

JavaScript

Parte III

Desarrollo de Aplicaciones Web



Eventos

- HTML provee atributos para ejecutar código JavaScript cuando ocurre un evento. Ejem. el atributo onload, onclick
- Configuración de atributos desde JavaScript.
- Los atributos de los elementos se convierten en propiedades de los objetos Element
- Utilizando el objeto Event definiremos los atributos en JavaScript

Ejemplo Event

```
<head>
<script>
function agregarevento() {
var elemento = document.querySelector("section > button");
elemento.onclick = mostrarmensaje;
}
function mostrarmensaje() {
alert("Presionó el botón");
}
window.onload = agregarevento;
</script>
</head>
```

```
<body>
<section>
<h1>Sitio Web</h1>
<button type="button">Mostrar</button>
</section>
</body>
</html>
```

Métodos de Event

- No se recomienda el uso de atributos de evento en elementos HTML porque es contrario al propósito principal de HTML5 que es el de proveer una tarea específica para cada uno de los lenguajes involucrados. HTML debe definir la estructura del documento, CSS su presentación y JavaScript su funcionalidad
- La definición como la del ejemplo anterior, tampoco se recomienda. Por estas razones, se han incluido nuevos métodos en el objeto Window para controlar y responder a eventos.

Métodos de Event

- **addEventListener(evento, listener, captura)**—Este método prepara un elemento para responder a un evento. El primer atributo es el nombre del evento (sin el prefijo on), el segundo atributo es una referencia a la función que responderá al evento (llamada *listener*) y el tercer atributo es un valor booleano que determina si el evento va a ser capturado por el elemento o se propagará a otros elementos (generalmente se ignora o se declara como false).
- **removeEventListener(evento, listener)**—Este método elimina Los nombres de los eventos que requieren estos métodos no son los mismos que los
- **Importante:** los nombres de los atributos no son los mismos que hemos utilizado hasta el momento. Los nombres de los atributos se han definido agregando el prefijo on al nombre real del evento. Por ejemplo, el atributo **onclick** representa el evento **click** el listener de un elemento.

Ejemplo addEventListener

```
<script>
function agregarevento() {
var elemento = document.querySelector("section > button");
elemento.addEventListener("click", mostrarmensaje);
}
function mostrarmensaje() {
alert("Presionó el botón");
}
window.addEventListener("load", agregarevento);
</script>
</head>
```

```
<body>
<section>
<h1>Sitio Web</h1>
<button type="button">Mostrar</button>
</section>
</body>
</html>
```

Depuración

- La depuración (o *debugging*) es el proceso de encontrar y corregir los errores en nuestro código.
- Existen varios tipos de errores, desde errores de programación hasta errores lógicos, e incluso errores personalizados generados para indicar un problema detectado por el mismo código).
- Algunos errores requieren del uso de herramientas para encontrar una solución y otros solo exigen un poco de paciencia y perseverancia.
- Para determinar qué es lo que no funciona en nuestro código es necesario leer las instrucciones una por una y seguir la lógica hasta detectar el error.

Depuración JavaScript - Navegadores

- Los navegadores ofrecen herramientas para ayudarnos a resolver estos problemas, y JavaScript incluye algunas técnicas que podemos implementar para facilitar este trabajo.
- **Consola:** Los tipos de errores que vemos a menudo impresos en la consola son errores de programación.



Ejemplo Depuración

- Por ejemplo, si llamamos a una función inexistente o tratamos de leer una propiedad que no es parte del objeto, se considera un error de programación y se informa de él en la consola.

```
<head>
<script>
funcionfalsa();
</script>
</head>
<body>
<section>
<h1>Sitio Web</h1>
</section>
</body>
</html>
```



Depuración: Objeto Console

- Como ya hemos mencionado, a veces los errores no son errores de programación, sino errores lógicos. El típico cartel, estoy en el while?? 😊
- **Log(valor)**—Este método muestra el valor entre paréntesis en la consola.
- **Assert(condición, valores)**—Este método muestra en la consola los valores que especifican los atributos si la condición especificada por el primer atributo es falsa.
- **Clear()**—Este método limpia la consola. Los navegadores también ofrecen un botón en la parte superior de la consola con la misma funcionalidad.

Ejemplo Objeto Console

- Para imprimir en la consola los valores generados por un bucle en cada ciclo para asegurarnos de que estamos creando la secuencia correcta de números.

```
<script>
```

```
var lista = [0, 5, 103, 24, 81];
```

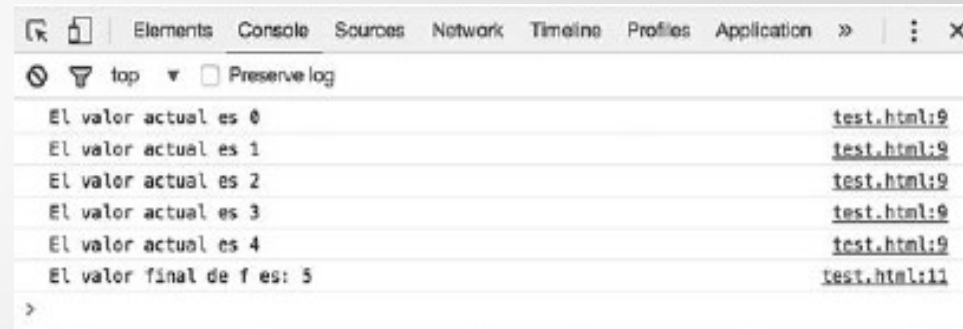
```
for(var f = 0; f < lista.length; f++) {
```

```
  console.log("El valor actual es " + f);
```

```
}
```

```
console.log("El valor final de f es: " + f);
```

```
</script>
```



Evento error

- En ciertos momentos, nos encontraremos con errores que no hemos generado nosotros. A medida que nuestra aplicación crece e incorpora librerías sofisticadas y API, los errores comienzan a depender de factores externos, como recursos que se vuelven inaccesibles o cambios inesperados en el dispositivo que está ejecutando nuestra aplicación.
- Con el propósito de ayudar al código a detectar estos errores y corregirse a sí mismo, JavaScript ofrece el evento error.


Depuración: Evento Error

- **Error**—Esta propiedad devuelve un objeto con información sobre el error.
- **Message**—Esta propiedad devuelve una cadena de caracteres que describe el error.
- **Lineno**—Esta propiedad devuelve la línea en el documento donde ha ocurrido el error.
- **Colno**—Esta propiedad devuelve la columna donde comienza la instrucción que ha producido el error.
- **Filename**—Esta propiedad devuelve la URL del archivo donde ha ocurrido el error.
- Con este evento, podemos programar nuestro código para que responda a errores inesperados.

Ejemplo Evento Error

```
<script>  
function mostrarerror(evento){  
  console.log('Error: ' + evento.error);  
  console.log('Mensaje: ' + evento.message);  
  console.log('Línea: ' + evento.lineno);  
  console.log('Columna: ' + evento.colno);  
  console.log('URL: ' + evento.filename);  
}  
window.addEventListener('error', mostrarerror);  
funcionfalsa();  
</script>
```

Ejemplo Evento error



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays a series of error messages related to a 'ReferenceError' for an undefined function named 'funcionfalsa'. The messages are as follows:

- Error: ReferenceError: funcionfalsa is not defined [test.html:8](#)
- Mensaje: Uncaught ReferenceError: funcionfalsa is not defined [test.html:9](#)
- Linea: 15 [test.html:18](#)
- Columna: 5 [test.html:11](#)
- URL: <file:///Users/macroid/Documents/Web%20Pages/HTML5%20Examples/test.html> [test.html:12](#)
- ▶ Uncaught ReferenceError: funcionfalsa is not defined at [test.html:15](#)

The console also shows a search filter 'top' and a 'Preserve log' checkbox. The bottom of the console shows a prompt character '>'.

Excepciones

- Para manejo de errores, cuando sabemos que existe la posibilidad de falla, se brinda un flujo alternativo de ejecución
- **Throw**—Esta instrucción genera una excepción.
- **Try**—Esta instrucción indica el grupo de instrucciones que pueden generar errores.
- **Catch**—Esta instrucción indica el grupo de instrucciones que deberían ejecutarse si ocurre una excepción.

Ejemplo Excepciones

```
var existencia=5;
function vendido(cantidad) {
  if (cantidad > existencia) {
    var error = {
      name: "ErrorExistencia",
      message: "Sin Existencia" };
    throw error;
  } else {
    existencia = existencia - cantidad; }
  }
  try {
    vendido(8);
  } catch(error) {
    console.log(error.message);
  }
}
```

Librerías y API

- Una librería es una colección de variables, funciones y objetos que realizan tareas en común, como calcular los píxeles que el sistema tiene que activar en la pantalla para mostrar un objeto tridimensional o filtrar los valores que devuelve una base de datos. Las librerías reducen la cantidad de código que un desarrollador tiene que escribir y ofrecen soluciones estándar que funcionan en todos los navegadores.
- Debido a su complejidad, las librerías siempre incluyen una interfaz, un grupo de variables, funciones y objetos que podemos usar para comunicarnos con el código y describir lo que queremos que la librería haga por nosotros. Esta parte visible de la librería se denomina *API* (del inglés, *Application Programming Interface*) y es lo que en realidad tenemos que aprender para poder incluir la librería en nuestros proyectos.

Librerías Nativas

- Lo que convirtió a HTML5 en la plataforma de desarrollo líder que es actualmente no fueron las mejoras introducidas en el lenguaje HTML, o la integración entre este lenguaje con CSS y JavaScript, sino la definición de un camino a seguir para la estandarización de las herramientas que las empresas facilitan por defecto en sus navegadores. Esto incluye un grupo de librerías que se encargan de tareas comunes como la generación de gráficos 2D y 3D, almacenamiento de datos, comunicaciones y mucho más.
- HTML5, ahora los navegadores incluyen librerías eficaces con API integradas en objetos JavaScript y, por lo tanto, disponibles para nuestros documentos. Implementando estas API en nuestro código, podemos ejecutar tareas complejas con solo llamar un método o declarar el valor de una propiedad.

Librerías Externas

- Antes de la aparición de HTML5, se desarrollaron varias librerías programadas en JavaScript para superar las limitaciones de las tecnologías disponibles en el momento.
- Algunas se crearon con propósitos específicos, desde procesar y validar formularios hasta la generación y manipulación de gráficos. Algunas de estas librerías se han vuelto extremadamente populares, y otras como Google Maps, son imposibles de imitar.
- Estas librerías no son parte de HTML5, pero constituyen un aspecto importante del desarrollo web, y algunas de ellas se han implementado en los sitios web y aplicaciones más destacados de la actualidad. Aprovechan todo el potencial de JavaScript y contribuyen al desarrollo de nuevas tecnologías para la Web. La siguiente es una lista de las más populares.

Librerías Externas

- **jQuery** (www.jquery.com) es una librería multipropósito que simplifica el código JavaScript y la interacción con el documento. También facilita la selección de elementos HTML, la generación de animaciones, el control de eventos y la implementación de Ajax en nuestras aplicaciones.
- **React** (facebook.github.io/react) es una librería gráfica que nos ayuda a crear interfaces de usuario interactivas.
- **AngularJS** (www.angularjs.org) es una librería que expande los elementos HTML para volverlos más dinámicos e interactivos.
- **Node.js** (www.nodejs.org) es una librería que funciona en el servidor y tiene el propósito de construir aplicaciones de red.

Librerías Externas

- **Modernizr** (www.modernizr.com) es una librería que puede detectar características disponibles en el navegador, incluidas propiedades CSS, elementos HTML y las API de JavaScript.
- **Moment.js** (www.momentjs.com) es una librería cuyo único propósito es procesar fechas.
- **Three.js** (www.threejs.org) es una librería de gráficos 3D basada en una API incluida en los navegadores llamada WebGL (Web Graphics Library).
- **Google Maps** (developers.google.com/maps/) es un grupo de librerías diseñadas para incluir mapas en nuestros sitios web y aplicaciones.

Ejemplo API Modernizr

```
<head>
<title>Modernizr</title>
<script src="modernizr-custom.js"></script>
<script>
function iniciar(){
var elemento = document.getElementById("subtitulo");
if (Modernizr.boxshadow) {
elemento.innerHTML = 'Box Shadow está disponible';
} else {
elemento.innerHTML = 'Box Shadow no está disponible';
}
}
window.addEventListener('load', iniciar);
</script>
```

Ir a www.modernizr.com, seleccione la característica Box Shadow (o las que quiera verificar), y haga clic en Build para crear su archivo.

Se descargará archivo llamado modernizr-custom.js en su ordenador. Mover el archivo al directorio de su documento y abra el documento en su navegador. Si el navegador soporta la propiedad CSS box-shadow, debería ver el mensaje "Box Shadow está disponible" en la pantalla.

API Formularios

- La API Formularios es un grupo de propiedades, métodos y eventos que podemos usar para procesar formularios y crear nuestro propio sistema de validación. La API está integrada en los formularios y elementos de formularios, por lo que podemos responder a eventos o llamar a los métodos desde los mismos elementos. Los siguientes son algunos de los métodos disponibles para el elemento `<form>`.
 - **submit()**—Este método envía el formulario.
 - **reset()**—Este método reinicializa el formulario (asigna los valores por defecto a los elementos).
 - **checkValidity()**—Este método devuelve un valor booleano que indica si el formulario es válido o no.

API Formulario

- La API también ofrece el siguiente evento para anunciar cada vez que se inserta un carácter o se selecciona un valor en un elemento de formulario.
 - **input**—Este evento se desencadena en el formulario o sus elementos cuando el usuario inserta o elimina un carácter en un campo de entrada, y también cuando se selecciona un nuevo valor.
 - **change**—Este evento se desencadena en el formulario o sus elementos cuando se introduce o selecciona un nuevo valor.
- Usando estos métodos y eventos, podemos controlar el proceso de envío del formulario desde JavaScript.

API Formulario

- La API Formularios es un grupo de propiedades, métodos y eventos que podemos usar para procesar formularios y crear nuestro propio sistema de validación. La API está integrada en los formularios y elementos de formularios, por lo que podemos responder a eventos o llamar a los métodos desde los mismos elementos. Los siguientes son algunos de los métodos disponibles para el elemento `<form>`.
 - **submit()**—Este método envía el formulario.
 - **reset()**—Este método reinicializa el formulario (asigna los valores por defecto a los elementos).
 - **checkValidity()**—Este método devuelve un valor booleano que indica si el formulario es válido o no.

API Formulario

- La API también ofrece el siguiente evento para anunciar cada vez que se inserta un carácter o se selecciona un valor en un elemento de formulario.
 - **input**—Este evento se desencadena en el formulario o sus elementos cuando el usuario inserta o elimina un carácter en un campo de entrada, y también cuando se selecciona un nuevo valor.
 - **change**—Este evento se desencadena en el formulario o sus elementos cuando se introduce o selecciona un nuevo valor.
- Usando estos métodos y eventos, podemos controlar el proceso de envío del formulario desde JavaScript.

Ejemplo API Formulario

- El siguiente ejemplo envía el formulario cuando se pulsa un botón.

```
<script>
```

```
function iniciar() {
```

```
var boton = document.getElementById("enviar");
```

```
boton.addEventListener("click", enviarformulario);
```

```
}
```

```
function enviarformulario() {
```

```
var formulario = document.querySelector("form[name='informacion']");
```

```
formulario.submit();
```

```
}
```

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

API Formulario

```
<body>
<section>
<form name="informacion" method="get" action="procesar.php">
<p><label>Correo: <input type="email" name="correo" id="correo"
required></label></p>
<p><button type="button" id="enviar">Registrarse</button></p>
</form>
</section>
</body>
</html>
```

Ejemplo API Formulario checkvalidity

```
<script>
function iniciar() {
var boton = document.getElementById("enviar");
boton.addEventListener("click", enviarformulario);      }
function enviarformulario() {
var formulario = document.querySelector("form[name='informacion']");
var valido = formulario.checkValidity();
if (valido) {
formulario.submit();
} else {
alert("El formulario no puede ser enviado"); }
}
window.addEventListener("load", iniciar);
</script>
```

Ejemplo API Formulario checkvalidity

```
<body>
<section>
<form name="informacion" method="get" action="procesar.php">
<p><label>Correo: <input type="email" name="correo" id="correo"
required></label></p>
<p><button type="button" id="enviar">Registrarse</button></p>
</form>
</section>
</body>
</html>
```

Validaciones Personalizados

- Existen diferentes maneras de validar formularios en HTML. Podemos usar campos de entrada del tipo que requieren validación por defecto, como email, convertir un campo regular de tipo text en un campo requerido con el atributo required, o incluso usar tipos especiales como pattern para personalizar los requisitos de validación.
- Sin embargo, cuando tenemos que implementar mecanismos de validación más complejos, como comparar dos o más campos o controlar el resultado de una operación, nuestra única opción es la de personalizar el proceso de validación usando la API Formularios.

Errores Personalizados

- Los objetos `Element` que representan elementos de formulario incluyen el siguiente método.
- **`setCustomValidity(mensaje)`**—Este método declara un error personalizado y el mensaje a mostrar si se envía el formulario. Si no se especifica ningún mensaje, el error se anula.
- Esto se utiliza para validaciones mas compleja.

Ejemplo Validación Personalizada

```
<script>  
var nombre1, nombre2;  
function iniciar() {  
  nombre1 = document.getElementById("nombre");  
  nombre2 = document.getElementById("apellido");  
  nombre1.addEventListener("input", validacion);  
  nombre2.addEventListener("input", validacion);  
  validacion();  
}
```

Ejemplo Validación Personalizada

```
function validacion() {  
  if (nombre1.value == "" && nombre2.value == "") {  
    nombre1.setCustomValidity("Inserte su nombre o su apellido");  
    nombre1.style.background = "#FFDDDD";  
    nombre2.style.background = "#FFDDDD";  
  } else {  
    nombre1.setCustomValidity("");  
    nombre1.style.background = "#FFFFFF";  
    nombre2.style.background = "#FFFFFF";  
  }  
}  
  
window.addEventListener("load", iniciar);  
</script>
```

Ejemplo Validación Personalizada

```
<body>
<section>
<form name="registracion" method="get" action="procesar.php">
<p><label>Nombre: <input type="text" name="nombre" id="nombre"></label></p>
<p><label>Apellido: <input type="text" name="apellido" id="apellido"></label></p>
<p><input type="submit" value="Registrarse"></p>
</form>
</section>
</body>
```

Objeto ValidityState

- En el ejemplo anterior no realiza una validación en tiempo real. Los campos se validan solo cuando se envía el formulario. Considerando la necesidad de un sistema de validación más dinámico, la API Formularios incluye el objeto ValidityState. Este objeto ofrece una serie de propiedades para indicar el estado de validez de un elemento del formulario.
- **valid**—Esta propiedad devuelve true si el valor del elemento es válido.
- La propiedad valid devuelve el estado de validez de un elemento considerando todos los demás estados de validez. Si todas las condiciones son válidas, la propiedad valid devuelve true.

Objeto `ValidityState`

- Si queremos controlar una condición en particular, podemos leer el resto de las propiedades que ofrece el objeto `ValidityState`.
- **`valueMissing`**—Esta propiedad devuelve `true` cuando se ha declarado el atributo `required` y el campo está vacío.
- **`typeMismatch`**—Esta propiedad devuelve `true` cuando la sintaxis del texto introducido no coincide con el tipo de campo. Por ejemplo, cuando el texto introducido en un campo de tipo `email` no es una cuenta de correo.
- **`patternMismatch`**—Esta propiedad devuelve `true` cuando el texto introducido no respeta el formato establecido por el atributo `pattern`.

Objeto `ValidityState`

- **`tooLong`**—Esta propiedad devuelve `true` cuando se ha declarado el atributo `maxlength` y el texto introducido es más largo que el valor especificado por este atributo.
- **`rangeUnderflow`**—Esta propiedad devuelve `true` cuando se ha declarado el atributo `min` y el valor introducido es menor que el especificado por este atributo.
- **`rangeOverflow`**—Esta propiedad devuelve `true` cuando se ha declarado el atributo `max` y el valor introducido es mayor que el especificado por este atributo.
- **`stepMismatch`**—Esta propiedad devuelve `true` cuando se ha declarado el atributo `step` y el valor introducido no corresponde con el valor de los atributos `min`, `max` y `value`.
- **`customError`**—Esta propiedad devuelve `true` cuando declaramos un error personalizado con el método `setCustomValidity()`

Ejemplo Objeto ValidityState

```
<script>
var formulario;
function iniciar() {
var boton = document.getElementById("enviar");
boton.addEventListener("click", enviarformulario);
formulario = document.querySelector("form[name='informacion']");
formulario.addEventListener("invalid", validacion, true);
formulario.addEventListener("input", comprobar);
}
function validacion(evento) {
var elemento = evento.target;
elemento.style.background = "#FFDDDD";
}
}
```


Ejemplo Objeto ValidityState

```
function enviarformulario() {  
  var valido = formulario.checkValidity();  
  if (valido) {  
    formulario.submit();  
  } }  
function comprobar(evento) {  
  var elemento = evento.target;  
  if (elemento.validity.valid) {  
    elemento.style.background = "#FFFFFF";  
  } else {  
    elemento.style.background = "#FFDDDD";  
  } }  
window.addEventListener("load", iniciar);  
</script>
```

Ejemplo Objeto ValidityState

```
<body>
<section>
<form name="informacion" method="get" action="procesar.php">
<p>
<label>Apodo:
  <input pattern="[A-Za-z]{3,}" name="apodo" id="apodo" maxlength="10"
  required></label></p>
<p><label>Correo: <input type="email" name="correo" id="correo"
  required></label></p>
<p><button type="button" id="enviar">Registrarse</button></p>
</form>
```

Ejemplo objeto `ValidityState` para saber exactamente qué ha producido el error

```
function enviarformulario() {  
  var elemento = document.getElementById("apodo");  
  var valido = formulario.checkValidity();  
  if (valido) {  
    formulario.submit();  
  } else if (elemento.validity.patternMismatch ||  
    elemento.validity.valueMissing) {  
    alert('El apodo debe tener un mínimo de 3 caracteres');  
  }  
}
```