JAVASCRIPT
(SINTAXIS Y SENTENCIAS)

Javascript

- Nacido como una alternativa a los lenguajes de programación web estáticos (HTML) y del lado del servidor (PHP, Ruby, Python) de los años 90.
- Es un lenguaje script multi-paradigma, basado en prototipos, dinámico, soporta estilos de programación funcional, orientada a objetos e imperativa.
- Es un lenguaje de programación ligero y es 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.
- 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 éste 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:

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.

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

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

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.

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.
- El valor de una variable puede cambiar, anteriormente se definían con la palabra reservada `var` pero debido a cómo fue diseñada en el lenguaje, permite la re-declaración de la variable o el uso de variables fuera de ámbito, lo que ocasiona errores de ejecución y falta de fiabilidad en el código, por lo que 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:

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.

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.

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 (`--`).

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 (`<=`).

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 (`=`)
- 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`.