DESARROLLO WEB EN ENTORNO CLIENTE

UD5 – EVENTOS Y FORMULARIOS

CICLO FORMATIVO DE GRADO SUPERIOR EN DESARROLLO DE APLICACIONES WEB

I.E.S. HERMENEGILDO LANZ – 2022/2023

PROFESORA: VANESA ESPÍN

CRITERIOS de evaluación del RA5	%UD	%Curso
a) Se han reconocido las posibilidades del lenguaje de marcas relativas a la captura de los eventos producidos.	**10%**	**15%**
b) Se han identificado las características del lenguaje de programación relativas a la gestión de los eventos.	**10%**	
c) Se han diferenciado los tipos de eventos que se pueden manejar.	**10%**	
d) Se ha creado un código que capture y utilice eventos.	**15%**	
e) Se han reconocido las capacidades del lenguaje relativas a la gestión de formularios Web.	**15%**	
f) Se han validado formularios Web utilizando eventos.	**15%**	
g) Se han utilizado expresiones regulares para facilitar los procedimientos de validación.	**15%**	
h) Se ha probado y documentado el código.	**10%**	

Resultados de Aprendizaje y Criterios de Evaluación

RA5. Desarrolla aplicaciones Web interactivas integrando mecanismos de manejo de eventos

Objetivos Didácticos de la Unidad 5

1. Conocer los distintos modelos de gestión de eventos.
2. Usar y validar formularios

IMPORTANTE: no todo lo que aprenderemos está en las diapositivas, HAY QUE SEGUIR LOS ENLACES DE REFERENCIA INDICADOS Y HACER LOS EJERCICIOS

Índice

Modelo 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 (DocumentObjectModel).
- Por tanto es el DOM el encargado de gestionar los eventos.

Modelo de Gestión de Eventos

- Para poder controlar un evento se necesita un manejador.
- El manejador es la palabra reservada que indica la acción que va a manejar.

- En el caso del evento ***click***, el manejador sería ***onclick***. Ejemplo:

```
<img src="mundo.jpg" onclick="alert('Click en imagen');">
```

Modelo de gestión de eventos en línea

Modelo de gestión de eventos

El evento anterior se podría hacer mediante una función

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

<body>
    <img src="mundo.jpg" onclick="func1();">
</body>
</html>
```

Modelo de gestión de eventos en línea

Modelo de gestión de eventos

En los navegadores antiguos, el modelo que se utilizaba era el modelo en línea.

Con la llegada de DHTML, el modelo se extendió para ser más flexible. En este nuevo modelo el evento pasa a ser una propiedad del elemento, ya podemos usar el siguiente código de JavaScript:

Ojo. No está estandarizado por el W3C

```
elemento.onclick = hacerAlgo; // cuando el usuario haga click en el objeto, se llamará a la función hacerAlgo()
```

- Para eliminar un gestor de eventos de un elemento, 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.click(); // así disparamos el evento click de forma manual y se ejecutará la función hacerAlgo()
```

Modelo de gestión de eventos avanzado

- El modelo de gestión de eventos en línea no se recomienda ya que estamos mezclando la estructura de la página web con la programación de la misma.
  Para evitar este problema, el W3C ofrece una manera sencilla de registrar los eventos que queramos, sobre un objeto determinado: el método ***addEventListener()***.
- Sintaxis
  - `eventTarget.addEventListener(event, function)`
  - `eventTarget.addEventListener(event, function, options)`
  - `eventTarget.addEventListener(event, function, useCapture)`

  Donde:
  - *event*. Cadena que representa el tipo de evento a escuchar.
  - *function*. El objeto que recibe una notificación cuando ocurre un evento de ese tipo. Puede ser *null*, objeto con método handleEvent o función JavaScript.
  - *options*. Diferentes opciones
  - *useCapture*. Cuando se captura evento. *true* (fase de captura) o *false* (fase de burbujeo), por defecto.

https://www.w3schools.com/jsref/met_document_addeventlistener.asp
https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener

Ejercicio 1. Ejemplo de addEventListener

HTML

```
<table id="outside">
    <tr>
      <td id="t1">one</td>
    </tr>
    <tr>
      <td id="t2">two</td>
    </tr>
  </table>
```

JAVASCRIPT

```
function modifyText() {
    const t2 = document.getElementById("t2");
    const isNodeThree = t2.firstChild.nodeValue === "three";
    t2.firstChild.nodeValue = isNodeThree ? "two" : "three";
  }

  // Add event listener to table
  const el = document.getElementById("outside");
  el.addEventListener("click", modifyText, false);
```

1. Programar este código para que funcione con dos archivos y explicar su funcionamiento ¿dónde debemos incluir la llamada al script? ¿por qué?
2. Añadir otro evento para que llame a la **función consola** que muestre un saludo por consola al pinchar la tabla
3. Añadir otro evento para que cambie el color del fondo de t1 a azul usando una **función anónima**

Ejercicio 1. cont. this y e.target

En este ejercicio vamos a diferenciar el uso de ***this*** y ***e.target*** en la llamada desde un evento.

1. Crea las funciones:

   saludar1 que salude con *alert* al identificador de ***this***

   saludar2 que salude con *alert* al identificador de ***e.target*** (***e*** debe pasarse como parámetro)

2. Crea los dos eventos correspondientes y pruébalos
3. ¿Cuál es la diferencia?
4. Crea una función *cambiaColor* para que cambie SOLO el color del texto de la fila pinchada de negro a azul o de azul a negro.

NOTA. Hemos añadido el evento sobre la tabla, sin embargo detecta los eventos en las filas → **Delegación de eventos**, que veremos más adelante

El evento load de window

- El inconveniente de esta forma de añadir eventos es que la página se debe cargar completamente antes de que se puedan utilizar las funciones DOM que asignan los manejadores a los elementos HTML. Por eso teníamos que poner la llamada al final, pero a veces tampoco es seguro su correcto funcionamiento.
- Una de las formas más sencillas de asegurar que cierto código se va a ejecutar después de que la página se cargue por completo es utilizar el evento ***onload***:

```
window.onload = function() {
  document.getElementById("pinchable").onclick = muestraMensaje;
}
```

OTRA FORMA: `window.addEventListener('load', inicializa);`

Ejercicio 1. fin

- Modificar el ejercicio 1 para asegurarnos de que los eventos se crean al cargarse la página usando *windows.onload* y subiendo la llamada al script a la cabecera del HTML.

Sintaxis de escucha de eventos en ES6

```
elemento.addEventListener('eventType', () =>
// eventHandler
);
```

Usamos función flecha anónima, donde:

- `eventType` es un evento o un tipo de evento
- `eventHandler` es la función/código que se ejecutará cuando el `eventType` se dispare.

Ten en cuenta que si usas una arrow function para el eventHandler no podrás capturar el elemento que dispara el evento usando this (como en ES5) por eso se recomienda usar e.target

Ejercicio 2

Añadir un evento a una imagen para que se agrande al hacer clic sobre ella usando la nueva sintaxis.

Eliminar un evento

Sintaxis

- `removeEventListener(type, listener)`
- `removeEventListener(type, listener, options)`
- `RemoveEventListener(type, listener, useCapture)`

Ojo: si un elemento tiene varios eventos habrá que eliminarlos uno a uno.

AÑADIR UN BOTÓN AL EJERCICIO 1 QUE ELIMINE LOS DOS EVENTOS SALUDAR

Cancelar un evento

- El método ***`preventDefault()`*** cancela el evento si es cancelable, lo que significa que la acción predeterminada que pertenece al evento no ocurrirá.
- Por ejemplo, esto puede ser útil cuando:
  - Al hacer clic en el botón "Enviar", evita que envíe un formulario
  - Al hacer clic en un enlace, evita que el enlace siga la URL
- Notas:
  - No todos los eventos son cancelables. Utilizamos la propiedad *`cancelable`* para averiguar si un evento es cancelable.
  - El método *`preventDefault()`* no previene la propagación de un evento a través del DOM. Use el método *`stopPropagation()`* para manejar esto.

Ejercicio 3

- Crea dos enlaces con href a otras páginas.
- Cancela uno de los enlaces mediante ***`preventDefault()`***

Tipos de eventos

- https://developer.mozilla.org/en-US/docs/Web/Events
- https://www.w3schools.com/tags/ref_eventattributes.asp
- Existen varios tipos de eventos según su origen:
  - Eventos del ratón (Mouse Event).
  - Eventos del teclado (Keyboard Event).
  - Eventos para formulario (aplicables a los elementos HTML).
  - Eventos de ventana.
  - Eventos de clipboard.
  - Eventos de drag & drop
  - … etc

Eventos del ratón

Métodos y propiedades en: https://www.w3schools.com/jsref/obj_mouseevent.asp

Event	Description
onclick	The event occurs when the user clicks on an element
oncontextmenu	The event occurs when the user right-clicks on an element to open a context menu
ondblclick	The event occurs when the user double-clicks on an element
onmousedown	The event occurs when the user presses a mouse button over an element
onmouseenter	The event occurs when the pointer is moved onto an element
onmouseleave	The event occurs when the pointer is moved out of an element
onmousemove	The event occurs when the pointer is moving while it is over an element
onmouseout	The event occurs when a user moves the mouse pointer out of an element, or out of one of its children
onmouseover	The event occurs when the pointer is moved onto an element, or onto one of its children
onmouseup	The event occurs when a user releases a mouse button over an element

Eventos del teclado

Métodos y propiedades en: https://www.w3schools.com/jsref/obj_keyboardevent.asp

Event	Description
onkeydown	The event occurs when the user is pressing a key
onkeypress	The event occurs when the user presses a key
onkeyup	The event occurs when the user releases a key

EJEMPLO: onkeypress + propiedad charCode	
charCode	Returns the Unicode character code of the key that triggered the event

```
<input type="text" size="40" onkeypress="myFunction(event)">
<script>
function myFunction(event) {
  alert ("The Unicode value is: " + event.charCode);
}
</script>
```

Eventos de formularios HTML

Attribute	Description
onblur	Fires the moment that the element loses focus
onchange	Fires the moment when the value of the element is changed
oncontextmenu	Script to be run when a context menu is triggered
onfocus	Fires the moment when the element gets focus
oninput	Script to be run when an element gets user input
oninvalid	Script to be run when an element is invalid
onreset	Fires when the Reset button in a form is clicked
onsearch	Fires when the user writes something in a search field (for <input="search">)
onselect	Fires after some text has been selected in an element
onsubmit	Fires when a form is submitted

Información sobre el evento

- La propiedad *type* nos dice el tipo de evento producido

```
var tipo = evento.type;
```

- Ejemplo:

HTML

```
<p>Press any key or click
the mouse to get the
event type.</p>
<p id="log"></p>
```

JAVASCRIPT

```
function getEventType(event) {
   const log = document.getElementById('log');
   log.innerText += `${event.type}\n`;
}

// Keyboard events
document.addEventListener('keydown', getEventType, false);
document.addEventListener('keyup', getEventType, false);

// Mouse events
document.addEventListener('mousedown', getEventType, false);
document.addEventListener('click', getEventType, false);
```

Ejercicio 4

- Dado el siguiente div:

```
<div id="contenidos" style="width:150px; height:60px; border:2px solid grey">
   Sección de contenidos...
</div>
```

- Asociarle los siguientes eventos:
  1. Cuando se pasa el ratón por encima se cambie el color del borde a verde
  2. Cuando deja de pasarse el ratón por encima el borde pasará a ser de color gris.
- Se usará una única función llamada *resalta* para los 2 eventos y se usará un switch para determinar el evento

Ejercicio 5

- Realizar una función para mostrar las diferentes propiedades del evento recibido al pulsar una tecla sobre un input.
- Las propiedades se mostrarán del modo:
  “la propiedad X tiene el valor Y”
  dentro de un div con identificador “info”.
- Para ello usaremos el evento “onkeydown”

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

Delegación de eventos

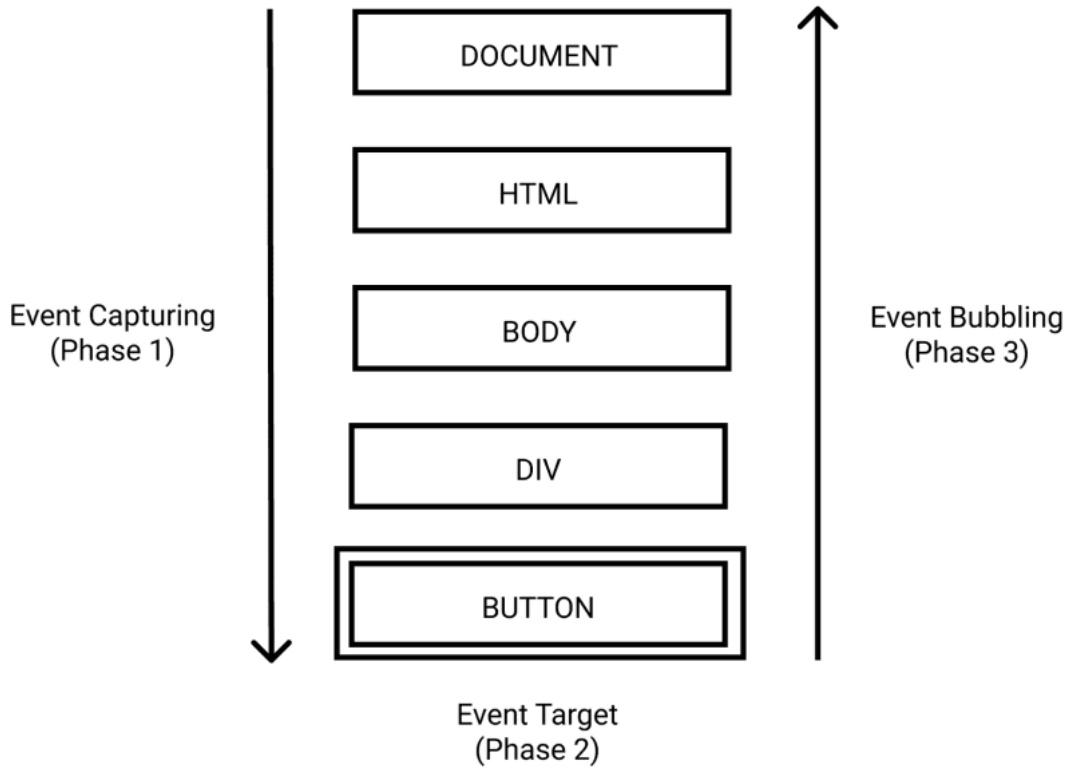
- El principal problema con los enfoques vistos es que solo aplica a los elementos existentes en el DOM al cargar la página.
- Si añadimos elementos dinámicamente, estos no se verán afectados por el evento.
- Para eso utilizamos la **Delegación de Eventos** que consiste en escuchar el evento en el elemento padre solamente para luego capturarlo cuando ocurra en sus hijos. Esto gracias a un comportamiento de los eventos llamado ***Bubbling***.

Bubbling y capturing

- El burbujeo es una fase de los eventos (la otra es *capturing*) que significa que cuando un evento ocurre en el DOM, es capturado en el elemento HTML más profundo posible, y de ahí se va elevando hacia sus padres en orden de jerarquía hasta llegar al objeto global (window).
- Por lo tanto, cuando un evento ocurre en un elemento, también está ocurriendo en sus padres. El bubbling es el comportamiento por defecto.

Resumen. Fases de captura del evento

- **Fase de Captura**: se busca el elemento mas profundo en el DOM que tenga registrado un evento en su listener.
- **Fase de Target**: ejecuta el evento del elemento en si.
- **Fase de Burbuja**: verifica si los elementos padre de dicho elemento tienen eventos registrados en sus listeners, si es así, ejecuta dichos eventos de manera automática.



Este es el funcionamiento por defecto, pero puede cambiarse

Ejercicio 6

```css
<style>
   a img{
      width:200px;
   }
   .gallery-container{
      display: flex;
      flex-wrap: wrap;
      column-gap: 10px;
   }
</style>
```

- Partiendo del siguiente código y sin modificarlo: añadir un evento en JavaScript para que al pinchar cualquier imagen de la galería se agrande a 500px. Probar su funcionamiento para demostrar la delegación de eventos modo “capturing” (por defecto)

```html
<div class="gallery-container">
    <a href="#"><img src="img/foto1.jpg" class="gallery-img"> <p>foto1</p></a>
    <a href="#"><img src="img/foto2.jpg" class="gallery-img"> <p>foto2</p></a>
    <a href="#"><img src="img/foto1.jpg" class="gallery-img"> <p>foto3</p></a>
    <a href="#"><img src="img/foto2.jpg" class="gallery-img"> <p>foto4</p></a>
    <a href="#"><img src="img/foto1.jpg" class="gallery-img"> <p>foto5</p></a>
    <a href="#"><img src="img/foto2.jpg" class="gallery-img"> <p>foto6</p></a>
    <a href="#"><img src="img/foto1.jpg" class="gallery-img"> <p>foto7</p></a>
    <a href="#"><img src="img/foto2.jpg" class="gallery-img"> <p>foto8</p></a>
    <a href="#"><img src="img/foto1.jpg" class="gallery-img"> <p>foto9</p></a>
</div>
```

Modificar fase de registro de evento

- 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);
elemento2.addEventListener('click',hacerAlgo2,false);
```

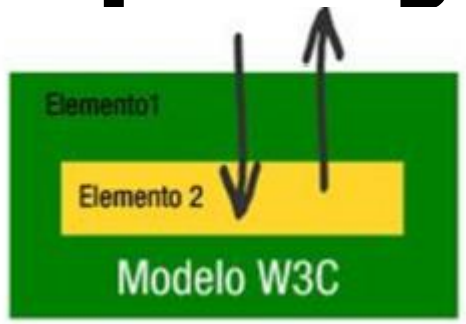
- Para detener la propagación del evento en la fase de burbujeo, disponemos del método `stopPropagation()`. En la fase de captura es imposible detener la propagación.

Más información en:

- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events#event_bubbling_and_capture
- https://javascript.info/bubbling-and-capturing

Ejemplo de Funcionamiento modo capturing

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

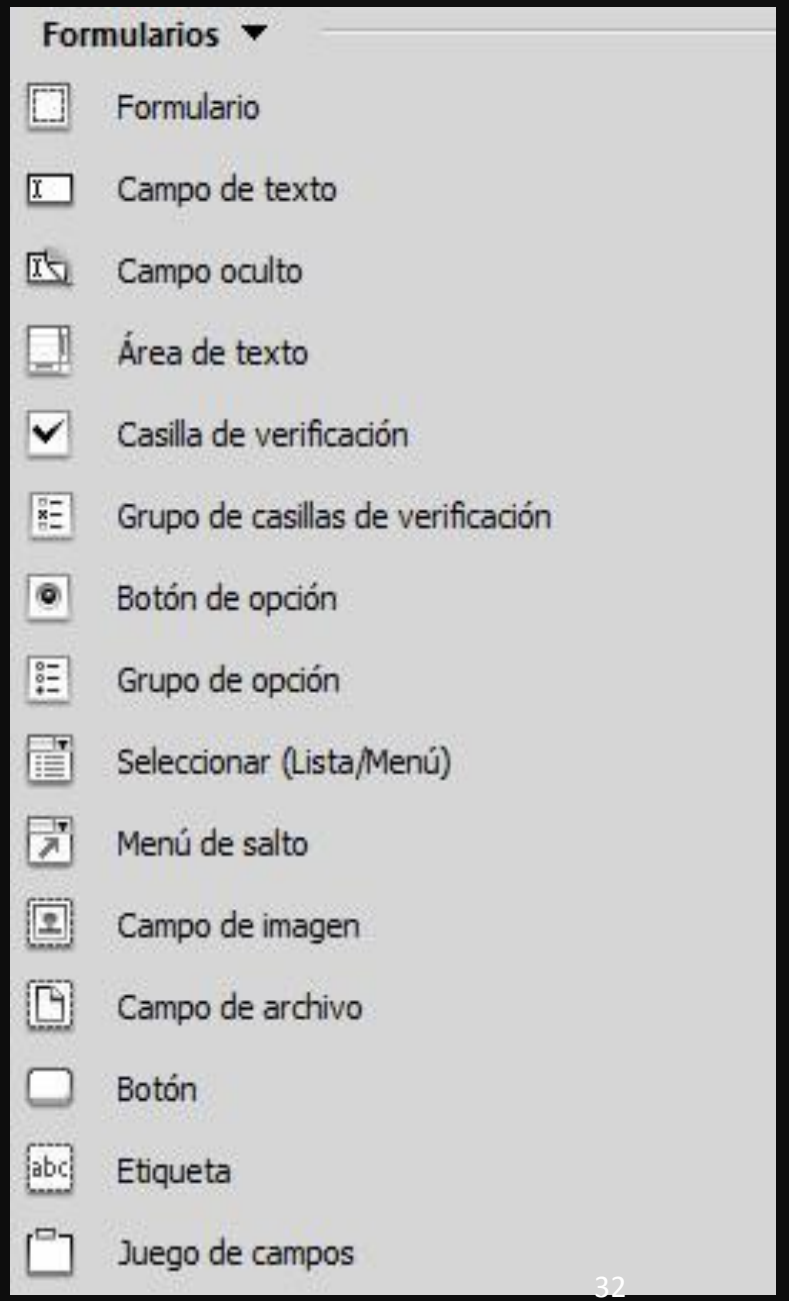


Si el usuario hace clic en el **elemento2** ocurrirá lo siguiente:

1. El evento clic comenzará en la fase de captura.
2. El gestor de eventos comprueba si hay algún ancestro del *elemento2* en el árbol del DOM que tenga programado un evento (true) *onclick* para la fase de captura
3. El gestor de eventos encuentra un *elemento1.hacerAlgo1* que ejecutará primero, pues su tercer argumento tiene un valor true.
4. El evento viajará hacia el destino (*elemento2*), pero no encontrará más eventos para la fase de captura. Entonces el evento pasa a la fase de propagación, y ejecuta *hacerAlgo2*, el cual está programado para esta fase (con un valor false en su tercer argumento).
5. El evento viaja hacia arriba de nuevo y chequea si algún ancestro tiene programado ese evento para la fase de propagación. Éste no será el caso, por lo que no hará nada más.
6. Para detener la propagación del evento en la fase de propagación, disponemos del método *stopPropagation()*. En la fase de captura es imposible detener la propagación.

Recordatorio: formularios

- La mayor parte de interactividad entre una página web y el usuario tiene lugar a través de un **formulario HTML**.
- Los formularios y sus controles, son objetos del **DOM** que tienen propiedades únicas, que otros objetos no poseen.
- El objeto de JavaScript ***form***, es una propiedad del objeto document. Se corresponderá con la etiqueta ***<form>*** del HTML.
- Un formulario podrá ser enviado llamando al método ***submit*** de JavaScript, o bien haciendo clic en el botón ***submit*** del formulario.
- Mediante JavaScript también podemos **crear y editar** formularios así como modificar su comportamiento.
- JavaScript permite examinar y **validar** las entradas de usuario directamente, en el lado del cliente.



RECORDATORIO. La propiedad form.elements

- La propiedad **elements[]** de formulario es una colección, que contiene todos los objetos input dentro de un formulario. Es otro array, con los campos input en el orden del documento.
- Empleando la propiedad **elements[]** podemos hacer un bucle que recorra un formulario. En el ejemplo siguiente se recorren todos y si el elemento es de texto lo pone en blanco.

```
var miFormulario = document.getElementById("contactar");
// guardamos la referencia del formulario en una variable.

if (! miFormulario) return false; // Si no existe ese formulario devuelve false.

for (var i=0; i< miFormulario.elements.length; i++)
{
     if (miFormulario.elements[i].type == "text")
     {
          miFormulario.elements[i].value = "";
     }
}
```

ELEMENTOS DEL DOM

PROPIEDADES Y MÉTODOS DEL OBJETO **ELEMENT** DEL DOM.
https://www.w3schools.com/jsref/dom_obj_all.asp

Recordemos los métodos:

- *createElement("elemento")* → crea un nuevo elemento
- *elemento.appendChild(hijo);* → añade un hijo al elemento
- *element*.setAttribute(*nombre*, *valor*) → modifica o varía un atributo del elemento

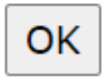
```
// Create element:
const para = document.createElement("p");
para.innerText = "esto es un párrafo.";

// Append to body:
document.body.appendChild(para);
```

```
myInput.setAttribute("type", "button");
```

Before:

OK

After:



Recordemos el ejercicio 3 de la unidad 3

En este ejercicio vamos a generar el formulario de la figura en JavaScript. Requisitos de nuestra página formulario.html:

a. creamos un elemento <section> con id="ContentFormulario"
b. Dentro creamos un botón "Generar Formulario" al pulsarlo, se generará el formulario, también dentro de la sección.
c. La función JavaScript se llamará GeneraFormulario()

Envío y validación de formularios

- **Validación**: comprobar que los datos del formulario son correctos (por ejemplo, email con formato válido).
- Puede hacerse en cliente (más rápida) o en servidor (más segura)
- Categorías típicas de validación:
  1. **Existencia**: comprueba cuando existe o no un valor.
  2. **Numérica**: que la información contenga solamente valores numéricos.
  3. **Patrones**: comprueba que los datos sigan un determinado patrón, como el formato de un e-mail, una fecha, un número de teléfono, un número de la seguridad social, etc. Mediante **EXPRESIONES REGULARES**.

La validación de un formulario en el lado del cliente puede ahorrar algunas idas y vueltas a la hora de enviar los datos, pero aún así, **tendrás que realizar la validación de datos en el servidor**, puesto que es allí realmente donde se van a almacenar esos datos y el origen de los mismos puede venir por cauces que no hemos programado.

Ver ejemplo de validación subido en Moodle

RECORDATORIO. Expresiones regulares

- Las expresiones regulares son **patrones de búsqueda**, que se pueden utilizar **para encontrar texto** que coincida con el patrón especificado.
- Cuando buscamos cadenas que cumplen un patrón en lugar de una cadena exacta, necesitaremos usar expresiones regulares. Podrías intentar hacerlo con funciones de *String*, pero al final, es más sencillo hacerlo con expresiones regulares, aunque su sintaxis es un poco extraña y no muy amigable.

- Las expresiones regulares se gestionan a través del objeto RegExp.
  Sintaxis:

```
var  expresion = /expresión regular/modificador;
```

Ejemplo: expresión regular para encontrar la palabra Aloe Vera

```
var  expresion = /Aloe\s+Vera/;
```

`\s+` = uno o varios espacios en blanco

Importante consultar:
https://www.w3schools.com/jsref/jsref_obj_regexp.asp

Expresiones regulares: modificadores y corchetes

Modifier	Description
g	Perform a global match (find all matches rather than stopping after the first match)
i	Perform case-insensitive matching
m	Perform multiline matching

Corchetes: para encontrar un RANGO de caracteres (recordar: no cadenas)

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any character between the brackets (any digit)
[^0-9]	Find any character NOT between the brackets (any non-digit)
(x|y)	Find any of the alternatives specified

Caracteres especiales más usados para expresiones regulares

(sigue)

Carácter	Coincidencias	Patrón	Ejemplo de cadena
^	Al inicio de una cadena	/^Esto/	Coincidencia en "Esto es...".
$	Al final de la cadena	/final$/	Coincidencia en "Esto es el final".
*	Coincide 0 o más veces	/se*/	Que la "e" aparezca 0 o más veces: "seeee" y también "se".
?	Coincide 0 o 1 vez	/ap?	Que la p aparezca 0 o 1 vez: "apple" y "and".
+	Coincide 1 o más veces	/ap+/	Que la "p" aparezca 1 o más veces: "apple" pero no "and".
{n}	Coincide exactamente n veces	/ap{2}/	Que la "p" aparezca exactamente 2 veces: "apple" pero no "apabullante".
{n,}	Coincide n o más veces	/ap{2,}/	Que la "p" aparezca 2 o más veces: "apple" y "appple" pero no en "apabullante".
{n,m}	Coincide al menos n, y máximo m veces	/ap{2,4}/	Que la "p" aparezca al menos 2 veces y como máximo 4 veces: "appppppple" (encontrará 4 "p").
.	Cualquier carácter excepto nueva línea	/a.e/	Que aparezca cualquier carácter, excepto nueva línea entre la a y la e: "ape" y "axe".
[...]	Cualquier carácter entre corchetes	/a[px]e/	Que aparezca alguno de los caracteres "p" o "x" entre la a y la e: "ape", "axe", pero no "ale".
[^...]	Cualquier carácter excepto los que están entre corchetes	/a[^px]/	Que aparezca cualquier carácter excepto la "p" o la "x" después de la letra a: "ale", pero no "axe" o "ape".
\b	Coincide con el inicio de una palabra	/\bno/	Que "no" esté al comienzo de una palabra: "novedad".

Caracteres especiales más usados para expresiones regulares
(cont)

Carácter	Coincidencias	Patrón	Ejemplo de cadena
\B	Coincide al final de una palabra	/\Bno/	Que "no" esté al final de una palabra: "este invierno" ("no" de "invierno").
\d	Dígitos del 0 al 9	/\d{3}/	Que aparezcan exactamente 3 dígitos: "Ahora en 456".
\D	Cualquier carácter que no sea un dígito	/\D{2,4}/	Que aparezcan mínimo 2 y máximo 4 caracteres que no sean dígitos: encontrará la cadena "Ahor" en "Ahora en 456".
\w	Coincide con caracteres del tipo (letras, dígitos, subrayados)	/\w/	Que aparezca un carácter (letra, dígito o subrayado): "J" en "JavaScript".
\W	Coincide con caracteres que no sean (letras, dígitos, subrayados)	/\W/	Que aparezca un carácter (que no sea letra, dígito o subrayado): "%" en "100%".
\n	Coincide con una nueva línea		
\s	Coincide con un espacio en blanco		
\S	Coincide con un carácter que no es un espacio en blanco		
\t	Un tabulador		
(x)	Capturando paréntesis		Recuerda los caracteres.
\r	Un retorno de carro		
?=n	Cualquier cadena que está seguida por la cadena n indicada después del igual.	/la(?= mundo)	Hola mundo mundial.

El objeto RegExp

```
let re = /ab+c/;
```

```
let re = new RegExp('ab+c');
```

- Es tanto un literal como un objeto de JavaScript, por lo que también se podrá crear usando un constructor:

```
var expresionregular = new RegExp("Texto Expresión Regular");
```

- Usar el literal cuando sabemos que la expresión no cambiará. Una versión compilada es mucho más eficiente.
- Usaremos el objeto cuando sabemos que la expresión regular va a cambiar o cuando vamos a proporcionarla en tiempo de ejecución.

Propiedad	Descripción
global	Especifica que sea utilizado el modificador "g".
ignoreCase	Especifica que sea utilizado el modificador "i".
lastIndex	El índice donde comenzar la siguiente búsqueda.
multiline	Especifica si el modificador "m" es utilizado.
source	El texto de la expresión regular RegExp.

Método	Descripción
compile()	Compila una expresión regular.
exec()	Busca la coincidencia en una cadena. Devolverá la primera coincidencia.
test()	Busca la coincidencia en una cadena. Devolverá true o false.

Tutorial Regular Expressions in JavaScript: https://www.youtube.com/watch?v=909NfO1St0A

End of unit 5