

---

# OBJETOS DE JAVASCRIPT

En JavaScript existen varios tipos de objetos, desde objetos de tipo lista de alto nivel hasta objetos complejos formados por propiedades y métodos, pero en general, la inmensa mayoría de los objetos heredan del objeto **Object**.

Una de las peculiaridades que presenta JavaScript es que todos los objetos tienen una propiedad **prototype** que mantiene un vínculo al objeto que le prototipó que, a su vez, tiene su propio prototipo y así sucesivamente. A esta idea o concepto se la suele denominar cadena de prototipos o modelo de prototipos.

Muchos programadores consideran que el modelo de prototipos es una de sus principales debilidades, sin embargo, este modelo es mucho más potente de lo que, a simple vista parece.

## 9.1 TIPOS DE OBJETO

---

- **Predefinidos:** son los que proporciona el lenguaje. Ejemplos de ello pueden ser el objeto **Date** para la gestión de fechas, **Math** para realizar operaciones matemáticas o **RegExp** para trabajar con expresiones regulares.
- **Personalizables:** son aquellos que permiten la declaración de **funciones**, **clases**, **objetos** o, también, la adición de otros objetos para implementar nuevas características.
- **Arrays:** son aquellos que permiten crear conjuntos de elementos a modo de matriz o lista de alto nivel.
- **JSON:** son aquellos que permiten crear conjuntos de elementos jerarquizables y que, en ocasiones, pueden proveer de métodos para trabajar en diferentes ámbitos o contextos.
- **Especiales:** son aquellos que han sido diseñados para tener una funcionalidad específica. Entre los más utilizados están **this** y **prototype**.

## 9.2 PROPIEDADES

Un objeto tiene, esencialmente, tres propiedades:

Propiedad	Descripción
<b>constructor</b>	Devuelve la función constructora nativa.
<b>length</b>	Devuelve la longitud del objeto, normalmente 1.
<b>prototype</b>	Permite añadir nuevas propiedades y métodos al objeto.

## 9.3 MÉTODOS

El número de métodos disponibles para los objetos es elevado y cambia en función de quién herede, sin embargo, existen algunos que son comunes a todo objeto. A continuación, se muestran los más frecuentes:

Método	Descripción y ejemplo
<b>assign</b>	<p>Copia el objeto referenciado por el segundo parámetro en el objeto referenciado por el primer parámetro y lo devuelve. Si el objeto es un elemento HTML, este método sólo copia el objeto o elemento en sí, no copia sus eventos asociados.</p> <pre>let b = {}; Object.assign(b, {a:1, b:2}); // Devuelve {a:1, b:2}</pre>
<b>create</b>	<p>Este método es otra forma de llamar al constructor de la clase.</p> <pre>Object.create({}); // Devolverá {}</pre>
<b>entries</b>	<p>Devuelve un array multidimensional que contiene los pares de clave valor en cada array subyacente. Esto es útil para poder iterar un objeto que, a primera vista, no parece iterable.</p> <pre>let json = {"name": "Pablo", "surname": "Fernández", "age": 18}; Object.entries(json); // Devuelve lo siguiente: [   0: ["name", "Pablo"]   1: ["surname", "Fernández"]   2: ["age", 18]   length: 3   ► __proto__: Array(0) ]</pre>

<b>getOwnPropertyDescriptor</b>	<p>Devuelve un objeto con toda la descripción de la propiedad que se envía como segundo parámetro. Las propiedades que se muestran son: el valor, si es enumerable, si es escribible y si es configurable.</p> <pre>Object.getOwnPropertyDescriptor(json, "surname"); // Devuelve un JSON con lo siguiente: {   value: "Fernández",   writable: true,   enumerable: true,   configurable: true, }</pre>
<b>getOwnPropertyDescriptors</b>	<p>Devuelve un objeto con la descripción de todas las propiedades que posee el objeto. Las propiedades que se muestran son las mismas que las devueltas por <code>getOwnPropertyDescriptor</code>.</p>
<b>getOwnPropertyNames</b>	<p>Devuelve un array con los nombres de las claves enumerables y no enumerables del objeto proporcionado por parámetro.</p> <pre>Object.getOwnPropertyNames(json); // Devuelve un array como ["name", "surname", "age"]</pre>
<b>hasOwnProperty</b>	<p>Devuelve un booleano si el objeto contiene la propiedad proporcionada por parámetro.</p> <pre>console.log(json.hasOwnProperty("name")); // Devuelve true</pre>
<b>keys</b>	<p>Devuelve un array con los nombres de las claves enumerables del objeto proporcionado por parámetro.</p> <pre>Object.keys(json); // Devuelve ["name", "surname", "age"]</pre>
<b>values</b>	<p>Devuelve un array con los valores de las claves enumerables del objeto proporcionado por parámetro.</p> <pre>Object.values(json); // Devuelve ["Pablo", "Fernández", 18]</pre>

## 9.4 ARRAYS

Los arrays, comúnmente, son un conjunto de elementos que se guardan en memoria de forma secuencial y que pueden ser accedidos a través de valores enteros o Strings denominados índices.

En JavaScript, sin embargo, un array no es precisamente esto, sino que, más bien, es un tipo objeto que tiene propiedades que hace que se asemeje a un array, y al que se le ha dotado de algunas características para que se maneje como si lo fuese.

A modo de curiosidad, aunque en la consola del navegador veamos índices que parecen números enteros, en realidad, internamente, se están convirtiendo a **String** y utilizados a modo de identificador de propiedad.

### 9.4.1 Creación de arrays

Una forma de crear o definir una matriz (o array) es, o a través de su constructor, o a través de unos corchetes:

```
let arr = new Array();  
let arr = [];
```

### 9.4.2 Acceso a elementos de un array

Para acceder a sus elementos podemos hacerlo a través del índice entre corchetes:

```
let arr = [1, 1, 2, 3, 5, 8];  
console.log(arr[0]); // Devuelve 1
```

El objeto **Array** es una estructura que empieza con el índice CERO, por tanto, para acceder a su último elemento deberemos acceder a su longitud menos uno.

El tipo de datos **String**, internamente, también es considerado como un array, por lo que podemos acceder a una posición concreta de una cadena como si fuese un array y recuperar el carácter que se encuentra en esa posición.

### 9.4.3 Inserción y almacenamiento de elementos en un array

Para almacenar un nuevo elemento podemos hacerlo a través del índice entre corchetes o utilizar el método **push**:

```
let arr = [1, 1, 2, 3, 5, 8];  
arr[6] = 13;  
arr.push(21);  
console.log(arr[7]); // Devuelve 21
```

Sin embargo, no es posible realizar inserciones directas en los arrays de un array. Esto es porque no se pueden insertar valores en los objetos que no hayan sido, previamente, definidos. Por ejemplo, si intentásemos insertar un valor en un elemento de un array no definido, pero que se supone que está dentro de otro array ya definido, nos saltaría un error de propiedad no definida. Para solucionarlo se debe definir el array antes de insertar el valor:

```
let arr = new Array();  
arr[0][0] = 1; // Devuelve un error de tipo  
  
let arr[0] = new Array();  
arr[0][0] = 1; // Ahora sí lo inserta
```



## 9.4.4 Eliminación de elementos de un array

Para eliminar un elemento de un objeto array podemos utilizar dos sentencias o instrucciones.

La sentencia **delete** no elimina realmente el dato, sino que establece su valor a vacío, pero conserva el espacio reservado en memoria.

```
let arr = [1, 1, 2, 3, 5, 8];
delete(arr[4]);

console.log(arr); // Devolverá [1, 1, 2, 3, empty, 8]
```

La sentencia **splice**, sin embargo, sí que elimina y reemplaza el elemento del array.

El método splice se alimenta de dos parámetros. El primer parámetro, indica la posición dónde empezar a eliminar. El segundo, indica cuantos elementos se deben eliminar desde la posición indicada por el primer parámetro.

```
let arr = [1, 1, 2, 3, 5, 8];
arr.splice(4, 1);

console.log(arr); // Devolverá [1, 1, 2, 3, 8]
```

## 9.4.5 Propiedades

Como buen objeto de JavaScript, el objeto Array tiene las típicas propiedades **CONSTRUCTOR**, **LENGTH** y **PROTOTYPE**, anteriormente comentadas.

## 9.4.6 Métodos

El número de métodos disponibles para los arrays es elevado. Por ello, a continuación, se muestran los más frecuentes:

Método	Descripción y ejemplo
<b>filter</b>	Devuelve un nuevo array que contiene los resultados que cumplen con la función indicada que se pasa como parámetro. <pre>let frutas = ["Manzana", "Kiwi", "Platano", "Pera"]; frutas.filter(function(x){ return x.length &gt; 4}); // Devuelve el array ["Manzana", "Platano"]</pre>
<b>join</b>	Devuelve una cadena que es el resultado de unir todos los elementos por el separador pasado por parámetro. Por defecto, el separador es el símbolo de coma. <pre>[1, 2, 3, 4, 5].join(" "); // Devuelve "1 2 3 4 5"</pre>

<b>indexOf</b>	<p>Devuelve la primera posición en la que aparezca el valor proporcionado por parámetro teniendo en cuenta que es sensible a mayúsculas y minúsculas. Si el resultado de la búsqueda fue infructuoso, el resultado será -1. Tiene un segundo parámetro opcional que indica desde qué posición empieza a buscar y que, por defecto, es 0.</p> <pre>[“Pablo”, “Elena”, “Pablo”].indexOf(“Pablo”); // Devuelve 0</pre>
<b>lastIndexOf</b>	<p>Devuelve la última posición en la que aparezca el valor proporcionado por parámetro teniendo en cuenta que es sensible a mayúsculas y minúsculas. Si el resultado de la búsqueda fue infructuoso, el resultado será -1. Tiene un segundo parámetro opcional que indica desde qué posición empieza a buscar y que, por defecto, es la longitud del array.</p> <pre>[“Pablo”, “Elena”, “Pablo”].lastIndexOf(“Pablo”); // Devuelve 2</pre>
<b>map</b>	<p>Devuelve un nuevo array que contiene los resultados tras haber sido tratados por la función que se proporciona por parámetro.</p> <pre>[1, 2, 3, 4, 5].map(function(x){ return x * x }); // Devuelve el array [1, 4, 9, 16, 25]</pre>
<b>pop</b>	<p>Elimina y devuelve el último elemento del array.</p> <pre>[1, 2, 3, 4, 5].pop(); // Devuelve 5 y deja el arr con los valores [1, 2, 3, 4]</pre>
<b>push</b>	<p>Añade el elemento al final del objeto y devuelve el array resultante.</p> <pre>[1, 2, 3, 4, 5].push(6); // Devuelve el array [1, 2, 3, 4, 5, 6]</pre>
<b>reverse</b>	<p>Devuelve el array en orden inverso.</p> <pre>[1, 2, 3, 4, 5].reverse(); // Devuelve el array [5, 4, 3, 2, 1]</pre>
<b>reduce</b>	<p>Permite realizar operaciones con arrays. Normalmente recibe dos parámetros (acumulador y elemento actual) y opera con el elemento sobre el acumulador. El resultado de la operación se devuelve en el acumulador.</p> <pre>[1, 2, 3, 4].reduce(function(acumulador, elemento){ return acumulador + elemento; }); // Devuelve 10</pre>
<b>shift</b>	<p>Elimina y devuelve el primer elemento del array.</p> <pre>let aux = [1, 2, 3]; aux.shift(); // Devuelve 1 y deja aux como [2, 3]</pre>

<b>sort</b>	<p>Devuelve un array unidimensional ordenado por valor. Antes de hacer la ordenación se hace una conversión a String, por lo que se ordena en formato de códigos Unicode, es decir, que el valor 80 está antes que el 9 y manzana está antes que plátano.</p> <pre>["Pablo", "Elena", "Adrian", "ana"].sort(); // Devuelve el array ['Adrian', 'Elena', 'Pablo', 'ana']</pre>
<b>slice</b>	<p>Devuelve un nuevo array que contiene el número de elementos proporcionado como segundo parámetro, empezando por la posición pasada como primer parámetro.</p> <pre>let aux = [3, 40, 200, 5, 1]; aux.slice(0,3); // Devuelve [3, 40, 200] y aux mantiene todos sus valores</pre>
<b>splice</b>	<p>Devuelve un nuevo array que contiene el número de elementos proporcionado como segundo parámetro, empezando por la posición pasada como primer parámetro. La diferencia con el método slice es que, splice, elimina del array original los elementos contenidos del array devuelto.</p> <pre>let aux = [3, 40, 200, 5, 1]; aux.splice(0,3); // Devuelve [3, 40, 200] y aux se queda con [5, 1]</pre>
<b>unshift</b>	<p>Añade, al principio del array, los elementos proporcionados por parámetro.</p> <pre>let aux = [1, 2, 3]; aux.unshift(4, 5); // Devuelve 5 y aux se queda como [4, 5, 1, 2, 3]</pre>

## 9.5 JSON

Según Wikipedia, JSON es un acrónimo de **JavaScript Object Notation** y resulta un formato de texto sencillo y ligero para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplio uso como alternativa al XML, se ha considerado un formato independiente del lenguaje.

Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos es que resulta mucho más sencillo desarrollar un analizador sintáctico, lo que se suele conocer como **parser**. En JavaScript, un objeto JSON puede ser analizado fácilmente usando la función **eval**, algo que (debido a la ubicuidad de JavaScript en casi cualquier navegador web) ha sido fundamental para que haya sido aceptado por parte de la comunidad de desarrolladores AJAX.

Resulta muy frecuente utilizar JSON en entornos donde el tamaño del flujo de datos entre cliente y servidor es de vital importancia (de aquí que sea utilizado por grandes compañías como Yahoo!, Google o Mozilla cuando la fuente de datos es de confianza y

donde no es importante el hecho de no disponer de procesamiento XSLT para manipular los datos en el cliente).

A modo de apunte introductorio final diremos que, si bien se tiende a considerar JSON como una alternativa a XML, lo cierto es que es frecuente el uso combinado de JSON y XML en algunas aplicaciones, como es el caso del servicio de Google Maps.

### 9.5.1 Sintaxis

Los tipos de datos disponibles en JSON puede ser valores numéricos, con el punto como separador de decimales, cadenas de texto entrecomilladas, valores booleanos true o false, valores nulos o arrays que suelen contener otros JSON.

```
[
{ "abbreviation": "Ene", "name": "Enero", "days": 31 },
{ "abbreviation": "Feb", "name": "Febrero", "days": 28 },
{ "abbreviation": "Mar", "name": "Marzo", "days": 31 },
{ "abbreviation": "Abr", "name": "Abril", "days": 30 },
{ "abbreviation": "May", "name": "Mayo", "days": 31 },
{ "abbreviation": "Jun", "name": "Junio", "days": 30 },
{ "abbreviation": "Jul", "name": "Julio", "days": 31 },
{ "abbreviation": "Ago", "name": "Agosto", "days": 31 },
{ "abbreviation": "Sep", "name": "Septiembre", "days": 30 },
{ "abbreviation": "Oct", "name": "Octubre", "days": 31 },
{ "abbreviation": "Nov", "name": "Noviembre", "days": 30 },
{ "abbreviation": "Dic", "name": "Diciembre", "days": 31 },
]
```

### 9.5.2 Creación de JSON

La forma más frecuente de crear o definir un objeto JSON es a través de dos llaves:

```
// Declaración de un JSON vacío
let persona = {};

// Declaración de un JSON con datos de una persona
let persona = {
  nombre: 'Pablo',
  apellido: 'Fernández',
  estatura: 1.60,
  edad: 18,
  trabaja: true
}
```

Sin embargo, también es posible crearlo a través de su constructor:

```
let persona = JSON.constructor();
persona.nombre = 'Pablo';
persona.apellido = 'Fernández';
persona.estatura = 1.60;
persona.edad = 18;
```



### 9.5.3 Acceso a elementos de un JSON

Para acceder a sus elementos podemos hacerlo a través de sus propiedades en formato objeto o en formato array:

```
console.log(persona['nombre']); // Devuelve 'Pablo'
console.log(persona.edad); // Devuelve 18
```

Al igual que pasa con el objeto **Array**, si el JSON está formado por un array de JSON su índice inicial será CERO, por tanto, para acceder a su último elemento deberemos acceder a su longitud menos uno.

### 9.5.4 Inserción y almacenamiento de elementos en un JSON

Para almacenar una nueva propiedad a un JSON, también podemos hacerlo en formato objeto o en formato array:

```
persona.talla = "5";
persona['nombre'] = "Elena";
```

### 9.5.5 Eliminación de elementos de un JSON

Para eliminar un elemento de un objeto array podemos utilizar la instrucción **delete** que elimina y reemplaza el elemento en el objeto donde se aplica.

```
delete(persona.talla);
console.log(persona);

// Devuelve lo siguiente:
{
  apellido: "Fernández"
  edad: 18
  estatura: 1.6
  nombre: "Elena"
  trabaja: true
  ► __proto__: Object
}
```

### 9.5.6 Envío y recepción de JSON

Los JSON son unos objetos con los que trabajamos de manera muy frecuente. Tanto es así, que incluso se les utiliza para enviar y recibir información al servidor o comunicarse con APIs, entre otras muchas posibilidades.

Dado que un JSON es también un objeto en JavaScript, este, hereda las propiedades y métodos propios del objeto como son el **constructor**, **hasOwnProperty** o **toString**.

Lo más normal es que, cuando se desea enviar información al servidor, el objeto que contiene esa información, sea transformado a un tipo String para que luego, en el servidor, pueda ser reconstruida y manipulada.

En lo referente a la recepción, lo habitual, es que el objeto que contiene esa información, venga en formato String. Sin embargo, puede ser convertido a un tipo concreto de objeto nada más ser recibido y, así, poder manipular dicha información.

Cuando se trata de JSON, en el proceso de envío y recepción, lo que se suele hacer es recurrir a los métodos **stringify** y **parse**.

#### 9.5.6.1 MÉTODO STRINGIFY

La sentencia **stringify** convierte un objeto analizable en una cadena de texto de tipo JSON.

Por ejemplo, uno de los usos más frecuentes de esta instrucción es utilizarla para almacenar datos en la memoria intermedia o para transferir datos entre el cliente y el servidor.

```
let objeto = {
  texto: 'valor',
  digito: 1
}

JSON.stringify(objeto); // Devolverá '{"texto":"valor","digito":1}'
```

Si el JSON está mal formado, en el método **stringify**, se provocará un error al intentar convertirlo a formato texto de JSON y devolverá un error de “Error de sintaxis”.

#### 9.5.6.2 MÉTODO PARSE

La sentencia **parse** convierte una cadena de texto de tipo JSON en un objeto analizable por JavaScript.

Por ejemplo, uno de los usos más frecuentes de esta instrucción es utilizarla para recuperar datos de la memoria intermedia o del servidor y, así, poder validar alguna propiedad.

```
let cadena = '{"texto":"valor","digito":1}';

let objeto = JSON.parse(cadena);
// Devuelve un String como:
{
  texto: "valor"
  digito: 1
  ► __proto__: Object
}

console.log(objeto.texto); // Devolverá "valor"
```

Si el JSON está mal formado, en el método **parse**, se provocará un error al intentar convertirlo a formato analizable y devolverá un error de “Error de sintaxis”.

## 9.6 ESPECIALES

En JavaScript hay objetos que se consideran especiales, bien porque tienen una funcionalidad muy concreta, bien porque son un enlace o vínculo con otros. Aquí vamos a describir algunos de ellos y que se utilizan durante el recorrido del libro.

### 9.6.1 El objeto window

El objeto window suele denominársele el objeto global ya que es la forma de referenciar a la ventana del navegador.

Cada vez que se accede a una página, el navegador crea un objeto window que, entre otras propiedades y métodos, contiene un objeto document con la información de la página solicitada.

El objeto window es único para cada pestaña del navegador, es decir, que los cambios que se puedan producir sobre el objeto window no se ven reflejados entre pestañas. En lo referente a marcos o frames el comportamiento del objeto window es idéntico al comportamiento entre pestañas, es decir, que también son independientes.

#### 9.6.1.1 MÉTODOS

Muchos de los métodos que utilizamos normalmente pertenecen al objeto window, sin embargo, aunque pertenecen a este, la mayoría de las veces no requieren que se ejecuten o llamen a través de él. Véase por ejemplo el caso de **console**.

#### 9.6.1.2 PROPIEDADES

Si nos ponemos a hablar sobre sus propiedades, la lista podría ser interminable. Por ello, aquí sólo describiremos las más frecuentes.

Propiedad	Descripción
<b>console</b>	Proporciona acceso a la consola del navegador. Este objeto, entre otras cosas, nos permite lanzar mensajes a la consola del navegador como, por ejemplo: <pre>console.log(objeto); // Mensaje sin estado console.warn(objeto); // Mensaje de advertencia console.error(objeto); // Mensaje de error</pre>
<b>innerHeight</b>	Devuelve el alto en píxeles de la ventana.
<b>innerWidth</b>	Devuelve el ancho en píxeles de la ventana.
<b>length</b>	Devuelve el número total de marcos ( <b>frames</b> o <b>iframes</b> ) que tiene la ventana.

<b>localStorage</b>	Provee acceso para gestionar el almacenamiento de datos permanentes, sin fecha de caducidad, aunque se cierre la pestaña o el navegador.
<b>opener</b>	Proporciona acceso a la ventana que fue abierta desde la ventana actual. Sólo es accesible cuando se realiza a través de <b>window.open</b> , en caso contrario, devuelve <b>null</b> .
<b>outerHeight</b>	Devuelve el alto en píxeles de la ventana incluyendo la barra de notificaciones y los bordes, si los hubiese.
<b>outerWidth</b>	Devuelve el ancho en píxeles de la ventana incluyendo la barra de notificaciones y los bordes, si los hubiese.
<b>pageXOffset, scrollX</b>	Devuelven la posición en píxeles del scroll horizontal, es decir, el valor del desplazamiento en píxeles de la barra de desplazamiento horizontal.
<b>pageYOffset, scrollY</b>	Devuelven la posición en píxeles del scroll horizontal, es decir, el valor del desplazamiento en píxeles de la barra de desplazamiento horizontal.
<b>screen</b>	Proporciona acceso a la interfaz <b>Screen</b> que provee toda la información disponible sobre el dispositivo dónde se muestra el documento. Entre sus propiedades podemos encontrar el alto y ancho en píxeles de la pantalla, la profundidad en bits de color y la orientación.
<b>screenX</b>	Devuelve la posición horizontal de la ventana en relación con el ancho de la pantalla.
<b>screenY</b>	Devuelve la posición vertical de la ventana en relación con el ancho de la pantalla.
<b>sessionStorage</b>	Provee acceso para gestionar el almacenamiento de datos temporales, que serán eliminados cuando se cierre la pestaña o el navegador.
<b>top</b>	Proporciona acceso al objeto <b>window</b> , marco o iframe superior de la ventana. Si no hay ningún nivel superior devolverá el objeto <b>window</b> actual.

## 9.6.2 El objeto document

El objeto **document** es quién representa a la página actualmente cargada. Dicho de otra manera, es quién proporciona acceso al DOM y describe los métodos y propiedades para poder manejar cualquier tipo de documento, sea del formato que sea (HTML, XHTML, SVG, ...).

Desde aquí, podemos acceder y manipular todo el DOM, como se verá más adelante.



### 9.6.2.1 MÉTODOS

El objeto `document` dispone de muchos métodos. Debido a ello, es mejor que se conozcan poco a poco según vayan surgiendo las necesidades. En este libro, veremos varios de ellos, pero si surgen dudas, lo mejor siempre seguirá siendo consultar la documentación oficial desde alguna fuente fidedigna.

### 9.6.2.2 PROPIEDADES

Al igual que pasa con el objeto `window`, si nos ponemos a hablar sobre sus propiedades, la lista podría ser interminable. Por ello, aquí sólo describiremos las más frecuentes.

Propiedad	Descripción
<b>all</b>	Devuelve un objeto <b>HTMLCollection</b> similar a un array que proporciona acceso a todos los elementos HTML que conforman el documento.
<b>activeElement</b>	Devuelve el elemento que actualmente está activo o tiene el foco.
<b>body</b>	Devuelve el elemento que representa y contiene el cuerpo del documento, en otras palabras, el elemento que se corresponde con la etiqueta <b>BODY</b> de la página actual.
<b>characterSet</b>	Devuelve el juego de caracteres que se utiliza en el documento. Lo habitual será que contenga el valor UTF-8.
<b>cookie</b>	Devuelve una lista con los nombres de las cookies que están asignadas o utiliza el documento. Van separadas por el símbolo punto y coma.
<b>contentType</b>	Devuelve el tipo de contenido MIME que se utiliza en el documento. Lo habitual será que contenga el valor <b>text/html</b> o <b>multipart/form-data</b> .
<b>defaultView</b>	Devuelve el objeto <b>window</b> al que pertenece.
<b>designMode</b>	Devuelve o establece la capacidad de editar todo el documento. Lo habitual es que su valor esté establecido a <b>off</b> . Si se cambia a <b>on</b> , todo lo que en principio era de sólo lectura (como un <b>LABEL</b> o un <b>H1</b> ), ahora será editable.
<b>docType</b>	Devuelve el DTD o Definición del Tipo de Documento del documento actual. Lo habitual será que contenga el valor <b>&lt;!DOCTYPE html&gt;</b> .
<b>documentElement</b>	Devuelve el elemento que representa y contiene el documento, en otras palabras, el elemento que se corresponde con la etiqueta <b>HTML</b> de la página actual.
<b>documentURI</b>	Devuelve la URL del documento actual.

<b>forms</b>	Devuelve un objeto <b>HTMLCollection</b> que provee acceso a todos los formularios que están definidos en el documento.
<b>head</b>	Devuelve el elemento que representa y contiene la cabecera del documento, en otras palabras, el elemento que se corresponde con la etiqueta <b>HEAD</b> de la página actual.
<b>height</b>	Devuelve el alto en píxeles del documento.
<b>images</b>	Devuelve un objeto <b>HTMLCollection</b> que provee acceso a todas las imágenes que están definidas en el documento.
<b>links</b>	Devuelve un objeto <b>HTMLCollection</b> que provee acceso a todos los hipervínculos que están definidos en el documento.
<b>onreadystatechange</b>	Es el evento que nos permite controlar el <b>readyState</b> , es decir, los diferentes estados por los que pasa la página durante su carga. Esta parte se verá más adelante en detalle.
<b>styleSheets</b>	Devuelve un objeto <b>StyleSheetList</b> que representa y contiene todas las hojas de estilo y bloques <b>STYLE</b> incluidos en el documento.
<b>referrer</b>	Devuelve la URL de la página desde donde se entró. Si hacemos una búsqueda en Google y pinchamos en uno de los resultados, lo normal es que, esta propiedad tenga su valor establecido a <i>https://www.google.com/</i> .
<b>readyState</b>	Devuelve el estado actual de carga del documento. Los posibles valores por los que puede pasar son <b>loading</b> , que indica que está cargándose todavía, <b>Interactive</b> , que indica que se ha terminado de cargar y el DOM está accesible, pero todavía faltan imágenes, estilos o iframes que no se han cargado y <b>complete</b> , que indica que el documento está totalmente cargado. Cuando <b>readyState</b> entra en el estado <b>Interactive</b> , el evento <b>DOMContentLoaded</b> se dispara automáticamente y cuando el documento entra en este estado, el evento <b>onload</b> se dispara automáticamente. El modo de conseguir los estados por los que pasa el documento es establecer un oyente o <b>listener</b> sobre el objeto <b>onreadystatechange</b> .
<b>scripts</b>	Devuelve un objeto <b>HTMLCollection</b> que provee acceso a todos los scripts que están definidos en el documento.
<b>width</b>	Devuelve el ancho en píxeles del documento.

### 9.6.3 El objeto Screen

El objeto **Screen** proporciona información sobre la pantalla del dispositivo en el que nos encontramos actualmente.

Las propiedades más importantes son:

Propiedad	Descripción
<b>availHeight</b>	Devuelve el alto disponible de la pantalla. Lo habitual es que no suela coincidir con la altura de la pantalla porque, esta propiedad, le resta la altura asignada a la barra de tareas de la interfaz gráfica del sistema (lo que viene siendo la barra de tareas de Windows, por ejemplo).
<b>availWidth</b>	Devuelve el ancho disponible de la pantalla. Lo habitual es que este valor coincida con el ancho de la pantalla.
<b>colorDeep</b>	Devuelve la profundidad de color en bits de la pantalla. Lo habitual es que contenga el valor 24.
<b>height</b>	Devuelve la altura en píxeles de la pantalla.
<b>orientation</b>	Devuelve un objeto <b>ScreenOrientation</b> que contiene la información acerca de la orientación de la pantalla. Entre las diferentes propiedades que se nos ofrece podemos encontrar la propiedad <b>angle</b> , que devuelve el ángulo de giro de la pantalla y la propiedad <b>type</b> , que muestra una serie de posibles valores que son <b>landscape-primary</b> , que indica que la pantalla está en posición horizontal natural, <b>landscape-secondary</b> , que indica que la pantalla está en posición horizontal natural girada 180 grados, <b>portrait-primary</b> , que indica que la pantalla está en posición vertical natural y <b>portrait-secondary</b> , que indica que la pantalla está en posición vertical natural girada 180 grados.
<b>width</b>	Devuelve el ancho en píxeles de la pantalla.

### 9.6.4 La interfaz Navigator

La interfaz **Navigator** es un objeto que contiene la información sobre el agente de usuario, es decir, la información sobre la identidad del usuario.

La interfaz **Navigator** permite consultar, entre otras muchas más cosas, el lenguaje seleccionado por el usuario, la plataforma que está utilizando, el sistema operativo y si admite cookies.



Las propiedades más importantes son:

Propiedad	Descripción
<b>appName</b>	Esta propiedad devuelve el nombre interno del navegador. Si se comprueba en diferentes navegadores, se puede observar que muchos, por no decir todos, tienen esta propiedad establecida a "Mozilla".
<b>appVersion</b>	Esta propiedad devuelve el nombre oficial del navegador. Si se comprueba en diferentes navegadores, se puede observar que muchos, por no decir todos, tienen esta propiedad establecida a "Netscape".
<b>connection</b>	Esta propiedad devuelve la versión del navegador. En varios navegadores, esta propiedad puede contener información adicional, como el sistema operativo.
<b>cookieEnabled</b>	Esta propiedad devuelve un booleano que indica si el navegador tiene habilitadas las cookies o no.
<b>hardwareConcurrency</b>	Si está disponible, esta propiedad devuelve el número de núcleos lógicos del procesador.
<b>language</b>	Esta propiedad devuelve el lenguaje que tiene seleccionado el navegador. El valor representado debe cumplir el estándar ISO 639-1.
<b>online</b>	Esta propiedad devuelve un booleano que indica si el navegador tiene acceso a la red, sea local o Internet.
<b>platform</b>	Esta propiedad devuelve la plataforma dónde se está ejecutando el navegador. Lo más frecuente es encontrar valores del tipo "Win32", "Win64", "MacIntel", ...
<b>productSub</b>	Si está disponible, esta propiedad devuelve el número de compilación del navegador.
<b>userAgent</b>	Esta propiedad devuelve un String que contiene la cadena que se corresponde con el agente de usuario. Lo habitual es que contenga mucha información, por lo que puede ser tratada para otros fines no "puramente identificativos".
<b>vendor</b>	Si está disponible, esta propiedad devuelve el fabricante del navegador.



## 9.6.5 La interfaz Location

La interfaz **Location** es un objeto que guarda toda la información referente a la URL actual. Los objetos **document** y **window** tienen, ambos, una propiedad que contiene los datos recuperados por la interfaz Location.

Las propiedades y métodos más importantes son:

Propiedad	Descripción
<b>href</b>	Contiene la URL completa.
<b>protocol</b>	Contiene el protocolo utilizado.
<b>hostname</b>	Contiene el dominio de la URL.
<b>port</b>	Contiene el puerto utilizado.
<b>pathname</b>	Contiene todo lo que no es el dominio.
<b>search</b>	Contiene los parámetros de la URL. Normalmente, le añade el símbolo de cierre de interrogación delante.
<b>reload</b>	Provoca la recarga de la página. <pre>location.reload();</pre>
<b>toString</b>	Devuelve la URL completa en formato String.

## 9.6.6 La interfaz HTMLElement

**HTMLElement** es una interfaz de JavaScript que representa a todos los elementos HTML del DOM. Hereda todas las propiedades del objeto **Element** e implementa las propiedades de los manejadores de eventos globales y táctiles. Además, es una interfaz muy útil cuando se desean añadir funcionalidades y/o sobrecargar métodos ya existentes.

A través de esta interfaz, se pueden leer y establecer la mayoría de los atributos que componen los elementos de HTML, así, por ejemplo, **HTMLElement** nos da acceso a atributos como los estilos del elemento (**style**), si está oculto (**hidden**), si es arrastrable (**draggable**), su orden de tabulación (**tabindex**), su combinación de teclado para acceder a él (**accesskey**), si su contenido es editable (**contentEditable**)... y así, casi una infinidad de propiedades.


Sirva como ejemplo que, gracias a esta interfaz, podríamos proveer a todos los elementos del DOM de un método que pueda eliminar nodos y elementos sin importar el navegador en el que nos encontramos, como se verá, más adelante, en el capítulo de “El DOM”.

### 9.6.6.1 PROPIEDADES

A continuación, se muestran las propiedades más importantes o más frecuentemente utilizadas.

Propiedad	Descripción
<b>accesskey</b>	Devuelve o establece la combinación de teclado asignada al elemento. Dependiendo del navegador, el establecimiento de esta propiedad puede variar.
<b>accesskeyLabel</b>	Devuelve el contenido de la combinación de teclado asignada al elemento.
<b>attributes</b>	Devuelve todos los atributos que presenta un elemento.
<b>childNodes</b>	Devuelve todos los nodos hijos que tiene un elemento.
<b>children</b>	Devuelve todos los elementos hijos que tiene un elemento.
<b>className</b>	Permite manipular las clases de un elemento.
<b>clientHeight</b>	Devuelve la altura interior en píxeles del elemento.
<b>clientWidth</b>	Devuelve el ancho interior en píxeles del elemento.
<b>contentEditable</b>	Permite que los elementos de sólo lectura que, en principio no deberían ser editables, puedan serlo. Sus posibles valores son <b>true</b> o <b>false</b> . Si, por ejemplo, establecemos esta propiedad a <b>true</b> en un DIV, todo su contenido será editable.
<b>dataset</b>	Devuelve un objeto DOMStringMap que da acceso a la creación y manipulación de los atributos personalizados de un elemento.
<b>draggable</b>	Devuelve un booleano que indica si el elemento es arrastrable o no.
<b>hidden</b>	Devuelve un booleano que indica si el elemento está oculto o no.
<b>id</b>	Devuelve o establece el identificador del elemento.
<b>innerHTML</b>	Devuelve el contenido de todos los nodos o elementos hijos del elemento seleccionado en formato HTML con codificación de caracteres. <b>Nota:</b> a codificación de caracteres significa que, el código, se devuelve en formato de entidad HTML, es decir, que convertirá los caracteres especiales como "<" en "&lt;" y ">" en "&gt;", entre otros.
<b>innerText</b>	Devuelve el contenido de todos los nodos o elementos hijos del elemento seleccionado en formato HTML sin codificación de caracteres. Es equivalente a utilizar <b>childNodes[0].nodeValue</b> . Nota: esta propiedad no devolverá los elementos que estén ocultos por CSS.
<b>isContentEditable</b>	Devuelve si el elemento es editable o no.
<b>lang</b>	Devuelve o establece el idioma asignado al elemento.

<b>localname</b>	Devuelve el nombre local del elemento. Se considera nombre local de un elemento a la parte de la etiqueta que está detrás del símbolo dos puntos, es decir, la que está detrás del prefijo. Si no tiene prefijo, es equivalente al valor que devuelve la propiedad <b>tagName</b> .
<b>name</b>	Devuelve o establece el nombre del elemento.
<b>namespaceURI</b>	Devuelve el espacio de nombres del elemento. Habitualmente devolverá <i>http://www.w3.org/1999/xhtml</i> si estamos trabajando con HTML5. Si no tiene ningún valor asignado, su valor será <b>null</b> .
<b>nextSibling</b>	Devuelve el nodo inmediatamente posterior. Si no tiene, su valor será <b>null</b> .
<b>nextElementSibling</b>	Devuelve el elemento inmediatamente posterior. Si no tiene, su valor será <b>null</b> .
<b>nodeName</b>	Devuelve el nombre del nodo. Si el nodo es un elemento HTML, el nombre coincidirá con el valor devuelto por la propiedad <b>tagName</b> .
<b>nodeType</b>	Devuelve un número entero que representa el tipo de nodo. En este caso, como todo son elementos, esta propiedad siempre contendrá el valor 1.
<b>nodeValue</b>	Devuelve o establece el valor del nodo. Esta propiedad será <b>null</b> para el propio elemento. Para los nodos de tipo texto, comentario o <b>CDATA</b> , devolverá el contenido del nodo. Si el nodo es un atributo, devolverá el valor del atributo.
<b>offsetHeight</b>	Devuelve la altura en píxeles del elemento.
<b>offsetLeft</b>	Devuelve la distancia en píxeles que hay entre el borde izquierdo del elemento padre y el borde izquierdo del elemento actual.
<b>offsetParent</b>	Devuelve el elemento padre desde dónde se calculan los valores de offset. Normalmente se corresponde con el elemento que tiene el último posicionamiento relativo ascendente más cercano.
<b>offsetTop</b>	Devuelve la distancia en píxeles que hay entre el borde superior del elemento padre y el borde superior del elemento actual.
<b>offsetWidth</b>	Devuelve la anchura en píxeles del elemento.
<b>outerHTML</b>	Devuelve el contenido del elemento seleccionado (incluyéndose a sí mismo, no sólo sus hijos) en formato HTML con codificación de caracteres.

 **NOTA**

La codificación de caracteres significa que, el código, se devuelve en formato de entidad HTML, es decir, que convertirá los caracteres especiales como "<" en "&lt;" y ">" en "&gt;", entre otros.




<b>ownerDocument</b>	Devuelve un objeto document al que está asociado el elemento. Normalmente, será el documento actual, aunque puede que esté establecido a null.
<b>parentNode</b>	Devuelve el nodo padre (o contenedor) del elemento. Si no tiene padre, su valor será <b>null</b> .
<b>parentElement</b>	Devuelve el elemento padre (o contenedor) del elemento. Si no tiene padre, su valor será <b>null</b> .
<b>previousSibling</b>	Devuelve el nodo inmediatamente anterior. Si no tiene, su valor será <b>null</b> .
<b>previousElementSibling</b>	Devuelve el elemento inmediatamente anterior. Si no tiene, su valor será <b>null</b> .
<b>scrollHeight</b>	Devuelve la altura en píxeles de la barra de desplazamiento vertical del elemento.
<b>scrollLeft</b>	Devuelve la posición actual de la barra de desplazamiento horizontal de un elemento, con respecto a la izquierda.
<b>scrollTop</b>	Devuelve la posición actual de la barra de desplazamiento vertical de un elemento, con respecto a la parte superior.
<b>scrollWidth</b>	Devuelve el ancho en píxeles de la barra de desplazamiento horizontal del elemento.
<b>selectedIndex</b>	Devuelve o establece el índice seleccionado de un elemento de tipo desplegable (select). Si su valor es -1 indica que no hay ninguno seleccionado.
<b>style</b>	Devuelve el objeto CSSStyleDeclaration que contiene todos los estilos asociados al elemento.
<b>tabIndex</b>	Devuelve o establece un valor que representa el orden de enfoque del elemento cuando se accede a través del tabulador. El valor 0 indica orden secuencial por definición o aparición en el código. El valor -1 indica que no puede tomar el foco.
<b>tagName</b>	Devuelve el nombre de la etiqueta del elemento. Ejemplos de ello pueden ser DIV, LABEL, FORM, INPUT, BUTTON, ...
<b>textContent</b>	Devuelve o establece el contenido del elemento y todos sus descendientes en formato de texto plano. Si se establece esta propiedad, los nodos hijos serán eliminados y se convertirá en un nodo de tipo texto. Nota: esta propiedad sí devolverá los elementos que estén ocultos por CSS.
<b>title</b>	Devuelve o establece el texto que se mostrará cuando el cursor del ratón o puntero se sitúe encima del elemento.
<b>value</b>	Devuelve o establece el valor en un elemento de formulario. Estos pueden ser INPUT, BUTTON, SELECT y DATALIST.



### 9.6.6.2 MÉTODOS MÁS IMPORTANTES

A continuación, se muestran los métodos más importantes o más frecuentemente utilizados.

Método	Descripción y ejemplo
<b>addEventListener</b>	<p>Añade o registra un manejador de evento a un elemento. No es posible utilizarlo con una colección de elementos directamente. Para ello habrá que recorrerla, elemento a elemento, con una estructura iterativa y asignar el listener de forma independiente, es decir, a cada elemento.</p> <pre>\$0.addEventListener("click", nombreFn, false);</pre>
<b>appendChild</b>	<p>Inserta un nodo como último hijo del elemento. El nodo a insertar puede ser también un elemento.</p> <pre>document.body.appendChild(el);</pre>
<b>cloneNode</b>	<p>Realiza la copia del elemento, incluyendo su contenido si está establecido a <b>true</b>.</p> <pre>el.cloneNode(true)</pre>
<b>closest</b>	<p>Realiza una búsqueda ascendente desde el elemento hasta el último de sus padres y llegar a la raíz del documento. Si encuentra el nodo que coincida con el CSS selector especificado lo devuelve, en caso contrario, devuelve null.</p> <pre>\$0.closest("body");</pre>
<b>dispatchEvent</b>	<p>Dispara el evento indicado por parámetro al elemento que se indique.</p> <pre>let evt = new Event('click'); \$0.dispatchEvent(event);</pre>
<b>getAttribute</b>	<p>Devuelve el valor del atributo proporcionado por parámetro del elemento indicado.</p> <pre>\$0.getAttribute("class");</pre>
<b>getElementsByClassName</b>	<p>Devuelve un objeto <b>HTMLCollection</b> con todos los elementos que tengan el nombre de clase proporcionada por parámetro.</p> <pre>\$0.getElementsByClassName("nombre_clase");</pre>
<b>getElementsByTagName</b>	<p>Devuelve un objeto <b>HTMLCollection</b> con todos los elementos que estén definidos con la etiqueta proporcionada por parámetro.</p> <pre>\$0.getElementsByTagName("nombre_etiqueta");</pre>
<b>querySelector</b>	<p>Devuelve el primer elemento descendiente del elemento que coincide con el grupo de selectores CSS especificado.</p> <pre>document.body.querySelector("div");</pre>

<b>querySelectorAll</b>	<p>Devuelve el conjunto de elemento descendientes del elemento que coinciden con el grupo de selectores CSS especificados.</p> <pre>document.body.querySelectorAll("div");</pre>
<b>hasAttribute</b>	<p>Devuelve si el elemento tiene establecido o no el atributo proporcionado por parámetro.</p> <pre>\$0.hasAttribute("class");</pre>
<b>hasChildNodes</b>	<p>Devuelve un booleano que indica si el elemento tiene hijos o no.</p> <pre>\$0.hasChildNodes();</pre>
<b>insertAdjacentHTML</b>	<p>Inserta el código HTML proporcionado como segundo parámetro en la posición especificada por el primer parámetro. Los posibles valores del primer parámetro son <b>afterbegin</b>, que inserta el código HTML como primer elemento hijo, <b>beforebegin</b>, que inserta el HTML antes del elemento, <b>afterend</b>, que inserta el HTML después del elemento, <b>beforeend</b>, que inserta el HTML como último elemento hijo.</p> <pre>\$0.insertAdjacentHTML('afterend', '&lt;i&gt;Texto&lt;/i&gt;');</pre>
<b>insertAdjacentElement</b>	<p>Inserta el elemento HTML proporcionado como segundo parámetro en la posición especificada por el primer parámetro. Los posibles valores del primer parámetro son <b>afterbegin</b>, que inserta el elemento como primer elemento hijo, <b>beforebegin</b>, que inserta el elemento antes del elemento, <b>afterend</b>, que inserta el elemento después del elemento y <b>beforeend</b>, que inserta el elemento como último hijo.</p> <pre>let p = document.createElement("p"); \$0.insertAdjacentElement('afterend', p);</pre>
<b>insertBefore</b>	<p>Permite insertar un elemento inmediatamente antes que el elemento indicado. Tiene una variación de comportamiento que permite realizar inserciones inmediatamente detrás. Esto se consigue si elemento adyacente tienen establecida la propiedad <b>nextSibling</b>.</p> <pre>nodoPadre.insertBefore(nuevoNodo, NodoIndicado);</pre>
<b>removeAttribute</b>	<p>Elimina el atributo solicitado del elemento indicado.</p> <pre>\$0.removeAttribute("class");</pre>
<b>remove</b>	<p>Elimina el elemento indicado.</p> <div>  <b>NOTA</b> </div> <p>No es válido para Internet Explorer 11 o inferiores.</p> <pre>\$0.remove();</pre>

<b>removeEventListener</b>	<p>Elimina el manejador de evento del elemento indicado. No es posible utilizarlo con una colección de elementos directamente. Para ello habrá que recorrerla, elemento a elemento, con una estructura iterativa y eliminar el listener de forma independiente, es decir, a cada elemento.</p> <pre>\$0.removeEventListener("click", nombreFn, false);</pre>
<b>setAttribute</b>	<p>Establece el valor del atributo proporcionado por parámetro al elemento indicado.</p> <pre>\$0.setAttribute("data-row", "1");</pre>

## 9.6.7 El objeto History

Este objeto proporciona una serie de métodos y propiedades que permiten controlar el historial del navegador.

Entre las propiedades que nos ofrece este objeto podemos encontrar:

Propiedad	Descripción
<b>constructor</b>	Devuelve la función constructora nativa.
<b>length</b>	Devuelve la longitud del array.
<b>prototype</b>	Permite añadir nuevas propiedades y métodos al objeto.
<b>scrollRestoration</b>	Permite establecer o configurar cómo serán los cambios de desplazamiento sobre la navegación del historial. Sus posibles propiedades son auto y manual. Su valor predeterminado es auto.
<b>state</b>	Permite recuperar el valor de estado que posee la parte superior de la pila del historial. En general, siempre será null hasta que se produzca una llamada a <b>pushState</b> o <b>replaceState</b> .

En lo referente a las acciones que se pueden realizar con este objeto, tenemos cinco métodos y un evento para controlar en avance o retroceso del historial.

### 9.6.7.1 MÉTODO BACK

Permite retroceder un paso atrás en la navegación como si le pulsásemos el botón de “atrás” del navegador.

```
history.back();
```

### 9.6.7.2 MÉTODO FORWARD

Permite avanzar un paso hacia adelante en la navegación como si le pulsásemos el botón de “adelante” del navegador.

```
history.forward();
```

### 9.6.7.3 MÉTODO GO

Permite saltar un número determinado de pasos en ambas direcciones, es decir, tanto retroceder como avanzar un número determinado de pasos en la navegación.

El número de pasos se indica como parámetro. Si su valor de este parámetro es negativo, retrocederá el número indicado de pasos en el historial. Si el valor resulta ser positivo, se avanzará el número indicado de pasos en el historial.

```
history.go(-2);
```

### 9.6.7.4 MÉTODO PUSHSTATE

Permite añadir registros o entradas en el historial del navegador.

Para poder configurarlo se deben suministrar tres parámetros.

Parámetro	Descripción
<b>datos</b>	Es un objeto que está asociado a la entrada y puede ser recuperable a través de la propiedad <code>state</code> desde dentro del evento denominado <b>PopStateEvent</b> . En este objeto sólo se puede almacenar hasta 640KB de texto y puede ser cualquier cosa que pueda ser convertida con el método <b>stringify</b> del objeto JSON.
<b>Título</b>	Es el nuevo título de la página que se va a insertar.
<b>URL</b>	Es la nueva URL de la página que se va a insertar en el historial.

```
history.pushState(null, 'Listado de Productos', './catalogo.html');
```

Este método puede ser muy interesante si lo que queremos es controlar los botones de avance y retroceso del navegador, sin embargo, todas las entradas que aquí se manejen deben seguir la política del mismo origen. El intento de introducir una entrada que no pertenezca al mismo dominio provocará una excepción del DOM.

```
history.pushState(null, 'mosquis!', "http://google.es");
```

Otra cosa que cabe destacar de este método es que no provocan la navegación ni la recarga de la entrada, es decir, aunque insertemos una entrada que no exista, el navegador no informará de ello hasta que forcemos la navegación a través del método **reload** del objeto **location** o, a través de la pulsación de F5.

#### NOTA

Es posible que algunos piensen que, en el fondo, el resultado de ejecutar este método es similar a ejecutar `window.location = "#..."`, sin embargo, `pushState` es una opción mejor porque, por ejemplo, permite cambiar la URL entera si se desea.



### 9.6.7.5 MÉTODO REPLACESTATE

Permite reemplazar la entrada actual del historial del navegador.

Para poder configurarlo se deben suministrar tres parámetros.

Parámetro	Descripción
<b>datos</b>	Es un objeto que está asociado a la entrada y puede ser recuperable mediante la propiedad <b>state</b> desde dentro del evento denominado <b>PopStateEvent</b> . En este objeto sólo se puede almacenar hasta 640KB de texto y puede ser cualquier cosa que pueda ser convertida con el método <b>stringify</b> del objeto JSON.
<b>Título</b>	Es el nuevo título de la página que se va a reemplazar.
<b>URL</b>	Es la nueva URL de la página que se va a reemplazar en el historial.

Imaginemos el supuesto caso de estar en la URL indicada anteriormente por el método **pushState**, “catalogo.html”. Si quisiéramos que apareciese “Catalogo” en vez de eso, podríamos realizar la siguiente acción:

```
history.replaceState(null, 'Mi Web', "/Productos/Catalogo");
```

Al igual que pasa con el método **pushState**, **replaceState** no provoca la navegación ni la recarga de la entrada, es decir, aunque insertemos una entrada que no exista, el navegador no informará de ello hasta que forcemos la navegación a través del método **reload** del objeto **location** o, a través de la pulsación de F5.

Si echamos una mirada atrás en el tiempo, igual alguno descubre que este tipo de acciones se realizaban a través del objeto **location** y su propiedad **hash**, que modificaban la parte del anclaje de una URL.

### 9.6.7.6 EVENTO ONPOPSTATE

Aunque este evento está asociado al historial, en realidad pertenece al objeto **window**.

Cada vez que el usuario pulsa en los botones de avance o retroceso del historial de navegación, se realiza una llamada al evento **onpopstate**.

El evento que recibe es un objeto **PopStateEvent** que contiene varias propiedades, no obstante, como se ha comentado antes, la propiedad que nos interesa es **state**. La propiedad **state** guarda el objeto con los datos que definimos a través de los métodos **pushState** y **replaceState**.

Si quisiéramos ver el valor de **state** cuando pulsamos en los botones de volver o avanzar en el historial podríamos verlo a través del siguiente código:

```
window.onpopstate = function(e){  
  console.log(e.state)  
}
```

## NOTA

Los métodos de `pushState` y `replaceState`, trabajan de forma coordinada con este evento y el objeto `History`.

### 9.6.7.7 EJEMPLO DE ANULACIÓN DEL BOTÓN VOLVER DEL NAVEGADOR

Como decíamos antes, en el pasado, la anulación del botón volver se realizaba a través del objeto `location` y su propiedad `hash`. Desde hace ya algunos años, una nueva forma de hacer esto es mediante el siguiente script:

```
let title = document.head.querySelector("title").innerHTML;

history.pushState(null, title, location.href);

window.onpopstate = function(e){
  history.forward();
}
```

### 9.6.8 El objeto `this`

El objeto **`this`** es un objeto genérico que provee acceso al objeto actual, ya sea una función u otro objeto. Si este objeto es llamado desde un contexto global, hará referencia al objeto `window`, mientras que, si se llama desde una función o evento, hará referencia al propio objeto destino.

En lo referente a su comportamiento, el objeto **`this`** puede provocar cierta confusión cuando se trata de manipular sus propiedades o métodos. Por ejemplo, puede ocurrir que un objeto trate de actualizar una de sus propiedades a través de `this` y el resultado sea que, aparentemente, no hace nada.

Supongamos el caso de un objeto que tiene que actualizar una de sus propiedades:

```
function producto(a, b){
  let p = a * b;
  this.p = p;
}
producto.p = 0;

// Ejecutamos la función y mostramos su propiedad "p"
producto(2,3);
console.log(producto.p); // Devolverá 0
```

La razón de porqué la ejecución de este código da como resultado su valor inicial es que, la asignación de **`this`**, espera un objeto instanciado. Si en vez de ejecutar la función directamente, la ejecutamos instanciando primero el objeto, veremos que la propiedad **`p`** sí se ha actualizado.

```
let prod = new producto(2,3);
console.log(prod.p); // Devolverá 6
```

Ahora bien, si lo que queríamos era utilizarlo sin tener que instanciarlo, entonces la respuesta es únicamente cambiar `this` por el nombre del objeto, en este caso `producto`.

```
function producto(a, b){
  let p = a * b;
  producto.p = p;
}
producto.p = 0;

// Ejecutamos la función y mostramos su propiedad "p"
producto(2,3);
console.log(producto.p); // Devolverá 6
```

Otra de las dudas que surgen cuando se realizan nuevos componentes en JavaScript es porqué, en ocasiones, **this** tiene un valor **undefined** o **null**. La respuesta a esa pregunta suele ser que se encuentra en modo estricto.

En modo no estricto, la ejecución del siguiente código devolvería verdadero, porque la función fue declarada en el contexto de `window`.

```
function valorThis(){
  console.log(this == window);
}
valorThis(); // Devolverá true
```

Sin embargo, en modo estricto, la ejecución del siguiente código devolvería falso, porque el objeto `window` es reemplazado por `null`.

```
function valorThis(){
  'use strict'
  console.log(this == window)
}
valorThis(); // Devolverá false
```

## 9.6.9 El objeto `globalThis`

Si el objeto `this` nos permite manejar el contexto actual, **globalThis** nos permite acceder al contexto global, que es dónde se encuentra el objeto `this`. Dicho de otra forma, si estamos dentro de una función de un objeto que fue declarado bajo el contexto de `window`, la propiedad `this` será la declaración de la función y `globalThis` será un alias de `window`.

```
let it = {name: 'IT', version: '1.0'}
it.imprimirContextos = function(){
  console.log("this: ", this);
  console.log("globalThis: ", globalThis);
}
```

Si ejecutamos el código, podremos comprobar que **this** representa al objeto **it** (que contiene la definición de `imprimirContextos`, `name` y `versión`), y que, **globalThis**, representa al objeto **window**.

Veamos otro ejemplo:

```
let it = function(){ console.log('IT inicializado'); }  
it.imprimirContextos = function(){  
  console.log("this: ", this);  
  console.log("globalThis: ", globalThis);  
}
```

Si ahora ejecutamos este código, podremos comprobar que **this** representa la declaración del **contenido de la función**, es decir, la función que contiene el método console mientras que, **globalThis**, representa al objeto **window**.

### 9.6.10 El objeto prototype

Todos los objetos en JavaScript provienen del objeto Object, por lo que, todos los objetos heredan sus métodos y propiedades. El problema surge cuando se desean incorporar nuevas funcionalidades o métodos a los objetos prototipados. Para eso JavaScript provee de la propiedad **prototype**, la cual permite precisamente esto.

El objeto **prototype** tiene bastantes propiedades obsoletas o no estandarizadas, pero hay 2 propiedades que no podemos ignorar:

Propiedad	Descripción
<b>constructor</b>	Especifica la función que creó el prototipo del objeto.
<b>__proto__</b>	Especifica la llamada de acceso provee acceso al interior del prototipo a través del cual se accede a ella.

En lo referente a sus métodos, hay algunos que cabe destacar:

Método	Descripción
<b>hasOwnProperty</b>	Devuelve un booleano que indica si la propiedad proporcionada por parámetro está presente en el objeto. <pre>let data = { id: 1, code: 0 }; data.hasOwnProperty("idd") // Devuelve false;</pre>
<b>isPrototypeOf</b>	Devuelve un booleano que indica si el objeto pertenece a la cadena de prototipos del objeto especificado. <pre>let data = { id: 1, code: 0 }; data.isPrototypeOf(JSON) // Devuelve false;</pre>
<b>propertyIsEnumerable</b>	Devuelve un booleano que indica si la propiedad tiene el atributo enumerable establecido. <pre>let data = { id: 1, code: 0 }; data.propertyIsEnumerable("id") // Devuelve true;</pre>
<b>toString</b>	Devuelve la definición del objeto convertido en formato cadena de texto. <pre>let data = { id: 1, code: 0 }; data.toString() // Devuelve '[object Object]';</pre>



A continuación, se muestra cómo se puede añadir la funcionalidad de fecha actual en formato español (Little Endian) al objeto Date de JavaScript:

```
Date.prototype.littleEndianFormat = function () {  
  const local = new Date(this);  
  
  // Se calcula el diferencial GMT  
  local.setMinutes(this.getMinutes()-this.getTimezoneOffset());  
  
  let aux = local.toJSON().slice(0, 10);  
  aux = aux.split('-')[2] + "-" +  
  aux.split('-')[1] + "-" +  
  aux.split('-')[0];  
  
  return aux;  
};  
  
new Date().littleEndianFormat();
```

La función **setMinutes** nos sirve para calcular el diferencial GMT.

Seguidamente, se transforma una cadena con la fecha y la hora en formato estándar de JavaScript a formato compatible con JSON y extraemos la subcadena del resultado desde la posición 0 hasta la posición 10.

Finalmente, y con ayuda de la función **split** convertimos la fecha, de formato inglés (Big Endian) a formato español (Little Endian) y, el resultado es lo que se devuelve.

## 9.7 OTRAS COSAS QUE SABER SOBRE LOS OBJETOS DE JAVASCRIPT

---

### 9.7.1 La herencia

Cuando se empieza a programar en un lenguaje como JavaScript, su sintaxis liberal puede causar muchos problemas de adaptación e, incluso, puede hacer que los desarrolladores rechacen el lenguaje.

El paradigma de JavaScript es, en varios aspectos, muy diferente a muchos lenguajes orientados a objetos. Sin embargo, no olvidemos que, en JavaScript, todo son objetos, incluyendo las entidades que, por definición, no deberían serlo.

Como decíamos en un capítulo anterior, en JavaScript todo hijo hereda de su padre y, casi todos, heredan de **Object**. El problema surge, cuando, se desean incorporar nuevas funcionalidades o métodos a los objetos prototipados, pero, para eso, JavaScript provee de algunas “técnicas” para ayudar durante el proceso.

Veámoslo con un ejemplo:

Si queremos crear un nuevo objeto desde cero, lo primero que debemos hacer es crear su constructor.

```
function Persona(nombre, apellidos, edad) {
  this.nombre = nombre + " " + apellidos;
  this.edad = edad;
};
```

Con esto hemos definido un objeto que hemos llamado **Persona** y al que le hemos dotado de unas pocas propiedades, no obstante, como no tiene ningún método, le creamos uno.

```
// Método para recuperar la edad de la persona
Persona.prototype.getEdad = function() {
  alert(this.nombre + ' tiene ' + this.edad + ' años!');
};

// Método para recuperar el nombre de la persona
Persona.prototype.getNombre = function() {
  alert('Mi nombre es ' + this.nombre);
};
```

Ahora lo que queremos hacer es, definir a esas personas como alumnos o profesores, por lo que tendremos que crear los objetos **Alumno** y **Profesor**.

```
function Alumno(persona, curso, asignaturas) {
  for(let key in persona){
    this[key] = persona[key];
  }

  this.curso = curso;
  this.asignaturas = asignaturas;
};

function Profesor(curso) {
  for(let key in persona){
    this[key] = persona[key];
  }
  this.curso = curso;
};
```

Ambos son Persona, por lo que tendrán que heredar de la clase que hemos definido antes. Para ello definimos cuál es el prototipo del que heredan y sobrescribimos su constructor para que las propiedades y métodos pertenezcan a ese objeto.

```
Alumno.prototype = new Persona();
Alumno.prototype.constructor = Alumno;

Profesor.prototype = new Persona();
Profesor.prototype.constructor = Alumno;
```

Ahora los objetos Alumno y Profesor son también Personas y, por tanto, ya hemos provocado la herencia de sus propiedades y métodos.

```
let pablo = new Persona('Pablo', 'Fernández', 18);
let alumno = new Alumno(pablo, '1º FP', '...');
```

Si ahora consultamos lo que contiene la variable `alumno`, veremos que tiene todas las propiedades del objeto `Alumno` y todas las propiedades y métodos del objeto `Persona`.

```
console.log(pablo);  
// devuelve lo siguiente:  
{  
  edad: 18  
  nombre: "Pablo Fernández"  
  ▶ __proto__: Object  
}  
  
console.log(alumno);  
  
// Devuelve lo siguiente:  
{  
  asignaturas: "..."  
  curso: "1º FP"  
  edad: 18  
  ▶ getEdad: f{ }  
  ▶ getNombre: f{ }  
  nombre: "Pablo Fernández"  
  ▶ __proto__: Persona  
}
```

### 9.7.2 Sentencias `get` y `set`

Permite definir un método que realizará una funcionalidad concreta cuando se acceda a una determinada propiedad.

```
get ultimo() {  
  if (this.log.length > 0) {  
    return this.log[this.log.length - 1];  
  } else {  
    return "";  
  }  
}
```

Si ejecutásemos el ejemplo anterior, podríamos observar que cuando se llama a la propiedad **ultimo**, el objeto nos devuelve el último elemento del array denominado `log`.

Y con la sentencia `set` pasa un poco lo mismo, podemos hacer que, cada vez que se actualice una propiedad, se ejecute una acción asociada de forma interna.

```
set mensaje(mensaje) {  
  this.log.mensaje(mensaje);  
}
```

Si ejecutásemos el ejemplo anterior, podríamos observar que, cuando se actualiza la propiedad `mensaje`, dicho mensaje se inserta en un array contenedor.

### 9.7.2.1 EJEMPLO COMPLETO DE GET Y SET

```
let Historico = {
  get ultimo() {
    if (this.log.length > 0) {
      return this.log[this.log.length - 1];
    } else {
      return "";
    }
  },
  set mensaje(mensaje) {
    this.log.push(mensaje);
  }, log: []
}

// Imprimimos el ultimo valor
console.log(Historico.ultimo); // Devuelve ""

// Añadimos un mensaje
Historico.mensaje = "hola que tal"; // Devuelve "Hola que tal"

// Imprimimos el ultimo valor
console.log(Historico.ultimo); // Devuelve "Hola que tal"
```

## 9.8 PRACTICA Y JUEGA

---

Test de JavaScript: Objetos	Código QR
<p>Juega a averiguar todas las respuestas correctas con el mínimo número de errores y en el menor tiempo posible.</p> <p><b><a href="https://codepen.io/pefc/full/GRXMZoB">https://codepen.io/pefc/full/GRXMZoB</a></b></p>	