Eventos

En la programación tradicional, las aplicaciones se ejecutan secuencialmente de principio a fin para producir sus resultados. Sin embargo, en la actualidad el modelo predominante es el de la programación basada en eventos. Los scripts y programas esperan sin realizar ninguna tarea hasta que se produzca un evento. Una vez producido, ejecutan alguna tarea asociada a la aparición de ese evento y cuando concluye, el script o programa vuelve al estado de espera.

JavaScript permite realizar scripts con ambos métodos de programación: secuencial y basada en eventos. Los eventos de JavaScript permiten la interacción entre las aplicaciones JavaScript y los usuarios. Cada vez que se pulsa un botón, se produce un evento. Cada vez que se pulsa una tecla, también se produce un evento. No obstante, para que se produzca un evento no es obligatorio que intervenga el usuario, ya que, por ejemplo, cada vez que se carga una página, también se produce un evento.

TIPOS DE EVENTOS

Cada elemento HTML tiene definida su propia lista de posibles eventos que se le pueden asignar. Un mismo tipo de evento (por ejemplo, pinchar el botón izquierdo del ratón) puede estar definido para varios elementos HTML y un mismo elemento HTML puede tener asociados diferentes eventos.

El nombre de los eventos se construye mediante el prefijo `on`, seguido del nombre en inglés de la acción asociada al evento. Así, el evento de pinchar un elemento con el ratón se denomina `onclick` y el evento asociado a la acción de mover el ratón se denomina `onmousemove`.

La siguiente tabla resume los eventos más importantes definidos por JavaScript:

Evento	Descripción	Elementos para los que está definido
`onblur`	Un elemento pierde el foco	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`onchange`	Un elemento ha sido modificado	`<input>`, `<select>`, `<textarea>`
`onclick`	Pulsar y soltar el ratón	Todos los elementos

ondblclick	Pulsar dos veces seguidas con el ratón	Todos los elementos
onfocus	Un elemento obtiene el foco	<button>, <input>, <label>, <select>, <textarea>, <body>
onkeydown	Pulsar una tecla y no soltarla	Elementos de formulario y <body>
onkeypress	Pulsar una tecla	Elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onload	Página cargada completamente	<body>
onmousedown	Pulsar un botón del ratón y no soltarlo	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos
onmouseout	El ratón "sale" del elemento	Todos los elementos
onmouseover	El ratón "entra" en el elemento	Todos los elementos
onmouseup	Soltar el botón del ratón	Todos los elementos

onreset	Inicializar el formulario	<form>
onresize	Modificar el tamaño de la ventana	<body>
onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar el formulario	<form>
onunload	Se abandona la página, por ejemplo al cerrar el navegador	<body>

Los eventos más utilizados en las aplicaciones web tradicionales son onload para esperar a que se cargue la página por completo, los eventos onclick, onmouseover, onmouseout para controlar el ratón y onsubmit para controlar el envío de los formularios.

Algunos eventos de la tabla anterior (onclick, onkeydown, onkeypress, onreset, onsubmit) permiten evitar la "acción por defecto" de ese evento. Más adelante se muestra en detalle este comportamiento, que puede resultar muy útil en algunas técnicas de programación.

Las acciones típicas que realiza un usuario en una página web pueden dar lugar a una sucesión de eventos. Al pulsar por ejemplo sobre un botón de tipo <input type="submit"> se desencadenan los eventos onmousedown, onclick, onmouseup y onsubmit de forma consecutiva.

MANEJADORES DE EVENTOS

Un evento de JavaScript por sí mismo carece de utilidad. Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado, por lo que la aplicación puede responder ante cualquier evento que se produzca durante su ejecución.

Las funciones o código JavaScript que se definen para cada evento se denominan *manejador de eventos* (*event handlers* en inglés).

HANDLERS Y LISTENERS

En las secciones anteriores se introdujo el concepto de "event handler" o manejador de eventos, que son las funciones que responden a los eventos que se producen. Además, se vieron tres formas de definir los manejadores de eventos para el modelo básico de eventos:

1. Código JavaScript dentro de un atributo del propio elemento HTML
2. Definición del evento en el propio elemento HTML pero el manejador es una función externa
3. Manejadores semánticos asignados mediante DOM sin necesidad de modificar el código HTML de la página

Cualquiera de estos tres modelos funciona correctamente en todos los navegadores disponibles en la actualidad. Las diferencias entre navegadores surgen cuando se define más de un manejador de eventos para un mismo evento de un elemento. La forma de asignar y *"desasignar"* manejadores múltiples depende completamente del navegador utilizado.

MANEJADORES DE EVENTOS DE DOM

La especificación DOM define otros dos métodos similares a los disponibles para Internet Explorer y denominados addEventListener() y removeEventListener() para asociar y desasociar manejadores de eventos.

La principal diferencia entre estos métodos y los anteriores es que en este caso se requieren tres parámetros: el nombre del *"event listener"*, una referencia a la función encargada de procesar el evento y el tipo de flujo de eventos al que se aplica.

El primer argumento no es el nombre completo del evento como sucede en el modelo de Internet Explorer, sino que se debe eliminar el prefijo on. En otras palabras, si en Internet Explorer se utilizaba el nombre onclick, ahora se debe utilizar click.

A continuación, se muestran los ejemplos anteriores empleando los métodos definidos por DOM:

```
function muestraMensaje() {
  console.log("Has pulsado el ratón");
}
var elDiv = document.getElementById("div_principal");
elDiv.addEventListener("click", muestraMensaje);

// Más adelante se decide desasociar la función al evento
elDiv.removeEventListener("click", muestraMensaje);
```

Asociando múltiples funciones a un único evento:

```
function muestraMensaje() {
  console.log("Has pulsado el ratón");
}

function muestraOtroMensaje() {
  console.log("Has pulsado el ratón y por eso se muestran estos mensajes");
}

var elDiv = document.getElementById("div_principal");
elDiv.addEventListener("click", muestraMensaje);
elDiv.addEventListener("click", muestraOtroMensaje);
```

Si se asocia una función a un flujo de eventos determinado, esa función sólo se puede desasociar en el mismo tipo de flujo de eventos. Si se considera el siguiente ejemplo:

```
function muestraMensaje() {
  console.log("Has pulsado el ratón");
}

var elDiv = document.getElementById("div_principal");
elDiv.addEventListener("click", muestraMensaje);

// Más adelante se decide desasociar la función al evento
elDiv.removeEventListener("click", muestraMensaje);
```

EL OBJETO EVENT

Cuando se produce un evento, no es suficiente con asignarle una función responsable de procesar ese evento. Normalmente, la función que procesa el evento necesita información relativa al evento producido: la tecla que se ha pulsado, la posición del ratón, el elemento que ha producido el evento, etc.

El objeto `event` es el mecanismo definido por los navegadores para proporcionar toda esa información. Se trata de un objeto que se crea automáticamente cuando se produce un evento y que se destruye de forma automática cuando se han ejecutado todas las funciones asignadas al evento.

El estándar DOM especifica que el objeto `event` es el único parámetro que se debe pasar a las funciones encargadas de procesar los eventos.

```
elDiv.onclick = function(event) {
  ...
}
```

El funcionamiento de los navegadores que siguen los estándares puede parecer "mágico", ya que en la declaración de la función se indica que tiene un parámetro, pero en la aplicación no se pasa ningún parámetro a esa función. En realidad, los navegadores que siguen los estándares crean automáticamente ese parámetro y lo pasan siempre a la función encargada de manejar el evento.

PROPIEDADES Y MÉTODOS

A pesar de que el mecanismo definido por los navegadores para el objeto event es similar, existen numerosas diferencias en cuanto a las propiedades y métodos del objeto.

PROPIEDADES DEFINIDAS POR DOM

La siguiente tabla recoge las propiedades definidas para el objeto event en los navegadores que siguen los estándares:

Propiedad/Método	Devuelve	Descripción
altKey	Boolean	Devuelve true si se ha pulsado la tecla ALT y false en otro caso
button	Número entero	El botón del ratón que ha sido pulsado. Posibles valores: 0 – Ningún botón pulsado 1 – Se ha pulsado el botón izquierdo2– Se ha pulsado el botón derecho3– Se pulsan a la vez el botón izquierdo y el derecho4– Se ha pulsado el botón central5– Se pulsan a la vez el botón izquierdo y el central6– Se pulsan a la vez el botón derecho y el central7` – Se pulsan a la vez los 3 botones
cancelable	Boolean	Indica si el evento se puede cancelar
charCode	Número entero	El código unicode del carácter correspondiente a la tecla pulsada
clientX	Número entero	Coordenada X de la posición del ratón respecto del área visible de la ventana
clientY	Número entero	Coordenada Y de la posición del ratón respecto del área visible de la ventana
ctrlKey	Boolean	Devuelve true si se ha pulsado la tecla CTRL y false en otro caso

currentTarget	Element	El elemento que es el objetivo del evento
detail	Número entero	El número de veces que se han pulsado los botones del ratón
isChar	Boolean	Indica si la tecla pulsada corresponde a un carácter
keyCode	Número entero	Indica el código numérico de la tecla pulsada
metaKey	Número entero	Devuelve true si se ha pulsado la tecla META y false en otro caso
pageX	Número entero	Coordenada X de la posición del ratón respecto de la página
pageY	Número entero	Coordenada Y de la posición del ratón respecto de la página
preventDefault()	Función	Se emplea para cancelar la acción predefinida del evento
relatedTarget	Element	El elemento que es el objetivo secundario del evento (relacionado con los eventos de ratón)
screenX	Número entero	Coordenada X de la posición del ratón respecto de la pantalla completa
screenY	Número entero	Coordenada Y de la posición del ratón respecto de la pantalla completa
shiftKey	Boolean	Devuelve true si se ha pulsado la tecla SHIFT y false en otro caso

target	Element	El elemento que origina el evento
timeStamp	Número	La fecha y hora en la que se ha producido el evento
type	Cadena de texto	El nombre del evento

Al contrario de lo que sucede con Internet Explorer, la mayoría de propiedades del objeto event de DOM son de sólo lectura. En concreto, solamente las siguientes propiedades son de lectura y escritura: altKey, button y keyCode.

La tecla META es una tecla especial que se encuentra en algunos teclados de ordenadores muy antiguos. Actualmente, en los ordenadores tipo PC se asimila a la tecla Alt o a la tecla de Windows, mientras que en los ordenadores tipo Mac se asimila a la tecla Command.

Prevenir el comportamiento por defecto

Una de las propiedades más interesantes es la posibilidad de impedir que se complete el comportamiento normal de un evento. En otras palabras, con JavaScript es posible no mostrar ningún carácter cuando se pulsa una tecla, no enviar un formulario después de pulsar el botón de envío, no cargar ninguna página al pulsar un enlace, etc. El método avanzado de impedir que un evento ejecute su acción asociada depende de cada navegador:

```
// Navegadores que siguen los estándares
objetoEvento.preventDefault();
```

TIPOS DE EVENTOS

La lista completa de eventos que se pueden generar en un navegador se puede dividir en cuatro grandes grupos. La especificación de DOM define los siguientes grupos:

- Eventos de ratón: se originan cuando el usuario emplea el ratón para realizar algunas acciones.
- Eventos de teclado: se originan cuando el usuario pulsa sobre cualquier tecla de su teclado.
- Eventos HTML: se originan cuando se producen cambios en la ventana del navegador o cuando se producen ciertas interacciones entre el cliente y el servidor.
- Eventos DOM: se originan cuando se produce un cambio en la estructura DOM de la página. También se denominan "eventos de mutación".

EVENTOS DE RATÓN

Los eventos de ratón son, con mucha diferencia, los más empleados en las aplicaciones web. Los eventos que se incluyen en esta clasificación son los siguientes:

Evento	Descripción
click	Se produce cuando se pulsa el botón izquierdo del ratón. También se produce cuando el foco de la aplicación está situado en un botón y se pulsa la tecla ENTER
dblclick	Se produce cuando se pulsa dos veces el botón izquierdo del ratón
mousedown	Se produce cuando se pulsa cualquier botón del ratón
mouseout	Se produce cuando el puntero del ratón se encuentra en el interior de un elemento y el usuario mueve el puntero a un lugar fuera de ese elemento
mouseover	Se produce cuando el puntero del ratón se encuentra fuera de un elemento y el usuario mueve el puntero hacia un lugar en el interior del elemento
mouseup	Se produce cuando se suelta cualquier botón del ratón que haya sido pulsado
mousemove	Se produce (de forma continua) cuando el puntero del ratón se encuentra sobre un elemento

Todos los elementos de las páginas soportan los eventos de la tabla anterior.

PROPIEDADES

El objeto event contiene las siguientes propiedades para los eventos de ratón:

- Las coordenadas del ratón (todas las coordenadas diferentes relativas a los distintos elementos)
- La propiedad type
- La propiedad srcElement (Internet Explorer) o target (DOM)

- Las propiedades shiftKey, ctrlKey, altKey y metaKey (sólo DOM)
- La propiedad button (sólo en los eventos mousedown, mousemove, mouseout, mouseover y mouseup)

Los eventos mouseover y mouseout tienen propiedades adicionales. Internet Explorer define la propiedad fromElement, que hace referencia al elemento desde el que el puntero del ratón se ha movido y toElement que es el elemento al que el puntero del ratón se ha movido. De esta forma, en el evento mouseover, la propiedad toElement es idéntica a srcElement y en el evento mouseout, la propiedad fromElement es idéntica a srcElement.

En los navegadores que soportan el estándar DOM, solamente existe una propiedad denominada relatedTarget. En el evento mouseout, relatedTarget apunta al elemento al que se ha movido el ratón. En el evento mouseover, relatedTarget apunta al elemento desde el que se ha movido el puntero del ratón.

Cuando se pulsa un botón del ratón, la secuencia de eventos que se produce es la siguiente: mousedown, mouseup, click. Por tanto, la secuencia de eventos necesaria para llegar al doble click llega a ser tan compleja como la siguiente: mousedown, mouseup, click, mousedown, mouseup, click, dblclick.

EVENTOS DE TECLADO

Los eventos que se incluyen en esta clasificación son los siguientes:

Evento	Descripción
keydown	Se produce cuando se pulsa cualquier tecla del teclado. También se produce de forma continua si se mantiene pulsada la tecla
keypress	Se produce cuando se pulsa una tecla correspondiente a un carácter alfanumérico (no se tienen en cuenta teclas como SHIFT, ALT, etc.). También se produce de forma continua si se mantiene pulsada la tecla
keyup	Se produce cuando se suelta cualquier tecla pulsada

PROPIEDADES

El objeto event contiene las siguientes propiedades para los eventos de teclado:

- La propiedad keyCode
- La propiedad charCode (sólo DOM)

- La propiedad srcElement (Internet Explorer) o target (DOM)
- Las propiedades shiftKey, ctrlKey, altKey y metaKey (sólo DOM)

Cuando se pulsa una tecla correspondiente a un carácter alfanumérico, se produce la siguiente secuencia de eventos: keydown, keypress, keyup. Cuando se pulsa otro tipo de tecla, se produce la siguiente secuencia de eventos: keydown, keyup. Si se mantiene pulsada la tecla, en el primer caso se repiten de forma continua los eventos keydown y keypress y en el segundo caso, se repite el evento keydown de forma continua.

EVENTOS HTML

Los eventos HTML definidos se recogen en la siguiente tabla:

Evento	Descripción
load	Se produce en el objeto window cuando la página se carga por completo. En el elemento <img> cuando se carga por completo la imagen. En el elemento` cuando se carga el objeto
unload	Se produce en el objeto window cuando la página desaparece por completo (al cerrar la ventana del navegador por ejemplo). En el elemento <object> cuando desaparece el objeto.
abort	Se produce en un elemento <object> cuando el usuario detiene la descarga del elemento antes de que haya terminado
error	Se produce en el objeto window cuando se produce un error de JavaScript. En el elemento <img> cuando la imagen no se ha podido cargar por completo y en el elemento <object> cuando el elemento no se carga correctamente
select	Se produce cuando se seleccionan varios caracteres de un cuadro de texto (<input> y <textarea>)
change	Se produce cuando un cuadro de texto (<input> y <textarea>) pierde el foco y su contenido ha variado. También se produce cuando varía el valor de un elemento <select>

submit	Se produce cuando se pulsa sobre un botón de tipo submit (<input type="submit">)
reset	Se produce cuando se pulsa sobre un botón de tipo reset (<input type="reset">)
resize	Se produce en el objeto window cuando se redimensiona la ventana del navegador
scroll	Se produce en cualquier elemento que tenga una barra de scroll, cuando el usuario la utiliza. El elemento <body> contiene la barra de scroll de la página completa
focus	Se produce en cualquier elemento (incluido el objeto window) cuando el elemento obtiene el foco
blur	Se produce en cualquier elemento (incluido el objeto window) cuando el elemento pierde el foco

Uno de los eventos más utilizados es el evento load, ya que todas las manipulaciones que se realizan mediante DOM requieren que la página esté cargada por completo y por tanto, el árbol DOM se haya construido completamente.

El elemento <body> define las propiedades scrollLeft y scrollTop que se pueden emplear junto con el evento scroll.

¿Qué es JSON?

Fuente: lenguajejs.com

Cuando trabajamos con mucha cantidad de información, se puede volver necesario aislar el código de programación de los datos. De esta forma, podemos guardar información en un fichero independiente, separado del archivo donde tenemos el código de nuestro programa. Así, si necesitamos actualizar o modificar datos, no tenemos que tocar el código de nuestro programa.

JSON son las siglas de **JavaScript Object Notation**, y no es más que un formato ligero de datos, con una estructura (*notación*) específica, que es totalmente compatible de forma nativa con JavaScript. Como su propio nombre indica, **JSON** se basa en la sintaxis que tiene JavaScript para crear objetos.

Un archivo **JSON** mínimo debe tener la siguiente sintaxis:

```
{
}
```

Esto simplemente es un objeto vacío. Un archivo JSON, puede contener varios tipos de datos:

```
{
  "name": "Manz",
  "life": 99,
  "dead": false,
  "props": ["invisibility", "coding", "happymood"],
  "senses": {
    "vision": 50,
    "audition": 75,
    "taste": 40,
    "smell": 50,
    "touch": 80
  }
}
```

Como se puede ver, en **JSON** todos los textos deben estar entrecomillados con «comillas dobles», y solo se pueden utilizar tipos de datos como **string**, **number**, **object**, **array**, **boolean** o **null**. Un valor **null**, simplemente, también sería un **JSON** válido.

> **OJO**: JSON no permite utilizar tipos de datos como **function**, **regexp** o valores **undefined**. Tampoco es válido incluir comentarios en un JSON.

Mucho cuidado con las **comillas mal cerradas** o las **comas sobrantes** (*antes de un cierre de llaves, por ejemplo*). Suelen ser motivos de error de sintaxis frecuentemente. Si tienes dudas sobre si la sintaxis del JSON que has construido es correcta, puedes utilizar JSONLint, una página que te permitirá pegar el código JSON y validarlo para saber si es correcto. También nos puede servir para indentar correctamente el JSON.

¿Cómo utilizar JSON?

Si analizamos bien la sintaxis de un JSON, nos daremos cuenta que es muy similar a algo a lo que ya deberíamos estar acostumbrados:

```
const o = {
  name: "Manz",
  life: 99,
};
```

Simplemente añadiendo **const o =** al principio, nos daremos cuenta (*si no era evidente ya*) de que se trata de un **objeto** de Javascript y que no debería ser muy sencillo pasar de JSON a Javascript y viceversa.

En Javascript tenemos una serie de métodos que nos facilitan esa tarea, pudiendo trabajar con **strings** que contengan JSON y objetos Javascript de forma indiferente:

Método	Descripción
JSON.parse(str)	Convierte el texto **str** (un JSON válido) a un objeto y lo devuelve.
JSON.stringify(obj)	Convierte un objeto Javascript **obj** a su representación JSON y la devuelve.

Convertir JSON a Objeto

La acción de **convertir JSON a objeto** Javascript se le suele denominar **parsear**. Es una acción que analiza una **string** que contiene un **JSON** válido y devuelve un objeto Javascript

con dicha información correctamente estructurada. Para ello, utilizaremos el método **JSON.parse()**:

```
const str = '{ "name": "Manz", "life": 99 }';

const obj = JSON.parse(str);

obj.name; // 'Manz'
obj.life; // 99
```

Como se puede ver, **obj** es un objeto generado a partir del JSON recogido en la variable **str** y podemos consultar sus propiedades y trabajar con ellas sin problemas.

Convertir Objeto a JSON

La acción inversa, **convertir un objeto JavaScript a JSON** también se puede realizar fácilmente haciendo uso del método **JSON.stringify()**. Este método difícil de pronunciar viene a ser algo así como «convertir a texto», y lo podemos utilizar para transformar un objeto de JavaScript a **JSON** rápidamente:

```
const obj = {
  name: "Manz",
  life: 99,
  saludar: function () {
    return "Hola!";
  },
};

const str = JSON.stringify(obj);

str; // {"name":"Manz","life":99}'
```

Observa que, como habíamos dicho, las funciones no están soportadas por **JSON**, por lo que si intentamos convertir un objeto que contiene métodos o funciones, **JSON.stringify()** no fallará, pero simplemente devolverá una **string** omitiendo las propiedades que contengan funciones.

localStorage y sessionStorage: ¿qué son?

El objeto Storage (API de almacenamiento web) nos permite almacenar datos de manera local en el navegador y sin necesidad de realizar alguna conexión a una base de datos.

localStorage y **sessionStorage** son propiedades que acceden al objeto Storage y tienen la función de almacenar datos de manera local, la diferencia entre éstas dos es que localStorage almacena la información de forma indefinida o hasta que se decida limpiar los datos del navegador y sessionStorage almacena información mientras la pestaña donde se esté utilizando siga abierta, una vez cerrada, la información se elimina.

Validar objeto Storage en el navegador

Aunque gran parte de los navegadores hoy en día son compatibles con el objeto Storage, no está de más hacer una pequeña validación para rectificar que realmente podemos utilizar dicho objeto, para ello utilizaremos el siguiente código:

```
if (typeof(Storage) !== 'undefined') {
  // Código cuando Storage es compatible
} else {
 // Código cuando Storage NO es compatible
}
```

Guardar datos en Storage

Existen dos formas de guardar datos en Storage, que son las siguientes:

localStorage

```
// Opción 1 ->  localStorage.setItem(name, content)
// Opción 2 ->localStorage.name = content
// name = nombre del elemento
// content = Contenido del elemento

localStorage.setItem('Nombre', 'Miguel Antonio')
localStorage.Apellido = 'Márquez Montoya'
```

sessionStorage

```
// Opción 1 ->  sessionStorage.setItem(name, content)
// Opción 2 ->sessionStorage.name = content
// name = nombre del elemento
// content = Contenido del elemento

sessionStorage.setItem('Nombre', 'Miguel Antonio')
sessionStorage.Apellido = 'Márquez Montoya'
```

Recuperar datos de Storage

Al igual que para agregar información, para recuperarla tenemos dos maneras de hacerlo

localStorage

```
// Opción 1 -> localStorage.getItem(name, content)
// Opción 2 -> localStorage.name

// Obtenemos los datos y los almacenamos en variables
let firstName = localStorage.getItem('Nombre'),
    lastName  = localStorage.Apellido

console.log(`Hola, mi nombre es ${firstName} ${lastName}`)
// Imprime: Hola, mi nombre es Miguel Antonio Márquez Montoya
```

sessionStorage

```
// Opción 1 -> sessionStorage.getItem(name, content)
// Opción 2 -> sessionStorage.name

// Obtenemos los datos y los almacenamos en variables
let firstName = sessionStorage.getItem('Nombre'),
    lastName  = sessionStorage.Apellido

console.log(`Hola, mi nombre es ${firstName} ${lastName}`)
// Imprime: Hola, mi nombre es Miguel Antonio Márquez Montoya
```

Eliminar datos de Storage

Para eliminar un elemento dentro de Storage haremos lo siguiente:

localStorage

```
// localStorage.removeItem(name)
localStorage.removeItem(Apellido)
```

sessionStorage

```
// sessionStorage.removeItem(name)
sessionStorage.removeItem(Apellido)
```

Limpiar todo el Storage

Ya para finalizar veremos la forma para eliminar todos los datos del Storage y dejarlo completamente limpio

localStorage

localStorage.clear()

sessionStorage

sessionStorage.clear()

Guardar y mostrar múltiples datos desde local/session Storage

La clave está en que **localStorage solo nos permite guardar un *string***. Así que primero debemos convertir nuestro objeto a string con *JSON.stringify()* como en el siguiente ejemplo:

```
let datos = ['html5','css3','js'];
// Guardo el objeto como un string
localStorage.setItem('datos', JSON.stringify(datos));
```

```
// Obtener el string previamente salvado
let datos = localStorage.getItem('datos');
console.log('Datos: ', JSON.parse(datos));
```

Las características de *Local Storage* y *Session Storage* son:

- Permiten almacenar entre 5MB y 10MB de información; incluyendo texto y multimedia
- La información está almacenada en la computadora del cliente y NO es enviada en cada petición del servidor, a diferencia de las cookies
- Utilizan un número mínimo de peticiones al servidor para reducir el tráfico de la red
- Previenen pérdidas de información cuando se desconecta de la red
- La información es guardada por domino web (incluye todas las páginas del dominio)