**Intro Js**

A inicios de los 98 – Módems 28KBps 🡪 Permitió aplicaciones web más complejas

Lenguaje de programación del lado del cliente.

Brendan Eich -> Netscape 🡪 Primera versión, llamado LiveScript

Por marketing le pusieron JavaScript debido a la popularidad de Java.

1997 se envió la especificación de JS 1.1 a la ECMA -> European Computer Manufacturers Association 🡪 TC39 creo el primer estándar llamado ECMA-262

JavaScript no es más que la implementación que realizó la empresa Netscape del estándar ECMAScript.

**Usos de JavaScript**

Hoy día pueden correr en el cliente y en el servidor.

Podemos crear webservers con la ayuda de node js.

Se pueden crear apps móviles con ayuda de ionic, react native, native script….

Tecnologías como Angular, View, React.

OJOO

Cada navegador tiene su propia implementación del estándar de ECMAScript

**Node nos permite correr JavaScript por fuera del navegador web**

**Ej:** Abro git bash y escribo node + enter 🡪 Escribo console.log(‘Hola Mundo’)

**Ej:** En visual estudio code creo archivo app.js y lo corro con la terminal 🡪 **node app.js**

**Recomendación:** Los archivos de JavaScript se recomienda ponerlos al final, para que cuando cargue la página web sea eso lo último que carga y no haga más lenta la carga de la página.

Por rendimiento poner los scripts al final del HTML

**Versiones de JavaScript:** 2009 -> ECMAScript 5 -> Soportada en todos los navegadores web  
2015 -> ECMAScript 6 🡪 Constantes, let, clases, etc -> Mayoría de navegadores web

De ahí decidieron hacer lanzamientos anuales.

**Polyfill**

Código que provee el funcionamiento de una nueva característica de JavaScript (ES6), en versiones viejas como ES5.

**Consola JavaScript**

JavaScript es un lenguaje interpretado -> Usualmente se lee de manera secuencial

**Depurar en el navegador** 🡪 En sources pongo un punto de debug en la parte del js que quiero visualizar

**Depurar en VSCode** 🡪 Run debug y seleccionar node

**USO DE VAR**

Al usar var se está colocando la variable dentro de un objeto global llamado window.

Usar var no es buena práctica.

El primer problema es que puedo nombrar una variable igual a alguna propiedad del objeto window y puede ser algo difícil de lidiar 🡪 Por eso es mejor let y const, porque no sobreescriben el objeto global window o donde sea que está corriendo JavaScript.

Con let somos capaces de ejecutar un código más limpio.

**Undefined 🡪** Valor de cualquier variable que no esté inicializada.

**Alert, Prompt y Confirm** son funciones que están dentro del objeto window. Si ejecutamos código en un lugar donde no esté el objeto window, no va a funcionar. Por ejemplo en node no funcionaría.

*En node ---------- > Global es similar a Window en un navegador web.*

**FUNDAMENTOS DE JAVASCRIPT**

JavaScript es un lenguaje débilmente tipado. Js intuye si es String o número, etc.  
Tipo de dato 🡪 Describe el contenido del valor que tiene la variable

**Primitivos** -> Es una información que no es un objeto y son inmutables

**Tipos De Datos Primitivos 🡪** Hay 6 tipos de datos primitivos en JS

Boolean -> true/false  
Null -> Sin valor en lo absoluto  
Undefined -> Una variable declarada a la que áun no se le asigna valor  
Number -> Integers, floats, etc.  
String-> Una cadena de caracteres  
Symbol -> Es un valor único que no es igual a ningún otro valor

**PALABRAS RESERVADAS**

let, for, switch, enum, const, etc …

**ARREGLOS**

Lista de información que contiene un grupo de elementos, usualmente del mismo tipo, pero en JS no es una regla que sean del mismo dato… [] .. Empiezan con la posición 0

**OBJETOS LITERALES**

JS tiene dos tipos: Objetos y primitivos

Objetos literales, objeto que tiene pares de valores

Al hacer const a = {} me bloquea que pueda hacer luego a = 123;

**FUNCIONES**

Su objetivo es centralizar cierta lógica y permitir la reutilización.

Es buena práctica que la definición de las funciones esté al inicio.

No se recomienda del todo definir la función con function saludar() porque luego con un var saludar = 123 la puedo dañar.

Es buena práctica asignarla como función anónima a const.

Las funciones tradicionales (definidas con function) tienen internamente un objeto llamado arguments, con las funciones de flecha eso no ocurre.

Funciones de flecha comenzaron con el estándar ES6

En las funciones de flecha es buena práctica dejar el paréntesis () => {}

**Retorno:** Si la función no tiene la palabra return entonces retorna undefined

**Nota**: Las funciones de flecha no tienen internamente el objeto arguments como si lo tienen las funciones tradicionales -🡪 Si quiero obtener los argumentos todos en un arreglo con una función de flecha entonces uso (…args) eso se llama parámetro Rest, no pueden tener ningún otro parámetro después de ese. Si necesito extraer algo aparte se pone adelante.

Cuando la función retorna un arreglo y yo quiero obtener los valores individualmente entonces hago:

*const* [casado, vivo, nombre, saludo] = imprimeArgumentos2(10, true, false, 'Daniel', 'Hola');

Con esto extraigo cada valor de forma independiente y me evito acceder posición por posición al arreglo.

También sirve con objetos, solo que haría const {nombre, apellido} = crearPersona(‘Daniel’, ‘Campu’);

**Desestructuración de objetos**

*const* tony = {

    nombre: 'Tony Stark',

    codeName: 'Ironman',

    vivo: false,

    edad: 40,

    trajes: ['Mark I', 'Mark V', 'Hulkbuster']

};

*const* imprimePropiedades = ({ *nombre*, *codeName*, *vivo*, *edad* = 15, *trajes* }) *=>* {

    console.log({ nombre });

    console.log({ codeName });

    console.log({ vivo });

    console.log({ edad });

    console.log({ trajes });

}

imprimePropiedades(tony);

**ESTRUCTURAS DE CONTROL DE FLUJO**

**Valor Y Referencia 🡪**

**Cuando trabajamos con primitivos y los mandamos a una función como argumento lo estamos mandando por valor.**

**En JS todos los objetos son pasados por referencia**, en js todo es un objeto exceptuando los primitivos.

Rompiendo la referencia: El … en este caso se llama operador spread y saca todos los valores del objeto. Ayuda a pasar por valor y no por ref, no confundir con el parámetro Rest.

*let* juan = { nombre: 'Juan' };

*let* ana = {...juan }; //Forma de romper la referencia y pasar el objeto por valor, es como crear una copia

En el caso de pasar un objeto como valor a una función se haría así:

*const* cambiaNombre = ({...*persona* }) *=>* {

    persona.nombre = 'Tony';

    return persona;

}

*let* peter = { nombre: 'Peter' };

*let* tony = cambiaNombre(peter);

**En arreglos:**

*const* frutas = ['Manzana', 'Pera', 'Pina'];

*const* otrasFrutas = [...frutas];

**Otra forma con arreglos:**

*const* otrasFrutas = frutas.slice();

**DOM – Document Object Model**

Conjunto de objetos anidados para crear un documento HTML 🡪 Tener acceso dinámico a todos los elementos a través de código.

document.querySelector('body')

document.querySelectorAll('img')

document.querySelector('small').innerText = '10'

document.getElementById('computadora-cartas');

document.querySelector('#computadora-cartas')

document.querySelector('.carta');

document.querySelectorAll('.carta')

document.getElementsByClassName('carta');

**Recomendado usar el query selector**

**Creando elementos en el DOM por consola**

const divBotones = document.querySelector('#divBotones')

const botonNuevo = document.createElement('button');

Agregar botón al div

divBotones.append(botonNuevo)

botonNuevo.innerText = 'Destruir el mundo';

botonNuevo.classList.add('btn')

botonNuevo.classList.add('btn-success')

**CALLBACK: Función que se ejecuta o dispara con la ocurrencia de una acción.**

**SECCION 6**

Patrón módulo -> Es el patrón mas común 🡪 Es compatible con casi cualquier versión ecmascript, también permite encapsular el código.

(() *=>* {

})();

Esta sentencia crea un nuevo scope el cual no puede ser llamado por nombre. Es una función anónima autoinvocada.

'use strict'

Se recomienda usarlo siempre cuando se implementa el patrón módulo.

**HACER PUBLICAS COSAS DE MI MODULO**

*const* miModulo = (() *=>* {

return {

        nuevoJuego: inicializarJuego

    };

})();

**Luego desde el index.html**

<script>

miModulo.nuevoJuego();

</script>

**Minificación de código**

Código comprimido a su máxima expresión.

Como cuando un baja una librería por ejemplo underscore-min.js

Es bueno hacerlo cuando el código va para la web o cuando lo van a tocar más personas

**CLASES EN JAVASCRIPT**

Prototype: \_\_proto\_\_ ……Es el ADN del objeto, quien es su padre, madre, de donde fueron originados. El objeto sabe de quien viene.

Actualmente se recomienda trabajar con clases en vez de prototipe 🡪 Class 🡪 ES6

WebPack transformar código moderno de JS a una versión globalmente aceptada.

**Problemática y Necesidad de Las Clases**

Antes de ES6 se tuvo una **alternativa para la herencia**

**Problema, herencia con prototype y tener que definir con new**

*function Persona(nombre, edad) {this.nombre = nombre; this.edad=edad;}*

*const maría = new Persona(‘Maria’, 18);*

**Clases básicas en JS**

El constructor se ejecuta en el momento que se crea una nueva instancia de la clase

El constructor es el único método que retorna una instancia de un objeto en vez de undefined