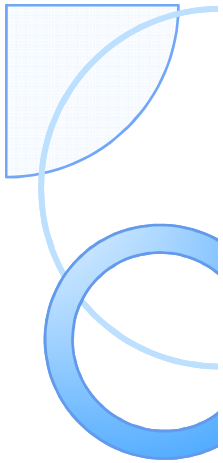
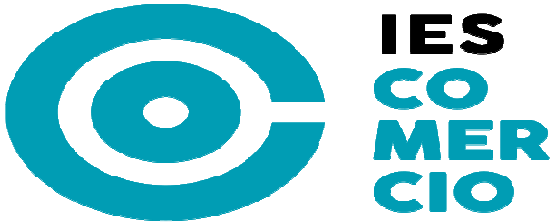




Tema 6:
Gestión de eventos y formularios en JavaScript.

Desarrollo Web en Entorno Cliente

(Curso 2016 - 2017)

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.

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:

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

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

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 izquierdo del ratón. El manejador de este evento es **`onclick`**.
  - Dblclick: Este evento se acciona cuando hacemos un doble click sobre el botón izquierdo 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:
  - 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 `dobleclick` sería la siguiente: `mousedown, mouseup, click, mousedown, mouseup, click, doblclick`.

Actividad 1

Crea un elemento 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.

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 `<img>` 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 `<img>` 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

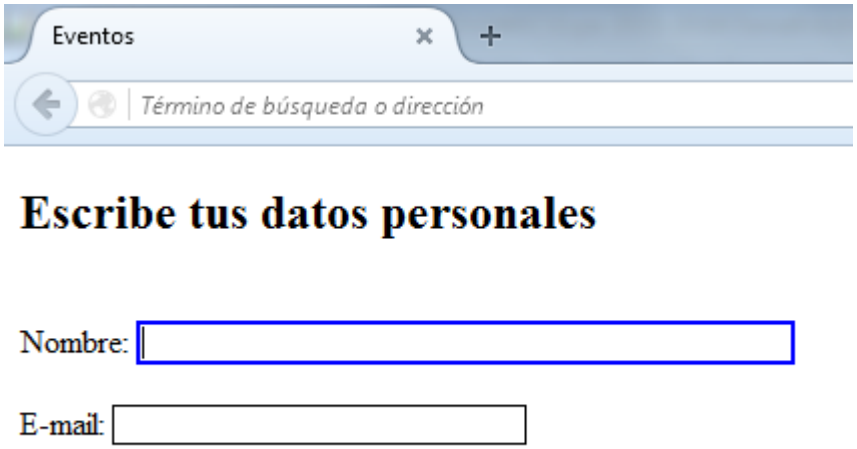
- Eventos HTML (3):
  - Reset: Este evento se produce cuando pulsamos sobre un botón de tipo reset. El manejador es **`onreset`**.
  - Resize: Este evento se produce cuando redimensionamos el navegador, actúa sobre el objeto `Window`. El manejador es **`onresize`**.
  - Scroll: Se produce cuando varía 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.

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.



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);
```

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 burbujean, 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);
```

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.