

# Diplomatura en programación web full stack con React JS



Módulo 3:

# JavaScript y maquetado avanzado

Unidad 4:

## JavaScript (parte 2)



## Presentación

En esta unidad vemos qué es el DOM, cómo manipularlo y responder a sus eventos así como también los distintos métodos de manipulación de arrays y objetos en Javascript.



## Objetivos

**Que los participantes logren...**

- Conocer el DOM para poder acceder, añadir y cambiar elementos del documento.
- Manipular las propiedades, funciones y eventos de los objetos.
- Explorar distintos métodos de manipulación de arrays y objetos en Javascript..



## Bloques temáticos

1. Funciones.
2. DOM.
3. Eventos.
4. Métodos de array: map, filter y find.
5. Interpolación de strings.
6. JSON.
7. Destructuring y operador spread.

# 1. Funciones

Cuando se desarrolla una aplicación compleja, es muy habitual utilizar una y otra vez las mismas instrucciones. Un script para una tienda de comercio electrónico, por ejemplo, tiene que calcular el precio total de los productos varias veces, para añadir los impuestos y los gastos de envío.

Cuando una serie de instrucciones se repiten una y otra vez, se complica demasiado el código fuente de la aplicación, ya que:

- El código de la aplicación es mucho más largo porque muchas instrucciones están repetidas.
- Si se quiere modificar alguna de las instrucciones repetidas, se deben hacer tantas modificaciones como veces se haya escrito esa instrucción, lo que se convierte en un trabajo muy pesado y muy propenso a cometer errores.

Las funciones son la solución a todos estos problemas, tanto en JavaScript como en el resto de lenguajes de programación. **Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente.**

En el siguiente ejemplo, las instrucciones que suman los dos números y muestran un mensaje con el resultado se repiten una y otra vez:



```
var resultado;

var numero1 = 3;
var numero2 = 5;

// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);

numero1 = 10;
numero2 = 7;
// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);

numero1 = 5;
numero2 = 8;


// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);
```

Aunque es un ejemplo muy sencillo, parece evidente que repetir las mismas instrucciones a lo largo de todo el código no es algo recomendable. La solución que proponen las funciones consiste en extraer las instrucciones que se repiten y sustituirlas por una instrucción del tipo "en este punto, se ejecutan las instrucciones que se han extraído".

Por lo tanto, en primer lugar se debe crear la función básica con las instrucciones comunes. Las funciones en JavaScript se definen mediante la palabra reservada `function`, seguida del nombre de la función. Su definición formal es la siguiente:


```
function nombre_funcion() {  
  
    .....  
  
}
```

El nombre de la función se utiliza para llamar a esa función cuando sea necesario. El concepto es el mismo que con las variables, a las que se les asigna un nombre único para poder utilizarlas dentro del código.



```
function suma_y_muestra() {  
    resultado = numero1 + numero2;  
    alert("El resultado es " + resultado);  
}
```

De esta forma, el ejemplo anterior quedaría así:




```
function suma_y_muestra() {  
    resultado = numero1 + numero2;  
    alert("El resultado es " + resultado);  
}  
  
var resultado;  
var numero1 = 3;  
var numero2 = 5;  
  
suma_y_muestra()  
  
numero1 = 10;  
numero2 = 7;  
  
suma_y_muestra();  
  
numero1 = 5;  
numero2 = 8;  
  
suma_y_muestra();
```




Otra posibilidad de las funciones es la de recibir parámetros. Estos son valores que pueden ser usados dentro del código que la función ejecuta. De esta manera las funciones se vuelven sumamente reutilizables. Cuando definimos la función, damos nombre a los parámetros que queremos recibir y estos estarán disponibles como variables dentro de la función.

Cuando llamemos a la función solo debemos pasar la variable o valor que deseemos usar como parámetro de la función en la posición correcta.



```
function saludar(nombre) {  
  alert("Hola " + nombre);  
}  
  
saludar("Gabriela")  
saludar("Antonio")
```

Por último, las funciones también puede devolver un valor o resultado mediante el uso de la palabra clave return. Dicho valor puede ser almacenado en una variable o utilizado en alguna de las estructuras de control o bucles que vimos previamente.



```
function numero_al_cubo(numero) {  
  return numero * numero * numero;  
}  
  
var tres_al_cubo = numero_al_cubo(3);  
alert(tres_al_cubo);
```

## 2. DOM

El **DOM** (Document Object Model en español Modelo de Objetos del Documento) es una API definida para representar e interactuar con cualquier documento HTML o XML. El DOM es un modelo de documento que se carga en el navegador web y que representa el documento como un árbol de nodos, en donde cada nodo representa una parte del documento (puede tratarse de un elemento, una cadena de texto o un comentario).

El DOM es una de las APIs más usadas en la Web, pues permite ejecutar código en el navegador para acceder e interactuar con cualquier nodo del documento. Estos nodos pueden crearse, moverse o modificarse. Pueden añadirse a estos nodos manejadores de eventos (event listeners en inglés) que se ejecutarán/activarán cuando ocurra el evento indicado en este manejador.

El DOM surgió a partir de la implementación de JavaScript en los navegadores. A esta primera versión también se la conoce como DOM o "Legacy DOM". Hoy en día el grupo WHATWG es el encargado de actualizar el estándar de DOM.

### Selectores de elementos

Los selectores api proveen métodos que hacen más fácil y rápido devolver elementos del nodo Element del DOM mediante emparejamiento de un conjunto de selectores. Esto es mucho más rápido que las técnicas anteriores, donde fuera necesario, por ejemplo usar un loop en un código JavaScript para localizar el ítem específico que quisieras encontrar.

#### **document.querySelector()**

Devuelve la primera coincidencia del (elemento) Element nodo dentro de las subramas del nodo. Si no se encuentra un nodo coincidente, se devuelve null.

### **document.querySelectorAll()**

Devuelve un listado de nodos NodeList conteniendo todos los elementos del nodo coincidentes( Element) dentro de las subramas del nodo, o Devuelve un Listado de Nodos vacío NodeList si no se encuentran coincidencias.

Estos métodos aceptan uno o más selectores separados por comas entre cada selector para determinar qué elemento o elementos deben ser devueltos. por ejemplo para seleccionar todos los elementos (p) del párrafo en un documento donde la clase CSS sea tanto warning or note, puedes hacer lo siguiente:

```
var special = document.querySelectorAll( "p.warning, p.note" );
```

También por usar query para etiquetas id. Por ejemplo:

```
var destacados = document.querySelector( "#principal,  
#bienvenidos, #notas" );
```

Luego de ejecutar el código de arriba, la variable contiene el primer elemento del documento, su ID puede ser uno de los siguientes: principal, bienvenidos, o notas.

Podes usar cualquier selector CSS con los métodos **querySelector()** y **querySelectorAll()**.

### **document.getElementById()**

Este método permite seleccionar una elemento del DOM usando el atributo id del mismo.



```
var mensaje = document.getElementById('mensaje');
```

### **document.getElementsByClassName()**

Este método devuelve un array o lista de elementos que contengan la clase que se especifique como parámetro.



```
var items = document.getElementsByClassName('item');
```

### **document.getElementsByTagName()**

Este método devuelve un array o lista de elementos cuya etiqueta html sea la que se especifique como parámetro.



```
var parrafos = document.getElementsByTagName('p');
```

## Crear elementos y agregarlos al DOM

Podemos crear cualquier tipo de elemento usando el método **document.createElement()**. Este método recibe como parámetro el nombre de la etiqueta del elemento que deseamos crear, por ejemplo li.

Una vez creado el elemento, este no será visible al usuario hasta que lo agreguemos explícitamente al DOM, para ello, los elementos o nodos del DOM cuentan con un método llamado **appendChild()** que recibe como parámetro el elemento que se desee agregar.



```
<body>
  <div id="div1">El texto siguiente es dinámico</div>

  <script>
    function agregarElemento() {
      var newDiv = document.createElement('div');
      newDiv.innerHTML = 'Hola!';

      document.body.appendChild(newDiv);
    }
    agregarElemento();
  </script>
</body>
```

## Eliminar elementos del DOM

Para eliminar elementos del DOM solo debemos obtener una referencia al elemento usando cualquiera de los métodos de selección que mencionamos previamente, y llamar a al método **remove()**.



```
<div id="div1">Este es el div1</div>
<div id="div2">Este es el div2</div>
<div id="div3">Este es el div3</div>

<script>
  var elemento = document.getElementById('div2');
  elemento.remove();
</script>
```

## innerHTML e innerText

Las propiedades de los elementos del DOM `innerHTML` e `innerText` hacen referencia al contenido de los nodos. Mientras que `innerText` **permite insertar** y manipular **contenido sencillo** dentro de los nodos, `innerHTML` nos permite **incluir código HTML** más complejo sin la necesidad de crear los elementos a mano.

```
<body>
  <p id="demo">Este elemnto tiene espacio extra y contiene
  <span>una etiqueta span</span></p>
  <script>

    function getHTML() {
      alert(document.getElementById('demo').innerHTML);
    }

    function getInnerText() {
      alert(document.getElementById('demo').innerText);
    }
  </script>
</body>
```

## Atributos de los elementos

Podemos acceder a los atributos de cualquier elemento que hayamos seleccionado simplemente usando el nombre del atributo como propiedad del nodo, Por ejemplo

```
var miLink = document.getElementsByTagName('a')[0];
alert(miLink.href);
```

De esta forma veremos el mensaje de alerta con el contenido del atributo **href** de nuestro link. Del mismo modo podemos escribir la propiedad y cambiar su valor según necesitemos.

## Estilos de los elementos

Los estilos CSS se pueden manipular mediante javascript accediendo a la propiedad **style** que tienen todos los elementos o nodos del DOM. Hay que tener en cuenta que las propiedades cuyo nombre lleva un guión en su nombre (estilo llamado kebab-case) cómo font-size se convierten al estilo camelCase, resultando en fontSize.




```
var intro = document.getElementById('intro');  
intro.style.backgroundColor = '#ff00ff';
```

## 3. Eventos

Javascript permite responder a eventos en el DOM mediante el uso de funciones. Existen 3 formas distintas de registrar gestores de eventos para un elemento del DOM:


### EventTarget.addEventListener



```
myButton.addEventListener('click', function(){  
    alert('Hello world');  
}, false);
```

Esta es la forma recomendada y la más moderna de las 3. Permite registrar más de un listener por evento, lo que brinda una mayor flexibilidad.

### Atributo HTML




```
<button onclick="alert('Hola Mundo')">Saludar</button>
```

Esta forma es la menos recomendada porque incrementa el tamaño del HTML y dificulta su lectura, además de que resulta muy complejo incluir múltiples instrucciones.



## Propiedad del elemento DOM



```
myButton.onclick = function(event){  
  alert('Hello world');  
};
```

Este método solo permite registrar un listener por evento.

## 4. Métodos de array: map, filter y find

### Array.map()

El método `map()` crea un nuevo array con los resultados de la llamada a la función indicada aplicados a cada uno de sus elementos.

```
1 const numeros = [1, 3, 8, 20];
2 const dobles = numeros.map(function(num) {
3   return num * 2;
4 });
5
6 console.log(dobles); // [2, 6, 16, 40];
7
8 // versión simplificada usando arrow functions
9 const dobles = numeros.map(num => num * 2)
```

### Array.filter()

El método `filter()` crea un nuevo array con todos los elementos que cumplan la condición implementada por la función dada.

```
1 const numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
2 const pares = numeros.filter(num => num % 2 === 0);
3 console.log(pares); // [2, 4, 6, 8, 10]
```

## Array.find()

El método **find()** devuelve el valor del primer elemento del array que cumple la función de prueba proporcionada, de lo contrario devuelve undefined.



```
1 const numeros = [3, 5, 10, 15, 28, 76, 149];  
2 const encontrado = numeros.find(num => num > 12);  
3 console.log(encontrado); // 15
```

## 5. Interpolación de strings

Las plantillas de cadena de texto (o template strings) son una característica implementada desde ES6 que nos permite, entre otras cosas, interpolar o intercalar variables o expresiones dentro de nuestras cadenas de texto sin tener que recurrir a la concatenación.

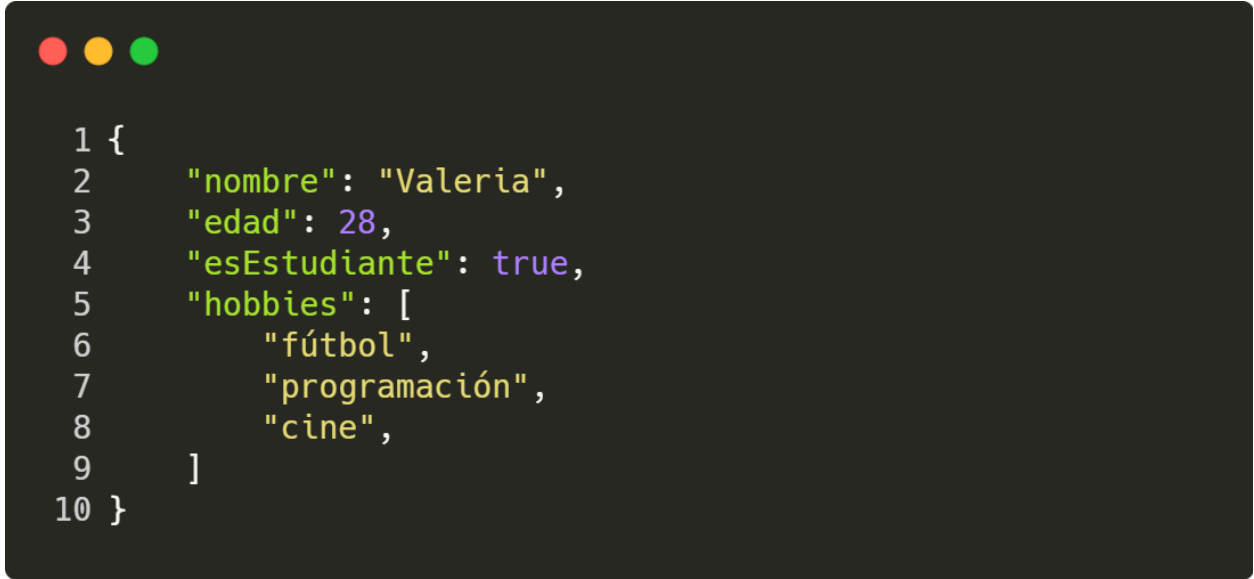
Para hacer uso de esta nueva funcionalidad tenemos que rodear nuestros strings con la tilde invertida (``) y las variables o expresiones que deseemos interpolar deberán estar contenidas dentro de la estructura `${}`.

```
1 var edad1 = 18;
2 var edad2 = 32;
3 var mensaje1 = `La cuota para personas de ${edad1} es de
  ${edad1 >= 21 ? 500 : 350}`;
4 // La cuota para las personas de 18 es de 350
5 var mensaje2 = `La cuota para personas de ${edad2} es de
  ${edad2 >= 21 ? 500 : 350}`;
6 // La cuota para las personas de 32 es de 500
```

## 6. JSON

JSON o JavaScript Object Notation es un formato de texto utilizado para definir estructuras de objetos y listas complejas. Es ampliamente utilizado para el intercambio de datos entre aplicaciones debido a la simplicidad de su implementación y la facilidad de lectura por parte de las aplicaciones y las personas. Es un subconjunto de la notación de objetos literal de Javascript, por lo cual existen múltiples similitudes entre ambos.

### Estructura básica



```
1 {  
2     "nombre": "Valeria",  
3     "edad": 28,  
4     "esEstudiante": true,  
5     "hobbies": [  
6         "fútbol",  
7         "programación",  
8         "cine",  
9     ]  
10 }
```

En el ejemplo de estructura podemos apreciar lo siguiente:

- Todo el objeto está rodeado por llaves `{}`.
- Los nombres de las propiedades están rodeados por comillas dobles `"`.
- Los tipos de valores posibles son: texto, número, booleano, array o incluso otros objetos.
- Las listas o arrays se rodean con corchetes `[]` y pueden contener cualquiera de los tipos de datos aceptados.




En javascript podemos definir lo que se denomina objeto literal usando una sintaxis similar a JSON. Una de las principales diferencias es que no es necesario utilizar comillas dobles en los nombres de propiedades. El ejemplo anterior podría escribirse en javascript de la siguiente manera.

```
1 const persona = {
2     nombre: "Valeria",
3     edad: 28,
4     esEstudiante: true,
5     hobbies: [
6         "fútbol",
7         "programación",
8         "cine",
9     ]
10 };
11
12 console.log(persona.nombre); // "Valeria"
13 console.log(persona.hobbies.length); // 3
```

## 7. Destructuring y operador spread

La desestructuración ( destructuring ) es una técnica utilizada para extraer y declarar varias variables a la vez. Podemos aplicar esta técnica a arrays, objetos, y otros tipos de estructuras nuevas en ES6 como maps y sets.

### Arrays



```
1 const colores = ["#ff0000", "#00ff00", "#0000ff"];
2
3 // En vez de hacer esto
4 const rojo = colores[0];
5 const verde = colores[1];
6 const azul = colores[2];
7
8 // Podemos hacer esto
9 const [rojo, verde, azul] = colores;
```



## Objetos

```
1 const pelota = {
2   posicion:{
3     x: 150,
4     y: 150
5   },
6   colorDeRelleno:"tomato",
7   radio: 25,
8 }
9
10 // En vez de hacer esto
11 const posicion = pelota.posicion;
12 const radio = pelota.radio;
13 const colorDeRelleno = pelota.colorDeRelleno;
14
15 // Podemos hacer esto
16 const { posicion, radio, colorDeRelleno } = pelota;
```

## Operador spread

El operador spread (...) sirve para obtener todas las propiedades de un objeto o los elementos de un array. Es de suma utilidad para cuando necesitamos hacer copias de objetos o listas modificando alguno de los valores o agregando nuevos.



## Arrays

```
1 const vocales = ['a', 'e', 'i'];
2 const vocalesCompletas = [...vocales, 'o', 'u'];
3 console.log(vocales); // ['a', 'e', 'i']
4 console.log(vocalesCompletas); // ['a', 'e', 'i', 'o', 'u']
```

## Objetos

```
1 const usado = {
2   marca: 'Chevrolet',
3   modelo: 'Corsa',
4   anio: 2012
5 };
6
7 const nuevo = {
8   ...usado,
9   anio: 2020
10 };
11
12 console.log(nuevo.marca); // Chevrolet
13 console.log(nuevo.anio); // 2020
```



## Bibliografía utilizada y sugerida

### Artículos de revista en formato electrónico:

**MDN Web Docs.** Disponible desde la URL: <https://developer.mozilla.org/>

### Libros y otros manuscritos:

**Eguiluz Pérez, Javier.** Introducción a JavaScript. España: [www.librosweb.es](http://www.librosweb.es) 2008.