¿Qué es ES6?

Todos conocemos JavaScript, uno de los lenguajes más utilizados por la comunidad mundial de desarrolladores, ¿pero sabemos sobre su historia, su creador u orígenes?

En 1995, Brendan Eich crea un lenguaje llamado “Mocha” cuando pertenecía al grupo de desarrollo del navegador Netscape.

En Septiembre de ese mismo año, deja el nombre de Mocha y lo renombra como LiveScript.

Cuando Netscape es adquirida por Sun Microsystems (propietaria del lenguaje Java), muy popular por aquellos años, es renombrado a JavaScript debido a una estrategia de marketing, aun cuando los lenguajes no están relacionados.

Para muchos la palabra ECMA no resulta conocida. Se trata de un acrónimo de “European Computer Manufacturers Association (ECMA)”, una organización internacional basada en membresías de estándares para la comunicación y la información. Actualmente, para expresar su alcance internacional, el nombre ya no se considera un acrónimo y no se escribe completamente en mayúsculas.

En el año de 1997 se crea un comité (TC39) en la ECMA para estandarizar JavaScript. A partir de entonces, los estándares de JavaScript se rigen como ECMAScript. No solo JavaScript se basa el lenguaje ECMAScript, existen otros como JScript y ActionScript 3 que también lo hacen. Haciendo una analogía, diremos que ECMAScript es el lenguaje y JavaScript, JScript y ActionScript 3 son dialectos de este lenguaje, siendo JavaScript su dialecto más conocido y utilizado.

Debido a esto no existen versiones propias de JavaScript, sino de su estándar contenedor ECMAScript, que desde el 2015 se encuentra en su versión número 6, que trae cambios importantes en la sintaxis del lenguaje.

Repaso de Variables

Una variable es como una caja, y dentro de ella podemos guardar cosas. Sólo que en las variables de JavaScript la caja sólo puede guardar una sola cosa a la vez.

Se las llama variables porque su contenido puede cambiar en cualquier momento durante el desarrollo de un programa. De esta forma, una variable en JavaScript puede contener distintas cosas según donde se encuentre en el programa: números, letras, fechas, y otros valores.

Para usar una variable, primero tenemos que crearla (a esto se le llama declarar una variable). Para hacerlo, escribimos la palabra clave **var** seguido del nombre que le quieres poner a la variable:

var nombre = “Ada”;  
var edad = 27;

Scope de Variables

El scope de una variable hace referencia al lugar donde va a vivir esta, o podrá ser accesible.

Cuando creamos una variable utilizando **var**, esta se crea con un scope de función, por lo que va a existir solo dentro del bloque de código de la función.

function unaFuncion() {   
 var nombre = 'Luke'  
 console.log(nombre); // 'Luke'  
}  
  
unaFuncion();  
  
console.log(nombre); // nombre is not defined

Como podemos ver en el ejemplo, la variable que declaramos con **var** llamada **nombre** no es accesible por fuera de la función.

Otro tipo de bloques (como los if/else, for, while, etc) no se consideran como un scope.

if(true) {   
 var nombre = 'Luke'  
}  
  
console.log(nombre); // 'Luke'

En este caso, utilizando **var**, podemos acceder a la variable **nombre** por fuera del bloque del if.

Con la introducción de ES6, se agregaron nuevas formas de crear una variable.

let, const y la introducción del scope de bloque

En ES6, **let** y **const** fueron introducidos como formas alternativas para crear una variable, teniendo las dos un scope de bloque.

En el scope de bloque, *cualquier* bloque (function, if, else, for, etc) va a ser considerado como un scope.

De esta forma, *function* sigue teniendo el mismo criterio al momento de declarar una variable.

function unaFuncion() {   
 let nombre = 'Luke'  
 console.log(nombre); // 'Luke'  
}  
  
unaFuncion();  
  
console.log(nombre); // nombre is not defined

Pero en el caso de otro tipo de bloque, ahora si clasifican como un *scope.*

if(true) {   
 let nombre = 'Luke'  
}  
  
console.log(nombre); // nombre is not defined

let

**let** funciona similar a **var**, pero la variable es creada con un scope de bloque. Esto quiere decir que esa variable solo vivirá en el bloque (global, function, if, for, etc) actual donde se está creando.

const

**const** funciona igual que **let**, con la diferencia que la variable que estamos declarando tiene que ser inicializada inmediatamente, y esa variable no va a poder ser reasignada después.

const nombre;

// SyntaxError: missing = in const declaration

const nombre = “Ada”;

nombre = “Grace”;  
 Uncaught TypeError: Assignment to constant variable.

**const** y **let** crean variables con scop de bloque (solo existen dentro del bloque interno que las rodea).

if(true) {   
 var nombreVar = 'Luke'  
}

if(true) {   
 let nombreLet = 'Luke'  
}

if(true) {   
 const nombreConst = 'Luke'  
}

console.log(nombreVar); // Luke

console.log(nombreLet); // Uncaught ReferenceError: nombreLet is not defined

console.log(nombreConst); // Uncaught ReferenceError: nombreConst is not defined

Strings - Template literals

Con la llega de ES6, tenemos una nueva forma de escribir strings en JavaScript, más cómoda y legible que cómo se hacía antes.

Antes, si queríamos imprimir un string con varias variables, teníamos que hacer lo siguiente:

let nombre = 'Ada;

let apellido = 'Lovelace';

console.log('Hola, mi nombre es ' + nombre + ' ' + apellido);

// Hola, mi nombre es Ada Lovelace

Con ES6 utilizando template strings podemos escribir lo mismo de una forma más cómoda utilizando las comillas invertidas **` `** y el símbolo del dólar $ seguido de las llaves {}, sin necesidad de usar el operador **+** para concatenar los strings.

console.log(`Hola, mi nombre es ${nombre} ${apellido}`);

// Hola, mi nombre es Ada Lovelace

Dentro de **${ }** podemos agregar cualquier cosa, incluso una expresión de JavaScript.

let nombre = 'Ada;

let apellido = 'Lovelace';

console.log(`Hola, mi nombre es ${nombre} ${apellido.toUpperCase()}`);

// Hola, mi nombre es Ada LOVELACE

**Strings Multilinea**

Antes de ES6, para escribir strings multilínea teníamos que hacerlo de una forma no muy cómoda utilizando el caractar **\n**

console.log('Esto es una línea\nEsto es otra línea\n y esto es la 3a línea');

O de esta forma para leerlo “mejor”:

console.log('Esto es una línea\n' +  
 'Esto es otra línea\n' +  
 'y esto es la 3a línea');

Con las comillas invertidas utilizadas en los *template strings* podemos añadir espacios y saltos de línea sin la necesidad de utilizar operadores extra ni el **+** para concatenar.

console.log(`Esto es una línea  
 Esto es otra línea  
 y esto es la 3a línea`);