FUENTE: <https://www.boardinfinity.com/blog/top-10-features-of-es6/>

**1. palabras clave let y const:**

La palabra clave "let" permite a los usuarios definir variables y, por otro lado, "const" permite a los usuarios definir constantes.

Las variables se declaraban previamente usando "var", que tenía alcance de función y se elevaban a la cima.

Significa que se puede utilizar una variable antes de la declaración. Pero las variables y constantes "let" tienen un alcance de bloque que está rodeado por llaves "{}" y no se pueden usar antes de la declaración.

**2. Arrow Functions**

ES6 proporciona una característica conocida como funciones de flecha. Proporciona una sintaxis más concisa para escribir expresiones de funciones eliminando las palabras clave "función" y "retorno".

|  |
| --- |
| // Arrow function  **let sumOfTwoNumbers = (a, b) => a + b;**  console.log(sum(10, 20)); // Output 30 |

Es evidente que no hay ninguna palabra clave "retorno" o "función" en la declaración de la función de flecha.

También podemos omitir el paréntesis en el caso de que haya exactamente un parámetro, pero siempre necesitaremos usarlo cuando tenga cero o más de un parámetro.

Pero, si hay varias expresiones en el cuerpo de la función, entonces debemos encerrarla entre llaves ("{}"). También necesitamos usar la declaración "return" para devolver el valor requerido.

**3. Multi-line Strings**

ES6 también proporciona cadenas multilínea. Los usuarios pueden crear cadenas de varias líneas utilizando comillas invertidas (`).

|  |
| --- |
| let greeting = `Hello World,  Greetings to all,  Keep Learning and Practicing!` |

**4. Default Parameters**

En ES6, los usuarios pueden proporcionar los valores predeterminados directamente en la firma de las funciones. Pero, en ES5, se debía utilizar el operador OR.

|  |
| --- |
| //ES6  let calculateArea = function(height = 100, width = 50) {  // logic  }  //ES5  var calculateArea = function(height, width) {  height = height || 50;  width = width || 80;  // logic  } |

**5. Template Literals**

ES6 introduce plantillas de cadenas muy simples junto con marcadores de posición para las variables. La sintaxis para usar la plantilla de cadena es ${PARAMETER} y se usa dentro de la cadena con tilde inverso.

|  |
| --- |
| let name = `My name is ${firstName} ${lastName}` |

**6. Destructuring Assignment**

La desestructuración es una de las características más populares de ES6. La asignación desestructurante es una expresión que facilita la extracción de valores de matrices o propiedades de objetos en variables distintas.

Hay dos tipos de expresiones de asignación de desestructuración, a saber, desestructuración de matrices y desestructuración de objetos. Se puede utilizar de la siguiente manera:

|  |
| --- |
| //Array Destructuring  let fruits = ["Apple", "Banana"];  let [a, b] = fruits; // Array destructuring assignment  console.log(a, b);  //Object Destructuring  let person = {name: "Peter", age: 28};  let {name, age} = person; // Object destructuring assignment  console.log(name, age); |

**7. Enhanced Object Literals**

ES6 proporciona literales de objetos mejorados que facilitan la creación rápida de objetos con propiedades dentro de las llaves.

|  |
| --- |
| function getMobile(manufacturer, model, year) {  return {  manufacturer,  model,  year  }  }  getMobile("Samsung", "Galaxy", "2020"); |

**8. Promises**

En ES6, las promesas se utilizan para la ejecución asincrónica. Podemos usar la promesa con la función de flecha como se muestra a continuación.

|  |
| --- |
| var asyncCall = new Promise((resolve, reject) => {  // do something  resolve();  }).then(()=> {  console.log('DON!');  }) |

**9. Classes**

Anteriormente, las clases nunca existían en JavaScript. Las clases se introducen en ES6 y se parecen a las clases de otros lenguajes orientados a objetos, como C++, Java, PHP, etc. Pero no funcionan exactamente de la misma manera. Las clases de ES6 simplifican la creación de objetos, implementan la herencia mediante el uso de la palabra clave "extends" y también reutilizan el código de manera eficiente.

En ES6, podemos declarar una clase usando la nueva palabra clave "clase" seguida del nombre de la clase.

|  |
| --- |
| class UserProfile {  constructor(firstName, lastName) {  this.firstName = firstName;  this.lastName = lastName;  }    getName() {  console.log(`The Full-Name is ${this.firstName} ${this.lastName}`);  }  }  let obj = new UserProfile('John', 'Smith');  obj.getName(); // output: The Full-Name is John Smith |

**10. Modules**

Anteriormente, no había soporte nativo para módulos en JavaScript. ES6 introdujo una nueva característica llamada módulos, en la que cada módulo está representado por un archivo ".js" separado. Podemos usar la declaración "importar" o "exportar" en un módulo para importar o exportar variables, funciones, clases o cualquier otro componente desde/hacia diferentes archivos y módulos.

|  |
| --- |
| export var num = 50;  export function getName(fullName) {  //data  };  import {num, getName} from 'module';  console.log(num); // 50 |

Spread

Fuente: <https://eloquentjavascript.net/05_higher_order.html>

**HighOrder Functions**

Las funciones que operan sobre otras funciones, ya sea tomándolas como argumentos o devolviéndolas, se denominan funciones de orden superior.

Como ya hemos visto que las funciones son valores regulares, no hay nada particularmente destacable en el hecho de que tales funciones existan.

El término proviene de las matemáticas, donde se toma más en serio la distinción entre funciones y otros valores.

**MAP**:

El método map transforma una matriz aplicando una función a todos sus elementos y construyendo una nueva matriz a partir de los valores devueltos. La nueva matriz tendrá la misma longitud que la matriz de entrada, pero su contenido se habrá asignado a una nueva forma.

**REDUCE:**

Otra cosa común que se puede hacer con las matrices es calcular un valor único a partir de ellas. Nuestro ejemplo recurrente, la suma de una colección de números, es un ejemplo de esto. Otro ejemplo es encontrar el guión con más personajes.

La operación de orden superior que representa este patrón se llama **reducir** (a veces también se llama **plegar**). Crea un valor tomando repetidamente un solo elemento de la matriz y combinándolo con el valor actual. Al sumar números, comenzarías con el número cero y, para cada elemento, lo sumarías a la suma.

Los parámetros a reducir son, además del array, una función combinadora y un valor inicial. Esta función es un poco menos sencilla que filtrar y mapear, así que mírala de cerca

**FOREACH**

La función de orden superior forEach() se utiliza para iterar sobre un objeto de matriz. Toma y regresa una vez que la función de devolución de llamada se analiza como argumento

El método forEach() toma tres parámetros de elemento, índice y matriz.

* elemento: especifica cada elemento de la matriz.
* índice: especifica el índice de base cero de los elementos de la matriz.
* matriz: especifica la matriz completa.
* forEach((element) => { /\* … \*/ })
* forEach((element, index) => { /\* … \*/ })
* forEach((element, index, array) => { /\* … \*/ })

MAS REFERENCIAS:

<https://dev.to/alexanie_/higher-order-array-methods-in-javascript-4hj2#:~:text=The%20forEach()%20higher%2Dorder,parse%20in%20as%20an%20argument>.

<https://www.codecademy.com/learn/game-dev-learn-javascript-higher-order-functions-and-iterators/modules/game-dev-learn-javascript-iterators/cheatsheet>

<https://www.freecodecamp.org/news/higher-order-functions-in-javascript/>

/\* Ejemplo de REDUCE

This method executes a callback function on every element in the array,

which results in a single value.

The method accepts two arguments:

Callback function

Initial value

The callback function accepts the following four parameters:

Accumulator

Current value

Current index

Source array

https://medium.com/@xadrijo/un-vistazo-a-la-funci%C3%B3n-reduce-en-javascript-f4459bc4e3aa

\*/