4.

Interacción con el usuario: eventos y formularios

Contenidos

1. Eventos

El modelo de **programación basada en eventos** consiste en que los scripts o programas esperan sin realizar ninguna tarea hasta que se produzca un evento. Una vez producido, ejecutan alguna tarea asociada a ese evento y, cuando concluye la tarea, el script o programa vuelve al estado de espera.

JavaScript permite realizar scripts siguiendo este modelo, que facilita la interacción con el usuario.

2. Modelos de eventos

Existen tres modelos de eventos, que dependen del navegador: el **modelo básico de eventos**, el **modelo de eventos estándar** y el **modelo de eventos de Internet Explorer**.

Veremos el **modelo básico de eventos**, que es el único compatible con todos los navegadores y, por tanto, el único que permite crear aplicaciones que funcionan de la misma manera en todos ellos.

NOTA: Los navegadores modernos tienden a cumplir los estándares de la Web, pero pueden seguir utilizándose navegadores más antiguos que no los soportan completamente o que los ignoran. Esto debe tenerse en cuenta, pues pueden producirse incompatibilidades entre distintos navegadores en los lenguajes HTML, CSS y/o JavaScript.

3. Modelo básico de eventos

En este modelo, cada elemento HTML tiene definida una lista de eventos a los que puede responder.

El nombre de cada evento se construye mediante el prefijo **on**, seguido del nombre de la acción asociada al evento. Por ejemplo, el evento de pulsar 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 algunos de los eventos definidos por JavaScript:

Evento	Descripción	Elementos en que está definido
onfocus	Seleccionar un elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onblur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
onclick	Pulsar y soltar el ratón	Todos los elementos
ondblclick	Pulsar dos veces seguidas el ratón y soltar	Todos los elementos

Evento	Descripción	Elementos en que está definido
onkeydown	Pulsar (presionar) una tecla	Elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onkeypress	Pulsación completa de una tecla	Elementos de formulario y <body>
onload	La página se ha cargado completamente	<body>
onmousedown	Pulsar (presionar) un botón del ratón	Todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	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
onselect	Seleccionar un texto	<input>, <textarea>
onreset	Inicializar el formulario (borrar todos sus datos)	<form>
onsubmit	Enviar el formulario	<form>

Las acciones típicas que realiza un usuario en una página web pueden dar lugar a una sucesión de eventos. Por ejemplo, si se pulsa sobre un botón de tipo **submit** se desencadenan los eventos **onmousedown**, **onmouseup**, **onclick** y **onsubmit**.

4. Manejadores de eventos

Para que los eventos resulten útiles, se les debe asociar código. Así, cuando se produce un evento, se ejecuta el código asociado y la aplicación puede responder al evento.

El código asociado a un evento, cuya finalidad es atender al evento cuando ocurra, se denomina **manejador de evento** o “**event handler**”. En JavaScript existen varias formas de asociar manejadores de eventos a los eventos:

- Manejadores como atributos de los elementos HTML.
- Manejadores como funciones externas.
- Manejadores semánticos.

4.1. Manejadores como atributos de los elementos HTML

El código se incluye en el propio elemento HTML, como valor de un atributo con el mismo nombre que el evento que se quiere capturar. Este suele ser el método **menos recomendado**.

Ejemplo

Se quiere mostrar un mensaje cuando el usuario pulse un botón. El nombre del evento es **onclick**, por lo que se incluye un atributo llamado **onclick** en el elemento HTML. El valor del atributo son las instrucciones JavaScript que hay que ejecutar cuando se produzca el evento.

```
<input type="button" value="Pulsa" onclick="alert('Has pulsado');">
```

4.1.1. Manejadores como atributos y la variable this

En estos manejadores de eventos puede utilizarse la variable **this**, que JavaScript crea automáticamente. En este caso, la variable **this** equivale al elemento que ha provocado el evento.

Ejemplo

Cuando el usuario pase el ratón por encima de un <**div**>, el texto de su interior se ha de mostrar en negrita. Cuando el ratón salga del <**div**>, el texto se ha mostrar de nuevo con un peso normal.

El elemento <**div**> original es el siguiente:

```
<div id="contenidos" style="width:150px; height:60px; font-weight:normal;">
   ...
</div>
```

Si no se utiliza la variable **this**, el código para modificar el peso de la fuente sería el siguiente:

```
<div id="contenidos" style="width:150px; height:60px; font-weight:normal;"
onmouseover="document.getElementById('contenidos').style.fontWeight='bold';"
onmouseout="document.getElementById('contenidos').style.fontWeight='normal';">
   ...
</div>
```

Utilizando la variable **this** puede simplificarse el código anterior, pues dentro del manejador de eventos, la variable **this** equivale al elemento que ha provocado el evento, en este caso el <**div**>:

```
<div id="contenidos" style="width:150px; height:60px; font-weight:normal;"
onmouseover="this.style.fontWeight='bold';"
onmouseout="this.style.fontWeight='normal';">
   ...
</div>
```

El código anterior es más compacto, más fácil de leer y de escribir, y funcionará correctamente aunque se modifique el valor del atributo **id** del <**div**>.

4.2. Manejadores como funciones externas

Cuando el código de un manejador es más complejo que unas pocas instrucciones, es más recomendable incluir su código en una función externa (llamada "**función manejadora**") y llamar a esta función desde el elemento HTML cuando se produce el evento. Además, de este modo, el código de los manejadores de eventos puede ser **reutilizado** (pues son funciones).

Ejemplo

```
<input type="button" value="Pulsa" onclick="alert('Has pulsado');">
```

se puede transformar en:

```
// Función externa
function muestraMensaje() {
   alert('Gracias por pulsar');
}

<input type="button" value="Pulsa" onclick="muestraMensaje();">
```

4.2.1. Manejadores como funciones externas y la variable this

En las funciones externas la variable **this no está disponible**. Por tanto, si quiere hacerse uso de ella, es necesario pasar la variable **this** como parámetro a la función externa.

Ejemplo

```
function resalta(elemento) {   // elemento almacenará el <div>
   switch(elemento.style.fontWeight) {
      case 'normal':
         elemento.style.fontWeight = 'bold';
         break;
      case 'bold':
         elemento.style.fontWeight = 'normal';
         break;
   }
}

<div style="width:150px; height:60px; font-weight:normal;"
onmouseover="resalta(this);" onmouseout="resalta(this)">
   ...
</div>
```

En el ejemplo anterior, **this** está disponible en el elemento HTML pero no en la función externa. Para acceder desde la función externa al elemento sobre el que se ha producido el evento, se le pasa **this** como parámetro en la llamada a la función, que lo recibirá en el argumento llamado **elemento**.

5. Manejadores de eventos semánticos

Utilizar los atributos HTML o las funciones externas para añadir manejadores de eventos “ensucia” el código HTML de la página.

Una buena práctica al crear páginas y aplicaciones web es separar el contenido (HTML) de la presentación (CSS). Siempre que sea posible, también se recomienda separar el contenido (HTML) del comportamiento (JavaScript), pues esto facilita la comprensión del código y su mantenimiento.

Los manejadores de eventos semánticos permiten separar el código JavaScript del marcado HTML. Esta técnica consiste en asignar funciones manejadoras mediante los métodos y propiedades del DOM de los elementos HTML.

Así, el siguiente ejemplo:

```
<input id="boton" type="button" value="Pulsa" onclick="alert('Has pulsado');">
```

Se puede transformar en:

```
// Función manejadora
function muestraMensaje() {
   alert('Gracias por pulsar');
}

// Asignar la función manejadora al elemento
document.getElementById('boton').onclick = muestraMensaje;
// Elemento HTML
<input id="boton" type="button" value="Pulsa">
```

Como se puede observar, el código HTML resultante no se mezcla con el código JavaScript.

La técnica de los manejadores semánticos consiste en:

1. Establecer una manera de identificar al elemento o elementos HTML (como el atributo **id**).
2. Crear una función manejadora encargada de manejar el evento.
3. Asignar la función manejadora al evento correspondiente del elemento HTML mediante los métodos y propiedades del DOM relacionados con eventos.

Asignar la función manejadora mediante el DOM es la clave de esta técnica. En primer lugar, se obtiene la referencia al elemento o elementos a los que se va a asignar el manejador:

```
document.getElementById('boton')
```

A continuación, se utiliza la propiedad del elemento con el mismo nombre que el evento que se quiere manejar. En este caso, la propiedad es **onclick**:

```
document.getElementById('boton').onclick = ...
```

Por último, se asigna la función manejadora. La asignación de una función a una variable o a una propiedad de un objeto **se hace escribiendo sólo el nombre de la función, SIN PARÉNTESIS** (o bien definiendo la función en la propia asignación, como **función anónima**).

Los paréntesis tras el nombre de una función son el **operador** de llamada a la función. Si se añaden, se llama a la función y se ejecuta, y lo que se asigna al evento es el valor devuelto por ella (o **null**).

```
// Asignar una función manejadora a un evento de un elemento
document.getElementById('boton').onclick = muestraMensaje;

// Ejecutar una función y asignar lo que devuelve (o null)
document.getElementById('boton').onclick = muestraMensaje();   // ¡ERROR!
```

Otra ventaja de esta técnica es que en las funciones manejadoras está disponible la variable **this**, que también equivale en ellas al **elemento que ha provocado el evento**.

```
// Función manejadora
function muestraMensaje() {
   this.value = 'Pulsado';   // this hará referencia al botón pulsado
}

// Asignar la función manejadora al elemento
document.getElementById('boton').onclick = muestraMensaje;

// Elemento HTML
<input id="boton" type="button" value="Pulsa">
```

Como los manejadores semánticos se asignan a los elementos HTML utilizando métodos y propiedades del DOM, sólo pueden utilizarse después de que la página se haya cargado **por completo** (que es cuando los nodos del árbol DOM están disponibles).

Una una manera de asegurarnos de esto es incluir el código JavaScript que accede al DOM antes del cierre del elemento <**body**> del documento.

Otra forma es utilizando el evento **onload** del objeto **window**. Este evento se produce después de que toda la página se ha cargado, es decir, después de que se ha descargado su código HTML, sus imágenes, los estilos, y cualquier otro objeto o recurso incluido en la página.

De este modo, el código JavaScript que accede al árbol DOM puede incluirse en cualquier lugar del documento (típicamente en el elemento <**head**>).

```
window.onload = function() {   // Se ejecutará tras cargarse la página
   document.getElementById('boton').onclick = muestraMensaje;
}
```

En el código anterior se utiliza la técnica de los manejadores semánticos **1)** para asignar una función manejadora (anónima) al evento **onload** del objeto **window**, y **2)** para asignar una función manejadora para el evento **onclick** del elemento con **id="boton"**.

En lugar de una función anónima, puede asignarse una función externa:

```
// Función principal
function main() {
   document.getElementById('boton').onclick = muestraMensaje;
}

window.onload = main;
```

5.1. Paso de parámetros a manejadores semánticos

Al asignar una función manejadora (con nombre) a un evento y tener que especificar sólo el nombre de la función, **sin paréntesis**, podemos ver que se impide llamar a la función manejadora pasándole parámetros. Una manera de solucionar este problema se muestra en el siguiente ejemplo.

Se asigna como función manejadora del evento una función anónima (sin parámetros o con un sólo parámetro donde JavaScript almacena un objeto **Event**), y se hace una llamada "normal" a la función manejadora original, en este caso con parámetros, dentro de la función anónima (la función anónima es una función como cualquier otra, y podemos escribir cualquier código en ella).

Ejemplo

```
// Función manejadora
function muestraMensaje1(parámetro1, ..., parámetroN) {
   // Código JavaScript de la función
}
```

```
window.onload = function() {
   let boton = document.getElementById('boton');

   boton.onclick = function() {
      muestraMensaje1(parámetro1, ..., parámetroN);   // Llamada "normal"
   }
}
```

6. Listeners

Otra forma de asignar manejadores de eventos a los eventos es mediante unos métodos llamados "**listeners**" o "**event listeners**". Estos métodos permiten asignar más de un manejador a un mismo evento de un elemento y eliminar manejadores previamente asignados como **listeners**.

En los siguientes ejemplos se emplea la siguiente página HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
   <meta charset="utf-8">
   <title>Ejemplo de Event Listeners</title>
</head>
<body>
   <div id="principal">Pulsa aquí</div>
</body>
</html>
```

Todos los elementos disponen de dos métodos para asociar y "desasociar" manejadores de eventos como "listeners": **addEventListener()** y **removeEventListener()**.

Estos métodos requieren tres argumentos: **1)** el nombre del evento (como cadena de caracteres), **2)** la función manejadora del evento y **3)** el tipo de flujo de eventos al que se aplica.

El primer argumento es el nombre del evento sin el prefijo **on**. Es decir, si en los elementos HTML se utiliza el nombre **onclick**, ahora debe utilizarse **click**.

El segundo argumento, la función manejadora, se asigna como es habitual (mediante su nombre **sin paréntesis** o definiéndola como función anónima).

El tercer argumento es opcional, y por defecto es **false**. Tiene que ver con un mecanismo llamado **flujo de eventos** o "**event flow**", que permite que varios elementos diferentes puedan responder a un mismo evento. Lo dejaremos sin especificar o con el valor **false**.

Cuando se utilizan manejadores de eventos semánticos suele recomendarse usar estos métodos.

Ejemplo

```
// Manejador de eventos
function muestraMensaje() {
   alert('Has pulsado el ratón');
}

window.onload = function() {

   // Se obtiene una referencia a un elemento
   let elemento = document.getElementById('principal');
   // Se asocia la función manejadora al evento "click" del elemento
   elemento.addEventListener('click', muestraMensaje);
   ...
   // Más adelante se decide desasociar la función del evento
   elemento.removeEventListener('click', muestraMensaje);
}
```

Si se quiere asociar más de una función manejadora al mismo evento:

```
// Manejador de eventos 1
function muestraMensaje() {
   alert('Has pulsado el ratón');
}

// Manejador de eventos 2
function muestraOtroMensaje() {
   alert('Has pulsado el ratón y por eso se muestran estos mensajes');
}

window.onload = function() {

   // Se obtiene una referencia a un elemento
   let elemento = document.getElementById('principal');

   // Se asocian varias funciones manejadoras al evento "click" del elemento
   elemento.addEventListener('click', muestraMensaje);
   elemento.addEventListener('click', muestraOtroMensaje);
}
```

Al pulsar sobre el ** **, se ejecutarán los dos manejadores en el orden en que han sido asignados al evento.

Sin embargo, no es posible asignar múltiples eventos mediante las propiedades del DOM:

```
window.onload = function() {
   let elemento = document.getElementById('principal');

   elemento.onclick = muestraMensaje;

   // Sólo se asignará este manejador ("onclick" es una propiedad, una variable)
   elemento.onclick = muestraOtroMensaje;
}
```

Otra forma de asegurar que cierto código se ejecute después de que la página se haya cargado es utilizar el evento **DOMContentLoaded** del objeto **document**. Este evento se produce después de se ha cargado completamente el HTML y el árbol DOM, pero antes de que se hayan descargado las imágenes, los estilos, y otros objetos o recursos incluidos en la página. Sólo puede utilizarse con el método **addEventListener()**.

Ejemplo

```
// Función principal
function main() {
   ...
}

document.addEventListener('DOMContentLoaded', main);
// No "document.onDOMContentLoaded = ..."   // ¡ERROR!
```

7. El objeto Event

Las funciones manejadoras pueden necesitar información relativa al evento producido: la tecla pulsada, la posición del ratón, el elemento que ha producido el evento, etc.

Esta información se almacena en un objeto llamado **Event**, que se crea automáticamente cuando se produce un evento y se destruye de forma automática cuando se han ejecutado todas las funciones manejadoras asociadas al evento.

El estándar DOM especifica que el **único parámetro** que se debe pasar a las funciones manejadoras de eventos es el objeto **Event**. Por tanto, se puede acceder al objeto **Event** incluyendo un parámetro en la definición de las funciones manejadoras que lo necesiten. A este parámetro se le asignará automáticamente el objeto **Event** cuando se llame a la función:

```
elemento.onclick = function(evento) {
   ...
}
```

Si queremos pasar más de un parámetro a una función externa, podemos realizar una llamada con todos los parámetros desde el interior de una función anónima.

```
function manejador(evento, numero) {
   ...
}

elemento.onclick = function(evento) {
   let x = 7;
   manejador(evento, x);
}
```

Las funciones manejadoras también tienen acceso al objeto **Event** sin recibirlo en un parámetro, pues está disponible en su interior como propiedad **event** del objeto **window** (**window.event**) o directamente como **event** (esto puede depender del navegador utilizado):

```
elemento.onclick = function() {
   // Si no se encuentra en "window.event" se busca como "event"
   let evento = window.event || event;   // O directamente event
}
```

7.1. Propiedades y métodos definidos por DOM

La siguiente tabla recoge algunas propiedades (la mayoría de sólo lectura) y métodos definidos para el objeto **Event** en los navegadores modernos que siguen los estándares. Algunas de ellas pueden aparecer o no (o pueden aparecer otras) dependiendo del tipo de evento:

Propiedad/Método	Devuelve	Descripción
type	String	El nombre del evento
button	Entero	En algunos eventos, el botón del ratón que ha sido pulsado
charCode	Entero	El código Unicode del carácter correspondiente a la tecla pulsada
key	String	El carácter pulsado
cancelable	Boolean	Indica si el evento se puede cancelar

Propiedad/Método	Devuelve	Descripción
preventDefault()	Función	Se emplea para cancelar la acción predefinida del evento
stopPropagation()	Función	Se emplea para detener el flujo de eventos de tipo **bubbling**
target	Element	El elemento que origina el evento
currentTarget	Element	El elemento que es el objetivo del evento
relatedTarget	Element	El elemento que es el objetivo secundario del evento (relacionado con los eventos de ratón)
timeStamp	Número	La fecha y hora en la que se ha producido el evento
pageX	Entero	Coordenada X de la posición del ratón respecto de la página
pageY	Entero	Coordenada Y de la posición del ratón respecto de la página
offsetX	Entero	Coordenada X de la posición del ratón respecto del elemento que origina el evento
offsetY	Entero	Coordenada Y de la posición del ratón respecto del elemento que origina el evento

8. Formularios

La programación de aplicaciones con formularios web siempre ha sido una tarea fundamental de JavaScript, sobre todo para validar los datos de los formularios directamente en el navegador.

9. Acceso a los formularios y sus elementos

Podemos obtener los datos de un formulario de varias maneras, pero la más recomendable consiste en acceder a los formularios y a sus elementos utilizando los métodos del DOM de acceso directo a los nodos: **getElementById()**, **querySelector()** o **querySelectorAll()**.

10. Propiedades básicas de los formularios

Cada **elemento de formulario** dispone de las siguientes propiedades:

- **type**: indica el tipo de elemento de que se trata.

  Para los elementos de tipo **<input>** (text, button, checkbox, etc.) coincide con el valor de su atributo HTML **type**.

Para las listas desplegables normales (elemento <**select**>) su valor es "**select-one**", y para las listas que permiten seleccionar varios elementos a la vez su valor es "**select-multiple**".

En los elementos de tipo <**textarea**>, el valor de **type** es "**textarea**".

- **form**: es una referencia directa al formulario al que pertenece el elemento.
- **name**: obtiene el valor del atributo **name** de HTML (no se permite modificarlo).
- **value**: permite leer y modificar el valor del atributo **value** de HTML.

  Para los campos de texto (<**input type="text"**> y <**textarea**>) contiene y permite modificar el texto que ha escrito el usuario.

  Para los botones contiene y permite modificar el texto que se muestra en el botón.

  Para los elementos **checkbox** y **radiobutton** no es muy útil.

11. Eventos más utilizados en los formularios

Algunos de los eventos más utilizados en los elementos de formulario son los siguientes:

- **onclick**: se produce cuando se pulsa con el ratón sobre un elemento. Normalmente se utiliza con cualquiera de los tipos de botones que permite definir HTML (<**input type="button"**>, <**input type="submit"**>, <**input type="image"**>).
- **onchange**: se produce cuando el usuario cambia el valor de un elemento de texto (<**input type="text"**> o <**textarea**>). También se produce cuando el usuario selecciona una opción en una lista desplegable (<**select**>) o selecciona un fichero (<**input type="file"**>).

  Generalmente, el evento se produce cuando el usuario pasa a otro campo del formulario tras realizar el cambio (cuando el campo de origen "pierde el foco").
- **onfocus**: se produce cuando el usuario selecciona un elemento del formulario.
- **onblur**: es el complementario de **onfocus**, pues se produce cuando el usuario "deselecciona" un elemento por haber seleccionado otro elemento del formulario.
- **onsubmit**: se produce cuando se envía el formulario (evento del formulario <**form**>).

12. Utilidades básicas de formularios

12.1. Obtener el valor de los campos de formulario

12.1.1. Cuadros de texto y textarea

El valor del texto mostrado por estos elementos se obtiene y se modifica con la propiedad **value**.

Ejemplo

```
<input type="text" id="texto" name="texto">
...
let valor = document.getElementById('texto').value;

<textarea id="parrafo" name="parrafo"></textarea>
...
let valor = document.getElementById('parrafo').value;
```

12.1.2. Botones de radio (“radio buttons”)

Un grupo de radio buttons se distingue porque los radio buttons que lo forman comparten el mismo valor en su atributo **name**. En un grupo de radio buttons, generalmente se quiere saber cuál de ellos se ha seleccionado.

Los radio buttons de un grupo son mutuamente excluyentes (sólo se puede seleccionar uno de ellos). La propiedad **checked** devuelve **true** para el radio button seleccionado y **false** en otro caso.

Por ejemplo, si se dispone del siguiente grupo de radio buttons:

```
<input type="radio" name="pregunta" id="pregunta_si" value="si"> SI
<input type="radio" name="pregunta" id="pregunta_no" value="no"> NO
<input type="radio" name="pregunta" id="pregunta_nsnc" value="nsnc"> NS/NC
<input type="button" id="boton" name="boton" value="Pulsa">
```

El siguiente código permite determinar si un radio button ha sido seleccionado o no:

```
function manejador() {

   let elementos = document.querySelectorAll("[name='pregunta']");

   for (let i = 0; i < elementos.length; i++) {
      if (elementos[i].checked) {
         alert('Elemento: ' + elementos[i].value + ' seleccionado.');
         break;   // Salir del bucle, no es necesario seguir comprobando
      }
   }
}

window.onload = function() {

   document.getElementById('boton').onclick = manejador;
}
```

12.1.3. Casillas de verificación (“check boxes”)

Los check boxes son similares a los radio buttons, pero pueden seleccionarse independientemente del resto. En este caso debe comprobarse cada checkbox para saber si se ha seleccionado.

Si se dispone de los siguientes check boxes:

```
<input type="checkbox" id="condiciones" name="condiciones" value="condiciones">
He leído y acepto las condiciones
<input type="checkbox" id="privacidad" name="privacidad" value="privacidad"> He
leído la política de privacidad
```

Utilizando la propiedad **checked**, es posible comprobar si cada checkbox ha sido seleccionado. La propiedad **checked** devuelve **true** si el checkbox está seleccionado y **false** en otro caso.

```
let elemento = document.getElementById('condiciones');
alert('Elemento: ' + elemento.value + '\n Seleccionado: ' + elemento.checked);

elemento = document.getElementById('privacidad');
alert('Elemento: ' + elemento.value + '\n Seleccionado: ' + elemento.checked);
```

12.1.4. Listas desplegables (select)

Si se dispone de una lista desplegable como la siguiente:

```
<select id="opciones" name="opciones">
   <option value="1">Primer valor</option>
   <option value="2">Segundo valor</option>
   <option value="3">Tercer valor</option>
</select>
```

Cada vez que se selecciona un nuevo elemento de la lista se produce el evento “**onchange**”. En general, lo que se quiere es obtener el valor del atributo **value** de la opción **<option>** seleccionada.

Para obtener el valor del atributo **value** de la opción seleccionada, puede consultarse directamente la propiedad **value** de la lista:

```
// Obtener la referencia a la lista
let lista = document.getElementById('opciones');

// Obtener el valor (value) de la opción seleccionada
let valorSeleccionado = lista.value;
```

Para obtener el elemento **<option>** seleccionado, pueden utilizarse las siguientes propiedades:

- **options**: es un array que contiene todos los elementos **<option> de la lista**. El primer **<option>** puede obtenerse con **document.getElementById("id_de_la_lista").options[0]**.
- **selectedIndex**: guarda el **índice** (posición) del array **options** que le corresponde al **<option>** seleccionado en la lista. Cuando se selecciona una opción, se actualiza automáticamente el valor de esta propiedad con el índice correspondiente al **<option>** seleccionado.

Una vez se tiene el **<option>** seleccionado, puede accederse a su atributo **value** (con la propiedad **value** del elemento) o accederse a su contenido (con la propiedad **text**).

Ejemplo

```
// Obtener la referencia a la lista
let lista = document.getElementById('opciones');

// Obtener el índice en el array "options" del <option> seleccionado
let indiceOptionSeleccionado = lista.selectedIndex;

// Con el índice y el array "options", obtener el <option> seleccionado
let optionSeleccionado = lista.options[indiceOptionSeleccionado];

// Obtener el valor (value) del <option> seleccionado
let valorSeleccionado = optionSeleccionado.value;

// Obtener el contenido del <option> seleccionado (el texto que muestra)
let textoSeleccionado = optionSeleccionado.text;
```

No obstante, normalmente se abrevian los pasos anteriores:

```
// Obtener el valor (value) del <option> seleccionado
let valorSeleccionado = lista.options[lista.selectedIndex].value;

// Obtener el contenido del <option> seleccionado (el texto que muestra)
let textoSeleccionado = lista.options[lista.selectedIndex].text;
```

No hay que confundir el valor de la propiedad **selectedIndex** con el valor de la propiedad **value** del **<option>** seleccionado. En el ejemplo, el primer **<option>** tiene un **value** igual a **1**. Pero si se selecciona esta opción, el valor de **selectedIndex** será **0**, al ser la primera opción del array **options**.

12.2. Establecer el foco en un elemento del formulario

Cuando un elemento está seleccionado y se puede escribir directamente en él o se puede modificar alguna de sus propiedades, se dice que “tiene el foco” del programa.

Si un cuadro de texto de un formulario tiene el foco, el usuario puede escribir directamente en él sin tener que pulsar previamente con el ratón en su interior. Igualmente, si una lista desplegable tiene el foco, se puede seleccionar una opción directamente subiendo y bajando con las flechas del teclado.

Al pulsar repetidamente la tecla **tabulador** sobre una página web, los diferentes elementos van obteniendo el foco (el elemento seleccionado suele mostrar un pequeño borde punteado).

NOTA: Si en una página web un formulario es el elemento más importante, como en una página de búsqueda o en una página para registrarse, se considera una buena práctica de usabilidad asignar automáticamente el foco al primer elemento del formulario cuando se carga la página.

Para asignar el foco a un elemento de HTML, se utiliza el método **focus()**.

```
document.getElementById('primero').focus();
```

```
<form id="formulario" action="#">
   <input type="text" id="primero" name="primero">
</form>
```

12.3. Evitar el envío duplicado de un formulario

Uno de los problemas con los formularios web es que el usuario pulse dos o más veces seguidas el botón de envío. Si la conexión es demasiado lenta o la respuesta del servidor tarda, el formulario sigue mostrándose en el navegador y el usuario tiene la tentación de volver a pulsar este botón.

El problema no suele ser grave y es posible controlarlo en el servidor, pero puede complicarse en formularios de aplicaciones importantes como las que implican transacciones económicas. Por esto, es una buena práctica deshabilitar el botón de envío después de la primera pulsación.

Ejemplo

```
function manejador() {
   this.disabled = true;   // Deshabilitar el botón
   this.value = "Enviando...";
   this.form.submit();     // Enviar el formulario con su método submit()
}

window.onload = function() {
   document.getElementById('boton').onclick = manejador;
}


<form id="formulario" action="#">
   ...
   <input type="button" id="boton" value="Enviar">   // El botón no es de envío
</form>
```

13. Validación

Validar los datos de los formularios en el cliente: **1)** previene el envío de datos incorrectos al servidor y **2)** permite notificar al usuario de forma inmediata si ha cometido algún error al rellenar un formulario (lo cual evita esperar la respuesta del servidor y tener que recargar la página sólo para informar al usuario de que ha cometido errores al rellenar un formulario).

Esto mejora la satisfacción del usuario con la aplicación (mejora la "experiencia de usuario") y ayuda a reducir la carga de procesamiento en el servidor.

NOTA: Los datos de los formularios deben validarse **siempre en el servidor**, pues el envío de los datos de un formulario puede realizarse por otro programa distinto de un navegador sin intervención del usuario, y esto implica "saltarse" la validación de los datos en el cliente.

Existen tantas posibles comprobaciones como elementos de formulario diferentes, pero algunas de ellas son muy habituales: que se rellene un campo obligatorio, que se seleccione el valor de una lista desplegable, que una dirección de e-mail o una fecha introducida sea correcta, etc.

NOTA: HTML5 introdujo nuevos elementos de formulario que ayudan en la validación de los datos introducidos (<**email**>, <**url**>, etc.).

La validación se basa en el comportamiento de ciertos eventos, como **onsubmit**, **onkeypress** y **onclick**. Estos eventos modifican su comportamiento normal en función del valor que devuelven. Si devuelven **true** (o nada), su comportamiento es el esperado. Si devuelven **false**, se cancela su comportamiento normal.

Por ejemplo, si se devuelve **false** en el evento **onkeypress** de un cuadro de texto, la tecla pulsada se descarta. Si se devuelve en el evento **onclick** de un enlace, el navegador no carga la página a la que apunta. Si se devuelve en el evento **onclick** de un formulario, el formulario no se envía, etc.

También puede cancelarse un evento explícitamente ejecutando su método **preventDefault()** (si es cancelable, lo que puede comprobarse consultando la propiedad **cancelable** del evento).

NOTA: Si se asigna un manejador de eventos con el método **addEventListener()**, devolver **false** para cancelar el evento no surtirá efecto. En ese caso, debe utilizarse el método **preventDefault()** del evento para cancelarlo, o asignar el manejador a la propiedad HTML del evento. Se recomienda utilizar el método **preventDeafult()** si el evento es cancelable, pues funciona independientemente de cómo se asigne el manejador.

Ejemplo

```
<textarea onkeypress="return false;"></textarea>
```

Como el valor devuelto por el evento **onkeypress** es **false**, se cancela el comportamiento por defecto del evento, es decir, la tecla presionada no se transforma en ningún carácter dentro del **textarea** (el **textarea** no permitirá escribir ningún carácter).

Un esquema general de una función de validación de un formulario completo puede ser el siguiente:

```
function validar(e) {   // e recibe el objeto Event del evento producido

   if (no se cumple la condición para el campo 1 del formulario) {
      alert('[ERROR] El campo debe tener un valor de ...');
      e.preventDefault();   // O return false;
   }
   if (no se cumple la condición para el campo 2 del formulario) {
      alert('[ERROR] El campo debe tener un valor de ...');
      e.preventDefault();   // O return false;
   }
   ...
   if (no se cumple la condición para el campo N del formulario) {
      alert('[ERROR] El campo debe tener un valor de ...');
      e.preventDefault();   // O return false;
   }

   // Si el script ha llegado a este punto, todas las condiciones
   // se han cumplido, por lo que se devuelve el valor true (o nada)
   return true;   // O nada
}
```

La función de validación de asigna al evento **onsubmit** del formulario que se desea validar.

```
window.onload = function() {
   document.getElementById('formulario').onsubmit = validar;
}
```

Esta técnica de validación consiste en validar un formulario completo cuando se procede a su envío, comprobando los elementos del formulario que corresponda. En cuanto se encuentra un elemento incorrecto se cancela el evento de envío del formulario (**onsubmit**) con el método **preventDefault()** del evento (o se devuelve **false**) y el formulario no se enviará. Si no se encuentra ningún error, se devuelve **true** (o nada) y el formulario se envía como lo haría normalmente.

Ejemplo

El siguiente formulario solicita la edad del usuario. Si no se introduce una edad comprendida entre los 18 y los 100 años, se cancela el envío del formulario.

```
function validar(e) {

   let edad = parseInt(document.getElementById('edad').value);

   if (isNaN(edad) || (edad < 18 || edad > 100)) {
      alert('[ERROR] El campo debe tener un valor numérico entre 18 y 100');
      e.preventDefault();  // O return false;
   }

   return true;  // O nada
}

window.onload = function() {
   document.getElementById("formulario").onsubmit = validar;
}


<form id="formulario" action="#">
   <input type="text" id="edad" name="edad" value="">
   <input type="submit" id="boton" value="Enviar">  // Botón de envío
</form>
```

Como los campos de formulario disponen de sus propios eventos, también pueden validarse los datos introducidos por el usuario conforme los va introduciendo, informándole de forma inmediata de los errores producidos sin tener que esperar a la validación del formulario completo. Por ejemplo, puede validarse una fecha tras ser introducida, la longitud de un texto tras ser introducido, la opción de una lista tras ser seleccionada, etc.

Esto también permite modificar de forma dinámica el contenido de un formulario dependiendo de los datos que el usuario va introduciendo. Por ejemplo, pueden variar las opciones de una lista en función de la opción seleccionada en otra lista, puede eliminarse un campo del formulario si se marca un determinado checkbox, etc.

En base a lo visto, en la práctica la gestión de errores puede llevarse a cabo de muchas maneras, dependiendo de cómo se deseen notificar los errores al usuario.

Ejemplo

El siguiente formulario solicita una marca y un modelo de producto. Cuando se realiza el envío del formulario, se comprueba si ambos campos están vacíos. Si un campo está vacío, se incluye el mensaje de error correspondiente en un array.

Tras comprobar los campos, si se han detectado errores, se cancela el evento de envío del formulario y se muestran todos los errores en un elemento reservado para este fin. Si no se han detectado errores, se envía el formulario.

```
function validar(e) {
   let errores = [];   // Array para almacenar los mensajes de error
   let marca = document.getElementById('marca').value;
   let modelo = document.getElementById('modelo').value;

   if (marca.trim().length <= 0) {
      errores.push('Debe introducir la marca del producto');
   }
   if (modelo.trim().length <= 0) {
      errores.push('Debe introducir el modelo del producto');
   }
   if (errores.length > 0) {
      e.preventDefault();
      let info = document.getElementById('info_errores');
      info.innerHTML = '';
      for (let error of errores) {
         info.innerHTML += '<p>' + error + '</p>';
      }
   }
}
window.onload = function() {
   document.getElementById('formulario').onsubmit = validar;
}

<form id="formulario" action="#">
   <input type="text" id="marca" name="marca" value="">
   <input type="text" id="modelo" name="modelo" value="">
   <input type="submit" id="boton" value="Enviar">
   <br>
   <div id="info_errores"></div>
</form>
```