Curso de Fundamentos de JavaScript

**Clase 1**

**Bienvenidos al Curso de Fundamentos de JavaScript**

A lo largo del curso aprenderemos las bases sólidas para dominar el lenguaje JavaScript. Veremos desde lo más básico como: variables, tipos de datos y operadores, e iremos avanzando con los condicionales y las estructuras de control. Luego veremos los conceptos relacionados con asincronismo, uno de los aspectos fundamentales de JavaScript, usando callbacks, promesas y async await. Y finalmente crearemos un juego de Simón dice en el que pondremos en práctica todo lo aprendido.

**Clase 2**

**Variables**

En este curso utilizaremos la consola de Google Chrome y el editor de texto Atom. También, crearemos una carpeta llamada fundamentos\_js.

Para crear un archivo independiente del html utiliza la etiqueta

**Clase 3**

**Variables: Strings**

Los strings son cadenas de texto. Para indicar que estamos usando una cadena de texto debemos de colocar las comillas simples.

En este curso utilizaremos los métodos:

* toUpperCase, que sirve para transformar un String a mayúsculas.
* toLowerCase, que sirve para transformar el string a minúsculas.

También usaremos el atributo:

* length, que nos indica la cantidad de caractéres que tiene un string.

Para concatenar dos strings se utiliza el símbolo (+)  
var nombreCompleto = nombre + ’ ’ + apellido

**Desafío:** Encuentra la última letra de su nombre

**Clase 4**

**Variables: Números**

En esta clase aprenderemos variables de tipo números y las operaciones más comunes en este tipo de variables.

En esta clase veremos operadores matemáticos como:

* suma ( + )
* resta ( - )
* multiplicación
* división

Recerda que en operaciones con decimales debemos realizar operaciones adicionales para conseguir un resultado preciso. Ej. var total = ( precioDeVino \* 100 \* 3) / 100

Para redondear una operación se utiliza la función: Math.round

**Clase 5**

**Funciones**

Las funciones son fracciones de código reutilizable. En esta clase aprenderemos a definir e invocar nuestras funciones. Para definir una función utilizaremos la palabra reservada ““function””.

Delimitamos el cuerpo de la función usando llaves { }. Los parámetros de la función son variables que se pasan a la función escribíendolos entre paréntesis ()

Definir funciones nos sirve para reutilizar código. JavaScript es un lenguaje interpretado, esto quiere decir que intentará ejecutar el código sin importar si los parámetros que le pasemos a la función estén invertidos o incluso incompletos.

**Clase 6**

**El alcance de las funciones**

En esta clase hablaremos del alcance que tienen las funciones. Si una variable no está definida dentro del cuerpo de una función hablamos de una variable global. Por el contrario, una variable definida dentro de una función es una variable local.

Para que la ejecución de una función no modifique una variable global usamos parámetros en lugar de pasar directamente la variable.

Es posible utilizar el mismo nombre para una variable global y para el parámetro de una función con un alcance local.

**Clase 7**

**Objetos**

Vamos a empezar a trabajar con objetos, veremos cómo declararlos, cuáles son sus ventajas, cómo asignarles atributos y cómo trabajar con ellos dentro de las funciones.

Los objetos se definen delimitados mediante llaves {}

Un atributo se compone de una clave (key) y un valor (value), que se separan entre sí por dos puntos “”:"". Los valores pueden ser de tipo string, número, booleano, etc. Cada atributo está separado del siguiente por una coma. Un objeto puede tener todos los atributos que sean necesarios.

Escribir el nombre de un objeto separado por un punto del nombre de un atributo, nos permite acceder al valor de dicho atributo para ese objeto. Un objeto también se puede pasar como atributo en una función.

Las últimas versiones de JavaScript nos permiten desglosar el objeto para acceder únicamente al atributo que nos interesa. Esto se consigue encerrando el nombre del atributo entre llaves { }.

**Clase 8**

**Desestructurar objetos**

En esta clase aprenderemos otra forma de acceder a los atributos de los objetos que es la desestructurización de los mismos.  
Para no duplicar las variables introducir el nombre de la variable como parámetro de la segunda variable. Ej var{nombre} = persona

**Clase 9**

**Parámetros como referencia o como valor**

Javascript se comporta de manera distinta cuando le pasamos un objeto como parámetro.

Cuando los objetos se pasan como una referencia, estos se modifican fuera de la función. Para solucionar esto se puede crear un objeto diferente. Esto lo podemos hacer colocando tres puntos antes del nombre. Ej …persona.

**Clase 10**

**Comparaciones en JavaScript**

Existen varias maneras de comparar variables u objetos dentro de javascript. En el primer ejemplo le asignamos a X un valor numérico y a Y un string. Para poder compararlos debemos agregar dos signos de igual (==). Esto los convierte al mismo tipo de valor y permite que se puedan comparar.

Cuando realizamos operaciones es recomendable usar tres símbolos de igual (===). Esto permite que JavasScript no iguale las variables que son de distinto tipo. Te recomendamos que uses el triple igual siempre que estés comparando variables.

Existen cinco tipos de datos que son primitivos:

1. Boolean
2. Null
3. Undefined
4. Number
5. String

**Clase 11**

**Condicionales**

En esta clase empezaremos a trabajar con estructuras de control, éstas nos permiten decidir el flujo de nuestro código.

Empezaremos con los condicionales. Los condicionales nos permiten decidir si un código se ejecuta o no. También introducimos un nuevo tipo de datos primitivos: el booleano, que determina si un valor es falso o verdadero. Mediante un condicional (if) decidiremos si se ejecuta una parte de nuestro código cuando se cumpla o no cierta condición.

**Clase 12**

**Funciones que retornan valores**

En esta clase seguiremos trabajando con condicionales para desglosar las funciones en funciones más pequeñas que retornen un valor.

Debemos de tener en cuenta que el número 18 esta incluido dentro del rango de edad. Para ello utilizamos los símbolos >=.

Return detiene la ejecución de una función y devuelve el valor de esa función.

Las variables definidas con const se comportan como las variables, excepto que no pueden ser reasignadas. Las constantes pueden ser declaradas en mayúsculas o minúsculas. Pero por convención, para distinguirlas del resto de variables, se escribe todo en mayusculas.

**Clase 13**

**Arrow functions**

En esta clase aprenderemos a utilizar Arrow Functions que permiten una nomenclatura más corta para escribir expresiones de funciones. Este tipo de funciones deben definirse antes de ser utilizadas.

Al escribir las Arrow Functions no es necesario escribir la palabra function, la palabra return, ni las llaves.

**Clase 14**

**Estructuras repetitivas: for**

En esta clase estudiaremos una de las estructuras básicas de control. El bucle for, se utiliza para repetir una o más instrucciones un determinado número de veces.

Para escribir un bucle for se coloca la palabra for seguida de paréntesis y llaves.  
Ej. for( ){ }. Dentro de los paréntesis irán las condiciones para ejecutar el bucle, y dentro las llaves irán las instrucciones que se deben repetir.

En este ejemplo la variable i la utilizamos como contador.

**Clase 15**

**Estructuras repetitivas: while**

En esta clase estudiaremos otra estructura repetitiva llamada while. While se ejecuta únicamente mientras la condición que se está evaluando es verdadera.

En ocasiones nuestro código puede fallar por errores de syntaxis o errores lógicos. En caso de que quieras verificar tu código, debes utilizar un debugger. El código se detiene cada vez que lee esta palabra.

**Clase 16**

**Estructuras repetitivas: do-while**

Otra estructura repetitiva es el do-while. A diferencia de la instrucción while, un bucle do…while se ejecuta una vez antes de que se evalúe la expresión condicional.

**Clase 17**

**Condicional múltiple: switch**

Una ultima estructura de control se llama switch. Switch se utiliza para realizar diferentes acciones basadas en múltiples condiciones.

Prompt, muestra un cuadro de mensaje que le pide al usuario que ingrese algúna información.

Break, sirve para que el browser se salte un bucle.

**Clase 18**

**Introducción a arrays**

Los arrays son estructuras que nos permiten organizar elementos dentro de una collección. Estos elementos pueden ser números, strings, booleanos, objetos, etc.

**Clase 19**

**Filtrar un array**

En esta clase aprenderemos como filtrar los arrays. Para filtrar siempre necesitamos establecer una condición. En este ejemplo nuestra condición es que la estatura de las personas sea mayor de 1.80mts.

El método filter ( ) crea una nueva matriz con todos los elementos que pasan la prueba implementada por la función proporcionada.

Recuerda que si no hay elementos que pasen la prueba, filter devuelve un array vacío.

**Clase 20**

**Transformar un array**

En esta clase veremos cómo transformar un array. El método map() itera sobre los elementos de un array en el orden de inserción y devuelve array nuevo con los elementos modificados.

**Clase 21**

**Reducir un array a un valor**

El método reduce() nos permite reducir, mediante una función que se aplica a cada uno de los elemento del array, todos los elementos de dicho array, a un valor único.

**Clase 22**

**Cómo funcionan las clases en JavaScript**

Las clases son funciones cuya sintaxis tiene dos componentes:

* expresiones
* declaraciones

En esta clase veremos el uso de this. Dentro de una función, el valor de this depende de cómo es llamada ésta.

**Clase 23**

**Modificando un prototipo**

En esta clase veremos cómo se modifican las clases de herencias. JavaScript funciona con una estructura orientada a objetos y cada objeto tiene una propiedad privada que mantiene un enlace a otro objeto llamado prototipo.

**Clase 24**

**El contexto de las funciones: quién es this**

En esta clase explicamos por qué al introducir el arrow function salió un error. El error del contexto de this en javascrip es uno de los errores más comunes.

Recuerda que dentro de la arrow function, this está haciendo referencia al espacio global, a windows.

**Clase 25**

**La verdad oculta sobre las clases en JavaScript**

Los objetos en JavaScript son “contenedores” dinámicos de propiedades. Estos objetos poseen un enlace a un objeto prototipo. Cuando intentamos acceder a la propiedad de un objeto, la propiedad no sólo se busca en el propio objeto sino también en el prototipo del objeto, en el prototipo del prototipo, y así sucesivamente hasta que se encuentre una propiedad que coincida con el nombre o se alcance el final de la cadena de prototipos.

**Clase 26**

**Clases en JavaScript**

Las clases de JavaScript son introducidas en el ECMAScript 2015 y son una mejora en la sintaxis sobre la herencia basada en prototipos de JavaScript.

La palabra clave extends se usa en declaraciones de clase o expresiones de clase para crear una clase que es hija de otra clase.

El método constructor es un método especial para crear e inicializar un objeto creado a partir de una clase.

**Clase 27**

**Funciones como parámetros**

En JavaScript, los parámetros de funciones son por defecto undefined. De todos modos, en algunas situaciones puede ser útil colocar un valor por defecto diferente que lo evalúe como verdadero.

**Clase 28**

**Cómo funciona el asincronismo en JavaScript**

JavaScript sólo puede hacer una cosa a la vez, sin embargo; es capaz de delegar la ejecución de ciertas funciones a otros procesos. Este modelo de concurrencia se llama EventLoop.

JavaScript delega en el navegador ciertas tareas y les asocia funciones que deberán ser ejecutadas al ser completadas. Estas funciones se llaman callbacks, y una vez que el navegador ha regresado con la respuesta, el callback asociado pasa a la cola de tareas para ser ejecutado una vez que JavaScript haya terminado todas las instrucciones que están en la pila de ejecución.

Si se acumulan funciones en la cola de tareas y JavaScript se encuentra ejecutando procesos muy pesados, el EventLoop quedará bloqueado y esas funciones pudieran tardar demasiado en ejecutarse.

**Clase 29**

**Cómo funciona el tiempo en JavaScript**

En principio, cualquier tarea que se haya delegado al navegador a través de un callback, deberá esperar hasta que todas las instrucciones del programa principal se hayan ejecutado. Por esta razón el tiempo de espera definido en funciones como setTimeout, no garantizan que el callback se ejecute en ese tiempo exactamente, sino en cualquier momento a partir de allí, sólo cuando la cola de tareas se haya vaciado.

**Clase 30**

**Callbacks**

En esta clase aprenderemos que son los callbacks y usaremos una librería externa que se llama jQuery.

Un callback es una función que se pasa a otra función como un argumento. Esta función se invoca, después, dentro de la función externa para completar alguna acción.

**Clase 31**

**Haciendo múltiples requests**

En esta clase accederemos a múltiples datos al mismo tiempo. Continuaremos trabajando con los jQuery y swapi.

**Clase 32**

**Manejando el Orden y el Asincronismo en JavaScript**

Una manera de asegurar que se respete la secuencia en que hemos realizado múltiples tareas es utilizando callbacks, con lo que se ejecutará luego, en cada llamada. Lo importante es que el llamado al callback se haga a través de una función anónima. Sin embargo, al hacerlo de esta manera generamos una situación poco deseada llamada CallbackHell.

**Clase 33**

**Manejo de errores con callbacks**

Para solucionar el problema de quedarnos sin conexión, u otro error similar, en medio de una sucesión de callbacks utilizamos el método fail().

**Clase 34**

**Promesas**

En esta clase veremos las promesas, que son valores que aun no conocemos. Las promesas tienen tres estados:

* pending
* fullfilled
* rejected

Las promesas se invocan de la siguiente forma:

**new** Promise( ( resolve, reject ) => {

// --- llamado asíncrono

**if**( todoOK ) {

// -- se ejecutó el llamado exitosamente

resolve()

} **else** {

// -- hubo un error en el llamado

reject()

}

} )

**Clase 35**

**Promesas Encadenadas**

A diferencia de los callbacks en el CallbackHell, que terminan estando anidados unos dentro de otros, cuando se usan Promesas la ejecución de las llamadas no se hacen de manera anidada sino de manera encadenada, al mismo nivel una debajo de la otra, lo que hace que el código sea mucho más legible y mantenible.

**Clase 36**

**Múltiples promesas en paralelo**

Para hacer el llamado a múltiples promesas, nos apoyamos en un array de ids con el que luego construimos otro arreglo de Promesas, que pasaremos como parámetro a Promise.all( arregloDePromesas ), con las promesas podemos encadenar llamadas en paralelo, algo que no es posible usando callbacks.

**Clase 37**

**Async-await: lo último en asincronismo**

Async-await es la manera más simple y clara de realizar tareas asíncronas. Await detiene la ejecución del programa hasta que todas las promesas sean resueltas. Para poder utilizar esta forma, hay que colocar async antes de la definición de la función, y encerrar el llamado a Promises.all() dentro de un bloque try … catch.

**Clase 38**

**Comenzando el juego**

Crearemos el juego Simon (Simón dice), en el que se van iluminando una secuencia de botones que el jugador tendrá que ir repitiendo, si se equivoca volverá a comenzar. El juego tendrá 10 niveles de dificultad, que deberán ser superados para ganar.

**Clase 39**

**Generando una secuencia de números**

Para generar la secuencia del juego usaremos un array con números aleatorios, que representarán el color del botón que se iluminará cada vez. Usamos new Array() para crear el arreglo de manera dinámica, y llamamos al método fill para rellenar ese array con ceros y poder luego iterar sobre éste con map()

**Clase 40**