UNIDAD 5

Eventos en JavaScript

Los eventos HTML son "cosas" que les suceden a los elementos HTML.

Cuando se utiliza JavaScript en páginas HTML, JavaScript puede "reaccionar" ante estos eventos.

Un evento es un mecanismo que se acciona cuando el usuario realiza un cambio sobre la página web.

Capturar un evento es programar una acción para que se realice una tarea.

El encargado de gestionar los eventos es el DOM (Document Object Model).

Listado de eventos HTML:

https://www.w3schools.com/tags/ref_eventattributes.asp

Objeto Event

Cuando ocurre un evento en HTML, el evento pertenece a un determinado objeto de evento, por ejemplo un evento de clic del mouse pertenece al objeto MouseEvent.

Todos los objetos de evento descienden del objeto Event y tienen sus métodos y propiedades.

https://www.w3schools.com/jsref/obj_event.asp

propiedades y métodos del objeto Event:

https://www.w3schools.com/jsref/dom_obj_event.asp

Palabra clave this

En JavaScript, **this** se refiere a un objeto, en referencia a la gestión de eventos this contiene al objeto en el que se desencadenó ese evento (por ejemplo un botón en el caso de hacer click sobre el) .

El objetos **this** es diferente dependiendo de cómo se invoque:

this sólo

Cuando se usa sólo, this se refiere al objeto global(window) .
Porqué this se ejecuta en el ámbito global.

En una ventana del navegador, el objeto global es [object Window]:
Ejemplo:

```
let x = this; //x contendrá el objeto window
```

En modo estricto this también se refiere al objeto global :
Ejemplo:

```
let x = this; //x contendrá el objeto window
```

this en una función (predeterminado)
En una función, el objeto global (window) es el contenido predeterminado para this.
Ejemplo:

```
function myFunction() {
  return this; //devuelve window
}
```

this en una función (estricta)
El modo estricto de JavaScript no permite el enlace predeterminado.
Entonces, cuando se usa en una función, en modo estricto, this es undefined.

Ejemplo:

```
"use strict";
function myFunction() {
  return this; //devuelve undefined
}
```

this en controladores de eventos
En los controladores de eventos HTML, this se refiere al elemento HTML que recibió el evento:
Ejemplo:

```
<button onclick="this.style.display='none'">
  Click to Remove Me!
</button>
```

Enlazar métodos de otros objetos (binding)

En este ejemplo this es el objeto persona:

```
class Person {
    constructor(firstName,lastName,id){
     this.firstname = firstname;
     this.lastname = lastname;
```

```
     this.id = id;
}
  fullName() {
    return this.firstname + " " + this.lastname;
  }
};
const person1 = new Person("John","Doe" ,5566);
```

Con los métodos globales call() y apply() podemos hacer un bind desde un objeto a métodos de otro, en el siguiente ejemplo utilizamos el método fullName del primer objeto con el segundo que no tiene método definido.

```
const person2 = {
  firstname:"Pepe",
  lastname: "Pérez"
}

// Return "Pepe Pérez":
person1.fullName.call(person2);
```

La diferencia entre call y apply es la forma en la que se pasan los argumentos.

bind() hace lo mismo que call() y apply() y la diferencia es que hace una copia del método, cuando call() y apply() lo ejecutan directamente.

Preservando el valor de this

Una de las funciones más importantes de bind() es preservar el contenido de this cuando las funciones se utilizan como callbacks:

En el siguiente ejemplo, el objeto persona tiene un método de visualización. En el método de visualización, this se refiere al objeto persona:

```
const person = {
  firstName:"John",
  lastName: "Doe",
  display: function () {
    let x = document.getElementById("demo");
      x.innerHTML = this.firstName + " " + this.lastName;
  }
}

person.display();
```

Cuando se utiliza una función como callback, this se pierde.

Este ejemplo intentará mostrar el nombre de la persona después de 3 segundos, pero en su lugar se mostrará indefinido :

```
const person = {
  firstName:"John",
  lastName: "Doe",
  display: function () {
    let x = document.getElementById("demo");
      x.innerHTML = this.firstName + " " + this.lastName;
  }
}

setTimeout(person.display, 3000);
```

El bind() método resuelve este problema.

En el siguiente ejemplo, el bind() método se utiliza para vincular person.display a person. Este ejemplo mostrará el nombre de la persona después de 3 segundos:

```
const person = {
  firstName:"John",
  lastName: "Doe",
  display: function () {
    let x = document.getElementById("demo");
      x.innerHTML = this.firstName + " " + this.lastName;
  }
}

let display = person.display.bind(person);
setTimeout(display, 3000);
```

Ver más en:
https://www.w3schools.com/js/js_this.asp

Modelo de registro de eventos en línea

Cada elemento del DOM tiene sus posibles eventos como una propiedad del mismo.

El nombre del evento es "on" seguido del nombre de la acción. Por ejemplo onClick

Este modelo no se recomienda, y aunque lo has visto en ejemplos que hemos utilizado hasta ahora, tiene el problema de que estamos mezclando la estructura de la página web con la programación de la misma, y lo que se intenta hoy en día es separar la programación en JavaScript, de la estructura HTML, por lo que este modelo no nos sirve.

Evitar la acción por defecto

A veces es interesante bloquear o evitar que se ejecute la acción por defecto.

Por ejemplo, en nuestro caso anterior podríamos evitar que nos conecte con la nueva pagina.html.

Cuando programamos un gestor de eventos, ese gestor podrá devolver un valor booleano true o false. Eso tendremos que programarlo con la instrucción return true|false. False quiere decir "no ejecutes la acción por defecto". Por lo tanto nuestro ejemplo quedará del siguiente modo:

```
<A href="pagina.html" onClick="alertar(); return false">Pulsa aqui</a>
```

De esa forma, cada vez que pulsemos en el enlace realizará la llamada a la función alertar() y cuando termine ejecutará la instrucción "return false", que le indicará al navegador que no ejecute la acción por defecto asignada a ese objeto (en este caso la acción por defecto de un hiperenlace es conectarnos con la página href de destino).

También sería posible que nos preguntara si queremos o no queremos ir a la pagina.html. Eso podríamos hacerlo sustituyendo true o false por una función que devuelva true o false según la respuesta que demos al "confirm":

```
<A href="pagina.html" onClick="return preguntar()">Pulsa aqui</a>
function preguntar()
{
    return confirm("¿Desea realmente ir a esa dirección?");
}
```

Ejemplos:

```
<!--Podemos cambiar atributos de una etiqueta-->
    <h3 id="cab1" onclick="this.innerHTML='Javascript en XHTML'"
onmouseover="this.style.background='red'"
onmouseout="this.style.background='white'">Pulsa aquí para ver lo que se
ejecuta</h3>
    <h3 id="cab1"
onclick="document.getElementById("cab1").innerHTML="...">...</h3>

    <!-- Podemos llamar a funciones externas -->
  <h3 onclick="cambiar(this)">Pulsa aquí para ver qué se ejecuta</h3>
  <script>
       function cambiar(elem) {
           elem.innerHTML = "Javascript en XHTML y función externa";
       }
  </script>

    <!-- Hay acciones que desencadenan varios eventos. Ej. submit desencadena
onmousedown, onclick, onmouseup, onsubmit.
    Para evitar que el navegador ejecute la acción por defecto necesitamos
añadir "return false;" -->
    <a href="http://www.google.com" onclick="alertar(); return false;">Pulsa
aquí para ver qué se ejecuta</a>
    <script>
        function alertar() {
            alert("Vamos a Google");
        }
    </script>
    <br/>
    <a href="http://www.google.com" onclick="return preguntar();">Pulsa aquí
para ver qué se ejecuta</a>
    <script>
        function preguntar() {
            return confirm("¿Deseas ir a Google?");
        }
    </script>

    <!-- Carga de página con evento onload -->
    <!-- Escribimos la etiqueta: <body onload="funcion_ejecutar"> -->
```

Modelo de registro de eventos tradicional

Esta forma de registro, no fue estandarizada por el W3C, pero debido a que fue ampliamente utilizada por Netscape y Microsoft, todavía es válida hoy en día. La ventaja de este modelo es que podremos asignar un evento a un objeto desde JavaScript, con lo que ya estamos separando el código de la estructura. Fíjate que aquí los nombres de los eventos sí que van siempre en minúsculas.

```
elemento.onclick = hacerAlgo;
```

Para eliminar un gestor de eventos de un elemento u objeto, le asignaremos null:

```
elemento.onclick = null;
```

Otra gran ventaja es que, como el gestor de eventos es una función, podremos realizar una llamada directa a ese gestor, con lo que estamos disparando el evento de forma manual. Por ejemplo:

```
elemento.onclick();
// Al hacer ésto estamos disparando el evento click de forma manual y se ejecutará la función hacerAlgo()
```

Sin paréntesis

Fíjate que en el registro del evento no usamos paréntesis (). El método onclick espera que se le asigne una función completa. Si haces: element.onclick = hacerAlgo(); la función será ejecutada y el resultado que devuelve esa función será asignado a onclick. Pero ésto no es lo que queremos que haga, queremos que se ejecute la función cuando se dispare el evento.

Uso de la palabra reservada this

En el modelo en línea podemos utilizar la palabra reservada this cuando programamos el gestor de eventos. Por ejemplo:

```
<A href="pagina.html" ID="mienlace" onClick="alertar(this)">Pulsa aqui</a>
<script type="text/javascript">
function alertar(objeto)
{  alert("Te conectaremos con la página: "+objeto.href); }
</script>
```

Su equivalente en el modelo tradicional sería:

```
<A id="mienlace" href="pagina.html">Pulsa aqui</a>
<script type="text/javascript">
document.getElementById("mienlace").onclick = alertar;
function alertar()
{     alert("Te conectaremos con la página: "+this.href);  }
</script>
```

Fíjate que estamos usando this dentro de la función, sin pasar ningún objeto (tal y como hacíamos en el modelo en linea). En el modelo tradicional el this que está dentro de la función, hace referencia al objeto donde hemos programado el evento.

También hay que destacar que en este modelo es importante que el hiperenlace sea declarado antes de programar la asignación onclick, ya que si por ejemplo escribimos el hiperenlace después del bloque de código de JavaScript, éste no conocerá todavía el objeto y no le podrá asignar el evento de onclick.

Ésto también se podría solucionar, programando la asignación de eventos a los objetos, en una función que se ejecute cuando el documento esté completamente cargado. Por ejemplo con window.onload=asignarEventos. (todo esto lo podemos aplicar también al siguiente modelo)

Nota:

En versiones más actuales de los navegadores disponemos de las etiquetas async y defer (que se añaden a la etiqueta <script src=”script.js” defer> que gestionan la carga del HTML, si ponemos defer la ejecución del script esperará a la carga previa del HTML

https://lenguajehtml.com/html/scripting/atributo-defer-async/

https://didacticode.com/que-diferencias-hay-entre-defer-y-async/

Si el script es type=”module” lleva defer por defecto.

Si por ejemplo queremos que cuando se produzca el evento se realicen llamadas a más de una función lo podremos hacer de la siguiente forma:

```
elemento.onclick = function ( ) { llamada1( ); llamada2( ) };
```

Ejemplos:

```
<h3 id="tradicional">Pulsa aquí para ver lo que se ejecuta</h3>
    <script>
        document.getElementById("tradicional").onclick = cambiar; //¡¡Sin
paréntesis!!
```

```
        function cambiar() {
            alert("Entramos en cambiar");
            document.getElementById("tradicional").innerHTML = "Modelo de
registro de eventos tradicional";
            document.getElementById("tradicional").onclick = null;
        }
    </script>
<h3 id="tradicional2">Pulsa aquí para ver qué se ejecuta</h3>
    <script>
        window.onload = function () {
            alert("La página ha cargado correctamente");
            document.getElementById("tradicional2").onclick = miMensaje;
        }

        function miMensaje() {
            document.getElementById("tradicional2").innerHTML = "Modelo de
registro de eventos tradicional";
        }
    </script>
```

Modelo de eventos avanzados del W3C

El W3C en la especificación del DOM de nivel 2, pone especial atención en los problemas del modelo tradicional de registro de eventos. En este caso ofrece una manera sencilla de registrar los eventos que queramos, sobre un objeto determinado.

La clave para poder hacer todo eso está en el método **addEventListener()**.

Este método tiene tres argumentos: el tipo de evento, la función a ejecutar y un valor booleano (true o false), que se utiliza para indicar cuándo se debe capturar el evento: en la fase de captura (true) o de burbujeo (false).

```
elemento.addEventListener('evento', función, false|true)
```

Por ejemplo para registrar la función alertar() de los ejemplos anteriores, haríamos:

```
document.getElementById("mienlace").addEventListener('click',alertar,fal
se);
function alertar()
{
    alert("Te conectaremos con la página: " + this.href);
}
```

La ventaja de este método, es que podemos añadir tantos manejadores como queramos. Por ejemplo:

```
document.getElementById("mienlace").addEventListener('click',alertar,fal
se);

document.getElementById("mienlace").addEventListener('click',avisar,fals
e);

document.getElementById("mienlace").addEventListener('click',chequear,fa
lse);
```

Por lo tanto, cuando hagamos click en "mienlace" se disparará la llamada a las tres funciones. Fíjate también, que el nombre de los eventos al usar addEventListener no lleva 'on' al comienzo.

También se pueden usar funciones anónimas (sin nombre de función), con el modelo W3C:

```
element.addEventListener('click', function () {
    this.style.backgroundColor = '#cc0000';
}, false)
```

Uso de la palabra reservada this

La palabra reservada this, tiene exactamente la misma funcionalidad que hemos visto en el modelo tradicional.

¿Qué eventos han sido registrados?

Uno de los problemas de la implementación del modelo de registro del W3C, es que no podemos saber con antelación, los eventos que hemos registrado a un elemento.

En el modelo tradicional si hacemos: alert(elemento.onclick), nos devuelve undefined, si no hay funciones registradas para ese evento, o bien el nombre de la función que hemos registrado para ese evento. Pero en este modelo no podemos hacer eso.

El W3C en el reciente nivel 3 del DOM, introdujo un método llamado eventListenerList, que almacena una lista de las funciones que han sido registradas a un elemento.

Para eliminar un evento de un elemento, usaremos el método removeEventListener():

```
elemento.removeEventListener('evento', función, false|true);
```

Para cancelar un evento, este modelo nos proporciona el método preventDefault().

Tambíen se pueden pasar parámetros a la función asignada al evento, para eso habría que pasarle una función anónima como en este ejemplo:

```
element.addEventListener("click", function() {
  myFunction(p1, p2);
```

```
});
```

Ejemplos:

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>Eventos. Modelo de registro de eventos avanzados del W3C</title>
    <!--
        EVENTO:
        Mecanismo que se acciona cuando el usuario realiza un cambio sobre la página
web.
        Capturar un evento es programar una acción para que se realice una tarea.
        El encargado de gestionar los eventos es el DOM(Document Object Model).

        MANEJADOR:
            Acción que se va a manejar. Por ejempo, en el evento click (hacer click
con el ratón) el manejador es onClick.
        -->
</head>

<body>
    <h1>Modelo de eventos avanzados del W3C</h1>
    <h3 id="w3c">Modelo del W3C</h3>
    <h3 id="w3canonima">Modelo del W3C con funciones anónimas</h3>

    <script>
        /* elemento.addEventListener("<evento_sin_on>",<funcion>,<false|true>); */
        document.getElementById("w3c").addEventListener("click", saludarUnaVez,
false);
        document.getElementById("w3c").addEventListener("click", colorearse, false);
        document.getElementById("w3c").addEventListener("mouseover", fondo, false);

        function saludarUnaVez() {
            alert("¡Hola, caracola!");
            document.getElementById("w3c").removeEventListener("click",
saludarUnaVez);
        }

        function colorearse() {
            document.getElementById("w3c").style.color = "red";
        }

        function fondo() {
            document.getElementById("w3c").style.background = "blue";
        }

        /* Funciones anónimas */
        /*
elemento.addEventListener("<evento_sin_on>",function(){<codigo_funcion}),false); */
        document.getElementById("w3canonima").addEventListener("click", function ()
```

```
{
            this.style.background = "#C0C0C0";
        });
    </script>
</body>

</html>
```

Eventos. Obtener información de un evento

```
<body>
    <h1 id="eventos">Obtener información de un evento</h1>
    <h2 id="parrafo1">Párrafo 1</h2>
    <h2 id="parrafo2">Párrafo 2</h2>
    <a href="https://www.w3schools.com/jsref/obj_event.asp">Objeto Event</a>
    <script>
        document.getElementById("eventos").addEventListener("mouseover",
manejador);
        document.getElementById("eventos").addEventListener("mouseout",
manejador);
        document.getElementById("parrafo1").addEventListener("click", saludo);
        document.getElementById("parrafo2").addEventListener("click", saludo);

        function manejador(e) {
            //Valoramos la posibilidad de que se utilice un navegador de
Microsoft anterior a versión 9 de IE
            if (!e) e = window.event;

            switch (e.type) {
            case "mouseover":
                this.style.color = "purple";
                break;
            case "mouseout":
                this.style.color = "yellow";
                break;
            }
        }
```

```
        function saludo(e) {
            //Valoramos la posibilidad de que se utilice un navegador de
Microsoft anterior a version 9 de IE
            if (!e) e = window.event;

            if (e.target.id == "parrafo1") alert("Has pulsado el primer
párrafo");
            else if (e.target.id == "parrafo2") alert("Has pulsado el segundo
párrafo");

            alert("Has pulsado el " + e.target.id);
        }
    </script>
</body>
```

Orden de disparo de los eventos

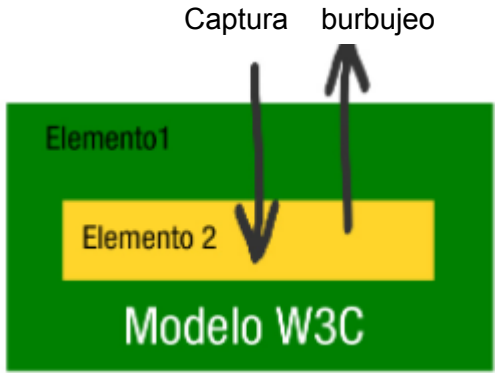
W3C decidió que, cuando se produce un evento en su modelo de eventos, primero se producirá la **fase de captura**(**desde el elemento más exterior al más interior**), y luego se producirá la **fase de burbujeo** (**desde el interior al exterior**).

Este modelo es el estándar, que todos los navegadores deberían seguir para ser compatibles entre sí.

Tú podrás decidir cuando quieres que se registre el evento: en la fase de captura o en la fase de burbujeo.

El tercer parámetro de **addEventListener** te permitirá indicar si lo haces en la **fase de captura (true)**, o en la **fase de burbujeo (false)**.

Por ejemplo:



```
elemento1.addEventListener('click', hacerAlgo1 ,true); → fase de captura
elemento2.addEventListener('click', hacerAlgo2, false);   → fase de burbujeo
```

Si el usuario hace click en el **elemento2** ocurrirá lo siguiente:

1. El evento de click comenzará en la fase de captura. El evento comprueba si hay algún ancestro del elemento2 que tenga un evento de **onclick** para la fase de captura (**true**).
2. El evento encuentra un **elemento1.hacerAlgo1()** que ejecutará primero, pues está programado a **true**.
3. El evento viajará hacia el destino, pero no encontrará más eventos para la fase de captura. Entonces el evento pasa a la fase de burbujeo, y ejecuta **hacerAlgo2()**, el cuál hemos registrado para la fase de burbujeo (**false**).
4. El evento viaja hacia arriba de nuevo y chequea si algún ancestro tiene programado un evento para la fase de burbujeo. Éste no será el caso, por lo que no hará nada más.

Para **detener la propagación del evento** en la fase de burbujeo, disponemos del método **stopPropagation()**. En la fase de captura es imposible detener la propagación.

Ejemplo:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .c1 {
            background-color: blue;
            position: absolute;
            top: 25px;
            left: 25px;
            width: 100px;
            height: 100px;
        }
        .c2 {
            background-color: rgb(255, 0, 140);
            position: absolute;
            top: 25px;
            left: 25px;
            width: 50px;
            height: 50px;
        }
    </style>
</head>
<body>
    <div id="exterior" class="c1">
        <div id= "interior" class="c2">
            div
        </div>
```

```
    </div>


    <script>
        const divext = document.getElementById("exterior");
        const divint = document.getElementById("interior");

        divext.addEventListener("click", clicendivext,true); //false fase de
burbujeo (dentro a fuera)
        divint.addEventListener("click", clicendivint,true)  //true fase de
captura (fuera d dentro)

        function clicendivext(e) {
            alert("exterior");
            //e.stopPropagation(); //detiene la propagación solo en burbujeo

        }
        function clicendivint(e) {
            alert("interior");
            //e.stopPropagation();
        }
    </script>
</body>
</html>
```

Validación Básica de Formularios con JavaScript

Seleccionamos el elemento HTML formulario en nuestro js.

Conociendo el id

```
var formulario = document.getElementById("miFormulario");
var formulario2 = document.forms["miFormulario"];
```

Conociendo el número de formulario dentro de la página

```
var formulario3 = document.getElementsByTagName("form")[0];
var formulario4 = document.forms[0];
```

Para seleccionar elementos de un formulario:

.elements[] → Devuelve un array(realmente es una colección) con todos los input del formulario
.getElementById("idElemento") → Devuelve un elemento con un id determinado
.getElementsByTagName("etiqueta") →Devuelve un array(colección) con elementos de un tipo de etiqueta (input, select, etc.)
.getElementsByName("nombre") → Devuelve un array con elementos que tienen el mismo nombre (por ejemplo, radiobutton).

Ejemplos:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="validacion.js" defer></script>
    <style>
        .error {
            border: solid 2px #FF0000;
        }
    </style>
</head>


<body>
    <h1>Formulario</h1>
    <form action="procesar.php" method="post" id="miFormulario">
        <table>
            <tr>
```

```
                <td>Nombre: </td>
                <td>
                    <input type="text" name="nombre" id="nombre" />
                </td>
            </tr>
            <tr>
                <td>Telefono: </td>
                <td>
                    <input type="text" name="telefono" id="telefono" />
                </td>
            </tr>
            <tr>
                <td>Fecha de nacimiento: </td>
                <td>
                    <input type="text" name="dia" size="2" id="dia" />
                    <input type="text" name="mes" size="2" id="mes" />
                    <input type="text" name="ano" id="ano" size="4" />
                </td>
            </tr>
            <tr>
                <td>Sexo : </td>
                <td>
                    <input type="radio" name="sexo" value="H" checked />
Hombre
                    <input type="radio" name="sexo" value="M" /> Mujer
                </td>
            </tr>
            <tr>
                <td>Mayor de 18:</td>
                <td>
                    <input type="checkbox" name="mayor" id="mayor" checked/>
                </td>
            </tr>
            </td>
            </tr>
        </table>
        <p>
            <input type="submit" value="Enviar" id="enviar" />
            <input type="reset" value="Borrar" id="borrar" />
        </p>
    </form>
</body>

</html>

//VALIDACION.JS
window.addEventListener("load",iniciar);


function iniciar() {
    document.getElementById("enviar").addEventListener('click', validar,
false);
}

function validaNombre() {
    var elemento = document.getElementById("nombre");
```

```
    limpiarError(elemento);
    if (elemento.value == "") {
        alert("El campo no puede ser vacío");
        error(elemento);
        return false;
    }
    return true;
}

function validaTelefono() {
    var elemento = document.getElementById("telefono");
    if (isNaN(elemento.value)) {
        alert("El campo teléfono tiene que ser numérico");
        return false;
    }
    return true;
}

function validaFecha() {
    var dia = document.getElementById("dia").value;
    var mes = document.getElementById("mes").value;
    var ano = document.getElementById("ano").value;

    var fecha = new Date(ano, mes, dia);
    if (isNaN(fecha)) {
        alert("Los campos de la fecha son incorrectos");
        return false;
    }
    return true;
}

function validaCheck() {
    var campoCheck = document.getElementById("mayor");
    if (!campoCheck.checked) {
        alert("Debes ser mayor de edad");
        return false
    }
    return true;
}

function validar(e) {
    if (validaNombre() && validaTelefono() && validaFecha() && validaCheck()
&& confirm("Pulsa aceptar si deseas enviar el formulario")) {
        return true;
    } else {
        e.preventDefault();
        return false;
    }
}

function error(elemento) {
    elemento.className = "error";
    elemento.focus();
}

function limpiarError(elemento) {
```

```
    elemento.className = "";
}
```

Validación de formularios con HTML5

Podemos utilizar elementos input especiales y atributos especiales de los input básicos para realizar una serie de validaciones. Las validaciones HTML5 solo funcionan al pulsar en un botón de tipo submit.

Ejemplos:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Formulario</h1>
    <form action="procesar.php" method="post" id="miFormulario">
        <table>
            <tr>
                <td>Nombre*: </td>
                <td>
                    <input type="text" name="nombre" id="nombre"
maxlength="15" pattern="[A-Za-z ]{2,15}" title="Introduce entre 2 y 15 letras"
required/>
                </td>
            </tr>
            <tr>
                <td>Edad*: </td>
                <td>
                    <input type="number" name="edad" id="edad" min="18"
max="100" required/>
                </td>
            </tr>
            <tr>
                <td>Telefono*: </td>
                <td>
                    <input type="text" name="telefono" id="telefono"
pattern="[0-9]{9}" title="Numero de 9 cifras" required/>
                </td>
            </tr>
        </table>
        <p>
```

```
            <input type="submit" value="Enviar" id="enviar" />
            <input type="reset" value="Borrar" id="borrar" />
        </p>
    </form>
</body>
</html>
```

Validación de formularios con HTML5 y JavaScript

Utilizaremos el método .checkValidity() y la propiedad .validity junto con las características de HTML5 para realizar las validaciones:

Ejemplos:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
 <script src="validacion.js"></script>
    <style>
        .error {
            border: solid 2px #FF0000;
        }
    </style>
</head>

<body>
    <h1>Formulario</h1>
    <form action="procesar.php" method="post" id="miFormulario">
        <table>
            <tr>
                <td>Nombre*: </td>
                <td>
                    <input type="text" name="nombre" id="nombre" maxlength="15"
pattern="[A-Za-z ]{2,15}" title="Introduce entre 2 y 15 letras" required/>
                </td>
            </tr>
            <tr>
                <td>Edad*: </td>
                <td>
                    <input type="number" name="edad" id="edad" min="18" max="100"
required/>
                </td>
            </tr>
            <tr>
                <td>Telefono*: </td>
```

```
                <td>
                    <input type="text" name="telefono" id="telefono"
pattern="[0-9]{9}" title="Numero de 9 cifras" required/>
                </td>
            </tr>
        </table>
        <p id="mensajeError"></p>
        <p>
            <input type="submit" value="Enviar" id="enviar" />
            <input type="reset" value="Borrar" id="borrar" />
        </p>
    </form>
</body>

</html>
```

```
window.onload = iniciar;

function iniciar() {
    document.getElementById("enviar").addEventListener('click', validar, false);
}

function validaNombre() {
    var elemento = document.getElementById("nombre");
    if (!elemento.checkValidity()) {
        if (elemento.validity.valueMissing) {
            error2(elemento, "Debe introducir un nombre")
        }
        if (elemento.validity.patternMismatch) {
            error2(elemento, "El nombre debe tener entre 2 y 15 caracteres");
        }
        //error(elemento);
        return false;
    }
    return true;
}

function validaEdad() {
    var elemento = document.getElementById("edad");
    if (!elemento.checkValidity()) {
        if (elemento.validity.valueMissing) {
            error2(elemento, "Debe introducir una edad")
        }
        if (elemento.validity.rangeOverflow) {
            error2(elemento, "El valor debe ser menor de 100")
        }
        if (elemento.validity.rangeUnderflow) {
            error2(elemento, "El valor debe ser mayor o igual que 18");
        }
        //error(elemento);
        return false;
    }
```

```
    return true;
}

function validaTelefono() {
    var elemento = document.getElementById("telefono");
    if (!elemento.checkValidity()) {
        if (elemento.validity.valueMissing) {
            error2(elemento, "Debe introducir un teléfono")
        }
        if (elemento.validity.patternMismatch) {
            error2(elemento, "El telefono debe tener 9 numeros");
        } //error(elemento);
        return false;
    }
    return true;
}

function validar(e) {
    borrarError();
    if (validaNombre() && validaEdad() && validaTelefono() && confirm("Pulsa aceptar
si deseas enviar el formulario")) {
        return true
    } else {
        e.preventDefault();
        return false;
    }
}

function error(elemento) {
    document.getElementById("mensajeError").innerHTML = elemento.validationMessage;
    elemento.className = "error";
    elemento.focus();
}

function error2(elemento, mensaje) {
    document.getElementById("mensajeError").innerHTML = mensaje;
    elemento.className = "error";
    elemento.focus();
}

function borrarError() {
    var formulario = document.forms[0];
    for (var i = 0; i < formulario.elements.length; i++) {
        formulario.elements[i].className = "";
    }
}
```

Expresiones regulares y Objetos RegExp

Las expresiones regulares son patrones de búsqueda, que se pueden utilizar para encontrar texto que coincida con el patrón especificado. Se utiliza el objeto **RegExp**.

Podemos instanciarlo así:

```
let  expresion = /expresión regular/;
```

La siguiente expresión regular coincidiría con la cadena “hola mundo” con cualquier número de espacios o tabulaciones en el medio de las dos palabras, indicado por el carácter especial \s+. (Listado de caracteres especiales)

```
let expresion = /hola\s+mundo/;
```

Las expresiones regulares también se pueden instanciar como un objeto

```
let expresionRegular = new RegExp("Texto Expresión Regular");
```

Pero esta forma solo se usará cuando la expresión regular se decida en tiempo de ejecución, en otro caso es más eficiente la primera forma de declararla.

Para utilizarlas podemos usar el atributo **pattern de HTML** o el método **match** en datos de tipo cadena:

`cadena.match(expresiónRegula)` devuelve un array con las coincidencias y si no encuentra nada devuelve null

Podemos usarlo así:

```
if (dni.match(patron))
    alert('DNI Correcto!!!!');
else
    alert('DNI INCORRECTO!!!!');
```

Por ejemplo en el atributo pattern lo utilizaremos así:

```
<input type="text" id="id1" pattern="[A-Za-z]{3}" title="Código de 3 letras">
```

El atributo title muestra el mensaje que le saldría al usuario si no cumple el patrón.

Aqui puedes comprobar tus expresiones regulares: https://regexr.com/

Cookies

¿Qué son las Cookies?

- Datos almacenados en nuestro ordenador en pequeños archivos de texto.
- Recuerdan la información de un usuario aunque se cierre el navegador o se desconecte del servidor.

- Podemos guardar el nombre de un visitante de la página, el número de veces que ha entrado, etc.
- Se guardan en forma de pares "nombre=valor" por ejemplo usuario = Manuel
- Limitadas a 4 kb
- Se pueden ver en las herramientas de depuración en la sección Aplicación

Crear una cookie

"nombre(nombre=); fecha de expiración -UTC-(expires=); edad máxima en segundos que queremos que esté activa (max-age=segundos); ruta(path=); dominio(domain=)"

Los atributos Domain y Path definen el alcance de una cookie: a qué URL se deben enviar las cookies ….
ej:
document.cookie ="usuario = Pepe; expires=Thu, 16 Nov 2025 12:00:00 UTC;";

```
<!--     <script>
        document.cookie = "usuario = Claudia;";
        document.cookie = "sexo = mujer;";

        //Leer una cookie
        var miCookie = document.cookie;
        alert(miCookie);


        //Modificar una cookie
        document.cookie = "sexo = hombre;";
        alert(document.cookie);

        //Borrar una cookie
        //document.cookie = "usuario=; expires=Thu, 01 Jan 1970 00:00:01 GMT";
        //document.cookie = "sexo=; expires=Thu, 01 Jan 1970 00:00:01 GMT";
        alert(document.cookie);
    </script>

</body>
```

Ejemplo de ver crear y borrar cookies

```
<body>
    <button type="button" id="verTodas">Ver todas las cookies</button>
    <button type="button" id="crearCookie1">Crear cookie 1</button>
    <button type="button" id="crearCookie2">Crear cookie 2</button>
    <button type="button" id="borrarCookie1">Borrar cookie 1</button>
    <button type="button" id="borrarCookie2">Borrar cookie 2</button>
    <script>
document.getElementById("verTodas").addEventListener("click",verTodas);
document.getElementById("crearCookie1").addEventListener("click",crearCookie);
document.getElementById("crearCookie2").addEventListener("click",crearCookie);
```

```
document.getElementById("borrarCookie1").addEventListener("click",borrarCookie
);
document.getElementById("borrarCookie2").addEventListener("click",borrarCookie
);
        function verTodas(){
            alert(document.cookie);
        }

        function crearCookie(e){
            if(!e) e = window.event;
            if (e.target.id=="crearCookie1")
                document.cookie ="nombre = Ada";
            else if (e.target.id=="crearCookie2")
                document.cookie ="apellido = Lovelace";
        }

        function borrarCookie(e){
            if(!e) e = window.event;
            if (e.target.id=="borrarCookie1")
                document.cookie ="nombre =;expires=Thu, 01 Jan 1970 00:00:00
UTC;";
            else if (e.target.id=="borrarCookie2")
                document.cookie ="apellido =;expires=Thu, 01 Jan 1970 00:00:00
UTC;";
        }
    </script>
   </body>
```

Ejemplo avanzado de acceso, creación y modificación de cookies

```
<body>
    <button type="button" id="verTodas">Ver todas las cookies</button>
    <button type="button" id="crearCookie">Crear cookie </button>
    <button type="button" id="modificarCookie">Modificar cookie </button>
    <button type="button" id="leerCookie">Leer cookie </button>
    <button type="button" id="borrarCookie">Borrar cookie </button>
    <script>

document.getElementById("verTodas").addEventListener("click",verCookies);

document.getElementById("crearCookie").addEventListener("click",crearModifCook
ie);

document.getElementById("modificarCookie").addEventListener("click",crearModif
Cookie);

document.getElementById("leerCookie").addEventListener("click",leerCookie);

document.getElementById("borrarCookie").addEventListener("click",borrarCookie)
;

        function verCookies(){
            alert("Cookies actuales:\n"+document.cookie);
        }

        function crearModifCookie(){
            var nombre = prompt("Introduzca el nombre de la cookie");
            var valor = prompt("Introduzca su valor");
```

```
            var expiracion = parseInt(prompt("Introduzca el número de días
para que expire"));
            setCookie(nombre, valor, expiracion);
            verCookies();
        }

        function leerCookie(){
            var nombre = prompt("Introduzca el nombre de la cookie a
consultar");
            var resultado = getCookie(nombre);
            alert(resultado);
        }

        function borrarCookie(){
            var nombre = prompt("Introduzca el nombre de la cookie a borrar");
            deleteCookie(nombre);
            verCookies();
        }

        function deleteCookie(nombre){
            setCookie(nombre,"",0);
        }

        function setCookie(nombre, valor, expiracion){
            var d = new Date();
            d.setTime(d.getTime()+expiracion*24*60*60*1000);
            var expiracion = "expires="+d.toUTCString();
            document.cookie = nombre+"="+valor+";"+expiracion+";path=/";
        }

        function getCookie(nombre){
            var nom = nombre+"=";
            var array = document.cookie.split(";");
            for (var i=0; i<array.length; i++){
                var c = array[i];
                while (c.charAt(0)==" "){
                    c = c.substring(1);
                }
                if (c.indexOf(nombre)==0){
                    return c.substring(nom.length, c.length);
                }
            }
            return "";
        }
    </script>

</body>
```

Webstorage (LocalStorage y SessionStorage)

- Permite almacenar datos localmente en el ordenador del usuario.
- Es más seguro y almacena más información que las cookies.

- El webstorage se almacena por origen (dominio y protocolo): todas las páginas del mismo origen pueden acceder a los mismos datos
- 5 megas de almacenamiento por dominio
- Para borrarlo en chrome F12 → Application → Localstorage (borraremos la clave que deseamos eliminar)

https://www.w3schools.com/html/html5_webstorage.asp

```
      //Comprobar si el navegador soporta webstorage
              if (typeof(Storage) !== "undefined"){
                  alert ("El navegador soporta webStorage");
              }else{
                  alert("El navegador NO soporta WebStorage");
              }

              //window.localStorage: almacena datos SIN fecha de expiración
              //Crear un elemento (¡¡siempre son cadenas!!)
              localStorage.setItem("nombre", "Ada"); //MEJOR
              localStorage.apellido = "Lovelace"; //PEOR

              //Consultar un elemento
              alert(localStorage.getItem("nombre")); //MEJOR
              alert(localStorage.apellido); //PEOR

              //Borrar un elemento
              localStorage.removeItem("apellido");
              alert(localStorage.getItem("nombre")+"
"+localStorage.getItem("apellido"));

              //Borrar todos los elementos
              localStorage.clear();
              alert(localStorage.getItem("nombre")+"
"+localStorage.getItem("apellido"));


      //window.sessionStorage: almacena los datos durante una sesión
      //si se cierra la ventana o el navegador, desaparecen
```

Ejemplo LocalStorage

```
<body>
    <button type="button" id="incrementar">Incrementar</button>
    <button type="button" id="decrementar">Decrementar</button>
    <button type="button" id="logout">Log out</button>
    <p id="saludo"></p>
    <p id="contador"></p>
    <script>
    //Comprobación de que el navegador soporta Web Storage
    if (typeof(Storage) !== "undefined") {
        alert("El navegador soporta WebStorage");
        if (localStorage.usuario!=null){
```

```
            document.getElementById("saludo").innerHTML="¡Bienvenido/a de
nuevo, "+localStorage.usuario+"!";
        }else{
            localStorage.usuario=prompt("¿Cómo te llamas?");
            document.getElementById("saludo").innerHTML="¡Tu primera visita,
"+  localStorage.usuario+"!";
        }

        if(!sessionStorage.getItem("contador"))
            sessionStorage.setItem("contador","0");

        document.getElementById("contador").innerHTML="Contador:
"+sessionStorage.getItem("contador");

document.getElementById("incrementar").addEventListener("click",incrementar);

document.getElementById("decrementar").addEventListener("click",decrementar);
        document.getElementById("logout").addEventListener("click",logout);
    } else {
        alert ("El navegador no soporta Web Storage");
    }

    function incrementar(){

sessionStorage.setItem("contador",Number(sessionStorage.getItem("contador"))+1
);
        document.getElementById("contador").innerHTML = "Contador: "+
sessionStorage.getItem("contador");
    }
    function decrementar(){

sessionStorage.setItem("contador",Number(sessionStorage.getItem("contador"))-1
);
        document.getElementById("contador").innerHTML = "Contador: "+
sessionStorage.getItem("contador");
    }
    function logout(){
        alert("Se ha cerrado la sesión de "+localStorage.getItem("usuario"));
        localStorage.removeItem("usuario");
        //sessionStorage.clear();
        document.getElementById("saludo").innerHTML = "";
    }
```

```
    </script>
</body>
```

Las cookies pueden ser accedidas desde el servidor y Webstorage no:

https://www.drauta.com/localstorage-y-cookies-diferencias-y-uso