# Curso de JavaScript Moderno – Guía definitiva.

## Conceptos - Fundamentos:

1. ECMAScript:

* Es la especificación para crear un ***lenguaje de scripting***, un lenguaje de scripting es interpretado en tiempo de ejecución por otro programa en lugar de ser procesado por el procesador de la computadora. Un **Script** requiere de un interprete mientras que un programa de un compilador.
* Los navegadores utilizan ECMAScript para interpretar el código JavaScript
  1. Versiones de ECMAScript:
* ES6: Nuevo estándar.
* ES7: Nuevas funciones.
* ES8: Nuevas funciones.
* ES. Next: Versión de desarrollo.

1. JavaScript

* Lenguaje de Scripting.
* JavaScript es el lenguaje de programación bajo el estándar de ES.
* ***Sección 2: Fundamentos de JavaScript***
* Al nombrar variables:
  + primerNombre => **camelcase**
  + primer\_nombre => **underscore**
  + PrimerNombre => **pascal case**
* Diferencia entre variables:

|  |  |  |
| --- | --- | --- |
| **VAR** | **LET** | **CONST** |
| El valor puede ser reescrito, y puede ser declarado dos veces. | El valor puede ser reescrito, pero no puede ser declaro dos veces. | Las variables const siempre deben tener un valor de inicio, y no se puede reescribir. |

* Métodos de las cadenas de texto:
  + indexOf(‘Palabra‘) => Trae la posición donde se encuentra la palabra.
  + replace(‘Palabra a reemplzar’, ‘Nueva palabra’)
  + includes(‘Palabra’) => Retorna true sí existe la Palabra o false si no.
* Tipos de datos:
  + No hay que declarar el tipo de datos qué contiene la variable, solo estructurarlo.
  + Para identificar el tipo de dato => **typeof**
* Operadores de comparación:

|  |  |
| --- | --- |
| Mayor qué | > |
| Menor qué | < |
| Igual | == |
| Igual estricto | === |
| Diferente | !== |

* Template String – Template Literals.
* Objetos en JavaScript:
  + **{ }**
  + {

nombre: “Cristian”,

edad: 18

}

* + Agregar un objeto dentro de un objeto.

{

nombre: “Cristian”,

edad: 18

ubicación: {

país: “Colombia”

ciudad: “Bogotá”

codigoPostal: 110111

dirección: “Calle 27ª Sur # 22-05}”

}

}

* + Agregar funciones.

{

nombre: “Cristian”,

edad: 18

ubicación: {

país: “Colombia”

ciudad: “Bogotá”

codigoPostal: 110111

dirección: “Calle 27ª Sur # 22-05}”

},

nacimiento: function() {

return new Date().getFullYear() – this.edad;

}

}

* Arreglos y sus métodos:
  + **[ ]**
  + Arreglos de números
  + Arreglos de cadenas de texto
  + Arreglos con los tipos de datos de JavaScript

Comprobar si es un arreglo:

* (Array.isArray(variable)) => Devuelve true si lo qué contiene la variable es un arreglo, o false si no lo es.
* variable**. push(‘**dato**’)** => Agrega el dato al final de arreglo.
* variable. **unshift(‘**dato**’)** => Agrega el dato al inicio del arreglo.
* variable.**pop ()** => Elimina el dato final del arreglo.
* variable.**shift ()** => Elimina el dato del inicio del arreglo.
* **concat** => Unir arreglos.
* Arreglos de objetos:
  + **[ {..}, {..}, {..} ]**
  + Se puede recorrer con un ciclo **for:**
    - **for**(let index = 0; i < arreglo.length; i++){ acción }
  + Si se accede al valor de forma individual se puede modificar.
* Window Object => Nuestra ventana de navegación.
* Scope => Visibilidad que tiene un valor en JavaScript.
  + Globales: No están dentro de una función, o dentro de un bloque de código.
  + Local: Cuando está dentro de una función su scope es local.
  + Bloque: Cuando se declara con **var** dentó de un par de llaves se reescriben los valores globales.
* ***Sección 3: DOM, Window, Scripting***
* DOM – Document Object Model
  + Document => root (html) => head – body
  + head => title => Texto
  + body => contenido de la página web
* DOM y Scripting
  + Traversing
  + CreateElement
  + DeleteElement
  + RemplaceElement
* ***Sección 4: Event Listeners***

Esperar que el usuario haga una acción para correr el código.

* Event Listener Click
* ***Sección 9: Objetos, Constructores y This***
* ***Sección 13: AJAX y HTMLRequest***
* API’s, REST y Request
  + API’s

API = Application Programming Interface.

Funciones, métodos que ofrece una librería para ser utilizada por otro software como una capa de abstracción.

Para acceder al servicio hay que enviar una petición estructurada.

* + REST API’s

REST = Representational State Transfer

Puede ser diseñada en cualquier lenguaje

REST Describe como deben ponerse a disposición lo recursos

La API debe responder a los Request de HTPP: GET, POST, PUT, PATCH, DELETE.

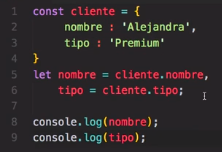
Cada API tiene sus propias reglas, métodos, estructuras.

* + REST API EndPoints y Request

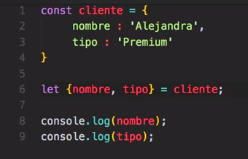
Una REST API cuenta con Endpoints para hacer operaciones CRUD

* ***Sección 15: JavaScript Asíncrono o Async JS***
* En el código asíncrono podemos tener 2 líneas de código y la línea 2 ejecutarse y finalizar antes que la línea uno termine de hacerlo.
* El código asíncrono se utiliza la mayor parte cuando se interactúa con una API o REST API.
* Para crear código asíncrono tenemos las siguientes funciones:
  + Callbacks
  + Promises
  + Async/Await
* ***Sección 16: Fetch API***
* ***Sección 17: Arrow Functions***
* = () => {}
* ***Sección 18: Async Await***
* Async – La función Asíncrona siempre requiere una promesa
* ***Sección 21: ES6***
  + ***Asignación por Destructuring***

Forma de interactuar más sencillo con las variables

Antes:

Después:



* ***Sección 23: Módulos Nativos en JavaScript***
* ¿Qué son módulos?

No permite agrupar la funcionalidad, conforme a nuestro programa crezca no permite darle soporte.

* Beneficios

Dividir el código en pequeños archivos

Un módulo puede ser utilizado fácilmente en otra aplicación

Dependencias:

Un modulo y puede estar conectado con otro módulo.

Evitan el conflicto con el nombre de las variables:

Si tenemos dos módulos con la función cliente, estos no generan un conflicto.

* Módulos y JavaScript

WebPack es la mejor opción para crear módulos

* ***Sección 25: Design Patterns o Patrones de Diseño en JavaScript***
* ¿Qué son?
  + Soluciones de código reutilizables que se utilizan en problemas comunes en aplicaciones de software
  + Se emplean patrones de diseño para el código.
* Categorías
  + Patrones de diseño de:
    - Creación de objetos
    - De estructura
    - De comportamiento
    - De arquitectura
* Constructor de patrones – Abstracción

* ***Sección 26: Higher Order Functions y Array Methods en JavaScript***
  + ***ForEach***
  + ***Map***
  + ***Filter***
  + ***Reduce***
  + ***Some***
  + ***Find***
* ***Sección 28: Client Side Storage – IndexedDB***
  + Storage en el cliente

Guardar datos en el navegador con JavaScript

No usar Cookies

Se recomienda almacenar pocos datos, nada que sea muy importante o crítico

* Opciones

Web Storage y IndexedDB

* Local y Session Storage
  + Ambos pueden guardar Strings o Cadenas de texto

Local Storage:

* + No se recomienda guardar información sensible
  + Es ideal para Productos temporales de un carrito

Session Storage:

* Recomendado para información sensible
* Información temporal

IndexedDB:

* Almacenar grandes cantidades de datos estructurados, incluyendo archivos
* Utiliza indexes para ser más rápido el acceso a los datos
* Recomendado para grandes cantidades de datos

Características:

* Es una base de datos completa
* Puedes guardar Arreglos, Objetos, Imágenes y Videos
* Datos estructurados con una llave
* Datos con llave -> valor.
* Asíncrono

Desventajas:

* No se puede sincronizar con el backend
* No se puede utilizar el modo privado
* El limite de datos los pone el navegador
* Si el usuario limpia el browser los datos se van a perder
* ***Sección 29: Introducción a webpack***
* ¿Qué es?
  + Crear paquetes de módulos para aplicaciones JavaScript
  + WebPack procesa la aplicación y mapea todas las dependencias de un módulo, con esto crea uno o varios paquetes o bundlers
* Conceptos
  + Entry
    - Especifica que modulo debe utilizar webpack para comenzar a crear la gráfica de dependencias, webpack se encargará de saber que dependencias tiene este módulo.
  + Output
    - En que parte se debe almacenar el paquete que se a creado
    - Normalmente la entrada esta en una carpeta llamada src/ y la salida a una carpeta llamada dist/
  + Loaders
    - WebPack solo entiende archivos JS pero son los loaders se pueden cargar otros formatos
    - **Test:** Que archivos deben ser transformados.
    - **Use:** Que loader estamos usando para este archivo.
  + Plugins
    - Optimizan los paquetes o bundle, administración avanzada de los assets
    - Un plugin se instala vía npm
* Crear un paquete
* node\_modules\.bin\webpack src\index.js