


# Javascript

01. Nivel inicial

03. Web Storage

A large yellow square centered on the page, containing the letters 'JS' in a bold, black, sans-serif font.

JS

14.1	Sistemas de almacenamiento	3
14.2	Session Storage	4
	Almacenando datos	5
	Leyendo datos	6
	Eliminando datos	7
14.3	Local Storage	9
	Evento storage	10

### 14.1 Sistemas de almacenamiento

La API Web Storage nos permite almacenar datos en el disco duro del usuario y acceder a los mismos cuando el usuario vuelve a visitar nuestro sitio web. El sistema de almacenamiento provisto por esta API se puede usar en dos situaciones particulares: cuando la información tiene que estar disponible solo durante una sesión y cuando se tiene que preservar hasta que lo determina el código o el usuario. Con el propósito de clarificar esta situación para los desarrolladores, la API se ha dividido en dos partes llamadas *Session Storage* y *Local Storage*.

**Session Storage**—Este es un mecanismo de almacenamiento que mantiene los datos disponibles solo durante una sesión. A la información almacenada a través de este mecanismo se accede desde una ventana o pestaña, y se conserva hasta que se cierra la ventana.

**Local Storage**—Este mecanismo trabaja de forma similar al sistema de almacenamiento de una aplicación de escritorio. Los datos son se preservan de forma permanente y siempre están disponibles desde la aplicación que los ha creado.

Ambos mecanismos trabajan con una interfaz similar y comparten las mismas propiedades y métodos, y ambos dependen del origen, lo que significa que la información solo estará disponible para el sitio web que la ha generado. Cada sitio web tiene asignado un espacio de almacenamiento, y la información se preserva o elimina dependiendo del mecanismo aplicado.

### 14.2 Session Storage

El sistema Session Storage es el más sencillo de todos. Este sistema almacena los datos solo para una sesión, lo cual significa que los datos se eliminarán cuando el usuario cierre la ventana o pestaña. Las aplicaciones pueden usarlo como soporte o para almacenar información que se puede solicitar más adelante en el proceso pero que se vuelve irrelevante si el usuario abandona el sitio web.

El siguiente es el documento que vamos a utilizar para probar esta API. Como ambos sistemas trabajan con la misma interfaz, solo vamos a necesitar un documento y un formulario sencillo para probar los ejemplos de este capítulo.

---

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>API Web Storage</title>
  <link rel="stylesheet" href="almacenamiento.css">
  <script src="almacenamiento.js"></script>
</head>
```

---

```

<body>
  <section id="cajaformulario">
    <form name="formulario">
      <p><label>Clave: <input type="text" name="clave"
id="clave"></label></p>
      <p><label>Valor: <textarea name="texto"
id="texto"></textarea></label></p>
      <p><button type="button" id="grabar">Grabar</button></p>
    </form>
  </section>
  <section id="cajadatos">
    <p>Información no disponible</p>
  </section>
</body>
</html>

```

---

### **Listado 14-1:** Creando un documento para trabajar con la API Web Storage

También necesitamos algunos estilos para diferenciar el formulario de la caja donde vamos a mostrar los datos.

```

#cajaformulario {
  float: left;
  padding: 20px;
  border: 1px solid #999999;
}
#cajadatos {
  float: left;
  width: 400px;
  margin-left: 20px;
  padding: 20px;
  border: 1px solid #999999;
}
#clave, #texto {
  width: 200px;
}
#cajadatos > div {
  padding: 5px;
  border-bottom: 1px solid #999999;
}

```

---

### **Listado 14-2:** Diseñando la interfaz



**Hágalo usted mismo:** cree un archivo HTML con el documento del Listado 14-1 y un archivo CSS llamado `almacenamiento.css` con los estilos del Listado 14-2. También necesitará un archivo JavaScript llamado `almacenamiento.js` para grabar y probar los códigos presentados a continuación.

## Almacenando datos

Los datos se almacenan como ítems, que se componen de un par nombre/valor. Estos ítems son como variables, cada uno con un nombre y un valor, que se pueden crear, modificar o

eliminar. Los siguientes son los métodos con los que cuenta la API para crear y leer un ítem en el espacio de almacenamiento.

**setItem(nombre, valor)**—Este método crea y almacena un ítem con el nombre y el valor especificados por los atributos. Si ya existe un ítem con el mismo nombre, se actualizará con el nuevo valor, por lo que este método también se puede usar para modificar datos almacenados con anterioridad.

**getItem(nombre)**—Este método devuelve el valor del ítem con el nombre especificado por el atributo.

El objeto **Window** incluye dos propiedades para facilitar acceso a los sistemas de almacenamiento: **sessionStorage** y **localStorage**. Para almacenar y leer ítems, tenemos que ejecutar los métodos correspondientes desde estas propiedades, como en el siguiente ejemplo.

---

```
function iniciar() {
    var boton = document.getElementById("grabar");
    boton.addEventListener("click", nuevoitem);
}
function nuevoitem() {
    var clave = document.getElementById("clave").value;
    var valor = document.getElementById("texto").value;
    sessionStorage.setItem(clave, valor);
    mostrar(clave);
}
function mostrar(clave) {
    var cajadatos = document.getElementById("cajadatos");
    var valor = sessionStorage.getItem(clave);
    cajadatos.innerHTML = "<div>" + clave + " - " + valor + "</div>";
}
window.addEventListener("load", iniciar);
```

---

### **Listado 14-3:** Almacenando y leyendo datos

En el código del Listado 14-3, la función **nuevoitem()** se ejecuta cada vez que el usuario hace clic en el botón Grabar del formulario. Esta función crea un ítem con la información insertada en los campos de entrada y luego llama a la función **mostrar()**. En esta función, el ítem se lee con el valor del atributo **clave** y el método **getItem()**, y luego el contenido del ítem se inserta en el elemento **cajadatos** para mostrarlo en la pantalla.



**Hágalo usted mismo:** copie el código del Listado 14-3 en su archivo almacenamiento.js y abra el documento del Listado 14-1 en su navegador. Inserte valores en los campos y pulse el botón para almacenarlos. Debería ver los valores del ítem que acaba de insertar dentro del elemento **cajadatos**.



**Lo básico:** los métodos y propiedades se pueden concatenar con notación de puntos. El intérprete procesa los componentes de la instrucción de izquierda a derecha. En el ejemplo del Listado 14-3, concatenamos el método **getElementById()** con la propiedad **value**. El intérprete primero ejecuta el método, obtiene una referencia al objeto **Element**, y luego lee el valor de la propiedad **value** de ese objeto. Este es simplemente un atajo, podríamos

haber almacenado la referencia al elemento en una variable y luego leer la propiedad desde esa variable en otra instrucción, como hemos hecho en ejemplos anteriores, pero de esta manera ahorramos algunas líneas de código. El resultado es el mismo, los valores que inserta el usuario en los campos se asignan a las variables **clave** y **valor**.

Además de estos métodos, la API también ofrece un atajo para crear y leer un ítem en el espacio de almacenamiento. Podemos usar el nombre del ítem como una propiedad de **sessionStorage** y acceder a su valor de esta manera. Como con cualquier otra propiedad, contamos con dos sintaxis: podemos encerrar la variable representando el nombre en corchetes (**sessionStorage[nombre] = valor**) o podemos usar notación de puntos (**sessionStorage.miitem = valor**).

---

```
function iniciar() {
    var boton = document.getElementById("grabar");
    boton.addEventListener("click", nuevoitem);
}
function nuevoitem() {
    var clave = document.getElementById("clave").value;
    var valor = document.getElementById("texto").value;
    sessionStorage[clave] = valor;
    mostrar(clave);
}
function mostrar(clave) {
    var cajadatos = document.getElementById("cajadatos");
    var valor = sessionStorage[clave];
    cajadatos.innerHTML = "<div>" + clave + " - " + valor + "</div>";
}
window.addEventListener("load", iniciar);
```

---

**Listado 14-4:** Usando un atajo para trabajar con ítems

## Leyendo datos

Los ítems se almacenan en un array, por lo que también podemos acceder a los valores con un índice o un bucle. La API ofrece esta propiedad y este método con dicho propósito.

**length**—Esta propiedad devuelve el número de ítems acumulados en el espacio de almacenamiento de la aplicación.

**key(índice)**—Este método devuelve el nombre del ítem en el índice especificado por el atributo.

Los ejemplos anteriores solo leen el último ítem almacenado. Aprovechando el método **key()**, vamos a mejorar el código para listar todos los valores disponibles en el espacio de almacenamiento.

---

```
function iniciar() {
    var boton = document.getElementById("grabar");
    boton.addEventListener("click", nuevoitem);
    mostrar();
}
```

---

```
function nuevoitem() {
    var clave = document.getElementById("clave").value;
    var valor = document.getElementById("texto").value;
    sessionStorage.setItem(clave, valor);
    document.getElementById("clave").value = "";
    document.getElementById("texto").value = "";
    mostrar();
}
function mostrar() {
    var cajadatos = document.getElementById("cajadatos");
    cajadatos.innerHTML = "";
    for (var f = 0; f < sessionStorage.length; f++) {
        var clave = sessionStorage.key(f);
        var valor = sessionStorage.getItem(clave);
        cajadatos.innerHTML += "<div>" + clave + " - " + valor + "</div>";
    }
}
window.addEventListener("load", iniciar);
```

---

#### **Listado 14-5:** Listando todos los ítems en el espacio de almacenamiento

El propósito de este ejemplo es mostrar la lista completa de ítems en la pantalla. La función `mostrar()` se ha mejorado con la propiedad `length` y el método `key()`. Dentro de un bucle `for`, se llama al método `key()` para obtener el nombre de cada ítem. Por ejemplo, si el ítem en la posición 0 del espacio de almacenamiento se ha creado con el nombre "miitem", la instrucción `sessionStorage.key(0)` devolverá el valor de "miitem". Si llamamos a este método desde un bucle, podemos listar todos los ítems en la pantalla, cada uno con su correspondiente nombre y valor.

A la función `mostrar()` también se le llama al final de la función `iniciar()` para mostrar los ítems que ya se encuentran en el espacio de almacenamiento tan pronto como se ejecuta la aplicación.



**Hágalo usted mismo:** actualice su archivo `almacenamiento.js` con el código del Listado 14-5 y abra el documento del Listado 14-1 en su navegador. Inserte nuevos valores en el formulario y pulse el botón para almacenarlos. Debería ver una lista con todos los valores que ha insertado hasta el momento dentro del elemento `cajadatos`.



**Lo básico:** puede aprovechar la API Formularios estudiada en el Capítulo 7 para controlar la validez de los campos de entrada y no permitir la inserción de ítems no válidos o vacíos.

## Eliminando datos

Los ítems se pueden crear, leer y, por supuesto, eliminar. La API incluye dos métodos para eliminar ítems del espacio de almacenamiento.

**removeItem(nombre)**—Este método elimina un ítem. El atributo **nombre** especifica el nombre del ítem a eliminar.

**clear()**—Este método elimina todos los ítems del espacio de almacenamiento.

El siguiente ejemplo incluye botones al lado de cada valor para eliminarlo.

---

```
function iniciar() {
    var boton = document.getElementById("grabar");
    boton.addEventListener("click", nuevoitem);
    mostrar();
}
function nuevoitem() {
    var clave = document.getElementById("clave").value;
    var valor = document.getElementById("texto").value;
    sessionStorage.setItem(clave, valor);
    document.getElementById("clave").value = "";
    document.getElementById("texto").value = "";
    mostrar();
}
function mostrar() {
    var cajadatos = document.getElementById("cajadatos");
    cajadatos.innerHTML = '<div><input type="button"
onclick="removerTodo()" value="Eliminar Todos"></div>';
    for (var f = 0; f < sessionStorage.length; f++) {
        var clave = sessionStorage.key(f);
        var valor = sessionStorage.getItem(clave);
        cajadatos.innerHTML += "<div>" + clave + " - " + valor + "<br>";
        cajadatos.innerHTML += '<input type="button"
onclick="removerItem(\'' + clave + '\')" value="Remover"></div>';
    }
}
function removerItem(clave) {
    if (confirm("Está seguro?")) {
        sessionStorage.removeItem(clave);
        mostrar();
    }
}
function removerTodo() {
    if (confirm("Está seguro?")) {
        sessionStorage.clear();
        mostrar();
    }
}
window.addEventListener("load", iniciar);
```

---

#### **Listado 14-6: Eliminando ítems en el espacio de almacenamiento**

Las funciones `iniciar()` y `nuevoitem()` del Listado 14-6 son las mismas que las del ejemplo anterior. Solo ha cambiado la función `mostrar()` para incorporar botones con el atributo de evento `onclick` con los que llamar a las funciones que eliminarán un ítem individual o todos juntos. El código crea un botón Eliminar para cada ítem de la lista y también un único botón en la parte superior para borrar el espacio de almacenamiento completo.

Las funciones `removerItem()` y `removerTodo()` son responsables de eliminar el ítem seleccionado o limpiar el espacio de almacenamiento, respectivamente. Cada función llama a la función `mostrar()` al final para actualizar la lista de ítems en la pantalla.





**Hágalo usted mismo:** con el código del Listado 14-6 podrá ver cómo se procesa la información con el sistema Session Storage. Abra el documento HTML del Listado 14-1 en su navegador, cree nuevos ítems y luego abra el mismo documento en otra ventana. La información será diferente en cada ventana. La primera ventana mantiene sus datos disponibles, pero el espacio de almacenamiento de la nueva ventana estará vacío. A diferencia de otros sistemas, Session Storage considera cada ventana como una instancia independiente de la aplicación y la información de la sesión no se comparte entre ellas.

## 14.3 Local Storage

Contar con un sistema fiable para almacenar datos durante una sesión puede ser útil en algunas circunstancias, pero cuando intentamos emular aplicaciones de escritorio en la Web, un sistema de almacenamiento temporario como este es generalmente insuficiente. Para mantener la información siempre disponible, la API Web Storage incluye el sistema Local Storage. Con Local Storage, podemos almacenar grandes cantidades de datos y dejar que el usuario decida si la información aún es útil y debemos conservarla o no.

Este sistema usa la misma interfaz que Session Storage; por lo tanto, cada método y propiedad estudiados en este capítulo están disponibles también en Local Storage. Solo necesitamos sustituir la propiedad `sessionStorage` por la propiedad `localStorage` para preparar los códigos.

---

```
function iniciar() {
    var boton = document.getElementById("grabar");
    boton.addEventListener("click", nuevoitem);
    mostrar();
}
function nuevoitem() {
    var clave = document.getElementById("clave").value;
    var valor = document.getElementById("texto").value;

    localStorage.setItem(clave, valor);
    mostrar();
    document.getElementById('clave').value = "";
    document.getElementById('texto').value = "";
}
function mostrar() {
    var cajadatos = document.getElementById("cajadatos");
    cajadatos.innerHTML = "";
    for (var f = 0; f < localStorage.length; f++) {
        var clave = localStorage.key(f);
        var valor = localStorage.getItem(clave);
        cajadatos.innerHTML += "<div>" + clave + " - " + valor + "</div>";
    }
}
window.addEventListener("load", iniciar);
```

---

### *Listado 14-7: Usando Local Storage*

En el Listado 14-7, tomamos uno de los códigos anteriores y reemplazamos la propiedad `sessionStorage` por la propiedad `localStorage`. Ahora, cada ítem creado está disponible en cada ventana e incluso después de que el navegador se cierra por completo.



**Hágalo usted mismo:** usando el documento del Listado 14-1, pruebe el código del Listado 14-7. Este código crea un nuevo ítem con la información en el formulario y automáticamente lista todos los ítems disponibles en el espacio de almacenamiento reservado para la aplicación. Cierre el navegador y abra el documento nuevamente. Aún debería ver todos los ítems de la lista.

## Evento storage

Debido a que la información almacenada por Local Storage está disponible en todas las ventanas donde se carga la aplicación, debemos resolver dos problemas: cómo se comunican estas ventanas entre sí y cómo actualizamos la información en la ventana que no está activa. La API incluye el siguiente evento para solucionar ambos problemas.

**storage**—La ventana desencadena este evento cada vez que ocurre un cambio en el espacio de almacenamiento. Se puede usar para informar a cada ventana abierta con la misma aplicación que se ha realizado un cambio en el espacio de almacenamiento y que se debe hacer algo al respecto.

Mantener la lista de ítems actualizada en nuestro ejemplo es fácil, solo tenemos que llamar a la función `mostrar()` cada vez que se desencadena el evento `storage`.

---

```
function iniciar() {
    var boton = document.getElementById("grabar");
    boton.addEventListener("click", nuevoitem);
    window.addEventListener("storage", mostrar);
    mostrar();
}

function nuevoitem() {
    var clave = document.getElementById("clave").value;
    var valor = document.getElementById("texto").value;
    localStorage.setItem(clave, valor);
    document.getElementById("clave").value = "";
    document.getElementById("texto").value = "";
    mostrar();
}

function mostrar() {
    var cajadatos = document.getElementById("cajadatos");
    cajadatos.innerHTML = "";
    for (var f = 0; f < localStorage.length; f++) {
        var clave = localStorage.key(f);
        var valor = localStorage.getItem(clave);
        cajadatos.innerHTML += "<div>" + clave + " - " + valor + "</div>";
    }
}

window.addEventListener("load", iniciar);
```

---

**Listado 14-8:** Respondiendo al evento `storage` para mantener la lista de ítems actualizada

En este ejemplo, la función `mostrar()` responde al evento `storage` y, por lo tanto, se ejecuta cada vez que se crea, actualiza o elimina un ítem. Ahora, si algo cambia en una ventana, se mostrará automáticamente en las otras ventanas que están ejecutando la misma aplicación.



**Hágalo usted mismo:** usando el documento del Listado 14-1, pruebe el código del Listado 14-8. El evento **storage** solo trabaja cuando la aplicación se almacena en un servidor o en un servidor local. Suba los archivos a su servidor y abra el documento en dos ventanas diferentes. Inserte o actualice un ítem en una ventana y abra la otra ventana para ver cómo el evento **storage** actualiza la información.

El evento **storage** envía a la función un objeto de tipo **StorageEvent** que contiene las siguientes propiedades para proveer información acerca de la modificación producida en el espacio de almacenamiento.

**key**—Esta propiedad devuelve el nombre del ítem afectado.

**oldValue**—Esta propiedad devuelve el valor anterior del ítem afectado.

**newValue**—Esta propiedad devuelve el nuevo valor asignado al ítem.

**url**—Esta propiedad devuelve la URL del documento que ha producido la modificación. El espacio de almacenamiento se reserva para todo el dominio, por lo que pueden acceder a él diferentes documentos.

**storageArea**—Esta propiedad devuelve un objeto que contiene todos los pares nombre/valor disponibles en el espacio de almacenamiento después de la modificación.

El siguiente ejemplo imprime en la consola los valores de estas propiedades para mostrar la información disponible.

---

```
function iniciar() {
    var boton = document.getElementById("grabar");
    boton.addEventListener("click", nuevoitem);
    addEventListener("storage", modificado);
    mostrar();
}

function nuevoitem() {
    var clave = document.getElementById("clave").value;
    var valor = document.getElementById("texto").value;
    localStorage.setItem(clave, valor);
    document.getElementById("clave").value = "";
    document.getElementById("texto").value = "";
    mostrar();
}

function modificado(evento) {
    console.log(evento.key);
    console.log(evento.oldValue);
    console.log(evento.newValue);
    console.log(evento.url);
    console.log(evento.storageArea);
    mostrar();
}

function mostrar() {
    var cajadatos = document.getElementById("cajadatos");
    cajadatos.innerHTML = "";
    for (var f = 0; f < localStorage.length; f++) {
        var clave = localStorage.key(f);
```

---

```
var valor = localStorage.getItem(clave);
cajados.innerHTML += "<div>" + clave + " - " + valor + "</div>";
}
}
window.addEventListener("load", iniciar);
```

---

#### ***Listado 14-9: Accediendo a las propiedades del evento***

En el Listado 14-9 agregamos una nueva función para responder al evento **storage**. Se llama a la función **modificado()** mediante el evento cuando el espacio de almacenamiento queda modificado por otra ventana. En esta función, imprimimos los valores de las propiedades uno por uno en la consola y al final llamamos a la función **mostrar()** para actualizar la información en la pantalla.

