

Tema 6:

Gestión de eventos y formularios en JavaScript.

Desarrollo Web en Entorno Cliente

Objetivos



- Capturar y gestionar los eventos producidos en una página web.
- Diferenciar los tipos de eventos que se pueden manejar.
- Crear código que capture y utilice eventos.
- Gestionar formularios web.
- Validar formularios web utilizando eventos y expresiones regulares.

Contenidos



1. Modelo de gestión de eventos.
2. Formularios.
3. Modificación de apariencia y comportamiento.
4. Validación y envío.
5. Expresiones regulares.
6. Utilización de cookies.



1.- Modelos de gestión de eventos

- Los eventos son mecanismos que se accionan cuando el usuario realiza un cambio sobre una página web.
- El encargado de crear la jerarquía de objetos que compone una página web es el DOM (*Document Object Model*).
- Por tanto es el DOM el encargado de gestionar los eventos.

1.- Modelos de gestión de eventos

- Para poder controlar un evento se necesita un manejador.
- Las funciones o código JavaScript que se definen para cada evento se denominan "*manejador de eventos*".
- En el caso del evento *click*, el manejador sería `onclick`.

— Ejemplo de manejador de eventos como atributo de un elemento html:

```

```

No es aconsejable

1.- Modelos de gestión de eventos

- Ejemplo de manejador de eventos como función externa:

```
<html>
  <head>
    <title>Pagina de Evento</title>
    <script type="text/javascript">
      function func1() {
        alert("Click en imagen");
      }
    </script>
  </head>
  <body>
    
  </body>
</html>
```

1.- Modelos de gestión de eventos

- Ejemplo de manejador de eventos semánticos:

```
<html>
  <head>
    <title>Pagina de Evento</title>
    <script type="text/javascript">
      function func1() {
        alert("Click en imagen");
      }
      window.onload = function()
      {
        document.getElementById("imgworld").onclick
          =func1;
      }
    </script>
  </head>
  <body>
    
  </body>
</html>
```

1.- Modelos de gestión de eventos

- Modelo de registro de eventos tradicional:

```
function hacerAlgo() {
```

```
}
```

```
. . .
```

```
//para asignar un evento a un elemento  
elemento.onclick = hacerAlgo;
```

```
//para eliminar el gestor de evento  
del elemento  
elemento.onclick = null;
```


1.- Modelos de gestión de eventos

- Modelo de registro avanzado de eventos según W3C:
 - Para asignar un evento a un elemento:

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

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

```
miDiv.addEventListener('click', muestraMensaje, false);
```

- Para desasociar la función de evento a un elemento:

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

```
miDiv.removeEventListener('click',muestraMensaje, false);
```

- Para cancelar un evento:

```
elemento.preventDefault;
```

Ejemplo anterior:

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

Modelo avanzado según W3C:

```
document.getElementById("mienlace").addEventListener(`cl
ic`,alertar, false);
```

La ventaja de este método es que podemos añadir tantos eventos como queramos:

```
document.getElementById("mienlace").addEventListener(`cl
ic`,alertar, false);
document.getElementById("mienlace").addEventListener(`cl
ic`,avisar, false);
document.getElementById("mienlace").addEventListener(`cl
ic`,chequear, false);
```

1.- Modelos de gestión de eventos

- Modelo de registro de eventos según Microsoft:
 - Para asignar un evento a un elemento:

```
elemento.attachEvent('evento', función)
```


- Este método tiene dos argumentos: el tipo de evento (el cual lleva en este caso el prefijo 'on', y la función a ejecutar.
- Los eventos siempre burbujan, no hay forma de captura.

```
miDiv.attachEvent('onclick', muestraMensaje);
```

- Para desasociar la función de evento a un elemento:

```
elemento.detachEvent('evento', función)
```

```
miDiv.detachEvent('onclick', muestraMensaje);
```



Comparando este modelo con el W3c encontramos dos diferencias importantes:

- Los eventos siempre burbujan, no hay forma de captura.
- La función que gestiona el evento está referenciada, no copiada, con lo que la palabra reservada **this** siempre hace referencia a window y será completamente inútil.

Como resultado de estas dos debilidades, cuando un evento burbujea hacia arriba es imposible conocer cuál es el elemento HTML que gestionó ese evento.

Listado de atributos de eventos (IE: Internet Explorer, F: Firefox, O: Opera, W3C: W3C Standard.)

Listado de atributos de eventos (IE: Internet Explorer, F: Firefox, O: Opera, W3C: W3C Standard.)

Listado de atributos de eventos					
Atributo	El evento de produce cuando...	IE	F	O	W3C
onblur	Un elemento pierde el foco.	3	1	9	Si
onchange	El contenido de un campo cambia.	3	1	No	Si
onclick	Se hace clic con el ratón en un objeto.	3	1	9	Si
ondblclick	Se hace doble clic con el ratón sobre un objeto	4	1	9	Si
onerror	Ocurre algún error cargando un documento imagen.	4	1	9	Si
onfocus	Un elemento tiene el foco.	3	1	9	Si
onkeydown	Se precisa una tecla del teclado.	3	1	No	Si
onkeypress	Se presiona una tecla o se mantiene presionada.	3	1	9	Si
onkeyup	Cuando soltamos una tecla.	3	1	9	Si
onload	Una página o imagen terminaron de cargarse.	3	1	9	Si
onmousedown	Se presiona un botón del ratón.	4	1	9	Si
onmousemove	Se mueve el ratón.	3	1	9	Si
onmouseout	Movemos el ratón fuera de un elemento.	4	1	9	Si
onmouseover	El ratón se mueve sobre un elemento.	3	1	9	Si
onmouseup	Se libera un botón del ratón.	4	1	9	Si
onresize	Se redimensiona una ventana o frame.	4	1	9	Si
onselect	Se selecciona un texto.	3	1	9	Si
onunload	El usuario abandona una página.	3	1	9	Si

1.- Modelos de gestión de eventos

- **Orden de disparo de los eventos.**

- Imagina que tenemos un elemento contenido dentro de otro elemento, y que tenemos programado el mismo tipo de evento para los dos (por ejemplo el evento click). ¿Cuál de ellos se disparará primero? Dependerá del tipo de navegador que tengamos.
- Si el usuario hace click en el elemento2, provocará un click en ambos: elemento1 y elemento2, pero ¿cuál es el orden de los eventos?



1.- Modelos de gestión de eventos

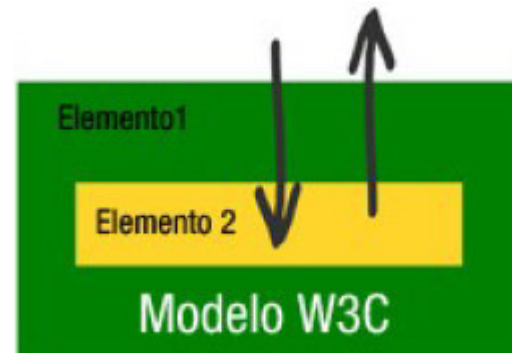
- **Orden de disparo de los eventos.**
 - Tenemos dos modelos propuestos por Nestcape y Microsoft en sus orígenes:
 - Nestcape: el evento en el elemento1 tendrá lugar primero. Es lo que se conoce como "**captura de eventos**".
 - Microsoft: el evento en el elemento2 tendrá precedencia. Es lo que se conoce como "**burbujeo de eventos**".



1.- Modelos de gestión de eventos

- **Orden de disparo de los eventos.**

- MODELO W3C: se decidió tomar una posición intermedia. Cuando se produce un evento, primero se producirá la fase de captura hasta llegar al elemento de destino, y luego se producirá la fase de burbujeo hacia arriba. Este modelo es el estándar, que todos los navegadores deberían seguir para ser compatibles.
- El programador decidirá cuando quiere que se registre el evento: en la fase de captura o en la fase de burbujeo.



1.- Modelos de gestión de eventos

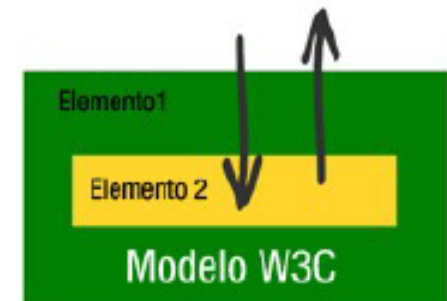
- Orden de disparo de los eventos.

Por ejemplo:

```
elemento1.addEventListener('click',hacerAlgo1,true);  
elemento2.addEventListener('click',hacerAlgo2,false);
```

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 cual 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.

1.- Modelos de gestión de eventos

- **El objeto event**

- Cuando se produce un evento, no es suficiente con asignarle una función para procesar ese evento. Además se necesita información relativa al evento producido: la tecla que se ha pulsado, la posición del ratón, el elemento que ha producido el evento, etc.
- El objeto ***event*** es el mecanismo definido por los navegadores para proporcionar toda esa información. Se trata de un objeto que se crea automáticamente cuando se produce un evento y que se destruye de forma automática cuando se han ejecutado todas las funciones asignada al evento.
- Existen numerosas diferencias en cuanto a las propiedades y métodos del objeto ***event*** dependiendo del navegador utilizado. Consultar el anexo del objeto.



En los navegadores tipo Internet Explorer, el objeto event se obtiene directamente mediante:

```
var evento = window.event;
```

En el resto de navegadores, se obtiene a partir del argumento que el navegador crea automáticamente:

```
function manejarEventos (evento)  
{  
    var evento = evento;  
}
```

Para programar una aplicacion que funcione en todos los navegadores:

```
function manejarEventos (evento)  
{  
    var evento = evento || window.event;  
}
```

Propiedades del objeto event

Propiedades del objeto evento	
Propiedad	Descripción
bubbles	Indica si el evento se propaga de manera ascendente por el DOM o no.
cancelable	Nos permite saber si el evento es cancelable. Un evento es cancelable si se puede detener, simulando que nunca ha ocurrido.
defaultPrevented	Toma valor true cuando el evento ha sido cancelado deliberadamente.
timeStamp	Nos dice el tiempo que ha transcurrido, en milisegundos, desde que se cargó la página hasta que se ha producido el evento.
type	Devuelve el tipo de evento que se ha producido como string ("click", "mouseup", "keydown", etc.)
target	Representa el elemento sobre el que se ha producido el evento.
currentTarget	Representa el elemento que ha lanzado el evento, es decir, el que tiene asociado el manejador.

Métodos del objeto event

Métodos del objeto evento	
Método	Descripción
preventDefault()	Cancela un evento, evitando así que se ejecute su acción por defecto. Esto no evita que el evento se siga propagando
stopPropagation()	Detiene la propagación de un evento hacia niveles superiores del DOM

Ejemplo:

```
function resalta(elEvento) {
    var evento = elEvento || window.event;
    switch(evento.type){
        case 'mouseover':
            this.style.borderColor = 'black';
            break;
        case 'mouseout':
            this.style.borderColor = 'silver';
            break;
    }
}

window.onload = function() {
    document.getElementById("seccion").onmouseover = resalta;
    document.getElementById("seccion").onmouseout = resalta;
}

<div id="seccion" style="width:150px; height:60px;border:thin
solid silver">
    Sección de contenidos...
</div>
```



1.- Modelos de gestión de eventos

- **Incompatibilidades entre navegadores:**

- Crear páginas y aplicaciones web resulta más complejo de lo que debería debido a las incompatibilidades entre los navegadores.
- La incompatibilidad más importante se da precisamente en el modelo de eventos del navegador.
- Se ha de tener en cuenta las diferencias existentes entre los navegadores actuales para el desarrollo de aplicaciones web.



1.- Modelos de gestión de eventos

- La especificación DOM define cuatro grupos de eventos dividiéndolos según su origen:
 - Eventos del ratón.
 - Eventos del teclado.
 - Eventos HTML.
 - Eventos DOM.

1.- Modelos de gestión de eventos

- **Eventos del ratón (1):**

- click: este evento se produce cuando pulsamos sobre el botón principal (izquierdo) del ratón. El manejador de este evento es `onclick`.
- dblclick: este evento se acciona cuando hacemos un doble click sobre el botón principal del ratón. El manejador de este evento es `ondblclick`.
- mousedown: este evento se produce cuando pulsamos un botón del ratón. El manejador de este evento es `onmousedown`.
- mouseout: este evento se produce cuando el puntero del ratón esta dentro de un elemento y este puntero es desplazado fuera del elemento. El manejador de este evento es `onmouseout`.

1.- Modelos de gestión de eventos

- Eventos del ratón (2):
 - mouseover: este evento al revés que el anterior se produce cuando el puntero del ratón se encuentra fuera de un elemento, y este se desplaza hacia el interior. El manejador de este evento es **onmouseover**.
 - mouseup: este evento se produce cuando soltamos un botón del ratón que previamente teníamos pulsado. El manejador de este evento es **onmouseup**.
 - mousemove: se produce cuando el puntero del ratón se encuentra dentro de un elemento. Es importante señalar que este evento se producirá continuamente una vez tras otra mientras el puntero del ratón permanezca dentro del elemento. El manejador de este evento es **onmousemove**.

1.- Modelos de gestión de eventos

- Eventos del ratón:

- En los eventos del ratón hay propiedades comunes con la teclas. Por ejemplo, también tenemos propiedades **AltKey**, **CtrlKey**, **ShiftKey** y **MetaKey**, para saber si la pulsación de alguna de estas teclas acompaña al evento del ratón.
- Además, la propiedad **button** devuelve el botón del ratón pulsado en el momento del evento. Estos valores son:

Valor	Significado
0	Botón principal del ratón.
1	Botón central del ratón (en muchos casos la rueda)
2	Botón secundario del ratón.
3	Cuarto botón (retroceder página).
4	Quinto botón (avanzar página).

1.- Modelos de gestión de eventos

- Eventos del ratón:
 - Cuando se pulsa un botón del ratón, la secuencia de eventos que se produce es la siguiente:
`mousedown, mouseup, click.`
 - Por tanto, la secuencia de eventos necesaria para llegar al `doubleclick` sería la siguiente:

<code>mousedown,</code>	<code>mouseup,</code>	<code>click,</code>
<code>mousedown,</code>	<code>mouseup,</code>	<code>click,</code>
<code>doubleclick.</code>		

Actividad 1



Crea un elemento `<div>` y muestra, cuando el usuario pase el ratón por encima, el borde destacado. Cuando el ratón salga del div, se volverá a mostrar el borde original.

```
<div id="elemento" style="padding: .2em; width: 150px; height: 60px; border:
thin solid silver"
onmouseover = "document.getElementById('elemento').style.borderColor =
'blue'"
onmouseout = "document.getElementById('elemento').style.borderColor =
'silver'">
Sección de contenidos...
</div>
```

Actividad 1.1



Crea una página web que tenga un texto que indique que al pulsarla Alt+F12, podremos colocar una imagen de fondo. El texto tiene que salir centrado.

Inicialmente aparece una pantalla color agua marina con el texto y hasta que el usuario no pulse esa tecla, la imagen no se muestra.

Tras pulsar ALT+F12 una imagen ocupará el fondo completo.

Actividad 1.2



Crea una aplicación web que muestre una capa centrada que ocupe el 50% del ancho y el alto de la ventana.

Inicialmente la capa será blanca y solo se verá el borde.

Al arrimar el ratón se colorea en verde.

Al hacer clic encima con el botón principal se colorea de rojo, pero solo mientras el botón principal esta abajo, sino se quite en color rojo.

Al hacer clic encima con el botón secundario ocurre lo mismo que en el caso anterior, pero se muestra el color azul. No se mostrara en ningún caso el menú de contexto.

1.- Modelos de gestión de eventos

- Eventos del teclado:
 - keydown: este evento se produce cuando pulsamos cualquier tecla del teclado. Si mantenemos pulsada una tecla de forma continua, el evento se produce una y otra vez hasta que soltemos la misma. El manejador de este evento es **onkeydown**.
 - keypress: este evento se produce si pulsamos una tecla de un carácter alfanumérico. (El evento no se produce si pulsamos enter, la barra espaciadora, etc...). En el caso de mantener una tecla pulsada, el evento se produce de forma continuada. El manejador de este evento es **onkeypress**.
 - keyup: este evento se produce cuando soltamos una tecla. El manejador de este evento es **onkeyup**.

1.- Modelos de gestión de eventos

- Eventos del teclado:
 - Cuando se pulsa una tecla correspondiente a un *carácter alfanumérico*, se produce la siguiente secuencia de eventos:
`keydown`, `keypress`, `keyup`.
 - Cuando se pulsa otro tipo de tecla, la secuencia es:
`keydown`, `keyup`.
 - Si se mantiene pulsada la tecla, para el primer caso (caracteres alfanuméricos) se repiten de forma continua los eventos: `keydown` y `keypress` y para el segundo caso, se repite el evento `keydown` de forma continua.

Actividad 2



Utiliza el evento adecuado para mostrar un aviso de que «*se ha soltado la tecla que has pulsado*» para cualquier tecla del teclado.

1.- Modelos de gestión de eventos

- Eventos HTML (1):

- load: el evento *load* hace referencia a la carga de distintas partes de la página. Este se produce en el objeto `Window` cuando la página se ha cargado por completo. En el elemento `` actúa cuando la imagen se ha cargado. En el elemento `<object>` se acciona al cargar el objeto completo. El manejador es **onload**.
- unload: el evento *unload* actúa sobre el objeto `Window` cuando la pagina ha desaparecido por completo (por ejemplo, si pulsamos el aspa cerrando la ventana del navegador). También se acciona en el elemento `<object>` cuando desaparece el objeto. El manejador es **onunload**.
- abort: este evento se produce cuando el usuario detiene la descarga de un elemento antes de que haya terminado, actúa sobre un elemento `<object>`. El manejador es **onabort**.

1.- Modelos de gestión de eventos

- Eventos HTML (2):

- error: el evento *error* se produce en el objeto `Window` cuando se ha producido un error en JavaScript. En el elemento `` cuando la imagen no se ha podido cargar por completo y en el elemento `<object>` en el caso de que un elemento no se haya cargado correctamente. El manejador es **onerror**.
- select: se acciona cuando seleccionamos texto de los cuadros de textos `<input>` y `<textarea>`. El manejador es **onselect**.
- change: este evento se produce cuando los cuadros de texto `<input>` y `<textarea>` pierden el foco y el contenido que tenían ha variado. También se producen cuando un elemento `<select>` cambia de valor. El manejador es **onchange**.
- submit: este evento se produce cuando pulsamos sobre un botón de tipo submit. El manejador es **onsubmit**.

1.- Modelos de gestión de eventos

- Eventos HTML (3):
 - reset: este evento se produce cuando pulsamos sobre un botón de tipo reset. El manejador es **onreset**.
 - resize: este evento se produce cuando redimensionamos el navegador, actúa sobre el objeto `window`. El manejador es **onresize**.
 - scroll: se produce cuando varía la posición de la barra de scroll en cualquier elemento que la tenga. El manejador es **onscroll**.
 - focus: este evento se produce cuando un elemento obtiene el foco. El manejador es **onfocus**.
 - blur: este evento se produce cuando un elemento pierde el foco. El manejador es **onblur**.

1.- Modelos de gestión de eventos

- Eventos DOM:
 - **DOMSubtreeModified.** Este evento se produce cuando añadimos o eliminamos nodos en el subárbol de un elemento o documento.
 - **DOMNodeInserted.** Este evento se produce cuando añadimos un nodo hijo a un nodo padre.
 - **DOMNodeRemoved.** Este evento se produce cuando eliminamos un nodo que tiene nodo padre.
 - **DOMNodeRemovedFromDocument.** Este evento se produce cuando eliminamos un nodo del documento.
 - **DOMNodeInsertedIntoDocument.** Este evento se produce cuando añadimos un nodo al documento.

1.- Modelos de gestión de eventos

- Eventos de movimientos en la ventana:

Eventos de movimiento en la ventana	
evento	Descripción
scroll	Ocurre cuando se ha desplazado la ventana a través de las barras de desplazamiento o usando el dispositivo táctil.
resize	Evento de window que se produce cuando se cambia el tamaño de la ventana.

- Eventos sobre carga y descarga de elementos:

Eventos sobre carga y descarga de elementos	
evento	Descripción
load	Se concluyó la carga del elemento. Es uno de los elementos más importantes a capturar para asegurar que el código siguiente funciona con la seguridad de que está cargado cuando se necesita. Cuando se aplica al elemento window , se produce cuando todos los elementos del documento se han cargado.
DOMContentLoaded	Similar al evento load . Se produce cuando el documento HTML ha sido cargado. A diferencia de load , se dispara sin esperar a que se terminen de cargar las hojas de estilos, imágenes y elementos en segundo plano. En general, para JavaScript, es más conveniente este evento.
abort	Se produce cuando se anula la carga de un elemento.
error	Sucede si hubo un error en la carga.
progress	Se produce si la carga está en proceso.
readystatechange	Ocurre cuando se ha modificado el estado del atributo readystatechange , lo cual ocurre cuando se ha modificado el estado de carga y descarga.

1.- Modelos de gestión de eventos

- Eventos sobre el historial:

Eventos sobre el historial	
evento	Descripción
popstate	Se produce si se cambia el historial.

- Eventos relacionados con la reproducción de medios:

Eventos relacionados con la reproducción de medios	
evento	Descripción
waiting	Sucede cuando el video se ha detenido por falta de datos.
playing	El medio está listo para su reproducción después de que se detuviera por falta de datos u otras causas.
canplay	Ocurre cuando se detecta que un video (u otro elemento multimedia) ya se puede reproducir (aunque no se haya cargado del todo)
canplaythrough	Se produce si se estima que el video ha cargado suficientes datos para poderse reproducir sin tener que esperar la llegada de más datos.
pause	El medio de reproducción se ha pausado.
play	Ocurre cuando el medio de reproducción se ha empezado a reproducir tras una pausa.
ended	Se produce si la reproducción del video o audio se ha detenido, sea por haber llegado al final o porque no hay más datos disponibles.
loadeddata	Se produce si se ha cargado el frame actual.
suspend	Se lanza si se ha suspendido la carga del medio.
emptied	Sucede si se ha vaciado el medio por un nuevo intento de carga por parte del usuario o por otras razones.
stalled	Sucede ante un fallo en la carga, pero sin que se detenga la misma.
seeking	Ocurre cuando se ha iniciado una labor de búsqueda en el medio.
seeked	Ocurre cuando se ha finalizado la labor de búsqueda.
loadedmetadata	Se han cargado los metadatos del medio.
durationchange	Sucede si se modifica el atributo duration del medio.
timeupdate	Ocurre si se ha modificado el atributo currentTime del medio.
ratechange	Se produce cuando el ratio del video se ha modificado.
volumechange	Se lanza si el volumen se ha modificado.

1.- Modelos de gestión de eventos

- Eventos de arrastre:

Están relacionados con la interface que es parte de HTML5. Para que un elemento pueda ser arrastrable debe tener el atributo **draggable** colocado a valor **true**:

```
<div id="capaArrastrable" draggable="true"> ¡Soy arrastrable! </div>
```

En este tipo de operaciones hay dos protagonistas: una capa arrastrable (la que realmente se arrastra) y un posible destino del arrastre.

1.- Modelos de gestión de eventos

- Eventos de arrastre:

Eventos de arrastre	
evento	Descripción
dragstart	Se produce cuando el usuario empieza a arrastrar el elemento.
drag	Ocurre una vez iniciado el arrastre, cada vez que se sigue arrastrando el elemento.
dragstop	Se produce cuando el arrastre finaliza.

Eventos que se producen en el elemento destino del arrastre:

Eventos de arrastre	
evento	Descripción
dragenter	Se produce cuando el elemento que se está arrastrando, entra en el elemento destino.
dragover	Ocurre cada vez que se continua, tras haber entrado, arrastrando el elemento origen sobre el destino.
dragleave	Ocurre cuando el elemento que se arrastra, sale del destino.
drop	Ocurre cuando el elemento origen se suelta dentro del destino. Para que ese evento se pueda capturar hay que eliminar el comportamiento por defecto del evento dragover .

1.- Modelos de gestión de eventos

- Eventos sobre animación y transiciones:

Eventos sobre animaciones y transiciones	
evento	Descripción
animationstart	Se produce cuando se inicia una animación sobre el elemento.
animationinteraction	Se produce justo cuando se repite la animación.
animationend	Se produce cuando finaliza a animación.
transitionrun	Se lanza cuando ya se ha preparado para empezar la transición.
transitionstart	Ocurre cuando se inicia una transición.
transitionend	Ocurre al finalizar la transición.

- Eventos del portapapeles:

Eventos del portapapeles	
evento	Descripción
cut	Se produce cuando el usuario intenta cortar contenido del elemento.
copy	Se produce cuando el usuario intenta copiar contenido del elemento.
paste	Se produce cuando el usuario intenta pegar contenido.

1.- Modelos de gestión de eventos

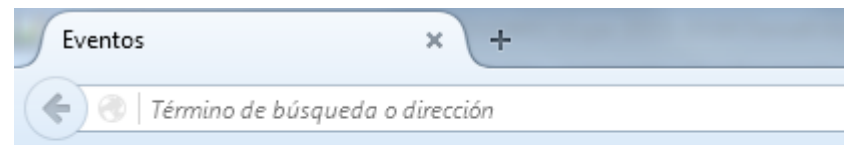
- Eventos especiales:

Eventos especiales	
evento	Descripción
offline	Solo funciona para el objeto <code>window</code> y se produce si el navegador se desconecta de la red.
online	Solo funciona para <code>window</code> y se produce si el navegador vuelve a conectarse a la red después de haber estado desconectado.
fullscreenchange	Ocurre cuando un elemento pasa a modo de pantalla completa.
fullscreenerror	Sucede si hay error al pasar un elemento a modo pantalla completa.
message	Evento que se asocia a numerosos elementos de envío de mensajes como los que se producen a través de las APIs <code>WebSockets</code> , <code>webWorkers</code> y otras.

Actividad 3



Resalta el campo que está activo en cada momento. Es decir, cuando el usuario se encuentre el campo nombre, el borde debe ser mas grueso y de color azul. Cuando el usuario pase a otro campo (pierda el foco), vuelva a su borde original.



Escribe tus datos personales

Nombre:

E-mail:

Actividad 3.1



Crea una aplicación que cree dos capas del mismo tamaño concretamente 200 pixels de ancho por 100 de alto.

La primera capa mostrará el texto “**Soy arrastrable**” y tendrá fondo amarillo y la segunda muestra “**Soy el destino**” y tiene el fondo blanco. Ambas tienen un borde de un pixel, negro.

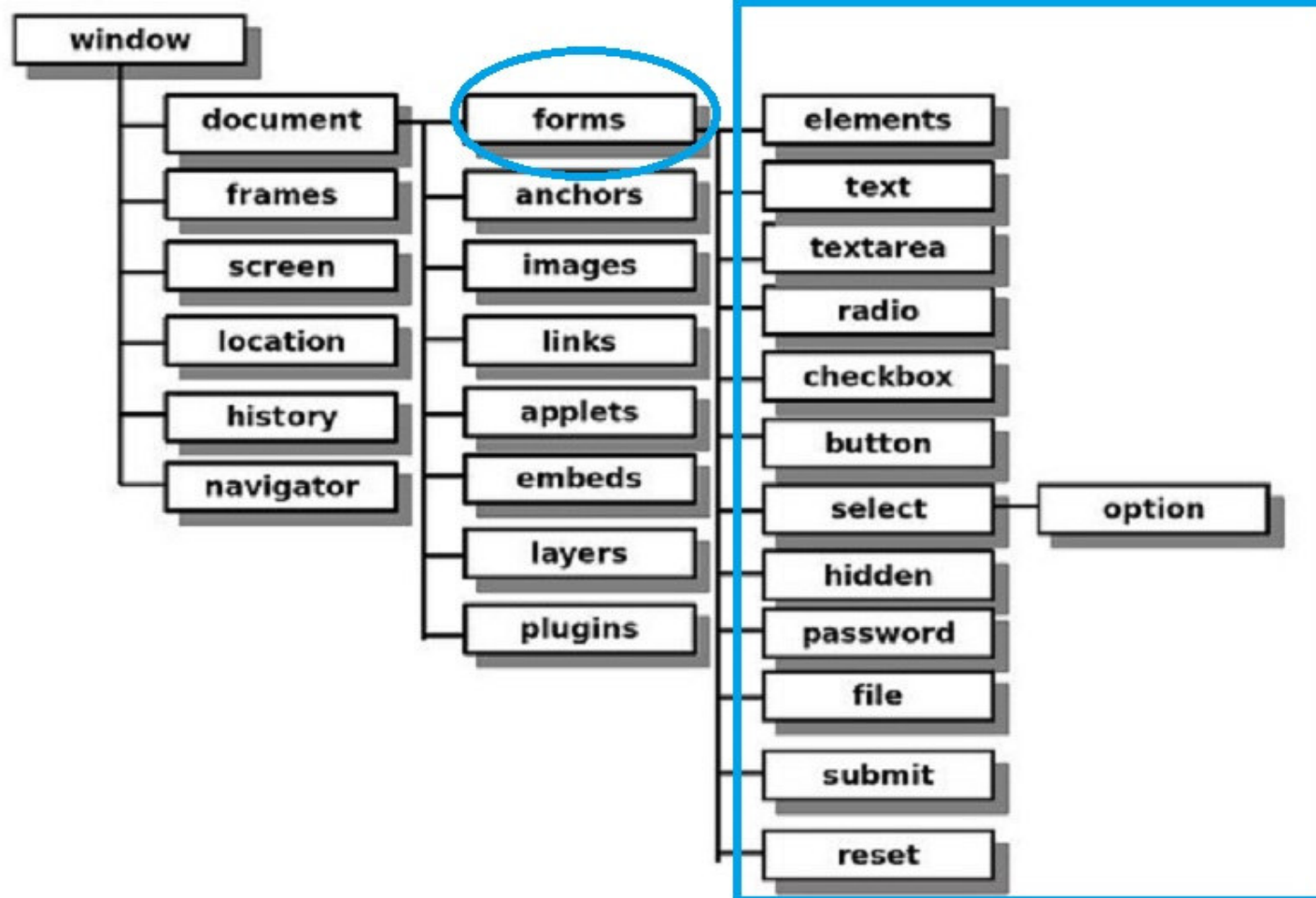
La aplicación permite arrastrar la primera sobre la segunda.

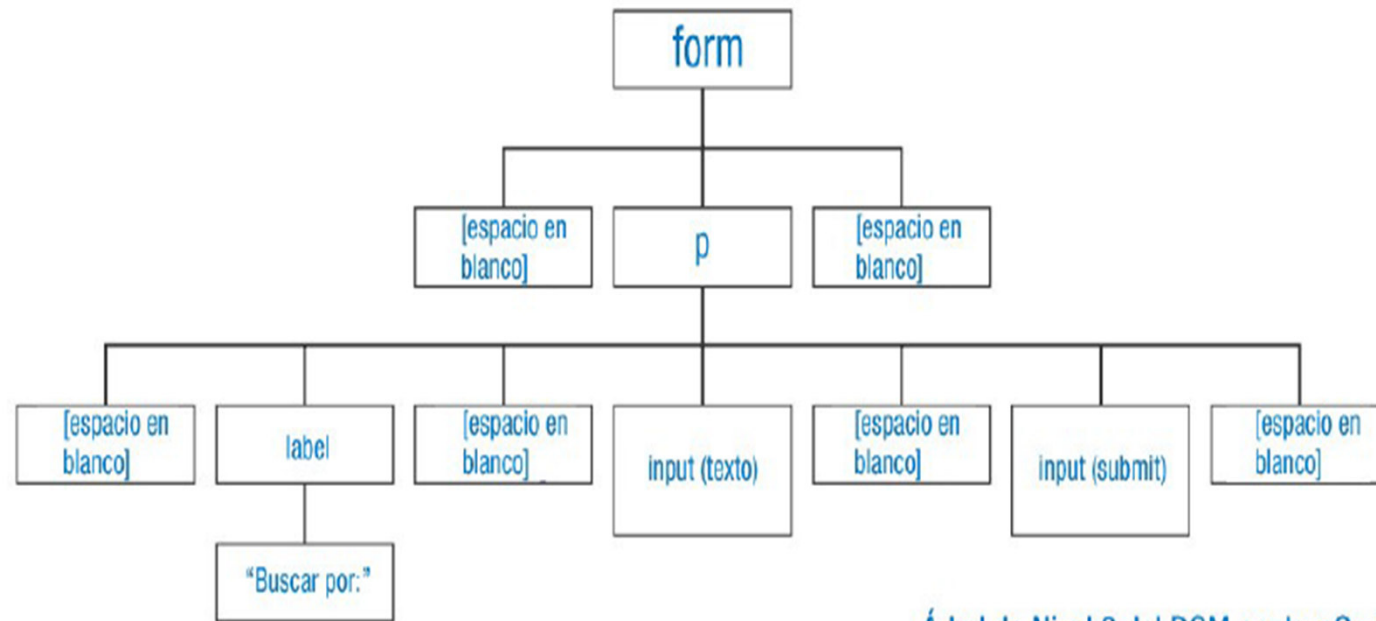
Durante el arrastre, la primera capa se mostrara con una opacidad del 50%. Al arrastrar sobre la segunda, esta (el destino) se muestra con fondo rojo. Al soltar en esta segunda capa, la primera desaparece y en la segunda aparecerá el texto “**Lo has logrado**”.



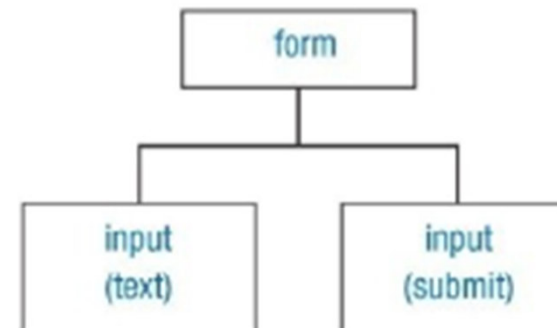
2.- Formularios

- Un formulario web sirve para enviar, tratar y recuperar datos que son enviados y recibidos entre un cliente y un servidor web.
- En JavaScript el objeto `form` depende en la jerarquía del objeto `document`.
- JavaScript dispone de numerosas propiedades y funciones que facilitan la programación de aplicaciones que manejan formularios.





Árbol de Nivel 2 del DOM en JavaScript



2.- Formularios

- Estructura de un formulario:
 - Los formularios se definen con etiquetas.
 - La etiqueta principal es `<form> </form>`.
 - Para que sea funcional, la etiqueta `<form>` necesita inicializar dos atributos:
 - `action` – Contiene la URL donde se redirigen los datos del formulario.
 - `method` – Indica el método por el cual el formulario envía los datos. Puede ser `POST` o `GET`.

2.- Formularios

- Ejemplo:

```
<html>
  <head><title>Ejemplo de formulario</title>
</head>
  <body>
    <h3>Formulario</h3>
    <form action="www.web.es/formulario.php"
      method="post">
      ...
    </form>
  </body>
</html>
```

2.- Formularios

- Cuando se carga una página web, el navegador crea automáticamente un array llamado `forms`, que contiene la referencia a todos los formularios de la página.
- Para acceder al array `forms`, se utiliza el objeto `document`, por lo que `document.forms` es el array que contiene todos los formularios de la página.
- Para acceder al primer formulario de la página:

```
document.forms[0];
```

2.- Formularios

- Además del array de formularios, el navegador crea automáticamente un array llamado `elements`, que contiene la referencia a todos los elementos (cuadros de texto, botones, listas desplegables, etc.) de ese formulario.
- Cada formulario de la página tendrá su array `elements` que nos permitirá acceder a sus elementos.
- Para acceder al primer elemento del primer formulario de la página:

```
document.forms[0].elements[0];
```

- Para acceder al ultimo elemento del primer formulario de la página:

```
document.forms[0].elements[document.forms[0].elements.length-1];
```

2.- Formularios

- Con el método de acceso a formularios anterior, si cambia el diseño de la página y el orden de los formularios, tendríamos que cambiar el código y el mantenimiento resulta tedioso.
- Accederemos a los formularios de una página a través de su nombre (atributo name) o a través de su atributo id.

```
var formPrincipal = document.mi_formulario;  
var formSecundario = document.otro_formulario;  
...
```

```
<form name="mi_formulario">  
</form>  
<form name="otro_formulario">  
</form>
```


2.- Formularios

- A través del método `getElementById()` del DOM nos permite acceder a un objeto a través de su atributo ID.

```
var fPri = document.getElementById("mi_formulario");  
var apellid = document.getElementById("apellidos");
```

...

```
<form id="mi_formulario">  
  <input type="text" id="apellidos" />  
</form>
```

2.- Formularios

- En resumen, para acceder a cualquier objeto dentro de nuestro documento o formulario:

```
document.getElementById("id-control") ;
```

```
document.nombreFormulario.name-del-control ;
```

```
<form id="formBusqueda" action="">  
  <input type="text" id="entrada" name="cEntrada" />  
  <input type="submit" id="enviar" name="enviar" />  
</form>
```

- Las siguientes referencias al campo de texto entrada, serán todas válidas:

```
document.getElementById("entrada");  
document.formBusqueda.cEntrada;  
document.formBusqueda.elements[0];  
document.forms["formBusqueda"].cEntrada;  
document.forms["formBusqueda"].elements["cEntrada"];
```

2.- Formularios: Elementos

- Cuadro de texto y textarea:
 - El valor de texto mostrado por estos elementos se obtiene y se establece mediante la propiedad `value`.

```
<input type="text" id="texto" />  
var valor = document.getElementById("texto").value;
```

```
<textarea id="parrafo"></textarea>  
var txt = document.getElementById("parrafo").value;
```

2.- Formularios: Elementos

- **Checkbox:**

- Para saber si un campo de tipo checkbox está o no marcado, disponemos de la propiedad `checked`.

```
<p>Colores favoritos</p>  
<input type="checkbox" id="rojo" name="rojo" value="ro"/> Rojo  
<input type="checkbox" id="azul" name="azul" value="az"/> Azul  
<input type="checkbox" id="verde" name="verde" value="ve"/> Verde
```

Colores favoritos

☐ Rojo

☐ Azul

☐ Verde

- Para marcar un elemento mediante código:

```
document.getElementById("verde").checked = true;
```

2.- Formularios: Elementos

- Radiobutton:

- Debemos asignar el mismo atributo `name` a cada uno de los botones del grupo.
- Lo importante es conocer cuál de todos los radiobuttons se ha seleccionado mediante la propiedad `checked`, la cuál devuelve `true` para el radiobutton seleccionado y `false` en cualquier otro caso.

Género

```
</br><input type="radio" name="genero" value="M"> Hombre  
</br><input type="radio" name="genero" value="F"> Mujer
```

Género

☐ Hombre

☐ Mujer

Actividad 4



Haciendo uso de un formulario, realiza un cuestionario al usuario para que seleccione su actor favorito de entre tres facilitados cualesquiera.

2.- Formularios: Elementos

- Select (listas desplegables):
 - En general, lo que se requiere es obtener el valor del atributo `value` de la opción seleccionada por el usuario.

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

- Para poder obtener el valor seleccionado se utilizarán las siguientes propiedades:

2.- Formularios: Elementos

- Select (listas desplegables):

- `options`, es un array creado automáticamente por el navegador para cada lista desplegable y que contiene la referencia a todas las opciones de esa lista.
- Para acceder a la primera opción de una lista:

```
document.getElementById("id-lista").options[0]
```

- `selectedIndex`, devuelve el índice de la opción seleccionada.

```
document.getElementById("id-lista").selectedIndex;
```

(El valor de `selectedIndex` para la primera opción de la lista será 0.)

Actividad 5



Utiliza una lista desplegable para que el usuario elija una provincia de entre cuatro y muestra mediante un mensaje de aviso la opción seleccionada.



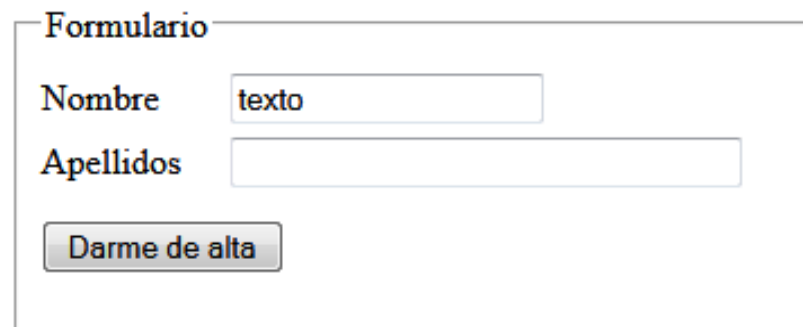
3.- Modificación de la apariencia y comportamiento de un formulario.

- Por defecto los formularios tienen unos estilos asignados. Estos estilos tienen unos colores y unos bordes determinados.
- Para modificar el aspecto de un formulario es necesario utilizar otros estilos.
- El lenguaje que maneja los estilos en HTML se llama *Cascading Style Sheets* (CSS). A través de hojas de estilo es posible mejorar notablemente el aspecto de los formularios.

3.- Modificación de la apariencia y comportamiento de un formulario.

- Organizar controles de un formulario:

```
<fieldset><legend>Formulario</legend>
  <div>
    <label for="nombre">Nombre</label>
    <input type="text" id="nombre" />
  </div>
  <div>
    <label for="apellidos">Apellidos</label>
    <input type="text" id="apellidos" size="35" />
  </div>
  <input class="btn" type="submit" value="Darme de alta"
/>
</fieldset>
```



The image shows a visual rendering of the HTML code provided. It features a rectangular box with a thin border. At the top left of the box is the title 'Formulario'. Below the title, there are two labels: 'Nombre' and 'Apellidos'. The 'Nombre' label is followed by a text input field that contains the placeholder text 'texto'. The 'Apellidos' label is followed by a longer text input field. At the bottom of the box, centered, is a button with the text 'Darme de alta'.



3.- Modificación de la apariencia y comportamiento de un formulario.

- Los formularios tienen unas acciones predeterminadas por defecto.
- Sin embargo es posible darle otro comportamiento a alguno de sus elementos.
- Por ejemplo podríamos querer que los datos se envíen a URLs diferentes en base a un dato que introduzca el usuario.

3.- Modificación de la apariencia y comportamiento de un formulario.

```
<script language="javascript">
    function enviar(form) {
        if (formulario.alta.checked == true) {
            formulario.action = "paginas/alta.html";
        }
        if (formulario.alta.checked == false) {
            formulario.action = "paginas/baja.html";
        }
        form.submit()
    }
</script>
```

- Para que la función enviar sea ejecutada, utilizaremos un input de tipo `button` que contenga el evento:
`onclick= "enviar(this.form)";`



4.- Validación y envío.

- El usuario puede cometer errores al rellenar un formulario.
- Si por ejemplo se espera un código postal y se introduce el nombre de una ciudad, se producirá un error.
- Con HTML5 se facilita este trabajo, aunque no todos los navegadores cumplen con todo el estándar.
- Para controlar estas situaciones se pueden usar las validaciones.

4.- Validación y envío.

- Propiedades de restricción:
 - **maxLength**: define la longitud máxima que puede tener un valor.
 - **pattern**: permite establecer una expresión regular que debe cumplir el campo.
 - **min, max**: establece respectivamente, el valor mínimo y máximo que puede tener un campo.
 - **step**: especifica los incrementos permitidos para un valor.

4.- Validación y envío.

- Validar un campo como obligatorio:

```
<script type="text/javascript">
  function validacion() {
    valor = document.getElementById("campo").value;

    if( valor == null || valor.length == 0 ){
      alert("El campo no puede ser vacío");
      return false;
    }
    return true;
  }
</script>
```

4.- Validación y envío.

- Validar un campo de texto como numérico:

```
<script type="text/javascript">
  function validaNum() {
    valor = document.getElementById("telefono").value;

    if( isNaN(valor) ) {
      alert("El campo tiene que ser numérico");
      return false;
    }
    return true;
  }
</script>
```

4.- Validación y envío.

- Validar un checkbox:

```
<script type="text/javascript">
  function validaCheck() {
    elemento = document.getElementById("campo");

    if( !elemento.checked ) {
      return false;
    }
    return true;
  }
</script>
```

4.- Validación y envío.

- Este tipo de validaciones se suelen realizar llamando a una función que analice si el dato cumple con las restricciones establecidas.
- ¿Cuándo se lanza la validación?
 - Cuando nos situamos en otro campo (pierde el foco)
 - Cuando pulsamos el botón de tipo **submit** del formulario.

El navegador se encarga de aplicar las restricciones, resaltando en rojo mientras no sea válida e impidiendo que se envíe el formulario.

4.- Validación y envío.

- JavaScript integra dos métodos sobre la validación de campos:
 - **checkValidity()**: devuelve un valor `true` si, tras lanzar las validaciones asociadas, el campo no presenta ningún error.
 - **setCustomValidity(mensaje)**: establece un mensaje personalizado de error para el campo.

Actividad 6.

Crea un formulario y valida sus campos.

- Validacion de un Formulario -

Nombre:	<input type="text"/>
Apellidos:	<input type="text"/>
Edad:	<input type="text"/>
Matricula Coche:	<input type="text" value="XXXX AAA"/>
Provincia:	<input type="text" value="Seleccione Provincia ▼"/>
<input type="button" value="Limpiar"/> <input type="button" value="Enviar"/>	

Los campos **Nombre**, **Apellidos** y **Edad** no pueden estar en blanco.

El campo **Edad**, es numérico y debe estar comprendido entre 0 y 105.

El campo **Matricula Coche** debe estar formado por 4 números espacio en blanco (opcional) y 3 letras de la A-Z en mayúsculas.

El campo **Provincia** mostrará una lista desplegable con el nombre de tres provincias. Deberá estar una de ellas seleccionada.

5. - Expresiones regulares

- Las expresiones regulares describen un conjunto de elementos que siguen un patrón.
- Las expresiones regulares son utilizadas para buscar, reemplazar y extraer información de las cadenas de caracteres.
- Un ejemplo podría ser todas las palabras que comienzan por la letra 'a' minúscula.
- JavaScript implementa expresiones regulares y facilita las comprobaciones de ciertos datos que deben seguir una estructura concreta.
- Al igual que la mayoría de elementos en JavaScript, las expresiones regulares son objetos.

5. - Expresiones regulares

- Los métodos que funcionan con expresiones regulares en JavaScript son:

Método	Descripción
exec	Un método <code>RegExp</code> que ejecuta una búsqueda por una coincidencia en una cadena. Devuelve un array de información.
test	Un método <code>RegExp</code> que verifica una coincidencia en una cadena. Devuelve <code>true</code> o <code>false</code> .
match	Un método <code>String</code> que ejecuta una búsqueda por una coincidencia en una cadena. Devuelve un array de información o <code>null</code> si no existe coincidencia alguna.
search	Un método <code>String</code> que verifica una coincidencia en una cadena. Devuelve el índice de la coincidencia, o <code>-1</code> si la búsqueda falla.
replace	Un método <code>String</code> que ejecuta una búsqueda por una coincidencia en una cadena, y reemplaza la subcadena encontrada con una subcadena de reemplazo.
split	Un método <code>String</code> que utiliza una expresión regular o una cadena fija para cortar una cadena y colocarlo en un array de subcadenas.

5. - Expresiones regulares

- En JavaScript las expresiones regulares se gestionan a través del objeto **RegExp**.
- Para crear un literal del tipo RegExp, hay que usar la siguiente sintaxis:

```
var expresion = /expresion_regular/;
```

- La expresión regular está contenida entre las barras /
- Las expresiones regulares están hechas de caracteres solos o en combinación con caracteres especiales.

5. - Expresiones regulares

- Caracteres especiales:
 - **^ Principio de entrada o línea.** Este carácter indica que las cadenas deberán comenzar por el siguiente carácter. Si este fuera una “a” minúscula como indicamos en el punto anterior la expresión regular sería **^a**.
 - **\$ Fin de entrada o línea.** Indica que la cadena debe terminar por el elemento precedido al dólar.

5. - Expresiones regulares

- Caracteres especiales:
 - *** El carácter anterior 0 o más veces.** El asterisco indica que el carácter anterior se puede repetir en la cadena 0 o más veces.
 - **+ El carácter anterior 1 o más veces.** El símbolo más indica que el carácter anterior se puede repetir en la cadena una o más veces.
 - **? El carácter anterior una vez como máximo.** El símbolo interrogación indica que el carácter anterior se puede repetir en la cadena cero o una vez.

5. - Expresiones regulares

- Caracteres especiales:
 - **.** **Cualquier carácter individual.** El símbolo punto indica que puede haber cualquier carácter individual salvo el de salto de línea.
 - **x|y** **x ó y:** La barra vertical indica que puede ser el carácter x o el y.
 - **{n}** **n veces el carácter anterior.** El carácter anterior a las llaves tiene que aparecer exactamente n veces.

5. - Expresiones regulares

- Caracteres especiales:
 - **{n,m}** Entre n y m veces el carácter anterior.
El carácter anterior a las llaves tiene que aparecer como mínimo n y como máximo m veces.
 - **[abc]** Cualquier carácter de los corchetes.
En la cadena puede aparecer cualquier carácter que este incluido en los corchetes.

5. - Expresiones regulares

- Caracteres especiales:
 - **[^abc]** Un carácter que no esté en los corchetes. En la cadena pueden aparecer todos los caracteres que no estén incluidos en los corchetes.
 - **\b** Fin de palabra. Este símbolo indica que tiene que haber un fin de palabra o retorno de carro.
 - **\B** No fin de palabra. El símbolo **\B** indica cualquiera que no sea un límite de palabra.

5. - Expresiones regulares

- Caracteres especiales:
 - **\d** **Cualquier carácter dígito.** Este símbolo indica que puede haber cualquier carácter numérico, de 0 a 9.
 - **\D** **Carácter que no es dígito.** Este símbolo indica que puede haber cualquier carácter siempre que no sea numérico.
 - **\f** **Salto de página.** Este símbolo indica que tiene que haber un salto de página.

5. - Expresiones regulares

- Caracteres especiales:
 - **\n Salto de línea.** Este símbolo indica que tiene que haber un salto de línea.
 - **\r Retorno de carro.** Este símbolo indica que tiene que haber un retorno de carro.
 - **\s Cualquier espacio en blanco.** Este símbolo indica que tiene que haber un carácter individual de espacio en blanco: espacios, tabulaciones, saltos de página o saltos de línea.

5. - Expresiones regulares

- Caracteres especiales:
 - **\S** **Carácter que no sea blanco.** Este símbolo indica que tiene que haber cualquier carácter individual que no sea un espacio en blanco.
 - **\t** **Tabulación.** Este símbolo indica que tiene que haber cualquier tabulación.
 - **\w** **Carácter alfanumérico.** Este símbolo indica que puede haber cualquier carácter alfanumérico.
 - **\W** **Carácter que no sea alfanumérico.** Este símbolo indica que puede haber cualquier carácter que no sea alfanumérico.

5. - Expresiones regulares

- Validar un formulario con expresiones regulares:
 - Combinando las anteriores expresiones se puede abordar una infinidad de patrones para validar datos en los formularios.
 - Se pueden validar por ejemplo campos como:
 - Correo electrónico
 - Teléfono
 - Código postal
 - DNI
 - Etc.

5. - Expresiones regulares

- **Validar un email:**

- nombre_usuario + @ + servidor + dominio
- Debe de empezar por letra o numero. Al menos tiene una letra o número. La letra o número se expresa mediante el carácter `\w`. Para asegurarnos de que la letra o número aparece al menos una vez utilizaremos el modificador `+`.
- Puede contener puntos y guiones `[\.-]` cero o una vez `?` y otras letras y números `\w`. Esta combinación podrá aparecer mas veces, así que utilizamos el modificador `*` (cero o varias veces). Insertaremos toda la combinación entre paréntesis.

`\w+([\.-]\w+)*`

- La expresión anterior nos vale para el usuario y servidor.

5. - Expresiones regulares

- **Validar un email:**

- Para el dominio:
- Irá al final, detrás de un punto `\.` . Podrá tener dos (`.es`, `.fr`, `.it`,...) o tres letras (`.com`, `.net`, `.org`,...) o cuatro (`.mobi`, `info`,...). Lo expresamos con el número entre los operadores `{ y }`
- Además podemos tener varios dominios seguidos (`.com.ar`, `.com.uk`,....) es por ello que deberemos de usar el modificador `+`. Ya que el dominio podrá aparecer varias veces.
- La expresión regular para el dominio sería:

`(\.\w{2,3,4})+`

- La expresión regular final para validar el email:

`/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3,4})+$/`

5. - Expresiones regulares

- Validar una dirección de correo electrónico:

```
<script type="text/javascript">
function validaEmail() {
    valor = document.getElementById("campo").value;

    if (/^\w+([\.-]?\w+)*@\w+([\.-]
]?\w+)*(\.\w{2,3,4})+$/ .test(valor)) {
        alert("La dirección de email " +valor+ " es
correcta.");
    } else {
        alert("La dirección de email es incorrecta.");
    }
}
</script>
```

5. - Expresiones regulares

- Validar un DNI:

```
<script type="text/javascript">
  function validaDNI() {
    valorDNI = document.getElementById("dni").value;
    var letras =
      ['T','R','W','A','G','M','Y','F','P','D','X',
       'B','N','J','Z','S','Q','V','H','L','C','K','E','T'];
    if( !(/^\d{8}[A-Z]$/.test(valorDNI)) ) { return false;
    }
    if(valorDNI.charAt(8) != letras[(valorDNI.substring(0,
8))%23]) { return false; }

    return true;
  }
</script>
```

5. - Expresiones regulares

- Validar un número de teléfono:

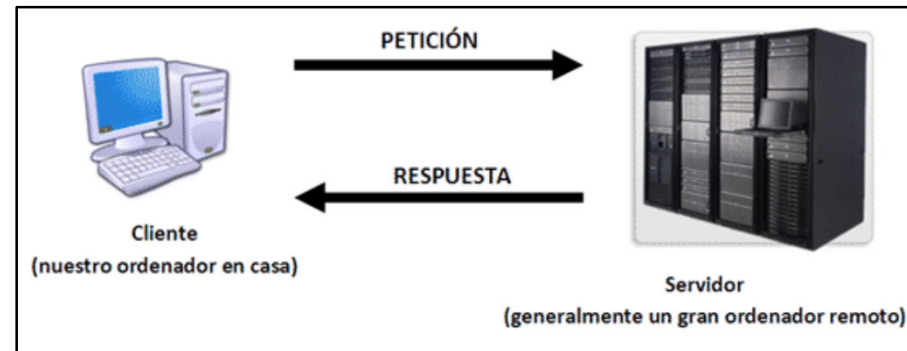
```
function validaTelefono() {  
    valor = document.getElementById("telefono").value;  
  
    if( !(/^\d{9}$/.test(valor)) ) {  
        return false;  
    }  
    return true;  
}
```

6. – Utilización de cookies.

- Las cookies surgieron como necesidad ante algunas ausencias tecnológicas del protocolo HTTP.
- Las cookies son información que se almacena en el navegador de forma persistente.
- Una cookie es un fichero de texto que se almacena en el navegador (ordenador del cliente) .
- Son un fichero propio de cada navegador.
- Surgieron con el fin de mantener la información de los carritos de compra virtuales a través de la web.
- Pueden ser creadas por:
 - El servidor y enviarlas al navegador para que las almacene.
 - Mediante JavaScript en el navegador, las almacena en el navegador y envíasalas posteriormente al servidor.

6. – Utilización de cookies.

Navegador sin cookies



Navegador con cookies





6. – Utilización de Cookies.

- Mantener opciones de visualización:
 - Las *cookies* son utilizadas en ocasiones para mantener unas preferencias de visualización.
 - Algunas páginas como *Google*, permiten que el usuario haga una configuración de su página de inicio en el buscador, a través de las *cookies* el servidor reconoce ciertos aspectos que el usuario configuró y conserva el aspecto.

6. – Utilización de cookies.

- Almacenar variables:
 - El servidor puede utilizar las *cookies* para almacenar variables que se necesiten utilizar en el navegador.
 - Un ejemplo sería, una página en la que nos solicitan unos datos, en la siguiente nos solicitan otros datos y así hasta la página final. Los datos de las páginas anteriores se irán almacenando en las *cookies* hasta que se finaliza el ciclo del formulario y el usuario envía los datos al servidor.
 - Antes del envío al servidor se recuperarán todos los campos del formulario que están guardados en las *cookies*.



6. – Utilización de cookies.

- Realizar un seguimiento de la actividad del usuario:
 - En ocasiones los servidores hacen uso de las *cookies* para almacenar ciertas preferencias y hábitos que el usuario tiene a la hora de navegar.
 - Con esta información, el servidor personaliza sus servicios y publicidad orientándolo a cada cliente en particular.
 - Estos fines no son del todo lícitos si la entidad que realiza estas actividades no avisa al usuario de que está realizando estas acciones.

6. – Utilización de cookies.

- Autenticación:
 - Autenticar a los usuarios es uno de los usos más habituales de las *cookies*.
 - A través de las *cookies*, el navegador guarda los datos del usuario, al realizar una petición al servidor, el navegador envía las *cookies* junto con la petición.
 - Las *cookies* tiene una caducidad, cuando pasa un periodo de tiempo establecido, estas desaparecen junto con el fichero de texto que guarda el navegador.
 - Habitualmente, como mecanismo de seguridad, las aplicaciones Web aplican un tiempo máximo de inactividad, por ejemplo de 15 minutos, tras el cual las *cookies* caducan, si no se produjo movimiento en la navegación de la aplicación web.



6. – Utilización de cookies.

- Lectura y escritura de las cookies:
 - Los dos procesos de implementación principales de una *cookie* son la escritura y la lectura de la misma.
 - A continuación se presentan tres funciones que sirven para:
 - Devolver el valor de una cookie.
 - Escribir una cookie.
 - Comprobar si existe un valor para la cookie.

6. – Utilización de cookies.


```
<html>
<head>
<script type="text/javascript">
function getCookie(c_name){
var
i,x,y,ARRcookies=document.cookie.split(";
");
for (i=0;i<ARRcookies.length;i++){

x=ARRcookies[i].substr(0,ARRcookies[i].in
dexOf("="));

y=ARRcookies[i].substr(ARRcookies[i].inde
xOf("=")+1);
x=x.replace(/^\\s+|\\s+$ /g, "");

if (x==c_name){
return unescape(y);
}
}
}
```

```
function setCookie(c_name,value,exdays){
var exdate=new Date();
exdate.setDate(exdate.getDate() + exdays);
var c_value=escape(value)+((exdays==null)?"" :";
expires="+exdate.toUTCString());
document.cookie=c_name + "=" + c_value;
}
function checkCookie(){
var username=getCookie("username");
if (username!=null && username!=""){
alert("Bienvenido " + username);
}else{
username=prompt("Por favor, Introduzca su
usuario:", "");
if (username!=null && username!=""){
setCookie("username",username,365);
}
}
}
</script>
</head>
<body><input type="button" name="chequeaCookie"
value="Chequear las cookies"
onclick="checkCookie();">
</body>
</html>
```



Fin de la Unidad 6

Gestión de eventos y Formularios