primer valor sea falso, si el segundo en **or** es verdadero el código será realizado. Se escribe con 2 ||:

**If(varN =** valor o texto **|| varN2 =** valor o texto**){**

realizar este código**;**

**}**

**And**

Este solo se activara si todas las condiciones del **if** son verdaderas. Se escribe con 2 &&:

**If(varN =** valor o texto **&& varN2 =** valor o texto**){**

realizar este código**;**

**}**

**Ternario**

Este se utiliza para crear un código de condición más simple y en una sola línea. Con **?** se activa si es verdadero, y con **:** si es falso. Ejemplo:

**Let varN =** true**;**

**Console.log(varN ? “**Verdadero**” : “**falso**”);**

**varN =** verdadero**.**

**Switch**

El **switch** es una transformación del **if** en el cual se escribe más ordenado el código. Con el **case** verificamos si una acción es verdadera y se acciona tal código. Con **default** proporcionamos un código por default si ninguna es verdadera. Con break separamos códigos. Se escribe:

**Switch(varN) {**

**case** texto o valor**:**

realizar este código**;**

**break;**

**case** texto o valor diferente**:**

realizar este código**;**

**break;**

**default:**

realizar este código

**break;**

**}**

Loops

Un loop es una función que corre hasta que una condición se cumple.

**For Loop**

Un for loop se escribe de esta forma:

**for (let i =** número de inicio**;** **i =** condición de cuando termina**; i =**  cuanto suma el i**) {**

código que se repetirá

**}**

EJ:

**for (let i = 0; i > 10; i++) {**

**console.log(`Numero: ${i}`);**

**}**

Solo el código terminara de repetirse cuando **i** sea igual a mayor de 10, esto sucederá 10 veces porque empieza por 0 y se suma de a un digito (**i++**)

**While loop**

Se escribe:

**let i =** numero de inicio

**while (i =** condición cuando termina**) {**

código que se repetira**;**

**i++;**

**}**

Ejemplo:

**let i = 0**

**while (i < 10) {**

**console.log(`Numero: ${i}`);**

**i++;**

**}**

Iteradores

**Iteradores con For**

**const pendientes = [“tarea”, “comida”, “cama”]**

**for(pendiente of pendientes) {**

**console.log(pendiente);**

**}**

Esto lo que hace es que crea un valor llamado **pendiente** de la lista de **pendientes** y cuando se sumen todos los objetos del arreglo se dejara de imprimir cada **pendiente**.

Ejemplo con arreglo con objetos:

**const carrito = [**

**{id: 1, producto: “libro”},**

**{Id: 2, producto: “camisas”},**

**{id: 3, producto: “gaseosa”}**

**];**

**for(producto of carrito) {**

**console.log(producto.producto)**

**}**

Ejemplo con objeto:

**const automóvil = {**

**modelo: “Nissan”,**

**motor: 6**

**};**

**for(info of Objects.values(automovil)) {**

**console.log(info);**

**}**

**Iterador For each**

Tambien existe un comando para que se imprima cada objeto de una function, arreglo, objeto, etc. Este se escriba:

**varN.forEach(function(varN-Objects) {**

codigo que se repetirá

**});**

Ejemplo con el arreglo:

**const pendientes = [“tarea”, “comida”, “cama”]**

**pendientes.forEach(pendiente =>**

**console.log(pendiente));**

Ejemplo con arreglo con objetos:

**const carrito = [**

**{id: 1, producto: “libro”},**

**{Id: 2, producto: “camisas”},**

**{id: 3, producto: “gaseosa”}**

**];**

**carrito.forEach(producto =>**

**console.log( producto.producto ))**

**});**

Ejemplo con objeto:

**const automóvil = {**

**modelo: “Nissan”,**

**motor: 6**

**};**

**Object.value(automovil).forEach(info =>**

**console.log(info))**

**Iterador Map**

Funciona de la misma forma que el For Each pero se crea una copia del arreglo al momento del crear el **.map.** Esta la podemos agregar a una nueva variable y referirnos a ella en un nuevo código. Se escribe:

**varN.map(function(varN-Objects) {**

codigo que se repetirá

**});**

Ejemplo con el arreglo de pendientes:

**let nuevoArreglo = pendientes.map (function(pendiente) {**

**return pendiente**

**});**

**console.log(nuevoArreglo)**

**Scope**

Esto es una propiedad de JavaScript en la cual una **var** puede ser resiginificada dentro de otro código y globalmente se cambiaría. Esto se evita utilizando **let**, de esta forma si tenemos un **letN** y luego lo re significamos dentro de otro código, su valor inicial global no cambiara, solo su local.

Object Destructuring

Con este podemos crear variables con los valores de las llaves de un objeto:

**const varN = {**

**llaveN: “texto”,**

**llaveN2: “texto”,**

**}**

**let {llaveN, llaveN2} = varN;**

Si el objeto es mas complejo y contiene llaves dentro de llaves también se puede utilizar

**const varN = {**

**llaveN: {**

**llaveN2: “texto”**

**}**

**}**

**let {llaveN: {llaveN2} } = varN;**

**Filter**

Este código filtra los objetos, llaves o arreglos para encontrar ciertos valores. Ejemplo con un object en un arreglo:

**const personas = [**

**{nombre: “Juan”, edad: 25}**

**{nombre: “Antonio”, edad: 17}**

**{nombre: “Josefina”, edad: 32}**

**{nombre: “Pepe”, edad: 18}**

**];**

**let mayores = personas.filter(persona => persona.edad > 25);**

**Find**

Encuentra un objeto, llave o arreglo en espefico. Ejemplo con un object en un arreglo:

**const personas = [**

**{nombre: “Juan”, edad: 25}**

**{nombre: “Antonio”, edad: 17}**

**{nombre: “Josefina”, edad: 32}**

**{nombre: “Pepe”, edad: 18}**

**];**

**let Alejandra = personas.reduce(persona => persona.nombre === “Alejandra”);**

**Reduce**

Reduce los valores de un objeto, arreglo, etc a un número. Tiene que tener un valor inicial. Ejemplo con un object en un arreglo:

**const personas = [**

**{nombre: “Juan”, edad: 25}**

**{nombre: “Antonio”, edad: 17}**

**{nombre: “Josefina”, edad: 32}**

**{nombre: “Pepe”, edad: 18}**

**];**

**let total = personas.reduce(( edadTotal, persona) =>**

**{return edadTotal + persona.edad},**

**0);**

Fetch API

Retornar valores de un sitio web para utilizarlos en mi código. Se crean funciones para darle valor a cada API que saquemos y luego se reestructuran para conseguir valores únicos utilizando object destructuring o iteradores. Ejemplo utilizando un api de un sitio web randomizable de usuarios

**descargarUsuarios(30)[[1]](#footnote-1)**

**function descargarUsuarios(cantidad[[2]](#footnote-2)) {**

**const api[[3]](#footnote-3) = ´https://api.randomuser.me/?nat=US&results=${cantidad[[4]](#footnote-4)}´;**

**//**llamado a Fetch

**fetch(api)[[5]](#footnote-5)**

**.then(respuesta => respuesta.json() )[[6]](#footnote-6)**

**.then(datos => imprimirHtml[[7]](#footnote-7)( datos.results) )[[8]](#footnote-8); }**

**Function imprimirHtml(datos) {**

**datos.forEach[[9]](#footnote-9)(usuario => {**

**console.log(usuario);**

**const li = document.createHtml(“li”)[[10]](#footnote-10);**

**const { name: {first}, name: {last}, picture: {medium}, nat[[11]](#footnote-11) } = usuario;**

**li.innerHtml[[12]](#footnote-12) =**

**´Nombre = ${first} ${last}**

**Pais = ${nat}**

**Imagen = <img src=”${medium}”>**

**´;**

**document.querySelector(‘#app’).appendChild(li)[[13]](#footnote-13); }**

**Uso Estricto**

Con este código obligamos a JavaScript a que solo se imprima cuando el html y Css se hayan cargado.

**(function(){**

**'use strict';**

**document.addEventListener('DOMContentLoaded', function(){**

**});**

**})();**

Existe una versión mejorada con mas soporte utilizando JQuery:

**$(document).ready(function() {**

**});**

**\***Los **varN** que seleccionan los id de los Select tienen que ir afuera del **document.addEventListener** ya que sino no funcionan.

**Eventos**

Eventos que realiza el usuario en nuestra página. Se escuchan o se activan con:

**varN.addEventListener(“click”,funN); :** Esto nos deja saber cuándo el usuario hace click en cierta id y mandar a que se active la función escrita dentro del paréntesis.

**varN.addEventListener(“blur”,funN); :** Esto nos deja saber cuándo el usuario cambia el valor de un input. Como podría ser el **input type=”number”**

**Código de eventos**

**function funN (event) {**

**event.preventDefault();**

**}; :**Cancela el evento si este es cancelable, sin detener el resto del funcionamiento del evento, es decir, puede ser llamado de nuevo.

**Código especializado**

**Ecuaciones y números**

Se pueden realizar ecuaciones entre **varN** si son números. Suma con +, resta con -, división con /, multiplicación con \*, etc. Tambien existe la clase **Math.** para esto:

**varN = Math.nombreDeLaFuncion(**numero o nombre de la clase con el numero**)**

**varN = Math.round():** Redondea un número a su entero más cercano.

**varN = Math.floor():** Redondea un número a su entero menor cercano.

**varN = Math.ceil():** Redondea un número a su entero mayor cercano.

**varN = Math.sqrt():** Raíz cuadrada.

**varN = Math.abs ():** Valor absoluto.

**varN = Math.pow (**número a potenciar**,** número que potencia**):** Potenciación de un número.

**varN = Math.min(**numero1**,** numero2**,** numero3): Te dice el valor mínimo de una serie de números.

**varN = Math.max(**numero1**,** numero2**,** numero3): Te dice el valor máximo de una serie de números.

**varN++ :** Aumenta de a 1 el número la primera vez que vuelve a ser impreso en la consola.

**varN +=** numero**; :** La suma se realiza con el numero escrito y no con 1.

**varN-- :** Resta de a 1 el número la primera vez que vuelve a ser impreso en la consola.

**varN -=** numero**; :** La resta se realiza con el numero escrito y no con 1.

Si **varN1=”10”** y **varN2= 10**

**console.log(number(varN1) + varN2):** Lo que realiza **number** es transformar un string en un número. Este puede convertir a **true** en **1** y **false** en **0**.

**console.log(parseInt (varN1) + varN2):** Lo que realiza **parseInt** es transformar un string en un número pero siempre a un entero.

**console.log(parseFloat (varN1) + varN2):** Lo que realiza **parseFloat** es transformar un string en un número no importa si es entero o decimal.

**varN2 = String(varN2); :** Pasa el número a string, contrario a los anteriores.

**varN.toFixed (**número de decimales que toma**);** : Toma del número tantos decimales como le indiques.

**Fechas**

**let varN = new Date();** : Esto nos da la fecha de hoy. Al igual que **Math, Date,** es una clase específica para fechas. Con este podemos escribir una determinada fecha y con los códigos posteriores sacar información de ella (**let varN = new Date(**mes**-**día**-**año**);**

**varN = varN.getMonth()**: Nos da el mes de hoy (Enero = 0).

**varN = varN.getDay ()**: Nos da el dia de hoy, en número de la semana (domingo = 0).

**varN = varN.getDate ()**: Nos da el dia de hoy, en número del mes.

**varN = varN.getFullYear ()**: Nos da el año actual.

**varN = varN.setFullYear (**Año**)**: Nos deja sobrescribir el año actual. Debe estar entre dos **.getFullYear().**

**varN = varN.getHours ()**: Nos da la hora actual.

**varN = varN.getMinutes ()**: Nos da el minuto actual.

1. Imprime 30 usuarios en la pagina [↑](#footnote-ref-1)
2. Parámetro que me deja cambiar la cantidad de usuarios que quiero [↑](#footnote-ref-2)
3. Esta constante nos dejara entrar al sitio web [↑](#footnote-ref-3)
4. Nos deja cambiar la cantidad de resultados con el parametro [↑](#footnote-ref-4)
5. Fetch uti9liza la url que está en la **const api** para sacar contenido de ella. [↑](#footnote-ref-5)
6. Me retorna las respuestas como un archivo Json para imprimirlo [↑](#footnote-ref-6)
7. Función creada para imprimir el Html de cierta forma [↑](#footnote-ref-7)
8. Se imprimen los datos en la pagina [↑](#footnote-ref-8)
9. Reestructuramos la información de los datos creando el parámetro **usuario** y dándole valores. [↑](#footnote-ref-9)
10. Crea cada Li de nuestro documento html para que los datos se vean en la página. [↑](#footnote-ref-10)
11. Como la página de API randomizer ya nos crea el arreglo debemos sacar sus valores a través del object destructuring. [↑](#footnote-ref-11)
12. Crea el contenido de cada li [↑](#footnote-ref-12)
13. Imprime el li en nuestro section utilizando su ID. [↑](#footnote-ref-13)