***MÓDULO 3: JavaScript Avanzado: Buenas Prácticas***

1. ***INTRODUCCIÓN:***

***1. Presentación del curso***

Las buenas prácticas son un conjunto de recomendaciones basadas en las rutinas de otros desarrolladores y gracias a ellas, tu trabajo será más comprensible, mejor organizado y tendrá un mejor rendimiento. En este curso, se conocerán las prácticas más populares y eficaces, que harán que tu trabajo como desarrollador de JavaScript mejore y sea más práctico al trabajar en una aplicación web. Estas prácticas se convertirán en el set de acciones que incorporarás en cada proyecto web y que ayudarán que tu código sea mejor.

***2. Objetivos de Buenas Prácticas con JavaScript***

Las prácticas van a ir desde: la integración con variables, algunas estructuras y objetos, cálculos que podemos evitar, el trabajo con estilo, formas de declarar una variable, para qué me sirve la indentación, como trabajar con punto y coma, como puedo trabajar con comentarios y también generar documentación de mi aplicación con tan sólo escribir el comentario; así como también algunos puntos de organización de código que me va a permitir optimizar mis consultas o cálculos así también como evitar la duplicidad o establecer bloques bastante bien definidos sobre el uso de funciones. También, algunas cuestiones de performance como el uso de alguna notación de objetos como es ***JSON*** o cómo podemos realizar pruebas de rendimiento directamente en nuestro código, trabajando con JavaScript y la consola de nuestros ***Browser***. Además, podrás aprender cómo podemos minificar y ofuscar código para que nuestro trabajo sea muchísimo más sencillo y el uso de algunas herramientas como puede ser ***JSHint*** o ***JSLink*** para poder encontrar errores y tener un código bastante limpio y sofisticado en nuestro proyecto de desarrollo web.

***3. Archivos base de Buenas Practicas***

El curso cuenta con material de apoyo en lo que encontraré los archivos de cada lección cada uno con el ***.js*** y el ***.html*** asociado.

***4. Requisitos antes de comenzar este curso de JavaScript***

Contar con conocimientos básicos de programación y haber trabajo un poco con JavaScript. También, es importante contar con un editor de texto, que será: ***SublimeText***. También es necesario contar con un conocimiento básico de HTML.

1. ***Prácticas generales en JavaScript***

***1. Aprende HTML y CSS***

La potencia de JavaScript está dada por la relación que tiene con otros lenguajes, por ejemplo: HTML y CSS. Por eso es necesario contar con el conocimiento básico del funcionamiento de HTML y CSS.

HTML se encarga de dar toda la estructura de una página o aplicación web. CSS3, provee toda la vista que el usuario va a contemplar al ingresar a esa página web, mientras que JavaScript permite interactuar al usuario con la pagina web.

***2. Primero JavaScript***

Muchos desarrolladores que intentan aprender JavaScript se equivocan al intentar aprender este lenguaje a través de un framework, es decir, buscan primero aprender el uso de herramientas como ***jQuery***¸ ***AngularJS*** o ***React***, p.ej. para poder trabajar directamente con JavaScript. Primero hay que aprender JavaScript dado que es el lenguaje que da pie a ***frameworks*** esta talle y a herramientas poderosas como ***Cordova***, ***PhoneGap*** o ***Ionic***.

***3. Especificación y documentación de JavaScript***

Una buena práctica es consultar la documentación, la especificación o referencias del lenguaje en el que estemos trabajando, p.ej.: JavaScript. Es recomendable ingresar directamente al documento original de especificación del lenguaje y consultar todas las características que tiene.

También se pueden consultar alguna referencia que exista en la web donde podrás encontrar cuestiones de cómo se aplican algunas características de la herramienta hasta detalles de la librería y estos cómo te van a poder ayudar.

La documentación es pieza clave en todo desarrollo web.

***4. Test de capítulo***

***A. Lo primero que tenemos que aprender a manejar para desarrollar nuestras aplicaciones web es:***

- JQuery

- Angular.js

- JavaScript

***ene dad***

***B. ¿Por qué dos lenguajes vi a la potencia que tiene JavaScript?***

- HTML y CSS

- Kotlin y C++

- Visual Basic.Net y PHP

- Sublime Text y C

***C. HTML se encarga de dar la estructura a las páginas web, CSS de la parte más visual y JavaScript de la interacción.***

- Falso

- Verdadero

1. ***Variables, estructuras y objetos en JavaScript***

***1. Uso de variables en JavaScript***

Para declarar variables, en recomendable declararlas al inicio del programa, es decir, en la parte superior de nuestro código.

Hay dos formas de declarar estas variables:

Primera:

***var*** mensaje;

***var*** estado;

Segunda:

***var*** mensaje, estado;

Recordar de inicializar las variables.

Una manera eficiente de declarar variables es la siguiente:

***var*** mensaje = “Hola Mundo”,

estado = “activo”,

resultado;

Esta forma de expresar facilita la administración del código y permite obtener un código mas limpio aunque sea más estricto dado que si olvido alguna de las comas, consola arrojará error y el programa no funcionará correctamente.

Otra ventaja que ofrece es que permite reducir la posibilidad de tener alguna redeclaración, es decir, alguna variable que declaramos en un principio y que estemos declarándola nuevamente; en este caso, la consola nos avisará con un error que estamos declarando nuevamente dicha variable.

También, permite fácilmente detectar todas las variables si las ubicamos en la parte superior de nuestro código y con esto, además, podemos evitar algunas variables globales y tener bien identificado el alcance de cada una de ellas.

***2. Variables globales en JavaScript***

Conforme vamos trabajando en un proyecto en el cual vamos adoptando nuevas tecnologías o simplemente vamos generando más código separándolo en más archivos, se va haciendo más latente el problema de las variables globales, es decir, variables que puede tener declaradas en un archivo con un valor, y en otro archivo las voy a sobreescribir, tal vez no en forma directa pero la variable pierde el ***scope***, el alcance que estoy utilizando.

Este tipo de riesgo de colisiones en los nombres de variables que yo estoy utilizando, voy a poder compensarlo o solucionarlo utilizando un objeto en el que defina los datos como propiedades o como métodos, según necesite, que vaya a utilizar.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Archivo1: ***app.js***

***var*** mensaje = “Mensaje global”;

resultado = 45;

Archivo2: ***app2.js***

***var*** mensaje = “Mensaje local”;

resultado += 2;

***function*** saludar () {

***console.log***(mensaje);

***console.log***(resultado);

}

saludar ()

***3. Palabras reservadas en JavaScript***

Las palabras reservadas son aquellas que cada lenguaje se guarda para su uso funcional, es decir, palabras que tienen un significado y una función específica dentro del lenguaje. Por lo cual, debemos tener cuidado en no utilizar dichas palabras reservadas.

Un ejemplo puede ser cuando estoy declarando una función y declaro el nombre de esta.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

***function*** catch (ball) {

***console.log***(ball);

}

catch (“beisball”);

***NOTA:*** Dado que ***catch*** es una palabra reservada, tanto ***app.js*** como ***consola***, emitirán un mensaje de error en el que no se puede utilizar catch como nombre de una función dado que es una palabra reservada.

***4. Cálculos innecesarios en JavaScript***

Cuando estamos programando, hay ciertos cálculos que hacemos de una manera lógica.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Se puede optimizar la instrucción original mediante el artilugio que se presenta en “Código Optimizado”

Código Original:

***var*** animales = [“perro”, “gato”, “pez”];

***for*** (***var*** i = 0; i < animales.***length*** ; i++) {

***console.log*** (animales[i]);

};

Código Optimizado:

***var*** animales = [“perro”, “gato”, “pez”];

***var*** totalAnimales = animales.***length***;

***for*** (***var*** i = 0; i < totalAnimales ; i++) {

***console.log*** (animales[i]);

};

***NOTA:*** En muchos registros de datos, este artilugio, permite mejorar el performance de la aplicación dado que facilita el cálculo mejorando el proceso de instrucciones.

***5. Objetos y funciones anónimas en JavaScript***

Trabajar con funciones anónimas es una actividad muy común dentro de JavaScript pero tiene un pequeño problema, cuando empezamos hacer uso de muchas funciones anónimas o funciones anónimas anidadas u otras funciones anidades dentro de una función anónima ya anidada, suele ser una práctica muy común aunque puede mermar en nuestro proceso de depuración, mantenimiento, pruebas o reuso del código que estemos trabajando. Por tanto, es recomendable que nosotros podamos crear un objeto donde estemos definiendo tantas propiedades y métodos, sí, es acercarse un poco hacia la programación orientada a objetos.

Aunque en la especifición de JavaScript ya vienen algunos elementos de orientación a objetos, declarar este tipo de variables para poder minimizar el uso de funciones anónimas, me va ayudar bastante y me va a permitir generar código más fácil de mantener y de reutilizar.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

• HTML:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Buenas practicas con JavaScript</title>

</head>

<body>

<button id="btnSaludar">Saludar</button>

<script type= "text/javascript" src="app.js"></script>

</body>

</html>

*• JavaScript:*

***var*** tienda = {

nombre: “Tienda las 4 esquinas”,

calcular: ***function*** (costo1, costo2) {

***return*** costo1 + costo2;

},

saludar: ***funciton*** () {

***var*** mensaje = “Hola”;

***return*** mensaje;

};

***console.log***(tienda.nombre);

***console.log***(tienda.calcular(20,32));

***console.log***(tienda.saludar());

***var*** btnSaludar = ***document.getElementById*** (“btnSaludar”);

btnSaludar.***addEventListener***(‘***click***’, ***function*** () {

});

***6. For in en JavaScript***

A veces es necesario iterar sobre la propiedad de un objeto, ya sea para validar si la propiedad existe o para trabajar con el valor que tenga almacenado. Para ello es que se utiliza una variante del ciclo For llamado **For … in** …

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

• HTML:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script type= "text/javascript" src="app.js"></script>

</body>

</html>

*• JavaScript:*

**var** persona **= {**

nombre: "Damian",

email: "damian.angelucci@hotmail.com",

twitter: "dami.an91",

saludar: **function() {**

**return** "Hola mundo"

**}**

**}**

**var** dato **= "";**

**for** (dato **in** persona) {

**console.log**(dato, persona[dato]);

}

**NOTAS:**

1. **for** (dato **in** persona){. Permite almacenar TODOS los objetos de "persona" en la variable "dato"

2. **console.log**(dato, persona[dato]);. Imprime los valores de las propiedades (nombre, email, etc) del objeto.

***7. Validar objetos en JavaScript***

A través de un condicional **if**, puedo evaluar si objeto existe o no, en tales casos consola imprimirá un ***true*** o un ***error*** respectivamente.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

• HTML:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script type= "text/javascript" src="app.js"></script>

</body>

</html>

*• JavaScript:*

**var** persona **= {}**;

**if** (persona) **{**

**console.log**("La persona existe");

**} else {**

**console.log**("La NO persona existe");

**}**

***8. Expresiones booleanas en JavaScript***

No es siempre necesario tener una evaluación como la de la lección anterior.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**var** estado = true,

resultado = “”;

**if** (estado) {

**console.log**(“continuar”);

} **else** {

**console.log**(“detenerse”);

}

***9. Operador Ternario en JavaScript***

Permite convertir un tipo de estructuras **if** y **else** de una sola instrucción en una sola línea.

El operador se denota con el símbolo “**?**”.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**var** estado = “true”;

**var** acción = estado  **?** ‘continuar’ **:** ‘detenerse’;

**NOTAS:**

1. **var** acción = estado  **?** ‘continuar’ **:** ‘detenerse’; //Si estado es Verdadero, entonces “continuar” sino “detenerse”.

2. El operador ternario, reemplazaría a la estructura:

**if** (estado) {

**console.log**(“continuar”);

} **else** {

**console.log**(“detenerse”);

}

***10. Igualdad en JavaScript***

En JavaScript existen los operadores “===” y “==”. El primero, evalua dos elementos basados en el tipo de dato y el valor; el segundo, trata hacer la comparación a como dé lugar, por lo tanto, es posible que ocurran errores.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**console.log**(1 **===** ‘1’);

**console.log**(1 **==** ‘1’);

**NOTA:** Para el primer valor, consola arroja un resultado **false** que es algo lógico dado que se compara un número entero con un carácter, mientras que el segundo valor, resulta **true** dado que NO tiene en cuenta que se trata de un carácter. Entonces, SIEMPRE es recomendable utilizar el operador “===”.

***11. Constructores de objetos en JavaScript***

A través de la palabra reservada **this**, se puede inicializar las propiedades de un objeto.

Ejemplo: **this.**nombre = nombre;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**var** Persona (nombre, apellidos) {

**this.**nombre = nombre;

**this.**apellidos = apellidos;

}

**var** persona = **new** Persona(“Damian”, “Angelucci”);

**console.log**(persona);

**console.log**(persona.nombre);

**console.log**(persona.apellidos);

**NOTA:**

1. **var** persona = **new** Persona(“Damian”, “Angelucci”); //Permite agregar las propiedades de una persona al objeto “Persona”.

***12. Try and catch con JavaScript***

El método **try** permite poner a prueba un bloque de código, si éste presenta algún error, puedo manejarlo utilizando **catch**.

Dentro del método **try**, puedo poner TODAS las funciones que quiero probar mientras que si llega a presentar algún error, podemos darle solución con el método **catch**.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**var** animales = [“perro”, “gato”, “pez”];

**var** totalAnimales = animales**.lenght**;

**try** {

*//Acciones a realizar por la excepción*

**for** (**var** i = 0; i < totalAnimales; i++) {

}

}

**catch** (e) {

*//Manejo de errores*

}

***13. Literales en JavaScript***

Una de las formas más prácticas de poder utilizar un objeto y un arreglo en JavaScript es a partir de las **literales**, es decir, cuando NO estemos utilizando el objeto **new**, que sólo lo emplearemos cuando se utilicen muchas propiedades de este objeto y no solo una fracción minúscula de él.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**var** persona = {};

persona.nombre = “Damian”;

**var** otraPersona = **new Object()**;

**var** lista = **new Array()**;

lista[0] = “manzana”;

lista[1] = “pera”;

**var** listaNueva = [“manzana”, “pera”];

**NOTAS:**

1. **var** otraPersona = **new Object()**; //Permite declarar un Objeto.

2. **var** lista = **new Array()**; //Permite declarar un Arreglo.

3. Las maneras de declarar un Objeto y/o un Arreglo en comparación a la utilización de **literales**, va a depender de la cantidad de información que se almacene. Para poca información, es suficiente utilizar **literales**.

***14. Test Capítulo 3:***

***A. La palabra reservada que nos va a permitir a realizar una inicialización de las propiedades de los objetos es***:

- this

- that

- object

- init

***B. Tenemos la siguiente secuencia de código:***

*var animales = ["perro", "gato", "pez"];*

*var totalAnimales = animales.length;*

*for (var i = 0; i < animales.length; i++)*

***en donde var totalAnimales = animales.length sirve para:***

- especificar el tamaño que tiene el array animales.

- mostrar el número de caracteres que componen el array animales.

- ver la posición que indican los registros del array animales.

- especificar los datos que contiene el array animales.

***C. Para no hacer uso de muchas funciones anónimas, lo más recomendable es crear*** …….. ***donde se definan propiedades y*** ……..

- una clase / funciones

- un objeto / métodos

- una variable / objetos

- una propiedad / clases

***D. Qué evalua el operador del triple igual?***

- varias cadenas de código

- los elementos basándose en tipo y valor

- dos condiciones

- dos valores, uno positivo y otro negativo.

***E. Todos los lenguajes de programación utilizan palabras reservadas. ¿Qué son estas palabras?***

- Las que no se utilizan nunca porque podrían provocar un fallo en la aplicación.

- Son aquellas que cada lenguaje se guarda para su uso funcional.

- Los nombres de algunas variables concretas.

- Las que se reservan por si hubiera que incorporarlas más adelante.

***F. La instancia***

var persona = {};

***hace referencia a la creación de un objeto al que agregamos …..***

- una serie de condiciones

- muchas propiedades o métodos

- expresiones booleanas

- pocas propiedades o métodos.

***G. ¿Cómo se pueden evitar errores en la validación de existencia de objetos?***

- Preguntando si el objeto existe.

- Ejecutando el código correspondiente.

- Comprobando que la sintaxis sea correcta.

- Usando la variable adecuada en cada caso.

***H. ¿En qué momento es recomendable declarar variables?***

- Al final de la cadena de código.

- Una vez hayamos escrito una cadena de código larga.

- Hacia mitad del código.

- Al inicio del programa.

***I. Las expresiones booleanas se usan porque pueden tener dos tipos de valores: positivo y ……..***

- negativo

- entero

- numérico

- alfanumérico

***J. Existen dos formas de declarar variables: usando la palabra reservada var y separando por ……..***

- punto y coma (;).

- barra inclinada (/).

- comas (,).

- asterisco (\*).

***K. El operador ……… va a permitir convertir las consultar if y else de una sola instrucción …….***

- binario / a dos operandos.

- unario / a un operando.

- condicional / a varias líneas.

- ternario / a una sola línea.

***L. Podemos evitar el riesgo de colisiones en los nombres de variables ……..***

- como no existe nada que evite este riesgo, tenemos que escribir cada variable con su propio nombre.

- utilizando especificadores en cada una de las variables usadas.

- utilizando un objeto donde se definan los datos como propiedades o como métodos.

- intercambiando mayúsculas y minúsculas, pues JavaScript discrimina entre ambas.

***M. A veces es necesario iterar sobre las propiedades de un objeto, tanto para validar si la propiedad existe como para trabajar con el valor que tenga asignado. ¿Qué ciclo vamos a usar para llevar a cabo esta iteración?***

- foro out.

- for in.

- for.

- for to.

***N. Una de las formas más prácticas de poder utilizar un objeto o un array en JavaScript es a partir del uso del ciclo for.***

- Verdadero.

- Falso.

***O. Una de las principales carencias del lenguaje JavaScript es que ……….***

- requiere trabajar juntamente con otros lenguajes de programación como Python o C++.

- necesita de CSS para poder utilizarse.

- no es un lenguaje fuertemente tipado.

- es un lenguaje fuertemente tipado.

1. ***ESTILO EN JAVASCRIPT***

***4.1 Formato de variables en JavaScript***

Para poder escribir nombres de variables o de funciones, podemos seguir también un estilo de formato, por ejemplo, podemos trabajar con el estilo **CamelCase** o el estilo **SnakeCase**.

El estilo **CamelCase** va de la siguiente manera:

**var** nombreCompleto = “Damian Angelucci”; //Variable

**function** saludarPersona() { } //Función

Es decir, indica que la primera letra que el nombre va en minúscula y la siguiente, en mayúscula “nombreCompleto”. Indica el cambio de palabra en la variable.

El estilo **SnakeCase** va de la siguiente manera:

**var** nombre\_completo = “Damian Angelucci”.

Nótese que utiliza el guión bajo para separar las palabras.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**var** nombreCompleto = “Damian Angelucci”;

**var** nombre\_Completo = “Damian Angelucci”;

**function** saludarPersona() {

}

***4.2 Constantes en JavaScript***

Para el caso de las constantes, será necesario seguir un patrón estándar que indica que el nombre de las constantes debe ir escrito en mayúsculas. En el caso que el nombre de una constante esté compuesta por dos palabras será necesario utilizar el estilo **SnakeCase**.

Ejemplo:

**var** LIMITE = 30;

**var** LIMITE\_TOTAL = 30;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**var** LIMITE = 30;

**var** LIMITE\_TOTAL = 30;

***4.3 Indentar código en JavaScript***

Cuando el código de nuestra aplicación comienza a crecer, va ser muy importante tenerlo bien indentado ayuda a entender bien el código del programa.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**var** tienda = {

nombre: “Tienda las 4 esquinas”,

calcular: **function**(costo1, costo2) {

**return** costo1 + costo2;

},

saludar: **function**() {

**var** mensaje = “Hola”;

**return** mensaje;

}

};

**var** btnSaludar = **document.getElementById**(“btnSaludar”);

btnSaludar.**addEventListener**(‘click’, **function**() {

**console.log**(tienda.saludar());

});

***4.4 Uso de punto y coma en JavaScript***

El punto y coma es el símbolo que me permite saber cuando una instrucción está terminando, es decir, en que momento estará terminando de ejecutarse o declararse.

JavaScript no exige la utilización del punto y coma pero, colocar dicho símbolo, es una buena práctica para este lenguaje de programación.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**var** tienda = {

nombre: “Tienda las 4 esquinas”,

calcular: **function**(costo1, costo2) {

**return** costo1 + costo2;

},

saludar: **function**() {

**var** mensaje = “Hola”;

**return** mensaje;

}

};

**var** btnSaludar = **document.getElementById**(“btnSaludar”);

btnSaludar.**addEventListener**(‘click’, **function**() {

**console.log**(tienda.saludar());

});

***4.5 Abreviatuas en JavaScript***

Cuando tenga mucho código y estoy haciendo interacciones continuas con HTML, es necesario utilizar algún tipo de abreviatura para poder entender las variables y el tipo de dato que este contiene.

Ejemplo:

**var** arrAnimales = [“perro”, “gato”, “pez”] //arr de Arreglo

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• HTML:*

**<!DOCTYPE html>**

**<html lang=**"en"**>**

**<head>**

**<meta charset="**UTF-8"**>**

**<meta http-equiv="X-UA-Compatible" content="IE=edge">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>**Buenas Prácticas en JavaScript**</title>**

**</head>**

**<body>**

**<form id=**"frmRegistro"**>** //Formulario

**<button id=**"btnSaludar"**>**Saludar**</button>** //Boton

**</form>**

**<script type=** "text/javascript" **src=**"app.js"**></script>**

**</body>**

**</html>**

*• JavaScript:*

**var** aarrAnimales = [“perro”, “gato”, “pez”]

**var** btnSaludar = **document.getElementById**(“btnSaludar”);

**var** frmRegistro = **document.getElementById**(“frmRegistro”);

***4.6 Comentarios en JavaScript***

Escribir comentarios permite entender mejor el objetivo de cada sección del programa que esté realizando. Para poder hacer comentarios hay dos maneras: la primera, es escribiendo doble barra (**//**) que indica que es el comentario de UNA sola línea y, la segunda comentando bloque de código de esta manera **/\* \*/**, permite comentar varias líneas de código.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**var** nombre = “Damian”,

email = “[damian.angelucci@hotmail.com](mailto:damian.angelucci@hotmail.com)”;

//COMENTARIOS BASADOS EN JSDOC

/\*\*

\*

\* Función para enviar saludos

\* @param {string} nombre - Nombre de la persona a saludar

\*/

//Tambien existe DOCBLOCK para generar documentación

**function** saludar (nombre) {

//Se retorna el mensaje formado

**return** “Hola “+ nombre;

}

/\*

\*/

**function** calcular(datoA, datoB, datoC) {

**var** resultado = datoA + datoB;

**return** resultado;

}

**NOTA:** Para documentar las variables o funciones del programa es recomendable hacerlo a partir de la normalización de JSDoc o DocBlock.

***4.7 Uso de comillas en JavaScript***

Las comillas me permiten trabajar y agrupar cadenas de texto. Hay dos tipos de comillas: las comillas dobles y las comillas simples, pero ambas pueden interactuar. Cuando comienzo con un tipo de comilla debo terminar con el mismo tipo.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**var** mensaje = “La actividad estuvo ‘divertida’”;

**var** mensajeCompleto = ‘La actividad estuvo “divertida”’;

**console.log**(mensaje);

**console.log**(mensajeCompleto);

**NOTA:** **var** mensajeCompleto = ‘La actividad estuvo ‘divertida’’; //En JavaScript, el uso de comillas simples o dobles dentro de las comillas simples o dobles respectivamente, indica el uso de una variable.

***4.8 Uso de JSLint***

JSLint es un servicio que permite validar el código contra reglas establecidas para generar un código limpio.

Entro a la página web de JSLint, copio el código a validar y la página web evalúa e indica las cosas que debería estar haciendo para tener un código limpio.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**var** tienda = {

nombre: “Tienda las 4 esquinas”,

calcular: **function**(costo1, costo2) {

**return** costo1 + costo2;

},

saludar: **function**() {

**var** mensaje = “Hola”;

**return** mensaje;

}

};

**var** btnSaludar = **document.getElementById**(“btnSaludar”);

btnSaludar.**addEventListener**(‘click’, **function**() {

**console.log**(tienda.saludar());

});

***4.9 Test Capítulo 4:***

***A. ¿Con qué símbolo vamos a determinar que una instrucción está a punto de terminar de ejecutarse o de declararse?***

- con punto y coma (;).

- con diagonal (/).

- con coma (,).

- con doble diagonal (//).

***B. Para validar el código contra reglas establecidas y generar un código limpio, usamos ……..***

- JSLint.

- JSON.

- Concat.

- JSHint.

***C. Si tengo declara la variable***

***var*** *nombreCompleto = “Pepito Pérez”;*

***¿qué estilo de escritura estoy usando?***

- Upper Case.

- Kebab Case.

- Snake Case.

- Camel Case.

***D. Cuando usamos constantes en JavaScript, su nombre debe ir escrito en ……..***

- minúsculas.

- es indiferente.

- mayúsculas.

- mayúscula la primera y el resto en minúsculas.

***E. Las abreviaturas sirven para …….. y el tipo de dato que contienen.***

- facilitar el trabajo al programador.

- entender las variables.

- ahorrar líneas de programación.

- escribir menos código.

***F. ¿Qué dos maneras conoces para escribir comentarios en JavaScript?***

- 1. Usando la doble diagonal (//). 2. Con los caracteres /\* \*/.

- 1. Usando los caracteres \*/ /\*. 2. Usando la expresión defComent.

- 1. Con la parabra reservada Coment. 2. Con /Coment.

- 1. Usando la diagonal (/). 2. Con un doble asterisco (\* \*).

***G. Cuando el nombre de una constante está formado por dos o más palabras, ¿qué formato de escritura es más adecuado utilizar?***

- Camel Case.

- Snake Case.

- Upper Case.

- Kebab Case.

***H. Con ………. trabajamos y agrupamos cadenas de texto.***

- los asteriscos.

- los corchetes.

- las comillas.

- las llaves.

***I. …….. un código significa dejar sangrías y espacios suficientes entre cada una de las expresiones que van a componer cualquier elemento de mi programa.***

- depurar.

- estructurar.

- indentar.

- actualizar.

1. ***ORGANIZACIÓN DE CÓDIGO EN JAVASCRIPT***

***5.1 Elimina consultas innecesarias en JavaScript***

Este tipo de consultas van a consumir recursos de una u otra manera de nuestros dispositivos. Así sea un recurso mínimo, siempre es importante ahorrar lo más posible.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• HTML:*

**<!DOCTYPE html>**

**<html lang=**"en"**>**

**<head>**

**<meta charset="**UTF-8"**>**

**<meta http-equiv="X-UA-Compatible" content="IE=edge">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>**Buenas Prácticas en JavaScript**</title>**

**</head>**

**<body>**

**<button id=**"btnSaludar"**>**Saludar**</button>**

**<script type=** "text/javascript" **src=**"app.js"**></script>**

**</body>**

**</html>**

*• JavaScript:*

//**window.onload = function() {**

**var** btnSaludar = **document.getEventById**(“btnSaludar”);

btnSaludar**.addEventListener**(‘click’, **function()** {

**console.log**(“Hola Mundo”);

})

//}

**NOTA:** Las líneas comentadas no son necesarias en el código optimizado.

***5.2 Archivos separados en JavaScript***

Siempre es importante mantener bien organizado nuestro código dado que generalmente vamos a ocupar muchísimos archivos, tanto de JS como CSS o HTML, por lo que es conveniente, realizar una separación adecuada y tengamos carpetas exclusivas para cada lenguaje y/o librerías “libs”. Esto permite que el código de JavaScript se va a trabajar de manera independiente. Esta separación se archivos se denomina “módulización”.

***5.3 Evitar la duplicidad en JavaScript***

En muchas ocasiones, en nuestro código, nos daremos cuenta de que estamos utilizando muchas instrucciones repetidas. Cuando caigamos en esta situación, es momento de trabajar con funciones, es decir, encapsular todas estas acciones en un solo objeto que yo pueda estar reutilizando después.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

Método 1:

**var** cuenta = 2 + 3;

**var** cuentaNueva = 20 + 45;

**var** totalFinal = cuenta + cuentaNueva;

**console.log**(totalFinal);

Método 2:

**function suma**(datoA, datoB) {

**var** resultado = datoA + datoB;

**return** resultado;

}

suma (2,3);

Método 3:

**var** cuentaFinal = suma(suma(2,3), suma(20,45));

**console.log**(cuentaFinal);

**NOTA:** El método 3 es el más efectivo para realizar la operación.

***5.4 Eventos y HTML en JavaScript***

Muchas veces nos vamos a encontrar en la situación donde necesitemos interactuar elementos de HTML con códigos de JavaScript, por ejemplo un botón.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• HTML:*

**<!DOCTYPE html>**

**<html lang=**"en"**>**

**<head>**

**<meta charset="**UTF-8"**>**

**<meta http-equiv="X-UA-Compatible" content="IE=edge">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>**Buenas Prácticas en JavaScript**</title>**

**</head>**

**<body>**

**<button id=**"btnSaludar"**>**Saludar**</button>**

**<script type=** "text/javascript" **src=**"app.js"**></script>**

**</body>**

**</html>**

*• JavaScript:*

**var** btnSaludar = **document.getElementById**(“btnSaludar”);

btnSaludar**.addEventListener**(‘click’, **function()** {

**console.log**(“Hola Mundo”);

}

**NOTA:** Una interacción posible entre HTML y JavaScript puede ser la creación de un botón, en este caso, “Saludar”.

***5.5 Bloques explícitos de funciones en JavaScript***

A trabajar con JavaScript y algunas estructuras de control, como lo es el **if**, vamos a trabajar, generalmente, con algunas formas abreviadas. Es decir, a veces no es necesario estar indicando el uso de llaves; estos son los casos cuando se trabaja con UNA sola línea.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**var** estado = **true**;

**if**(estado)

**console.log**(“avanzar”);

***5.6 Test Capítulo 5:***

***A. Cuando en la consola pulsamos varias veces un botón, aparece un número al lado del texto que aparecerá al realizar esa acción de pulsar. ¿Puedes explicar qué significa ese número?***

- La cantidad de veces que se va a repetir el texto al pulsar el botón.

- Las líneas de código necesarias para que se ejecute la acción.

- La cantidad de veces que hemos pulsado el botón.

- La cantidad de pruebas que hemos hecho para que el botón funcione correctamente.

***B. ¿En qué parte de la estructura de la página escribimos normalmente la carga de los scripts?***

- En la etiqueta <body>.

- En la etiqueta <html>.

- En la etiqueta <head>.

- En la etiqueta <title>.

***C. Tenemos la sentencia***

**var** estado = true;

**if** (estado)

**console.log**(“avanzar”);

**var** resultado = 2+2;

***y queremos que en la consola se muestre el valor avanzar, pero lo único que aparece es un error. ¿Por qué ocurre esto?***

- Porque faltan las llaves después de if(estado) y de var resultado = 2+2;

- Porque esta sentencia solo aplica cuando trabaja con una sola línea.

- Porque hemos escrito mal el código. En lugar de if(estado) debería ser for(estado).

- Porque var estado = true es incorrecto. Debería ser var estado = false.

***D. La carpeta libs acoge todas las librerías relacionadas con JavaScript y con …….***

- CSS

- HTML

- JSON

- media query

***E. Para evitar utilizar muchas instrucciones repetidas, vamos a trabajar con un código de depuración***.

- Falso.

- Verdadero.

1. ***PERFORMANCE EN JAVASCRIPT***

***6.1 Usa JSON***

Trabajar con datos cuando estamos desarrollando una aplicación con JavaScript es una tarea sencilla cuando utilizamos JSON. JSON, como sus iniciales indican, “JavaScript Object Notation”, es un formato para el intercambio de datos. Básicamente, JSON va a describir la información que nosotros queremos utilizar. JSON nació como una alternativa a XML, el cual es más lento dada la cantidad de etiquetas que se tienen que parsear e imprimir cuando nuestra aplicación se está ejecutando.

JSON trata los datos como objetos, por lo que la implementación con JavaScript es más sencilla, rápido y transparante.

Una de las ventajas que tiene trabajar con JSON es la popularidad que tiene trabajar con este formato tanto que se ha vuelto un estándar entre los desarrolladores para poder transmitir datos entre distintas aplicaciones. Por ejemplo, podemos revisar una API del clima quien me puede regresar un JSON y, de hecho, esta API tiene una consula donde puedo armar mi propia consulta.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• HTML:*

**!DOCTYPE html>**

**<html lang=**"en"**>**

**<head>**

**<meta charset="**UTF-8"**>**

**<meta http-equiv="X-UA-Compatible" content="IE=edge">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>**Buenas Prácticas en JavaScript**</title>**

**</head>**

**<body>**

**<script scr=** "//code.jquery.com/jquery-1.11.3.min.js"**></script>**

**<script scr=** "//code.jquery.com/jquery-migrate-1.2.1.min.js"**></script>**

**<script type=** "text/javascript" **src=**"app.js"**></script>**

**</body>**

**</html>**

*• JavaScript – Ejemplo 1: JSON local:*

**var** url = "dirección del clima";

**$.getJSON**(url, **function**(productos) {

**console.log**(productos);

})

*• JavaScript – Ejemplo 2: JSON API del clima:*

**$.getJSON**("productos.json", function(productos) {

**console.log**(productos);

**var** totalProductos = productos**.length**;

**for** (var i = 0; i < totalProductos; i++) {

**console.log**(productos[i].nombre);

}

})

*• JSON:*

[

{"nombre":"manzanas"},

{"nombre":"peras"},

{"nombre":"zanahorias"},

{"nombre":"uvas"},

{"nombre":"naranjas"},

{"nombre":"cafe"},

{"nombre":"agua"},

]

**NOTA:** NO FUNCIONÓ EL PROGRAMA.

***6.2 Join vs Concat***

Los arreglos de JavaScript poseen métodos que nos van a facilitar mucho la vida. En algunos de estos casos, nosotros necesitamos, por ejemplo, hacer una concatenación entre elementos de un arreglo.

El método **join** permite unir elementos de un arreglo a través de un símbolo que le indique, por ejemplo la coma. Su resultado es una cadena de texto. El método **concat** permite trabajar con los elementos separados de un arreglo o agregar nuevos elementos en él. Su resultado es un arreglo.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**var** animales = [“perro”, “gato”, “pez”];

**var** listaAnimales = animales**.join**(“,”);

**var** nuevaLista = animales**.concat**();

**console.log**(listaAnimales);

**console.log**(**typeof** listaAnimales);

**console.log**(nuevaLista);

**console.log**(**typeof** nuevaLista);

***6.3 Pruebas de rendimiento de JavaScript***

Regularmente comenzamos a escribir el código y a veces lo hacemos de una manera tan automática que simplemente no nos damos cuenta de si estamos impactando o no en el performance de nuestra aplicación con ciertos bloques de código. Por tanto, es muy recomendable medir el “performance” de toda línea que nosotros estemos ejecutando. Para esto, vamos a utilizar la consola, pero vamos a trabajar con un método que se llama **time**. Este método, debe finalizar con un **timeEnd** en la última línea de código.

El método mencionado arrojará el tiempo que tarde el ejecutarse el código. Dicho tiempo, depende también del navegador que estemos utilizando.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

**console.time**(‘revisión’)

**var** animales = [“perro”, “gato”, “pez”];

**var** l = animales**.length**;

**for** (**var** i = 0; i < 10000; i++) {

animales[i] = i;

}

**console.timeEnd**(‘revisión’)

**NOTA:** El argumento del método **time** debe ser el mismo tanto en el inicio como en el final de la prueba.

***6.4 Minificar código de JavaScript***

Minificar el código es una práctica que se ha vuelto muy popular, no solamente porque hace que el código cambie sino porque me permite tener un código bastante optimizado y que la ejecución mucho mejor. Por ejemplo, cuando utilizo un código de JavaScript que está minificado, ese tipo de códigos no va a incluir espacios en blanco, saltos de líneas, etc. De hecho, cuando ves un código que aparece que está todo en línea seguida y que no hay apertura para poder leerlo, es cuando decimos que este código minificado. Las ventajas apuntan, en primera instancia, hacia el performance de la ejecución, es decir, el tiempo que va a tardar en cargar y ejecutarse cada una de las líneas de código que haya desarrollado.

En internet, hay muchas herramientas para la comprensión de códigos, por ejemplo **jscompress.com**. También existen frameworks que cuentan con este tipo de herramientas.

Otra de las ideas de tener el código minificado es que puedas, no entenderlo. Es decir, simplemente a veces queremos tener una cierta capa de “seguridad” dado que representa una dificultad para que otro usuario pueda entender lo que está pasando en ese archivo de JavaScript.

Para minificar el código hay que dirigirse a la página jscompress.com, entonces:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Luego de la minificación del código, hemos notado que algunos parámetros han cambiado de nombre, esto se lo llama **ofuscación**. Esto sucede para que algunos nombres de variables, parámetros, etc. no entren en conflicto con algunas otras cosas. De manera online, se puede realizar la ofuscación del código a través de la página **javascriptobfuscator.com**. Luego de realizar la operación, el código no se entiende porque se traduce a una expresión **hexadecimal** pero sigue funcionando sin problemas.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript – código original:*

**var** tienda = {

nombre: “Tienda las 4 esquinas”,

calcular: **function**(costo1, costo2) {

**return** costo1 + costo2;

},

saludar: **function**() {

**var** mensaje = “Hola”;

**return** mensaje;

}

};

**var** btnSaludar = **document.getElementById**(“btnSaludar”);

btnSaludar.**addEventListener**(‘click’, **function**() {

**console.log**(tienda.saludar());

});

**console.log**(tienda.nombre);

**console.log**(tienda.calcular(20,32));

**console.log**(tienda.saludar());

*• JavaScript – código minificado:*

var tienda={nombre:"Tienda las 4 esquinas",calcular:function(n,a){return n+a},saludar:function(){return"Hola"}},btnSaludar=document.getElementById("btnSaludar");btnSaludar.addEventListener("click",function(){console.log(tienda.saludar())});

**console.log**(tienda.nombre);

**console.log**(tienda.calcular(20,32));

**console.log**(tienda.saludar());

*• JavaScript – código ofuscado:*

var \_0xa449=["\x54\x69\x65\x6E\x64\x61\x20\x6C\x61\x73\x20\x34\x20\x65\x73\x71\x75\x69\x6E\x61\x73","\x48\x6F\x6C\x61","\x62\x74\x6E\x53\x61\x6C\x75\x64\x61\x72","\x67\x65\x74\x45\x6C\x65\x6D\x65\x6E\x74\x42\x79\x49\x64","\x63\x6C\x69\x63\x6B","\x73\x61\x6C\x75\x64\x61\x72","\x6C\x6F\x67","\x61\x64\x64\x45\x76\x65\x6E\x74\x4C\x69\x73\x74\x65\x6E\x65\x72"];var tienda={nombre:\_0xa449[0],calcular:function(\_0xd344x2,\_0xd344x3){return \_0xd344x2+ \_0xd344x3},saludar:function(){var \_0xd344x4=\_0xa449[1];return \_0xd344x4}};var btnSaludar=document[\_0xa449[3]](\_0xa449[2]);btnSaludar[\_0xa449[7]](\_0xa449[4],function(){console[\_0xa449[6]](tienda[\_0xa449[5]]())})

**console.log**(tienda.nombre);

**console.log**(tienda.calcular(20,32));

**console.log**(tienda.saludar());

***6.5 Uso de JSHint***

En muchas ocasiones el navegador que estemos utilizando para poder trabajar en nuestros desarrollos pueden quedarnos algo cortos al momento de detectar errores. Para eso se desarrolló **JSHint**. Este permite detectar errores y potenciales problemas en el código de JavaScript, y envía una lista de lo que está sucediendo. Lo único que hay que hacer es ingresar al sitio web **jshint.com** y, a partir de aquí, agregar el código que necesita revisión.

Captura de pantalla de un celular

Descripción generada automáticamente

***6.6 Librerías de JavaScript***

En la actualidad, todos nuestros desarrollos hacen uso generalmente de una o varias librerías externas. Este tipo de librerías que nos van a facilitar muchísimo el trabajo y el desarrollo de nuestra aplicación. No podemos estar eligiendo cualquier tipo de librería que encontremos en internet. Debemos garantizar que la librería que estemos utilizando tenga soporte y esté siendo actualizada regularmente. Por ejemplo, algunos tipos de librerías basadas en JavaScript que tienen soporte regular y que se están actualizando regularmente pueden ser **jquery** (jquery.com), **jquery mobile** (jquerymobile.com), **angularjs** (angularjs.org), **React**, **Bootstrap**, **Foundation** (foundation.zurb.com), **Codova** (cordova.apache.org), **PhoneGap** (móvil), **Ionic**(móvil)

***6.7 Testing con JavaScript***

Una de las mejores maneras para validar que el código funcione adecuadamente es realizarle pruebas a través de un framework llamado **mochajs** (mochajs.org), el cual corre a través de un servidor con **Node.js**, y podrás verificar los resultados en un navegador, haciendo este tipo de pruebas asíncronas bastante simples y sencillas.

Para descargarlo, es necesario tener previamente instalado **Node.js**.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*• JavaScript:*

***6.8 Despedida y conclusiones de Buenas prácticas con JavaScript***

***6.9 Test Capítulo 6:***

***A. ¿Cuál es el resultado del método Concat?***

- un error.

- un array.

- una expresión inválida.

- una cadena de texto.

***B. ¿Qué vamos a usar para medir el performance de toda línea que estamos ejecutando?***

- console.log().

- console.time().

- console.timeout().

- console.writeline().

***C. Una de las mejores maneras para validar que el código funcione adecuadamente es realizar pruebas a través del framework mochajs.***

- Verdadero.

- Falso.

***D. ¿Qué es necesario tener instalado previamente para poder ejecutar mochajs?***

- JQuery.

- Node.js

- C++.

- Python

***E. Las ventajas de minificar el código en JavaScript apuntan, en primera instancia, …………***

- a la compilación del código.

- al tiempo de respuesta de la aplicación.

- a evitar errores de código.

- al performance de la ejecución.

***G. Vamos a usar JSON cuando necesitemos ……….., a los que trata como ……..***

- mejorar la calidad del código / errores.

- intercambiar datos / objetos.

- disminuir el número de etiquetas / datos.

- depurar el código / scripts.

***H. ¿Qué hace JSHint?***

- Detectar errores y potenciales problemas en el código de JavaScript.

- Compilar el código para ejecutarlo en cualquier máquina física o virtual.

- Transpilar código, es decir, generarlo en un lenguaje a partir de otro diferente.

- Aumentar la velocidad de carga y visualización del código.

***I. Tenemos una lista formada por diferentes elementos y queremos concatenarlos todos. El método que usaremos será ……..***

- concat.

- link.

- join.

- assemble.

***J. ¿Qué tipo de librerías externas podemos utilizar en nuestros desarrollos?***

- Aquellas que garanticen un soporte y una actualización regular.

- Aquellas que tengan una relación directa con JavaScript.

- Cualquiera que encontremos en internet y que sea gratis.

- Las que ofrezcan una cantidad de contenidos.