**¿Cómo nace JavaScript?**

Nace como la necesidad de generar dinamismo en las páginas web y que a su vez los usuarios y las empresas pudieran interactuar unos con otros.

**¿Qué es JavaScript?**

Es un lenguaje interpretado, orientado a objetos y débilmente tipado y dinámico.

* Débilmente tipado: Se pueden hacer operaciones entre tipos distintos de datos (enteros con string, booleanos con enteros, etc.).
* Dinámico: Corre directamente en la etapa de RunTime sin una etapa de compilación previa. Esto permite probar el código inmediatamente, pero también es lo que hace que los errores se muestren mientras se ejecuta el programa.

**¿Realmente es JavaScript un lenguaje interpretado?**

Sí, y la razón es que el navegador lee línea por línea nuestro código el cual le indica lo que tiene que hacer, sin la necesidad de compilar. Todo esto es controlado por el motor de JavaScript 8 del navegador.

**JavaScript es Backwards Compatible**

Todas las funciones nuevas que salen de **JavaScript** no dañaran el trabajo ya hecho, pero no se podrá utilizar en nuestro entorno de trabajo inmediatamente. Para solucionar esto está \*\*Babel\*\* que permite utilizar las nuevas características del lenguaje pero lo transforma a una versión que el navegador puede entender.

**Nota**: Forward quiere decir que es compatible con versiones futuras, esto significa que incluir una adición al lenguaje en un programa no causaría que se rompa si se ejecuta en un motor JS anterior. JS no es compatible con versiones futuras.

**¿Por qué estudiar JavaScript?**

Por 2019 JavaScript es parte de los 3 lenguajes estándares para construir sitios web junto a HTML y CSS. HTML lo utilizamos para maquetar y mostrar cualquier información que queramos en una página, como texto, imágenes, videos, etc. CSS es un lenguaje de estilo que utilizamos para darle, valga la redundancia, el estilo a nuestra página. Y JavaScript era el único lenguaje de programación que se podía utilizar para hacer tu página dinámica o generar una aplicación web. Pero a finales del 2019 la W3C decidió y subió como nuevo lenguaje de programación \*\*WebAssembly\*\*, el cual es un lenguaje nuevo para construir productos web y con este ya no tendrás que usar HTML, CSS y JavaScript para poder hacer productos webs.

JavaScript tiene una comunidad enorme de desarrolladores que te pueden ayudar a generar diferentes cosas. Para trabajar en aplicaciones web puedes utilizar muchos frameworks y librerías construidas en JavaScript que te van a ayudar a hacer proyectos de forma mucho más rápida, eficiente y robusta. También se puede utilizar JavaScript para generar aplicaciones en dispositivos móviles a través de **React Native** así como también aplicaciones de escritorio con **Electron** y por último, también puedes ser desarrollador backend con nodejs o IOT (Internet of things).

**Elementos de un lenguaje de Programación: Variables, Funciones y Sintaxis.**

Hay 2 elementos que vamos a utilizar en un lenguaje de principal, estos son dos componentes principales:

1. Data que guardamos en memoria y estos datos se convierten en tipos de datos, números, string, objetos, boolean, etc.

2. Tares (Funciones) que haremos con esa data.

**Nota: typeof** es una palabra reservada en JavaScript para validar que tipo de data es el valor que quieres validar. Existen diferentes tipos de datos, los primitivos y no primitivos.

Primitivos:

* Números
* String
* Booleanos
* Vacios.

No Primitivos:

* Arrays
* Objetos

**Variables**

Una variable se puede entender como una representación en algún lugar en memoria que vamos a utilizar para guardar un valor.

* **var**: Es una palabra reservada que cuando el navegador lo lee detecta que es una variable y le reserva un espacio en la memoria para guardar un valor.

**Funciones**

Cuando hablamos de funciones en JavaScript, tenemos dos tipos de funciones: Funciones Declarativas (function declaration / function statement) y Expresiones de función (function expression / funciones anónimas).

* **Funciones Declarativas:** En las funciones declarativas, utilizamos la palabra reservada **function** al inicio para poder declarar la función:
* **Expresión de función**: En la expresión de función, la declaración se inicia con la palabra reservada **var**, donde se generará una variable que guardará una función anónima.

En la expresión de función, la función podría o no llevar nombre, aunque es más común que se hagan anónimas.

**Diferencias**

A las funciones declarativas se les aplica **hoisting**, y a la expresión de función, no. Ya que el **hoisting** solo se aplica en las palabras reservadas **var** y **function**. Lo que quiere decir que, con las funciones declarativas, podemos mandar llamar la función antes de que ésta sea declarada, y con la expresión de función, no, tendríamos que declararla primero, y después mandarla llamar.

**Scope**

Es el alcance de las variables que declaramos. Existe el scope global y local, el primero es una variable que tiene un alcance en todo nuestro programa este se genera una vez que nuestro archivo de JavaScript se inicializa en el navegador, y el segundo es una variable que tiene alcance limitado, es decir que solo existen dentro de la función que se declara. Todo lo que este declarado en el scope global puede ser consumido en uno local pero no al revés.

**Hoisting**

Es cuando las variables y las funciones se declaran antes de que se procese cualquier tipo de código. El hoisting solo pasa a conversiones pasadas de JavaScript, desde ECMAScript 5 hacia atrás. Desde ECMAScript 6 en adelante el hoisting ya no sucede, ya que se gatilla con dos palabras claves que serían variable y function. En la versión de ECMAScript 6 y superiores se presentaron dos nuevas variables, cuales son **const** y **let**.

**Coerción**

Significa que es la forma en la que podemos cambiar un tipo de valor a otro. Existe la **Coerción Implícita** que es cuando el lenguaje nos ayuda a cambiar de un tipo de valor a otro. Y también está la **Coerción Explicita** que es la forma en que nosotros obligamos a que un valor de un tipo cambie a otro.

**Truthy y Falsy**

**Valores FALSY:**

* 0 es FALSY/FALSO Boolean(0)
* null es FALSY/FALSO Boolean(null)
* NaN es FALSY/FALSO Boolean(NaN)
* undefined FALSY/FALSO Boolean(undefined)
* false FALSY/FALSO Boolean(false)
* "" FALSY/FALSO Boolean("")

**Valores THRUTHY:**

* Cualquier cadena no vacia THRUTHY Boolean("hola")
* " " (El espacio tambien cuenta como true) THRUTHY Boolean(" ")
* Números mayores a 0 THRUTHY Boolean(22)
* Un Arreglo THRUTHY Boolean([])
* Un Objeto THRUTHY Boolean({})
* Cualquier funcion THRUTHY Boolean(function(){})
* True THRUTHY Boolean(true)

**Operadores: Asignación, Comparación y Aritméticos**.

**ARITMETICOS**

* + // Suma a + b, afirmación unitaria +a : Positivo, concatenación "a" + "la" : "ala"
  + // Resta a - b, negación unitaria -a : Negativo
* // Producto a \* b
* / // Divisor a / b
* %// Residuo a % b
* \*\* // Potencia a \*\* b, a elevado a b
* ++ // Incremento (suma uno) a++ : a = a + 1
* -- // Decremento (resta uno) a-- : a = a - 1

**ASIGNACIÓN**

* = // Asignación a = b
* += // Asignación de adición a += b : a = a + b
* -= // Asignación de sustracción a -= b : a = a - b
* \*= // Asignación de multiplicación a \*= b : a = a \* b
* /= // Asignación de división a /= b : a = a / b
* %= // Asignación de residuo a %= b : a = a % b
* \*\*= // Asignación de potencia a \*\*= b : a = a \*\* b
* <<= // Asignación de desplazamiento a la izquierda a <<= b : a = a << b
* >>= // Asignación de desplazamiento a la derecha a >>= b : a = a >> b
* >>>= // Asignación sin signo de desplazamiento a la derecha a >>>= b : a = a >>> b
* &= // Asignación AND a &= b : a = a & b
* ^= // Asignación XOR a ^= b : a = a ^ b
* |= // Asignación OR a |= b : a = a | b

**COMPARACIÓN**

* == // Igualdad a == b, a tiene el mismo valor que b
* != // Distinto a != b, a tiene un valor diferente a b
* === // Identidad a === b igual valor igual tipo de dato
* !== // Sin Identidad a !== b igual valor o igual tipo de dato
* >// Mayor que a > b
* >= // Mayor o igual que a >= b
* < // Menor que a < b
* <= // Menor o igual que a <= b

**LOGICOS**

* && // AND, Y, a && b : a y b
* || // OR, O, a || b : a o b
* ! // NOT, Negación, a = true : !a = false

**LOGICOS A NIVEL DE BITS**

* << // Desplazamiento a la izquierda a << b
* >> // Desplazamiento a la derecha a >> b
* >>> // Desplazamiento a la derecha sin signo a >>> b
* & // AND a & b
* ^ // XOR a ^ b
* | // OR a | b
* ~ // NOT a | b```

**Arrays**

El array es una estructura de datos que también se conoce como una lista de datos que va a guardar ciertos valores. Pueden guardar valores del tipo numéricos, texto, objetos e incluso otros arrays.

**Loops o Ciclos**

El ciclo es una manera rápida y sencilla de hacer que una tarea pueda repetirse sin hacerse de manera manual hasta que una condición se cumpla. Los objetos se manejan con propiedades, que es una combinación de palabras claves y valores. También pueden estar conformados con métodos para realizar algún tipo de acción en específico.

**Objetos**

Un objeto en programación es una representación abstracta de un objeto en la vida real, sin embargo, también puede entenderse como un contenedor de datos.