Hay otra funcionalidad que puedes gestionar desde Javascript y es la utilización de APIs. Las APIs son la exposición de una serie de funcionalidades del mismo grupo, y desarrolladas en bloque, que resuelven problemas que te encuentras en tus propias aplicaciones y que, de otro modo, tendrías que programar tú.

Por ejemplo, el API que expone métodos de reconocimiento de voz para pasarla a texto está disponible para que la uses directamente en tus programas. ¿Imaginas tener que programar de cero esas funcionalidades?. Sí,  mola, pero igual está fuera del alcance de los tiempos que has pactado con tu Product Owner.

Generalmente los API’s se dividen en dos categorías:

* Las **Browser** API’s:
  + **DOM:** para manejar los nodos de la estructura HTML que ya hemos estudiado previamente.
  + **Canvas:** creación de gráficos 2D dentro de un contexto gráfico
  + **WebGL:** una pequeña abstracción de OpenGL dentro del navegador para realizar tareas con gráficos 2D/3D.
  + **Geolocalización**: para poder geolocalizar al cliente de tus aplicaciones.
  + Etc
* Las **third-party**API’s:
  + **Google Maps:** permite la inserción de mapas dentro de nuestros sitios (¡cuidado si tienes éxito con tu sitio que no es del todo gratis!  )
  + **Twitter:** para incluir los últimos tweets de alguien dentro de tu sitio web.
  + etc

|  |  |  |
| --- | --- | --- |
| **Variable** | **Explicación** | **Ejemplo** |
| [**String**](https://developer.mozilla.org/en-US/docs/Glossary/String) | Una cadena de texto. Para indicar que la variable es una cadena, debes  escribirlo entre comillas. | var miVariable = ‘Bob’; |
| [**Number**](https://developer.mozilla.org/en-US/docs/Glossary/Number) | Un número. Los números no tienen comillas. | var miVariable = 10; |
| [**Boolean**](https://developer.mozilla.org/en-US/docs/Glossary/Boolean) | Tienen valor verdadero/falso. true/false son palabras especiales en JS, y no necesitan comillas. | var miVariable = true; |
| [**Array**](https://developer.mozilla.org/en-US/docs/Glossary/Array) | Una estructura que te permite almacenar varios valores en una sola referencia. | var miVariable = [1,’Bob’,’Steve’,10];  Llama a cada miembro del array así: miVariable[0], miVariable[1], etc. |
| [**Object**](https://developer.mozilla.org/en-US/docs/Glossary/Object) | Básicamente cualquier cosa. Todo en JavaScript es un objeto y puede ser almacenado en una variable. Mantén esto en mente mientras aprendes. | var miVariable = document.querySelector(‘h1’);  Todos los ejemplos anteriores también. |

**||**es el operador**“or” (o)**

**&&**es el operador**“and” (y)**

**!**es el operador**“not” (no)**

1. var a = "Hola";
2. alert**(typeof(**a**))**;
3. // sacaría un mensaje por pantalla con el tipo String como valor

|  |
| --- |
|  |
| igualdad | == | x == 10  x == “10”  x == 5 | True  True  false | |
| Igualdad verificando el tipo | === | x === 10  x === “10” | True  False | |
| Menor que | < | x < 11 | true | |
| Mayor que | > | x > 11 | false | |
| Menor o igual que | <= | x <= 10 | True | |
| Mayor o igual que | >= | x >= 11 | false | |
| Distinto de | != | x != 10  x != 11 | false  true | |
| Distinto de con tipo | !== | x !== “10” | true | |
| Operador ternario | ? | Cond ? true : false |  |

**Sintaxis**[**Sección**](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/Conditional_Operator#Sintaxis)

*condición* ? *expr1* : *expr2*

Parámetros[Sección](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/Conditional_Operator#Par%C3%A1metros)

**condición**

Una expresión que se evalúa como true o false.

**expr1, expr2**

Expresión con valores de algún tipo.

**Descripción**[**Sección**](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/Conditional_Operator#Descripci%C3%B3n)

Si la condición es true, el operador retorna el valor de la expr1; de lo contrario,  devuelve el valor de expr2. Por ejemplo, para mostrar un mensaje diferente en función del valor de la variable *isMember,* se puede usar esta declaración:

"La Cuota es de: " + (isMember ? "$2.00" : "$10.00")

También puedes asignar variables dependiendo del resultado de la condición ternaria:

var elvisLives = Math.PI > 4 ? "Sip" : "Nop";

También es posible realizar evaluaciones ternarias múltiples (Nota: El operador condicional es asociativo):

var firstCheck = false,

secondCheck = false,

access = firstCheck ? "Acceso Denegado" : secondCheck ? "Acceso Denegado" : "Acceso Permitido";

console.log( access ); // muestra "Acceso Permitido"

También puede usar operaciones ternarias en espacio vacío con el propósito de realizar diferentes operaciones:

var stop = false, age = 16;

age > 18 ? location.assign("continue.html") : stop = true;

También puede realizar más de una operación por caso, separándolas con una coma:

var stop = false, age = 23;

age > 18 ? (

alert("OK, puedes continuar."),

location.assign("continue.html")

) : (

stop = true,

alert("Disculpa, eres menor de edad!")

);

También puede realizar más de una operación durante la asignación de un valor. En este caso, ***el último valor separado por una coma del paréntesis*será el valor asignado**.

var age = 16;

var url = age > 18 ? (

alert("OK, puedes continuar."),

// alert devuelve "undefined", pero será ignorado porque

// no es el último valor separado por comas del paréntesis

"continue.html" // el valor a ser asignado si age > 18

) : (

alert("Eres menor de edad!"),

alert("Disculpa :-("),

// etc. etc.

"stop.html" // el valor a ser asignado si !(age > 18)

);

location.assign(url); // "stop.html"

1. var a = "Si algún **\"**afectado**\"** está en la sala...";

Podemos saber cuál es la **longitud de una cadena** con la propiedad length.

1. var a = "Hola";
2. alert**(**a.length**)**;
3. // Nos mostrará el mensaje con 4 como resultado

También podemos rescatar un carácter concreto de una cadena indicando su posición y entendiendo que el primer índice no es 1 sino 0.

1. alert**(**a**[**1**])**;
2. // Nos mostrará la "o" como resultado, no la “H”

Podemos realizar la comprobación de una subcadena, dentro de la cadena original, con la función indexOf y el parámetro “subcadena a encontrar”, de la siguiente forma:

1. alert**(**a.indexOf**(**"la"**))**;
2. // Nos mostrará 2 ya que es el posición donde comienza la subcadena dentro de Hola
3. alert**(**a.indexOf**(**"le"**))**;
4. // Nos mostrará -1 ya que no existe la subcadena dentro de la original

Podemos extraer una subcadena de la original, sabiendo su posición de comienzo y la longitud de lo que queremos extraer:

1. alert**(**a.slice**(**0,2**))**;
2. // Nos mostrará Ho

Y, por último en este repaso, voy a mostrar el cambio de todos los caracteres a minúsculas o a mayúsculas con las funciones toLowerCase() y toUpperCase() y cómo sustituir parte de una cadena por otra parte concreta.

1. alert**(**a.toUpperCase**())**;
2. // Nos mostraría un mensaje con HOLA
3. alert**(**a.toLowerCase**())**;
4. // Nos mostraría un mensaje con hola
5. alert**(**a.replace**(**“Ho”, “ha”**))**;
6. // Nos mostraría un mensaje con hala
7. alert**(**b**[**1**])**;

En el caso de tener arrays multidimensionales, es decir de más de una dimensión, tendremos que otorgar el índice concreto de dicha dimensión.

Si queremos acceder al elemento 4 del array b tendremos que indicar lo siguiente:

1. alert**(**b**[**2**][**1**])**;

Los arrays comparten ciertos métodos con las cadenas, por lo que podemos realizar lo siguiente para saber la longitud del array:

1. alert**(**b.length**)**;
2. // Nos mostraría que este array tiene 3 elemento

## mostrarlos en una tabla de datos. Para hacer esto, podemos usar el método [split()](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/String/split).

En su forma más simple, esto toma un único parámetro, el carácter que quieres separar de la cadena, y devuelve las subcadenas entre el separador como elementos en una matriz.

1. var a = "1,2,3,4,5";
2. alert**(**a.split**(**","**))**;
3. // Nos crearía un array [1, 2, 3, 4, 5]

También puedes ir en la dirección opuesta usando el método [join()](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/join). Prueba lo siguiente:

1. var a = **[**1, 2, 3, 4, 5**]**;
2. alert**(**a.join**(**","**))**;
3. // ahora nos va a crear una cadena unidad por "," que es el separador propuesto

No hemos dicho cómo cambiar un elemento concreto de un array, pero es muy fácil. Por ejemplo:

1. var a = **[**1,3,3**]**;
2. a**[**1**]** = 2;
3. // ahora la lista queda como [1, 2, 3]

También es muy fácil meter y sacar elementos de un array gracias a las funciones push y pop. Por ejemplo:

1. var a = **[**1,2**]**;
2. a.push**(**3**)**;
3. // ahora la lista queda como [1, 2, 3]
4. var a = **[**1, 2, 3**]**;
5. a.pop**()**;
6. // ahora la lista queda como [1, 2]

Podemos realizar ambas operaciones, al principio de un array, con las funciones **unshift()** y **shift():**

1. var a = **[**2, 3**]**
2. a.shift**(**1**)**;
3. // ahora la lista queda como [1, 2, 3]
4. a.unshift**()**;
5. // ahora la lista vuelve a quedar como [2,3]

* El ya comentado **window** object. Que, básicamente, es el tab donde cargamos nuestras páginas. Este objeto nos ofrece un montón de información para poder jugar a adaptar nuestras necesidades a sus propiedades.
* El objeto **Navigator**. Desde aquí podemos acceder a otro tipo de información útil como la geolocalización del usuario, el user-agent, el lenguaje preferido del usuario…

El objeto **Document**. Este es el objeto que quiero tratar aquí y que va de cómo manipular el árbol del DOM.

# Estos son los principales elementos que debes entender sobre el  DOM.

* **Elemento Nodo**: un elemento como existe dentro del DOM.
* **Nodo Root**: El elemento padre o top del árbol de nodos. En el caso de un HTML es siempre un HTMLNode, en otros vocabularios como SVG, XML tienen un root distinto.
* **Nodo hijo**: Un nodo que se encuentra directamente debajo de otro nodo.
* **Nodo descendiente**: Un nodo que se encuentra debajo de otro nodo pero sin que la condición sea que esté directamente debajo. Es decir, puede estar debajo de un nodo hijo.
* **Nodo padre**: Un nodo que tiene dentro de él, uno o más nodos.
* **Nodos hermanos**: Nodos que están en el mismo nivel dentro del árbol DOM (por ejemplo, los nodos de primer nivel del body)
* **Nodo texto**: Un nodo que contiene texto dentro.

Ya, más o menos, sabrías esto después de haber usado estos conceptos dentro del HTML e incluso del CSS.

Imagina que tenemos la siguiente estructura HTML.

1. <!DOCTYPE html>
2. **<html>**
3. **<head>**
4. **<meta** charset**=**"utf-8"**>**
5. **<title>**Manipulando el DOM**</title>**
6. **</head>**
7. **<body>**
8. **<a** href**=**”www.elmundo.com”**>**Esto es un enlace al periódico EL MUNDO**</a>**
9. **</body>**
10. **</html>**

E incluimos esto después del **<a href…>**:

1. **<script>**
2. var enlace = document.querySelector('a');
3. **</script>**

Tendremos dentro de enlace una instancia de un objeto **HTMLAnchorElement**, que hereda de **HTMLElement**y de **Node**, a su vez. La clase Node tiene una propiedad llamada textContent que nos permite definir precísamente el contenido del nodo de texto que, en este caso, está dentro del **HTMLAnchorElement**.

Vamos también a cambiar el enlace (URL) ya que el periódico El Mundo tiene un dominio **.es** y no .**com**. Ahora atacaremos la propiedad href del objeto **HTMLAnchorElement**.

1. **<script>**
2. var enlace = document.querySelector('a');
3. enlace.textContent = “Aquí. ELMUNDO.es”;
4. enlace.href = “http://www.elmundo.es”;
5. **</script>**

**document.querySelector** es la forma moderna de realizar búsquedas para encontrar nodos HTML, ya que permite búsquedas al estilo CSS.

Por ejemplo, podemos buscar elementos de las siguientes maneras, siguiendo el estilo CSS:

1. **<p** class**=**"parrafo-ejemplo"**>**Esto es un párrafo de ejemplo**</p>**
2. **<div** id**=**"contenedor"**>**
3. **<img** src**=**"https://via.placeholder.com/350x150" width**=**"350" height**=**"150"**>**
4. **<img** src**=**"https://via.placeholder.com/450x250" width**=**"450" height**=**"250"**>**
5. **</div>**
6. var parrafo = document.querySelector**(**".parrafo-ejemplo"**)**;
7. var contenedor = document.querySelector**(**"#contenedor"**)**;
8. var primeraImagen = document.querySelector**(**"#contenedor img:first-child"**)**;
9. var segundaImagen = document.querySelector**(**"img + img"**)**;
10. parrafo.style.border = "1px solid #F00";
11. contenedor.style.border = "3px solid blue";
12. primeraImagen.style.border = "2px solid green";
13. segundaImagen.style.border = "2px solid green";

**document.querySelectorAll**tiene la misma utilidad pero para recibir colecciones de elementos que hagan match con el selector propuesto dentro de sus parámetros.

Estas son las sustitutas de los famosos: **getElementById, getElementsByTagName**.

La función **createElement** del objeto Document es la encarga de hacer esta acción. Para ello, le pasamos el nombre de la etiqueta a crear (ojo, que no todos los nodos se crean con esta función).

1. **<div** id**=**"targetDiv"**>**Hola**</div>**
2. **#targetDiv {**
3. width: 200px;
4. height:100px;
5. background-color: #fabada;
6. padding: 20px;
7. **}**
8. **#targetDiv p {**
9. font-weight: bold;
10. **}**
11. var divObjetivo = document.querySelector**(**'#targetDiv'**)**;
12. var pHijo = document.createElement**(**"p"**)**;
13. pHijo.textContent = "Ejemplo de creación de un p";
14. divObjetivo.appendChild**(**pHijo**)**;

# **Una vez que conseguimos la referencia al nodo, o bien le creamos desde cero, podemos tener acceso a sus estilos y atributos.**

1. nodoRef.style.color = 'white';
2. nodoRef.style.backgroundColor = 'black';
3. nodoRef.style.padding = '10px';
4. nodoRef.style.width = '250px';
5. nodoRef.style.textAlign = 'center';

Todos los elementos que pertenecen a HTMLElement tienen la propiedad **style**, que es del tipo **Document.stylesheets** y que devuelve un array **CSSStyleSheet**.

Como sabemos. Esto es útil para las veces que lo es. No siempre es la mejor forma de añadir estilos a un elemento. Sí, en los casos que ya te cité en el otro módulo de CSS.

Para hacerlo más elegante, procederíamos añadiendo una clase CSS al elemento en cuestión que, obviamente, estaría registrada previamente por alguno de los métodos que ya conocemos.

1. **nodoRef.setAttribute('class', 'nombre\_de\_la\_regla');**
2. **// aunque yo prefiero**
3. **nodoRef.classList.add("nombre\_de\_la\_regla");**
4. **// aunque también se puede hacer**
5. **nodoRef.className = "nombre\_de\_la\_regla";**
6. // aunque... ya es suficiente!! <img draggable="false" class="emoji" alt="" src="https://s.w.org/images/core/emoji/12.0.0-1/svg/1f642.svg">

Para este nivel, demuéstrame que sabes hasta aquí. Por lo que te voy a proponer una práctica antes de la última lección.

# **Tenemos una lista de películas dentro de un objeto JSON. De la siguiente forma:**

1. var pelis = **[**
2. **{**
3. “Nombre”: “El señor de los anillos”,
4. “Genero”: “Acción”
5. **}**,
6. **{**
7. “Nombre”: “Regreso al Futuro”,
8. “Genero”: “Acción”
9. **}**,
10. **{**
11. “Nombre”: “Los increibles 2”,
12. “Genero”: “Animación”
13. **}**
14. **]**;

JSON:

<https://www.digitalocean.com/community/tutorials/how-to-work-with-json-in-javascript>

1. var btn = document.querySelector**(**'button'**)**;
2. function random**(**number**)** **{**
3. **return** Math.floor**(**Math.random**()**\***(**number+1**))**;
4. **}**
5. btn.onclick = function**()** **{**
6. var rndCol = 'rgb(' + random**(**255**)** + ',' + random**(**255**)** + ',' + random**(**255**)** + ')';
7. document.body.style.backgroundColor = rndCol;
8. **}**

Con este pequeño snippet cambiamos el color de fondo del botón que seleccionamos. Vemos los pasos:

1. Seleccionamos la referencia al botón
2. Creamos un función que devuelve un número aleatorio (lo vamos a usar para crear el color a través de sus componentes RGB)
3. Nos enganchamos al evento “click” del botón.
4. Definimos el callback del evento
5. Asignamos el color de fondo tal y como hemos aprendido antes.

Básicamente, para poder “engancharnos” a los eventos, necesitamos conocer el identificador del evento (que en el fondo es una cadena de identificación) y definir nuestro **callback** de acción.

Hay otras acciones que -evidentemente- también se suelen hacer como consultar datos del evento, saber quién es el target, etc. Pero, en este nivel, nos vale con saber lo básico.

Otra lista pequeña de eventos podría ser:

* **Blur:** cuando un elemento pierde el foco
* **Focus:** cuando recibe el foco
* **Dblclick:** cuando se realiza doble click
* **Keypress:** cuando se presiona una tecla
* **Keydown:** cuando se presiona sin soltar
* **Keyup:** cuando se levanta la tecla
* **Mouseover:** cuando entra el puntero del ratón en un área
* **Mouseout:** cuando sale el puntero del ratón de un área

Para ser más genéricos, conociendo la cadena que se va a reconocer como identificador del evento, podemos utilizar siempre la función **addEventListener**. Aunque existe la posibilidad de crear manejadores de eventos llamados “inline”. Te va sonando todo esto, ¿verdad?.

Vamos a realizar dos ejemplos de todo lo comentado en el párrafo anterior:

1. var btn = document.querySelector**(**'button'**)**;
2. function bgChange**()** **{**
3. var rndCol = 'rgb(' + random**(**255**)** + ',' + random**(**255**)** + ',' + random**(**255**)** + ')';
4. document.body.style.backgroundColor = rndCol;
5. **}**
6. btn.addEventListener**(**'click', bgChange**)**;

**Ejemplo 1.** Con la función addEventListener

Como puedes comprobar todo es muy similar a lo que ya hemos visto. La excepción es que usamos para la referencia del botón la función comentada. Nota que ya no existe el prefijo “on” por delante. Simplemente ponemos el nombre del evento: “*click*”.

1. ...
2. **<button** onclick**=**"bgChange()"**>**Press me**</button>**
3. ...
4. **<script>**
5. function bgChange() {
6. var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';
7. document.body.style.backgroundColor = rndCol;
8. }
9. **</script>**

**Ejemplo 2**. Listener añadido de forma “*inline*”

Aquí lo mismo. Usamos el atributo onclick que, por debajo, se convertirá en una llamada a la función del objeto que realiza la asignación del listener. Le pasamos como valor el nombre de la función callback que atiende al evento.

Existe una operación, evidentemente, de borrado del listener. Esto tiene sentido si en un momento dado, de forma dinámica, queremos dejar de escuchar ciertos eventos, en ciertos controles.

Para ello existe **removeEventListener**. En nuestro caso:

1. btn.removeEventListener**(**'click', bgChange**)**;

**Ejemplo 3.** Manera de eliminar un listener concreto. En este caso, del control referenciado por “*btn*”.

Hay ciertos momentos en los que verás que dentro de un manejador de evento (callback function que vengo llamando desde hace unas líneas) aparecerá un parámetro. Este parámetro es el objeto **event**. Este objeto, como ya comenté, tiene información extra que nos puede resultar de interés.

Ahora nos vamos a centrar únicamente en el atributo *target*. Este atributo nos va a identificar el control afectado por dicho evento.

Por ejemplo:

1. function bgChange**(**e**)** **{**
2. var rndCol = 'rgb(' + random**(**255**)** + ',' + random**(**255**)** + ',' + random**(**255**)** + ')';
3. e.target.style.backgroundColor = rndCol;
4. console.log**(**e**)**;
5. **}**
6. btn.addEventListener**(**'click', bgChange**)**;

**Ejemplo 4.** El atributo target del objeto Event.

Aquí estamos viendo cómo identificamos quién está afectado por el evento (***e.target***). Una vez que poseemos dicha referencia, cambiamos su **backgroundColor** al estilo que ya conocemos y, claro está, siempre en respuesta al evento “*click*” en este caso.

# **Eventos y la fase de la burbuja**

## La burbuja, que no la inmobiliaria que es más preocupante, es un término que sirve de símil para explicar la propagación de eventos dentro de los controles.

Vamos a ver un ejemplo que clarificará todo esto. Tenemos un contenedor y un contenido. Ambos se suscriben a un evento “*click*”.

Si pulsamos el contenedor, no hay problema. El **callback** que responde, lo hará y punto. Pero, ¿qué pasa si pulsamos el contenido?. Aparentemente, debería funcionar igual (si es así como nos interesa). Pero no. El evento sube hacia su nodo padre. Si este tiene un evento “click” asociado, también lo ejecuta (y tiene sentido que lo haga, si te das cuenta). Y así, sucesivamente, hasta que llega al nodo root (¡qué gustazo hablar con términos que ya entiendes!).

1. ...
2. **<div** class**=**"contenedor"**>**
3. **<p** class**=**"contenido"**>**
4. Esto es un ejemplo de contenido
5. **</p>**
6. **</div>**
7. **<script>**
8. var contenedor = document.querySelector('.contenedor');
9. var contenido = document.querySelector('.contenido');
10. contenedor.addEventListener('click', function () {
11. alert("Hola. Has hecho click en el contenedor");
12. });
13. contenido.addEventListener('click', function (e) {
14. alert("Hola. Has hecho click en el contenido");
15. });
16. **</script>**
17. ...

**Ejemplo 7.** Propagación de eventos hacia el top de nodos

Cuando hacemos “click” en contenedor (div), el mensaje que se ejecuta es el suyo: «*Hola. Has hecho click en el contenedor*«.

Cuando hacemos “click”, por el contrario, en el contenido (p), se ejecutan ambos mensajes:

* «*Hola. Has hecho click en el contenedor*«.
* «*Hola. Has hecho click en el contenido*«.

Para solucionar esto, si es así como deseas que se comporte, has de añadir lo siguiente para evitar que el evento se propague hacia arriba:

1. contenido.addEventListener**(**'click', function **(**e**)** **{**
2. alert**(**"Hola. Has hecho click en el contenido"**)**;
3. e.stopPropagation**()**;
4. **})**;

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>