ECMAScript 6

EcmaScript 6

- El lenguaje que habitualmente conocemos como **Javascript**, formalmente se le conoce como **EcmaScript**.

- La versión 6 (ES6) ofrece multitud de nuevas características para extender el potencial del lenguaje.

- ES6 todavía no está aceptado por la mayoría de navegadores, por lo que para usarlo, debemos transformarlo a ES5.

- Existen numerosos traductores para realizar esta acción. Angular 2 utiliza **TypeScript** para llevar a cabo este trabajo.

ES6

- Javascript fue creado en 1995, pero sigue siendo uno de los lenguajes más importantes actualmente.

Sep 2016	Sep 2015	Change	Programming Language
1	1		Java
2	2		C
3	3		C++
4	4		C#
5	5		Python
6	7	▲	JavaScript
7	6	▼	PHP
8	11	▲	Assembly language
9	8	▼	Visual Basic .NET
10	9	▼	Perl

ES6

- Alguna de las características más importantes agregadas en esta nueva versión son las siguientes:
  - Clases
  - Arrow Functions
  - Template Strings
  - Herencia
  - Constantes declaradas dentro de un ámbito.
  - Módulos
  - …

ES6 - Clases

- Como en la mayoría de lenguajes, en ES6 utilizamos las clases para poder definir cómo se van a implementar ciertos objetos que utilizaremos en nuestras aplicaciones.

- La posibilidad de utilizar clases, facilita la implementación de herencia dentro de los componentes de nuestro código.

```
class Persona{
  constructor(){
    // Constructor de la clase Persona
  }

  hablar(){
    // Método
  }
}
```

ES6 - Clases

- Podemos usar la palabra reservada **this** para hacer referencia a la instancia de la clase.

```
class Persona{
  constructor(){
    // Constructor de la clase Persona
  }

  hablar(){
    // Método
  }

  acciones(){
    // Llamamos al método hablar de la misma clase
    this.hablar();
  }
}
```

ES6 - Arrow Functions

- Tenemos una nueva característica que nos permite definir funciones anónimas de manera mucho más sencilla. Veamos la reescritura del siguiente ejemplo:

```
items.forEach(function(x)
  {  console.log(x);
  incrementedItems.push(x+1);
});

// Podríamos transformarlo en:

items.forEach((x) => {
  console.log(x);
  incrementedItems.push(x+1);
});
```

ES6 - Arrow Functions

- Esto hace extremadamente sencillo la definición de funciones que calculan su resultado en función de una única expresión.

```
incrementedItems = items.map((x) => x+1);
```

- Sería equivalente a:

```
incrementedItems = items.map(function (x)
  {  return x+1;
});
```

ES6 - Arrow Functions

- Hay que tener cuidado con el uso de este tipo de funciones ya que no tienen sus propias instancias de objetos como **this** o **super**, si no que reciben su valor del ámbito superior donde queden enmarcadas.

ES6 - Template Strings

- Tradicionalmente, en Javascript, las cadenas de caracteres vienen definidas por comillas simples o comillas dobles.

- No existía la posibilidad de insertar cualquier tipo de variable dentro de estas cadenas

- Si queríamos hacer algún tipo de concatenación, debíamos proceder de la siguiente manera:

```
var nombre = "Mario";
var edad = 32;
console.log("Hola, me llamo " + nombre + " y tengo " + edad + " años");
```

ES6 - Template Strings

- ES6 incluye un nuevo tipo de Strings, delimitado por el símbolo `.
- Estas cadenas, al contrario que las anteriores, pueden incluir nuevas líneas.
- Aparte, existe un nuevo mecanismo para la inclusión de variables dentro de estos strings.

```
let nombre = "Mario";
let edad = 32;
console.log(`Hola, me llamo ${nombre} y tengo ${edad} anyos`);
```

ES6 - Ámbito

- ES6 incluye un nuevo concepto de ámbito dentro de nuestros bloques.
- En ES5 y anteriores, las variables limitaban su ámbito a las funciones donde se encontraban definidas y podían acceder al ámbito superior del contexto de su función.

```
var cinco = 5;
var otroTres = tres; // ERROR

function ambito1(){
  var tres = 3;
  var otroCinco = cinco;  // CORRECTO
  var otroSiete = siete;  // ERROR
}

function ambito2(){
  var siete = 7;
  var otroCinco = cinco; // CORRECTO
  var otroTres = tres; // ERROR
}
```

ES6 - Ámbito

- En ES6, **var** sigue funcionando de la misma manera, pero aparecen dos nuevas maneras para declarar variables: **const** y **let.**

- Las variables definidas con los modificadores **const** y **let**, utilizan los bloques creados a partir de llaves ({ }) como contenedores de ámbito.

- Dichas variables quedarían definidas dentro de esos bloques y no serían accesibles desde fuera de los mismos:

```
var i = 0;
for (i= 0; i<10; i += 1{
  var j = i;
  let k = i;
}
console.log(j); // 9
console.log(k); // undefined
```

ES6 - Ámbito

- **let** funciona de la misma manera que **var.**
- Define una variable, cuyo resultado puede ser leído y escrito.
- Las variables definidas con **const** son de sólo lectura.
- Una vez que se ha utilizado **const** con algún identificador, éste no puede ser reasignado.

```
const literal = {};

literal.atributo = "prueba"; // CORRECTO
literal = [] // ERROR
```

ES6 - Operador de Propagación

- El operador de propagación *spread* nos permite que una expresión sea expandida en situaciones donde se esperan múltiples argumentos (llamadas a funciones) o múltiples elementos (arrays/objetos literales).

- El símbolo para representar dicho operador son tres puntos (…) delante de los elementos que vayamos a incluir.

- Podemos verlo, por ejemplo, a la hora de pasarle los argumentos a una función:

```
const agregar = (a, b) => a + b;
let argumentos = [3, 5];
console.log(agregar(...argumentos));
// Equivale a
console.log(agregar(3,5));
```

ES6 - Operador de Propagación

- Las funciones no son el único elemento donde podemos usar este nuevo elemento.

- Los arrays, se pueden concatenar de manera muy sencilla utilizando los tres puntos:

```
let cde = ['c', 'd', 'e'];
let letras = ['a', 'b', ...cde, 'f', 'g'];
console.log(letras); // ['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

ES6 - Operador de Propagación

- Lo mismo podemos hacer con los objetos definidos de manera literal:

```
let mapABC = { a: 5, b: 6, c: 3};
let mapABCD = { ...mapABC, d: 7}; // { a: 5, b: 6, c: 3, d: 7 }
console.log(mapABCD.a); // 5
```

ES6 - Argumentos variables

- No debemos confundir el operador anterior con el uso que tienen los tres puntos en la definición de argumentos en la declaración de una función.

- Con esta forma de definir los argumentos estamos indicando que, al método concreto se le pueden pasar múltiples variables.

```
function mostrarElementos (...elementos)
  {  console.log(elementos);
  console.log(elementos[0]);
}

mostrarElementos(2,5,1);
mostrarElementos(3);
```

ES6 - Valores por defecto de los parámetros

- De manera similar a otros lenguajes, podemos indicar los valores que tienen ciertos parámetros de manera predeterminada.

- Para ello, en la definición de la función, hemos de asignar a la variable el valor que deseemos.

```
function suma (x, y = 0)
    {  return x + y;
}
suma(10) === 10;
```

ES6 - Destructuring

- Se trata de una manera rápida de poder extraer los datos de las colecciones definidas con { } o [ ] sin tener que escribir mucho código.

- Podemos transformar el siguiente código:

```
let foo = ['uno', 'dos', 'tres'];

let uno = foo[0];
let dos = foo[1];
let tres = foo[2];
```

- Por el siguiente:

```
let foo = ['uno', 'dos', 'tres'];

let [uno, dos, tres] = foo;
console.log(uno);
```

ES6 - Destructuring

- Podemos utilizar esta técnica cuando trabajamos con objetos más complejos:

```
let myModule = {
    drawSquare: function drawSquare(length) { /* código */ },
    drawCircle: function drawCircle(radius) { /* código */ },
    drawText: function drawText(text) { /* código */ },
};

let {drawSquare, drawText} = myModule;
drawSquare(5);
drawText('hello');
```

ES6 - Import y Export

- Podemos importar librerías y exportar funciones o variables.

```
// fichero lib/math.js
export function sum (x, y) { return x + y };
export var pi = 3.141593;

// Podemos importar todo el fichero
import * as math from "lib/math";
console.log("2π = " + math.sum(math.pi, math.pi));

// O solo los elementos que nos interesan
import { sum, pi } from "lib/math";
console.log("2π = " + sum(pi, pi));
```