**M1S5: Fundamentos de JavaScript**

**Hola, te encuentras en la 5 semana, ¿cómo vas con tu primer proyecto?, ¿ya lo entregaste? recuerda que debes entregarlos todos para poder aprobar el Bootcamp, hasta ahora aprendiste y aplicaste HTML y CSS. A continuación, daremos inicio a un nuevo contenido y es el que corresponde a JavaScript.**

**Sigue adelante... 😄**

**ÍNDICE**

* [JavaScript](https://github.com/U-Camp/BOOT-M1-SEM5#javascript)
* [Sintaxis y Sentencias](https://github.com/U-Camp/BOOT-M1-SEM5#sintaxis-y-sentencias)
  + [Variables y Constantes](https://github.com/U-Camp/BOOT-M1-SEM5#variables-y-constantes)
  + [Operadores Aritméticos, de Comparación y Lógicos](https://github.com/U-Camp/BOOT-M1-SEM5#operadores-aritm%C3%A9ticos-de-comparaci%C3%B3n-y-l%C3%B3gicos)
  + [Tipos de datos](https://github.com/U-Camp/BOOT-M1-SEM5#tipos-de-datos)
  + [Estructuras de Control](https://github.com/U-Camp/BOOT-M1-SEM5#estructuras-de-control)
* [Funciones](https://github.com/U-Camp/BOOT-M1-SEM5#funciones)

**JavaScript**

JavaScript es un lenguaje script multi-paradigma, basado en prototipos, dinámico, soporta estilos de programación funcional, orientada a objetos e imperativa. Nacido como una alternativa a los lenguajes de programación web estáticos (HTML) y del lado del servidor (PHP, Ruby, Python) de los años 90, JavaScript, es un lenguaje de programación ligero, interpretado, basado en prototipos y orientado a objetos; normalmente confundido con Java pero con características completamente alejadas de éste, es uno de los lenguajes favoritos y más usados en el mundo del desarrollo de software, su uso principal es como lenguaje de script para páginas web, ya que anteriormente se ejecutaba exclusivamente del lado del cliente (exploradores web), pero actualmente tiene implementaciones en muchos entornos sin navegador web, tal como NodeJS.

Antes de que existiera NodeJS, con JavaScript sólo se podían construir aplicaciones web, las cuales se ejecutan en un navegador web haciendo uso de un JavaScript Engine que interpreta el código en tiempo de compilación, convirtiéndolo en un bytecode entendible para las máquinas.

Además de JavaScript Engine, los navegadores poseen un Runtime Environment el cual otorga acceso a APIs nativas de los navegadores mediante las cuales un programador puede construir una aplicación web, ya que este entorno de ejecución provee objetos tales como el DOM, WINDOW, DOCUMENT, LOCATION o el objeto XMLHttpRequest (AJAX) con el cual se pueden realizar llamadas al servidor asíncronas. Gracias a la conjunción de esos dos elementos (JavaScript Engine y Runtime Environment) es que se puede ejecutar JavaScript.

El código de JavaScript se ejecuta asíncronamente por defecto, en un único proceso (single thread), lo que permite que las operaciones devuelvan el control del programa antes de que terminen mientras se siguen operando en segundo plano, esto agiliza el proceso de ejecución, pero complica el razonamiento sobre el programa. Sin embargo, la arquitectura de JavaScript permite este tipo de ejecución en favor de la rapidez, no importando el entorno de ejecución, el diseño de esa arquitectura se describe a continuación:

**o Call Stack:** Es una estructura de datos, una pila de llamadas (LIFO) donde se almacenan el seguimiento de un script que ejecuta secuencia de instrucciones.

**o Event Queue:** Es una cola que se adjunta a cada programa de JavaScript, la cual es la lista de mensajes pendientes por ser procesados.

**o Worker Threads:** Son procesos esclavos que, mediante el Runtime Environment, nos permiten ejecutar librerías fuera o concurrente fuera del proceso principal de JavaScript.

**o Event Loop:** El bucle de eventos es el encargado de orquestar toda la arquitectura; es el encargado de monitorizar si se ha producido algún nuevo evento y ejecutar el callback correspondiente.

Además de las ventajas inherentes a esta arquitectura, JavaScript es un lenguaje de programación sencillo y ligero, por lo que su curva de aprendizaje es poco pronunciada.

A continuación, se muestra un ejemplo de JavaScript, el cual accede al DOM para obtener todos los párrafos definidos en un documento HTML y manda un mensaje con el total de párrafos dentro del documento:

Ejemplo:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Ejemplo HTML y JavaScript</title>

</head>

<body>

<h1>

Encabezado 1.

</h1>

<hr />

<h2>

Encabezado 2.

</h2>

<p>

Párrafo para encabezado 2.

</p>

<hr />

<h5>

Encabezado 5.

</h5>

<p>

Párrafo para encabezado 5.

<br />Con salto de línea

</p>

<!--

La etiqueta `script` permite ejecutar código

JavaScript directamente en el navegador

-->

<script>

// Por medio de un selector `('p')`, JavaScript accede al DOM,

// el cual regresa un arreglo de elementos que cumplen con el selector

// en este caso el conjunto de 2 párrafos que existen en el DOM.

alert(document.querySelectorAll('p').length);

</script>

</body>

</html>

*Ejemplo de HTML con código Javascript*

**Si deseas puedes consultar y descargar esta infografía que muestra de forma más ilustrada lo que te acabo de explicar sobre JavaScript,**[**presiona aquí**](https://github.com/U-Camp/BOOT-M1-SEM5/blob/main/infografias/Semana_1_Info1_Javascript.pdf)

**Seguimos...**

Encuestas realizadas por Stack Overflow en el 2019 ("Stack Overflow Developer Survey 2019 ", s.f.) indican que, entre los programadores de todo el mundo, **JavaScript** es el lenguaje más popular y usado, con un porcentaje del 67.8 %. Al ser un lenguaje con alta aceptación, este puede ser utilizado por profesionales y por aquellos que apenas se inician en el mundo de la programación, la tendencia de uso indica que este lenguaje goza de una alta aceptación en la comunidad, por lo que la curva de aprendizaje es mínima, así como el soporte y los recursos de apoyo que existen son extensos, razones de sobra para elegirlo como el lenguaje de programación tanto para el frontend (funcionalidad visible para el usuario final) como para el backend (funcionalidad de lógica del negocio) y con el cual se pretende impartir el presente bootcamp.

El lenguaje JavaScript tiene distintas palabras reservadas con las que se puede codificar y hacer aplicaciones web enriquecidas, permite el uso de variables, constantes, operaciones aritméticas y lógicas, así como comparaciones y estructuras de control de flujo, las cuales se verán a continuación.

**Sintaxis y Sentencias**

**Variables y Constantes**

Las variables son contenedores de datos o de un valor en específico que sirven para realizar operaciones o evaluar expresiones, como una suma o una parte de una oración. El valor de una variable puede cambiar, anteriormente se definían con la palabra reservada var pero debido a como fue diseñada en el lenguaje, ya que permite la re-declaración de la variable o el uso de variables fuera de ámbito, lo que ocasionaba errores de ejecución y falta de fiabilidad en el código, se adoptó el uso de la palabra reservada let para especificar variables y la palabra reservada const para especificar constantes, las cuales no cambian de valor, seguidas del nombre de la variable o constante (que pueden contener letras, dígitos y guiones bajos, preferiblemente usando camelCase) hasta este punto se le llama declaración; después de la declaración de la variable o constante, se puede asignar un valor, para lo cual se usa el signo igual (=) y el valor a asignar (ejemplo let variable = 123; o const constante = 'cadena constante';). Dentro de esto existen dos conceptos que hay que tener en cuenta:

**o Scope:** El scope es el ámbito de una variable o constante, es hasta donde el lenguaje puede leer que ésta se encuentra declarada, es decir que no existe más allá del bloque de código en el que se le declara, por lo que, si se accede fuera de ese bloque de código, el lenguaje lanzará un error.

**o Hoisting:** El hoisting es la elevación de variables, lo que quiere decir que para evitar posibles errores y re-declaraciones de variables, hay que definir las variables y las constantes al inicio de cada bloque de código.

**Operadores Aritméticos, de Comparación y Lógicos.**

**JavaScript** admite varios tipos de operadores con los cuales se evalúan expresiones, como lo son los aritméticos, los de comparación o los lógicos. Los operadores aritméticos realizan operaciones algebraicas sobre las variables o constantes, con lo que al final se devuelve un resultado (ejemplo const suma = a + b;), los más usados son la adición (+), la resta (-), la multiplicación (\*), la división (/), residuo (%), incremento (++) y decremento (--).

Los operadores de comparación evalúan 1 o más expresiones y devuelven un resultado booleano indicando el resultado de esa comparación (ejemplo const esMayor = a > b; si a es mayor a b, la constante esMayor será igual a true, si b es mayor que a, entonces esMayor será igual a false); los más usados son la igualdad (== y ===, el triple igual es más estricto en cuanto a que compara el tipo de dato), desigualdad (!= y !==) mayor que (>), menor que (<), mayor o igual que (>=), menor o igual que (<=).

En cuanto a los operadores lógicos, estos se aplican también sobre variables o constantes, y también devuelven un resultado booleano que puede ser usado en comparaciones de datos, normalmente son usados para comparar dos o más expresiones de comparación (ejemplo const resultadoBooleano = a > b && b < c;); los más usados son comparación AND (&&), comparación OR (||) y negación NOT (!).

Existen otro tipo de operadores que se usan naturalmente como parte del lenguaje, es decir, que se comprenden por sí mismos como lo son los operadores de agrupación (()), el operador de asignación (=) o el operador ternario (?); hay otros que casi no se usan como los operadores a nivel bit (>>, <<) o el de exponenciación (\*\*).

Todo tipo de expresión ya sea aritmética, comparativa o lógica, usa la precedencia de operadores para ser evaluada.

**Tipos de Datos**

Un valor en JavaScript siempre está relacionado a algún tipo de dato, por ejemplo, una cadena de texto o un número.

Aunque JavaScript no es un lenguaje altamente tipado, todas las variables y constantes están asociados a algún tipo de dato específico.

Los tipos de datos principales son:

* Number, cualquier tipo de dato numérico (ejemplo const numero = 1;);
* String, cualquier tipo de dato de texto, el cual debe estar rodeado por comillas simples o dobles (ejemplo const cadena = 'Una cadena';);
* Boolean, que es normalmente el resultado de una comparación y que a nivel de cómputo significa un uno o un cero (ejemplo const verdadero = true; o const comparacion = 4 > 1;);
* Object es un tipo de dato compuesto, los anteriores son tipos de datos primitivos ya que almacenan una solo cosa, sin embargo el objeto puede contener múltiples valores de los cuales está compuesto (ejemplo const obj = { propiedadNumerica: 1, propiedadCadena: 'Una cadena' };);
* Undefined y null son tipos de datos especiales que definen el estado de una variable o constante en particular, por ejemplo si se declara una variable sin un valor específico, éste será igual a null (ejemplo const sinValor;) así mismo si se accede a una variable no declarada ni asignada en ninguna parte del código, está será undefined.

**Estructuras de Control**

Dentro de JavaScript existen estructuras que permiten el control del flujo que llevan los datos, los cuales continúan la ejecución de una manera no lineal, evaluando expresiones que permiten realizar una u otra acción dependiendo del resultado de la evaluación, así mismo el flujo puede ser orientado a que se ejecute una sola vez con el uso de estructuras condicionales o a que se itere una y otra vez hasta que se cumpla una condición específica con el uso de estructuras cíclicas.

**Condicionales**: Se usan las palabras reservadas if, else, switch, break y default para realizar comparaciones y ejecutar una acción si se cumplen esas condiciones.

const edad = 18;

if(edad >= 18) {

// Esta condición se cumple, por lo que saldrá el alert.

alert("Eres mayor de edad");

}

else {

alert("Todavía eres menor de edad");

}

switch (edad) {

case 18:

// Esta condición se cumple, por lo que saldrá el alert.

alert('Tienes 18 años');

break;

case 20:

alert('Tienes 20 años');

break;

default:

alert('Sigues siendo joven');

}

*Ejemplo de estructuras de control condicionales*

**Cíclicos:** Se usan las palabras reservadas for, while y do para realizar iteraciones mientras se cumpla una condición. const edad = 10;

// Imprimirá en consola el valor de `i` 10 veces.

for (let i = 0; i < edad; i += 1) {

console.log(i);

}

// Imprimirá en consola el valor de `cont` 10 veces.

let cont = 0;

while (cont < edad) {

cont++;

console.log(cont);

}

// Imprimirá en consola el valor de `inc` 10 veces.

let inc = 0;

do {

inc++;

console.log(inc);

} while (inc < edad);

*Ejemplo de estructuras de control cíclicas*

**Hola, ¿cómo te sientes?, ¿te parece si tomamos un descanso y nos dedicamos 5 minutos a respirar, escuchando una canción relajante?, puedes cerrar los ojos y escuchar la siguiente música [presiona aquí](https://www.youtube.com/watch?v=Pd3TcScm6UU)**

**¿Cómo te sientes?, espero que mucho mejor y ahora sí continuemos...**

**Funciones**

Las funciones en JavaScript, son bloques de código que realizan una acción específica, es decir, que se encargan de realizar cierto conjunto de sentencias relacionadas entre sí. Esto sirve para poder dividir el código en pequeños pedazos que hagan cosas específicas, lo cual permite reusar ese código en distintas partes del proceso, lo que evita que reescribamos. Por lo que, si se detecta que se están repitiendo líneas de código en varios lados, es mejor invertir tiempo reescribiendo esas líneas en forma de función e invocando la función en cada parte en la que se le necesite (refactorizar).

Una función se define con la palabra reservada function seguida del nombre de la función (que pueden contener letras, dígitos y guiones bajos, preferiblemente usando camelCase), seguido por paréntesis.

Los paréntesis se usan para recibir parámetros dentro de la función, estos se separan por comas y se usan como variables / constantes dentro de la función.

El código que se encuentra dentro de una función es ejecutado cuando esta función se invoca, esto puede ocurrir en 3 posibles escenarios: Cuando la función es invocada desde un evento (onclick cuando el usuario hace clic en un botón, por ejemplo), cuando la función es invocada desde otra parte del código o cuando se invoca a sí misma (recursión).

Así mismo, una función puede regresar un valor o un resultado del proceso que realizó, en este caso se usa la palabra reservada return la cual detiene la ejecución de la función hasta el punto en la que ésta se encuentre y regresa cualquier valor computado dentro de la función o incluso puede no regresar nada.

function nombreFuncion(parametro1, parametro2, parametro3) {

// Las variables y constantes locales

// no son accesibles fuera de la función

let variableLocal = 'Variable local';

console.log('Parámetro 1: ', parametro1);

console.log('Parámetro 2: ', parametro2);

console.log('Parámetro 3: ', parametro3);

variableLocal = 'Valor de variable cambiado';

// Se regresa el valor procesado en la función

// En este caso la suma del `parámetro2` que es numérico

// Más el atributo `valor` del objeto `parámetro3`.

const constanteLocal = parametro2 + parametro3.valor;

return constanteLocal;

}

// Aquí se re-usa la función 3 veces,

// evitando escribir el mismo código esas mismas 3 veces

// Aun cuando regrese un valor

// se puede mandar llamar directamente la función

nombreFuncion('Los argumentos', 1, { tipo: 'Objeto', valor: 4 });

// Se puede llamar y almacenar el valor regresado en una constante

const suma = nombreFuncion('Pueden ser', 2, { tipo: 'Objeto', valor: 5 });

console.log('Suma: ', suma);

// O se puede llamar, almacenar el valor

// en una variable y hacer algo con la misma

let res = nombreFuncion('De cualquier tipo', 3, { tipo: 'Objeto', valor: 6 });

const resta = res - 2;

console.log('Resta: ', resta);

*Ejemplo de Funciones*

**Si deseas puedes consultar y descargar esta infografía que muestra de forma más ilustrada lo que te acabo de explicar sobre JavaScript (SINTAXIS Y SENTENCIAS),**[**presiona aquí**](https://github.com/U-Camp/BOOT-M1-SEM5/blob/main/infografias/M1_S5_Infografia%2001.pdf)

**Sería todo el contenido de esta semana, recuerda que si quieres realizar prácticas puedes copiar y pegar los códigos anteriores en VSC o en CodePen e ir ejecutando cada ejemplo para que los puedas visualizar y si tienes alguna duda puedes consultar con tus coaches desde Microsoft Teams.**

**Nos vemos en la siguiente semana.**