Pagina web compementaria: https://www.w3schools.com/js/default.asp

# 0. Introducción Universidad Desarrollo Web

A tener en cuenta: Algunas clases no van a tener anotaciones ya que son conceptos que ya adquirí. A diferencia de las otras 2 secciones esta va a tener apuntes sueltos y no toda una expliación.

# 1. Introducción a JavaScript

## 1. Introducción a JavaScript

JavaScript es un lenguaje de programación interpretado, el cual puede ser ejecutado y entendido por navegadores web. Es uno de los lenguajes mas utilizados para navegadores.
Toda la interacción que tengamos con el navegador se lo considera Front-End. JavaScript posee frameworks para el front-end que facilitan y agilizan el coding de una pagina. Entre los frameworks se encuentran Angular.js (Google), Vue.js y React.js (Facebook).
JavaScript también puede ser utilizado para el back-end con el framework node.js.
El lenguaje se basa en el estándar Eggman Script y cada año se realiza una actualizacion que evoluciona el lenguaje.

## 2. Tecnologias a utilizar.

Vamos a utilizar node.js para el backend y ejecutar en tiempo real los programas que hagamos con JavaScript.
Vamos a instalar la extension de JavaScript para Visual Studio Code, Quokka.js, ESLint, Prettier - Code formatter, Bracket Pair Colorizer 2, Better Comments, Live Server y el Visual Studio IntelliCode.
Quokka.js is a developer productivity tool for rapid JavaScript / TypeScript prototyping. Runtime values are updated and displayed in your IDE next to your code, as you type.3. Diferencia entre var y let en JavaScript

## 3. Diferencia entre var y let en JavaScript

let te permite declarar variables limitando su alcance (scope) al bloque, declaración, o expresión donde se está usando. a diferencia de la palabra clave var la cual define una variable global o local en una función sin importar el ámbito del bloque. Source:
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/let

# 2.Tipos de Datos en JavaScript

Las variables declaradas en JavaScript se conocen como variables dinamicas. Esto quiere decir que en cualquier momento vamos a poder asignarle un nuevo valor a una variable previamente asignada con distinto tipo de dato.
Symbol es un tipo de datos cuyos valores son únicos e immutables. Dichos valores pueden ser utilizados como identificadores (claves) de las propiedades de los objetos.  Cada valor del tipo Symbol tiene asociado un valor del tipo String o Undefined que sirve únicamente como descripción del símbolo. Source:
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Symbol
Las clases tambien se consideran funciones en JavaScript.
Las variables Undefined son variables declaradas que no estan asociadas a ningún valor.

Null no es un tipo de dato es un tipo object.
Los arrays en JavaScript son de tipo object.
La palabra reservada var la utilizamos para declarar variables, sin embargo hoy en dia no se considera una buena práctica. Sustituyendo esta palabra tenemos las palabras reservadas let para una variable y const para una constante.
En JavaScript es posible declarar una variable sin asignarle una palabra reservada. No se recomienda ya que no es una buena práctica. Buenas prácticas son usar let y const hoy en dia.
El operador de comparación === y !== revisa los valores pero también los tipos. Ejemplo: a = 3 - c = "3" -> a==c -> true | a===c -> false.
Dentro de la clase Number podemos parsear strings a numeros.
Para que las conversiones puedan realizarse de manera correcta debemos verificar que la cadena sea un numero, para ello tenemos la herramientat IsNan que significa IsNotANumber.

## 3. Sentencias de Control en JavaScript

Podemos utilizar etiquetas o labels para indicar donde quiero que vaya el programa despues de, por ejemplo, un continue. No es una buena practica. Se declara de la siguiente manera: Inicio:. Se la conoce como goto.

## 4. Ciclos en JavaScript

Podemos utilizar etiquetas o labels para indicar donde quiero que vaya el programa despues de, por ejemplo, un continue. No es una buena practica. Se declara de la siguiente manera: Inicio:. Se la conoce como goto.

## 5. Arreglos en JavaScript

Los arrays en JavaScript se comportan como listas dinamicas de C#
let autos = new Array("BWM", "Mercedes Benz", "Volvo"); Es una practica vieja para declarar una array y ya no se recomienda utilizarla.
const autosDos = []; Es la forma que se recomienda. La constante va a almacenar la referencia de memoria del array.

## 6. Funciones en JavaScript

Definicion funciones anonimas: An anonymous function is a function without a name. An anonymous function is often not accessible after its initial creation.
Parametros vs Argumentos: A parameter is the variable which is part of the method's signature (method declaration). An argument is an expression used when calling the method.
Las funciones aplican el concepto de Hoisting. El Hoisting definir nuestra funcion en cualquier parte del programa y la vamos a poder mandar a llamar, ya sea antes de definirla y despues.
Ejemplo de declaracion de funcion: function NOMBRE(PARAMETROS){CUERPO}
Las funciones de tipo expresion se utilizan cuando definimos una variable y declaramos una funcion en la misma linea, esta variable va a llamar a la funcion cuando la utilicemos, guarda la referencia de la variable. Se la considera una funcion anonima.
Ejemplo: let sumar = function(a,b){return a+b}; | let resultado = sumar(4,4);
Las funciones self invoking se mandan a llamar a si mismas. Ejemplo:

```
(function (VARIABLES) {
   console.log("Ejecutando la funcion");
})(PARAMETROS);
```

Con los parentesis al final se manda a llamar a si misma, es una funcion anonima que no se puede reutilizar.

Las funciones puede ser descritas como objetos y los objetos en JavaScript pueden tener propiedades y métodos asociados a cada uno de los objetos.
La propiedad arguments.length dentro de la funcion nos dice cuantos elementos recibe nuestra funcion.
La función flecha es una función anonima que aplica Lambda. Es recomendable que la variable asociada a nuestro Lambda sea de tipo const.
Ejemplo de declaracion: const NOMBRE = (PARAMETROS) => (CUERPO); El return puede ser implicito si es una sola linea de codigo o explicito si tenga mas de una linea de codigo.
En JavaScript no es requerido que coincidan el numero de argumentos (variables a la hora de llamar a la funcion) con el numero de parametros (firma de la funcion).
Las funciones pueden tener parametros con valores predefindos, en caso de no pasar argumentos o pasar menos de los que tengo en mi funcion estos se mantienen y no se pisan.
Los tipos primitivos son aquellos datos que no tienen propiedades ni metodos, ni se le pueden asociar.

## 7. Objetos en JavaScript

Básicamente la diferencia entre un tipo primitivo, como puede ser un tipo numérico y un objeto, es que el tipo primitivo no contiene propiedades ni tampoco métodos.
Un objeto puede contener propiedades y métodos.
Otra manera de acceder a propiedades en JS, aparte del ., es accediendo como si fuese un array poniendo entre los corchetes el nombre de la propiedad. Esto nos permite recorrer la matriz con un for.
Ejemplo accediendo a la propiedad: persona['nombre'];
Ejemplo recorrer con un for:

```
for(propiedades in persona){
   console.log(persona);//Muestro todas las propiedades, genera que se itere un string con todos los datos
* cantida de propiedades
   console.log(persona[propiedades]); //Muestro linea por linea cada una de las propiedades.
}
```

En JavaScript se puede agregar propiedades simplemente haciendo OBJETO.PROPIEDAD, con el operador . como se ve. Permeable a muchisimos errores.
Con la palabra reservada delete OBJETO.PROPIEDAD podemos eliminar una propiedad, tambien es muy permeable a errores.

El metodo Object.values(OBJETO) nos devuelve un array con todas propiedades definidas en nuestro objeto.
El metodo JSON.stringify(OBJETO) nos devuelve una cadena en formato JSON con las propiedades y sus valores.
La sintaxis del get es muy similar a la de C#. Utilizamos la palabra reservada get. Ejemplo de get: get nombre() {return this.nombre;} - get nombreCompleto (){return this.nombre + " " + this.apellido}.
Los setters de igual manera que en C# funcionan de manera muy similar. Ejemplo: set Language(lang) {this.idioma = lang.toUpperCase();}
Si queremos generar constructores POR FUERA de una clase podemos generar una funcion. Ejemplo: function Persona(nombre, apellido, email){this.nombre = nombre; this.apellido = apellido;this.email = email;} y utilizando la palabra new se lo asignamos a una variable: let persona = new Persona("Juan", "Perez", "jperez@mail.com");
En JavaScript se puede poner directamente los simbolos {} para genarar el espacio en memoria, sin utilizar new. Ejemplo: let miObjeto = {}; Lo mismo aplica para los demas tipos de datos con sus caracteres respectivos, por ejemplo con un array: let miArray = [];
En general cada tipo de dato tiene su constructor de clase pero muchas veces no lo utilizamos, utilizamos directamente las conversiones implicitas que nos provee JavaScript.

Si queremos agregar un atributo/propiedad durante el transcurso del programa que afecte a todos los objetos del mismo tipo y no queremos editar el constructor (Horrible practica de programacion), podemos utilizar la propiedad de Prototype. Ejemplo: Persona.prototype.tel;

El metodo call nos va a permitir llamar a un metodo que esta definido en un objeto desde otro objeto. Es decir, vamos a poder utilizar una funcion definida en OBJETO1 y utilizarla con los parametros de OBJETO2.
Ejemplo: OBJETO1.FUNCION.call(OBJETO2);
Tambien podemos pasarle argumentos a nuestra funcion call cuando vayamos a utilizar metodos definidos en otro objeto, esto se logra simplemente poniendo una , luego del objeto que queremos aplicarle el metodo de otro objeto. Ejemplo: OBJETO1.MEOTOD.call(OBJETO2, ARGUMENTO, N CANTIDAD DE ARGUMENTOS);
El metodo apply funciona como el metodo call pero con una diferencia. El metodo apply requiere de un array con los argumentos que vayamos a utilizar.
Ejemplo: OBJETO1.FUNCION.apply(OBJETO2, ARRAY);

## 8. Clases en JavaScript

Las clases se declaran como en C# pero sin definir el scope.
Ejemplo: class X {Constructor-Atributos-Metodos}
Para generar un constructor vamos a utilizar la palabra reservada constructor.
Ejemplo: constructor(PARAMETROS){CUERPO}
A diferencia de C# los atributos los podemos definir dentro del constructor con this.ATRIBUTO.
Se recomienda usar un _ antes del nombre del atributo para que este no coincida con la propiedad.
Cuando trabajamos con clases no se aplica el concepto de Hoisting. Primero debemos definir nuestra clase para poderla utilizar.
https://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado
La sintaxis de herencia en JavaScript es: class HIJA extends PADRE.
Para llamar al constructo de la clase padre utilizamos la palabra reservada super en la primera linea despues de declrar el constructor en la clase hija. Ejemplo:

```
constructor(nombre, apellido, departamento){
   super(nombre, apellido);
}
```

Simplemente poniendo el nombre del metodo ya se esta definiendo.
Ejemplo: NombreCompleto(){}

Para sobreescribir un metodo podemos hacer volvemos a utilizar la palabra super. Ejemplo: NombreCompleto(){ return super.NombreCompleto() + " " + this._departamento}. Podemos obviar el super si es una clase padre.
En JavaScript por defecto todas las clases heredan implicitamente de Object, esto ocurre si no tiene un extends explicito.

## 9. Palabra Static en JavaScript

Para definir un metodo estatico (de clase) simplemente utilizamos la palabra static al comienzo de la firma del metodo.
Para definir un atributo estatico debe estar precedido por la palabra static.
Podemos agregar atributos por fuera de nuestra constructor, siendo estos atributos de los objetos.
Se pueden crear atributos estaticos al boleo si cometemos un error a la hora de llamar a la clase con la variable asociada.
Para definir un metodo statico que se mantenga constante podemos hacer un static get VARIABLE(){CUERPO} para poder tener un atributo constante estatico no modificable.

## 10. Ejercicio de Herencia en JavaScript

En JavaScript para que podamos ejecutar clases en diferentes archivos necesitamos manejar el concepto de módulos. Y para manejar el concepto de módulos necesitamos desplegar nuestra aplicación en un servidor. Más adelante vamos a poder como podemos separar las clases en diferentes archivos y ejecutarlas por separado.

## 11. Sistema de Ventas con JavaScript

Iterar una lista en JavaScript se hace definiendo una variable y utilizando la palabra reservada of. Ejemplo: let item of this._productos.

## 12. Proyecto Mundo PC con JavaScript

## 13. Modo Strict en JavaScript

El modo estricto de JavaScript no deja que usemos variables sin antes haberlas declarado. Se utiliza poniendo al principio de nuestro archivo de extensión .js "use strict";

## 14. POO en JavaScript

Utilizando el mismo nombre de un metodo padre en una clase hija estamos sobrescribiendo el metodo (polimorfismo o nuevo metodo). Para acceder al elemento padre tanto para constructores como para metodos utilizamos la palabra reservada super.

La palabra instanceof nos permite filtrar a que tipo de instancias queremos acceder en nuestro codigo. Parecido al typeof. La palabra instanceof responde a una relacion padre-hijo.
Ejemplo: PARAMETRO instanceof(CLASE).

## 15. Manejo de Errores en JavaScript

Los errores los manejamos con bloques try-catch-finally. El catch se hace a un objeto error (es el mas generico) que nos devuelve un mensaje cuando lo accedemos.
Ejemplo:

```
"use strict";
try{
   x = 10;
} catch(error){
   console.log(error);
}finally{
   console.log("El programa continua");
}
```

El throw envia un objeto de tipo error pero personalizando el mensaje que se envia en el objeto. Los errores tiene la propiedad name y message, si enviamos nuestro propio throw no se aplican estas propiedades.

## 16. Funciones Flecha (Arrow Functions) en JavaScript

Con la funcion flecha no se aplica el concepto de Hoisting.
Si utilizamos una sola linea de codigo no hace falta utilizar los simbolos {}.

## 17. Funciones Callback en JavaScript

Una función de tipo callback es una función que se pasa como parametro a otra función y dentro de una función vamos a poder llamar a otra función.
Aplica el concepto de Hoisting.
Ejemplo función callback:

```
function Sumar(op1, op2, funcionCallBack) {
   funcionCallBack(op1 + op2);
}
```

El objetivo de las funciones de tipo callback es que puedan ser utilizadas para procesos que se ejecutan de manera asincronica.
Cuando hablamos de procesos sincronicos, quiere decir que ejecuta una linea a la vez. Cuando tratamos con procesos asincronicos quiere decir que vamos a poder ejectuar varios procesos por separado.
https://stackoverflow.com/questions/8963209/does-async-programming-mean-multi-threading
https://itblogsogeti.com/2016/03/22/async-vs-multithread/

## 18. Promesas en JavaScript

En promesas utilizamos funciones de tipo callback.
Una promesa básicamente es código que tiene varios estados, así que vamos a poder lanzar una petición para procesar un código. En dado caso que la promesa se resuelta correctamente se resuelve el codigo y en caso que haya tenido problemas se manda a llamar el error.
Por lo tanto podemos decir que existen dos caminos: Un en el que se ejecuto el codigo correctamente y se resolvio el mismo y otro en el cual falla el codigo y se manda a llamar el error en el catch.
Mientras la promesa no haya terminado de ejecutar el codigo se encuentra en estado de pendiente. Una vez ejecutado el codigo si se puede resolver utilizamos la funcion .then(), en caso de que no se haya podido procesar utilizaremos la funcion .catch().
Sintaxis: Mirar el código.
Pagina de ejemplos y definiciones:
https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Using_promises
Una Promise (promesa en castellano) es un objeto que representa la terminación o el fracaso de una operación asíncrona. Dado que la mayoría de las personas consumen promises ya creadas, esta guía explicará primero cómo consumirlas, y luego cómo crearlas.
La palabra async nos va a permitir el uso de promesas. Al poner la palabra antes de la definición de un método significa que esta obligado a regresar una promesa.
La palabra reservada await solo puede utilizarse dentro de una función aysnc.

## 19. Manejo del DOM HTML con JavaScript

DOM: https://en.wikipedia.org/wiki/Document_Object_Model
El Document Object Model (DOM) son todos los elementos y sub elementos que componen un archivo html. Cuando trabajamos desde JavaScript cada uno de los elementos y sub elementos del archivo HTML se tratan como objetos. El document es el objeto mas generico y conforme desarrollamos la pagina se desprenden los demas objetos, mirar la imagen como referencia.

Los elementos script (si bien vamos a ver otra forma mas practica de utilizar JS) deben estar despues de todos los tags que componen el body para poder acceder a estos.
La propiedad getElementById("STRING-NOMBRE"), nos devuelve el tipo de objeto del tag. Para filtrar su contenido utilizamos el .innerHTML, esta propiedad podemos utilizarla tanto para lectura como para escritura.
Con la propiedad document.getElementsByTagName("ETIQUETA"); podemos tomar todos los tags que concuerden con la etiqueta. Si pasamos una etiqueta invalida lo tomara como si no existiera, o con 0.
Con la propiedad document.getElementsByClassName("NOMBRE DE LA CLASE"); podemos tomar todos los tags que concuerden con la clase CSS. Si pasamos una etiqueta invalida lo tomara como si no existiera, o con 0.
La propiedad document.querySelectorAll(TAG.CLASE); nos permite tomar todos los tags con una clase especifica.
Para acceder a los elementos de un formulario utilizamos el metodo de document.forms["ID DEL FORM"];
La propiedad document.write nos permite escribir con JavaScript sobre el HTML. Sin embargo al utilizarlo puede quitar el ultimo elemento HTML que encuentre. Depende de como se utilice.

## 20. Manejo de Eventos con DOM HTML y JavaScript

Las cookies son archivos que guardan información cuando estamos navegando por sitios web. Y esta información se guarda en el formato de key:value. Asi podemos asociar por ejemplo un nombre de usuario con el usuario (usuario:"nombre") y almacenarlo en una cookie.
Para no colocar cada uno de los eventos que queremos utilizar dentro de las etiquetas HTML podemos utilizar el metodo addEventListener. El evento asociado solo lleva el nombre y no la accion que indica el nombre (si el evento es onfocus solo lleva el focus).
Ejemplo: document.getElementById("ID").addEventListener("EVENTO", METODO-ASOCIADO);