JavaScript

ANEXO 5.-

JQuery:

Manejo de eventos

DISTRIBUIDO POR:

CENTRO DE INFORMÁTICA PROFESIONAL S.L.

C/ URGELL, 100
08011 BARCELONA
TFNO: 93 426 50 87

C/ RAFAELA YBARRA, 10
48014 BILBAO
TFNO: 94 448 31 33

www.cipsa.net

Manejo de eventos

Eventos

Javascript permite crear páginas web interactivas que reaccionan a las acciones del usuario sobre sus elementos. Esto significa que en la página se ejecutan cambios y operaciones en respuesta a una acción del usuario.

Un ejemplo básico es una página provista de un formulario en los que el usuario puede introducir una serie de valores como su nombre, edad y dirección de correo electrónico. Cuando el usuario pulsa el botón de envío se ejecuta una función de Javascript que valida los datos introducidos. En consecuencia, si son correctos se envían, de lo contrario se mostraría un mensaje de error al usuario. En este caso, la pulsación del botón constituye el *evento* que desencadena la validación de los datos.

- Un *evento* representa una acción del usuario sobre un elemento de la página. Por ejemplo; el evento "*click*" representa la pulsación del botón izquierdo del ratón sobre un elemento. A cada evento de un elemento puede asignarse un *manejador*.
- Un *manejador* es una función de Javascript que se ejecuta cuando se da el evento al que se asignó y que implementa las operación en respuesta, por ejemplo; validar un formulario.

Registro de eventos

Javascript define el conjunto de eventos que pueden detectarse sobre los diferentes tipos de elementos de una página web:

Evento	Descripción	Elementos para los que está definido
blur	Deseleccionar el elemento	<button>, <input>, <label>, <select> ,<textarea>, <body>
change	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
click	Pinchar y soltar el ratón	Todos los elementos
dblclick	Pinchar dos veces seguidas con el ratón	Todos los elementos
focus	Seleccionar un elemento	<button>, <input>, <label>, <select> ,<textarea>, <body>
keydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>
keypress	Pulsar una tecla	Elementos de formulario y <body>

Evento	Descripción	Elementos para los que está definido
keyup	Soltar una tecla pulsada	Elementos de formulario y `<body>`
load	La página se ha cargado completamente	`<body>`
mousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
mousemove	Mover el ratón	Todos los elementos
mouseout	El ratón *"sale"* del elemento (pasa por encima de otro elemento)	Todos los elementos
mouseover	El ratón *"entra"* en el elemento (pasa por encima del elemento)	Todos los elementos
mouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
reset	Inicializar el formulario (borrar todos sus datos)	`<form>`
resize	Se ha modificado el tamaño de la ventana del navegador	`<body>`
select	Seleccionar un texto	`<input>`, `<textarea>`
submit	Enviar el formulario	`<form>`
unload	Se abandona la página (por ejemplo al cerrar el navegador)	`<body>`

Para cada uno de éstos eventos, JQuery define una función que permite asignar una *función manejadora* para una selección de elementos. Algunas de estas funciones son:

La función ***$().click()*** que permite asignar a los elementos de una selección una *función manejadora* equivalente:

```
// Asignacion de función manejadora al evento click del botón id="boton"
$("#boton").click(function (event) {
    alert("PULSADO.");
});
```

Acceso al elemento origen del evento desde la función manejadora. (*this*)

Las funciones de registro de eventos se ejecutan para cada uno de los elementos de la selección contra la que se invocan. Esto es útil cuando se tienen varios elementos que se desea que realicen la misma acción ante un evento.

El siguiente código asigna la misma función manejadora a todos los elementos *<input type="button">* presentes en la página. Si fuera necesario obtener el valor o modificar el elemento botón que ha sido pulsado, puede emplearse ***this*** para acceder al elemento que ha originado el evento.

```
$("input:button").click(function () {
    // Código a ejecutar para cada uno de los botones al ser pulsado
});
```

Ejemplo: La siguiente página muestra tres botones cada uno de los cuales al ser pulsado se tiñe de amarillo mostrando un mensaje con su nombre:

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title></title>
    <script src="Scripts/jquery-3.0.0.js"></script>
    <meta charset="utf-8" />
    <script>
        $().ready(function () {
            // Para todos los botones cuando sean pulsados
            $("input:button").click(function () {
                // Cambia el color del botón pulsado
                $(this).css("background-color", "#FFFF00");
                // Muestra el título del botón pulsado
                alert($(this).val());
            });
        });
    </script>
</head>
<body>
    <div id="panel"></div>
    <input type="button" value="BOTON 1" /><br/>
    <input type="button" value="BOTON 2" /><br />
    <input type="button" value="BOTON 3" /><br />
</body>
</html>
```

La función ***$().on()*** permite asignar una función manejadora a múltiples eventos de uno o varios elementos al mismo tiempo:

Ejemplo: Registro de un evento con su función manejadora

```
// Registro del evento click sobre todos los párrafos de la página.
$("p").on("click", function () {
    …
});
```

Ejemplo: Registro de varios eventos a misma función manejadora

```
// Registro del evento mouseenter y mouseleave para todas las capas.
$("div").on("mouseenter mouseleave", function () {
    …
});
```

Ejemplo: Registro de múltiples eventos con funciones manejadoras propias.

```
            $("div").on({
                mouseenter: function () {
                    alert("El puntero del ratón entra en la capa");
                },
                mouseleave: function () {
                    alert("El puntero del ratón sale de la capa");
                },
                click: function () {
                    alert("El puntero del ratón sale de la capa");
                }
            });
```

Registro de eventos de una sola acción (*$().one()*)

Existen ocasiones en las que se quiere que un evento se ejecute una única vez. Para ello debe emplearse la función ***$().one()*** para registrarlo indicando como argumentos el evento y la función de retrollamada con la acción a ejecutar:

Ejemplo: La siguiente página muestra una capa y un botón que al ser pulsado la primera vez modifica el color de la capa y lanza un mensaje. Si se pulsa más veces el botón no surte ningún efecto.

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title></title>
    <script src="Scripts/jquery-3.0.0.js"></script>
    <meta charset="utf-8" />
    <style>
        .capa {
            width:400px;
            height:400px;
            background-color: #000000;
        }
    </style>
    <script>
        $().ready(function () {
            // Registro del evento click del boton id="elemento"
            // para una sola ejecución.
            $("#boton").one("click", function () {
                $(".capa").css("background-color", "#ff0000");
            });
        });
    </script>
</head>
<body>
    <div class="capa"></div><br />
    <input type="button" id="boton" value="CAMBIAR COLOR" />
</body>
</html>
```

En el caso de que se desee que un evento ejecute una funcion manejadora determinada la primera vez y otra distinta el resto de las veces, puede hacerse una reasignación:

Ejemplo: El siguiente código asigna una función manejadora al evento click para que se ejecute una única vez. No obstante, en la misma función manejadora puede asignarse otra para que se ejecute el resto de las veces.

```
        $('p').one('click', function () {
            console.log('click primera vez');
             $(this).click(function () {
                 console.log('click resto veces');
              });
        });
```

Supresión de eventos

La función ***$().off()*** permite eliminar un evento de los elementos de la selección contra la que se invoca. Requiere como único argumento el nombre del evento a eliminar.

```
    // Desregistra el evento de click de todos los párrafos de la página
    $("p").off("click");
```

Disparo de eventos

Existen ocasiones en las que puede ser necesario provocar desde código la ejecución de la tarea asociado a un evento sin que éste se suceda. Para estas situaciones JQuery dispone de la función ***$().trigger()***.

Esta función requiere como argumento el nombre del evento que se quiere provocar:

```
  // Registro del evento "click" sobre un elemento.
  $("#foo").on("click", function () {
        alert($(this).text());
  });

   // Disparo del evento click del elemento desde código
   $("#foo").trigger("click")
```

No obstante, esta práctica se considera desaconsejada siendo preferible declarar una función de manera explicita y registrarla como función manejadora del evento de modo que pueda ser igualmente llamada al margen del evento:

```
var foo = function (e) {
     if (e) {
          console.log(e);
     } else {
          console.log('esta ejecucción no provino desde un evento');
     }
};
$('p').click(foo);
// Ejecución de la función manejadora en vez de ejecutar $('p').trigger('click')
foo(); // en lugar de realizar $('p').trigger('click')
```

Registro de eventos con datos vinculados

La función $().on() también permite pasar datos vinculados a la funcion manejadora. Esto resulta especialmente útil cuando se emplea una misma funcion manejadora para múltiples eventos.

Para ello se emplea la siguiente sintaxis:

$().on(<evento>, <objeto datos vinculados>, <fn_manejadora>)

Ejemplo: Supóngase que tenemos una determinada función (handler) que queremos asignar como función manejadora del evento de click de varios botones incluyendo objeto de datos vinculado con el atributo "*color*" y un valor según el botón. El código sería el siguiente:

```
$("#btn_rojo").on("click", { color: 0 }, handler);
$("#btn_azul").on("click", { color: 1 }, handler);
$("#btn_verde").on("click", { color: 2 }, handler);
```

En la función manejadora debe emplearse la propiedad *data* del objeto origen del evento para acceder a los datos vinculados:

```
function handler(ev) {
    // Obtencion del dato vinculado color
    var valor_color = ev.data.color;
}
```

Ejemplo: Supóngase el siguiente código en el que se muestra una capa y tres botones para los que se designa una misma funcion manejadora (*handler*).

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <style>
        #capa { width: 500px; height: 200px }
    </style>
    <meta charset="utf-8" />
    <script src="Scripts/jquery-3.1.0.js"></script>
    <script>
        // Funcion manejadora
        function handler(ev) {
            colores = ["#ff0000", "#0000ff", "#00ff00"]
            $("#capa").css('background-color', colores[ev.data.color]);
        }
        $(function () {
            $("#btn_rojo").on("click", { color: 0 }, handler);
            $("#btn_azul").on("click", { color: 1 }, handler);
            $("#btn_verde").on("click", { color: 2 }, handler);
        });
    </script>
</head>
<body>
    <div id="capa">

    </div>
    <button id="btn_rojo">ROJO</button>
    <button id="btn_verde">VERDE</button>
    <button id="btn_azul">AZUL</button>
</body>
</html>
```

Delegación de eventos

La delegación de eventos es un mecanismo que permite asignar funciones manejadoras a elementos antes de que siquiera existan. Este es el caso de elementos que no forman parte del código HTML de la página, sino que son añadidos mediante JQuery, como por ejemplo; los elementos de una lista.

Supóngase una página que muestra una lista *<ul>* y dos elemento *<li>* presentes en el código HTML, más un botón que se desea que al ser pulsado añada un nuevo elemento *<li>* a la lista mediante JQuery:

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title></title>
    <script src="Scripts/jquery-3.0.0.js"></script>
    <script>
        $().ready(function () {
            // Evento de pulsación sobre cada elemento de la lista
            $("#lista li").on("click", function () {
                // Muestra mensaje con el contenido del elemento <li> pulsado
                alert("HAS PULSADO: " + $(this).text());
            })
            // Eventio de pulsación sobre el botón de añadir.
            $("input:button").click(function () {
                var cantidad = $("#lista li").length + 1;
                // Añade un nuevo elemento <li> a la lista.
                $("#lista").append("<li>opcion "+cantidad+"</li>");
            });
        });
    </script>
</head>
<body>
    <ul id="lista">
        <li>opcion 1</li>
        <li>opcion 2</li>
    </ul>
    <input type="button" value="añadir" />
</body>
</html>
```

Para ello se asigna al evento click del botón una función manejadora que obtiene el número de elementos *<li>* presentes en la lista y añade uno nuevo. También se asigna al evento click de todos los elementos *<li>* de la lista una función manejadora que muestra el mensaje "*HAS PULSADO*" seguido del contenido del elemento seleccionado.

Si ejecutamos el código veremos que al hacer click sobre los elementos *<li>* "*opcion1*" y "*opcion2*" que forman parte del HTML, el evento click se ejecuta correctamente. Sin embargo, no funciona para los elementos <li> creados al pulsart el botón "*añadir*".

Esto sucede por que *los eventos asignados directamente a los elementos de una selección, sólo se asignan a los elementos existentes en ese instante*. Los elementos añadidos posteriormente no son registrados.

Para poder asignar eventos a elementos que no existentes en el momento de la asignación debe emplearse la *delegación de eventos*. Esto consiste en asignar el evento a un elemento padre que los contiene en vez de a ellos directamente.

En este caso, podemos asignar el evento *click* a los elementos <*li*> hijos del elemento <*ul*> que representa la lista empleando la función ***$().on()*** del siguiente modo:

$(selección elemento padre).on("evento", "elementos hijos", función manejadora)

```
    <script>
        $().ready(function () {
            $("#lista").on("click", "li", function () {
                alert("HAS PULSADO: " + $(this).text());
            })
            $("input:button").click(function () {
                var cantidad = $("#lista li").length + 1;
                var $lista = $("#lista");
                $lista.append("<li>opcion "+cantidad+"</li>");
            });
        });
    </script>
```

De este modo, el evento *click* se asigna a todos los elementos <*li*> contenidos en el elemento <*ul*> actuales y futuros. Esto es posible gracias a la *propagación de los eventos.*

La propagación de los eventos

La propagación de eventos significa que los eventos de un elemento se propagan por defecto hacia el elemento padre que lo contiene. Por ejemplo; sea el siguiente HTML:

```
<!DOCTYPE html>
<html>
<head>
    …
</head>
<body>
    <div>
        <ul id="lista">
            <li>opcion 1</li>
            <li>opcion 2</li>
        </ul>
    </div>
</body>
</html>
```

Al pulsar con el raton sobre el elemento <*li*>, se produce una cascada de eventos *click* en el siguiente orden:

*** → → → <body> → <html> → raiz del documento web***

Esto significa que los eventos de los elementos <*li*> de la lista <ul> anterior, "pasan" a través del elemento <*ul*> permitiendo su captura mediante delegación con $().on().

(*) Para comprobar la propagación de eventos basta con agregar una función manejadora al mismo evento *click* de los diferentes elementos y ver el orden en que se ejecutan.

Cancelación de la propagación

La propagación de los eventos sucede siempre de manera predeterminada. Sin embargo puede interrumpirse si una función manejadora asignada al evento que se propaga invoca el método ***stopPropagation()*** del objeto *origen del evento*. En tal caso, la propagación del evento se detiene no saltando al elemento padre siguiente.

Ejemplo: El siguiente código captura el evento de envío (*submit*) de un formulario, tal que si no se verifican las validaciones se cancela tanto el envío como la propagación del evento a otros elementos padre que pudieran recoger el evento.

```
        // Captura del evento de envío de un formulario
        $("form").on("submit", function (e) {
            // Comprobación de validación
            if (!validacion()) {
                // Cancela la accion predeterminada del evento → envío del formulario.
                e.preventDefault();
                // Cancela la propagación → Ningún elemento padre recibe evento submit
                e.stopPropagation();
            }
        });
```

Registro de eventos en elementos creados dinámicamente

Es posible registrar eventos en elementos creados dinámicamente. Par ello sólo es necesario seleccionarlos tras su creación y registrar el evento:

Ejemplo: El siguiente código muestra una capa a la que se le añade una cabecera <h2> registrando a continuación el evento “click” para que se muestre un mansaje al ser pulsada:

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
        <meta charset="utf-8" />
    <script src="scripts/jquery-3.1.0.js"></script>
    <script>
         $(function () {
             // Creacion de la cabecera e inclusión en la capa.
             var elemento = $("#contenido").append("<h2>texto</h2>");
             // Registro del evento "click" a la cabecera.
             elemento.find("h2").on("click", function () { alert("PULSADO"); });
         });
    </script>
</head>
<body>
    <div id="contenido" style="background-color: aqua">
    </div>
</body>
</html>
```

El objeto origen del evento (*Event Object*)

A veces las funciones registradas a eventos necesitan información sobre el evento para poder desempeñar su tarea. Estas informaciones pueden ser tales como conocer la tecla pulsada cuando se produce un evento *keypress*, o la posición del puntero del ratón en la pantalla en el momento de producirse un evento *click*. La forma de obtener estos datos es mediante el objeto origen del evento (*Event Object*).

En JQuery las funciones manejadoras de cualquier evento pueden incluir como parámetro un *objeto origen del evento*. Este objeto dispone de las propiedades necesarias para obtener información sobre el evento según su tipo:

- *pageX*, *pageY* → La posición del puntero del ratón en el momento que el evento ocurrió, relativo a las zonas superiores e izquierda de la página.
- *type* → El tipo de evento (por ejemplo “click”).
- *which* →El botón o tecla presionada.
- *data* → Alguna información pasada cuando el evento es ejecutado.
- *target* → El elemento DOM que inicializó el evento.
- *preventDefault()* → Método que cancela la acción predeterminada del evento (por ejemplo: seguir un enlace).
- *stopPropagation()* → Método que detiene la propagación del evento sobre otros elementos.

Puede consultarse la lista completa de los eventos capturables por los elementos HTML, así como las propiedades del objeto origen del evento (*Event Object*) para cada uno de los diferentes tipos de eventos en la siguiente página de referencia:
http://www.w3schools.com/jsref/dom_obj_event.asp

Información sobre el evento (*type*)

El objeto ***event*** dispone de una propiedad ***type*** devuelve una cadena con el identificador del evento sucedido. Esta propiedad puede utilizarse para manejar distintos eventos mediante una misma función manejadora.

Ejemplo: El siguiente código muestra una página con un elemento <img> cuya imagen mostrada cambia cuando el puntero del ratón entra y sale de su área.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <script src="Scripts/jquery-3.1.0.js"></script>
     <script>
          function accion(ev) {
              if (ev.type == "mouseover") {
                  $(this).attr("src", "images/green_light.png");
              }
              if (ev.type == "mouseout") {
                  $(this).attr("src", "images/red_light.png");
              }
          }

          $(function () {
              $("#imagen").on("mouseover mouseout", accion);
          });
     </script>
</head>
<body>
    <img id="imagen" src="images/red_light.png" />
</body>
</html>
```

La función *accion()* obtiene el objeto imagen y el objeto ***event***. En función del tipo de evento detectado (***mouseover***, ***mouseout***), la función asigna uno u otro fichero al objeto imagen de la página.

La función *$().hover()*

Esta funcion permite asignar dos *funciones manejadoras* para los eventos de entrada del puntero del ratón (*mouseenter*) y salida (*mouseleave*):

```
$().hover( <fn_manejadora_mouseenter>, <fn_manejadora_mouseleave> )
```

Con ella, el código anterior podría simplificarse del siguiente modo:

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <script src="Scripts/jquery-3.1.0.js"></script>
    <script>
          $(function () {
              $("#imagen").hover(function () {
                  $(this).attr("src", "images/green_light.png");
              }, function () {
                  $(this).attr("src", "images/red_light.png");
              });
          });
    </script>
</head>
<body>
    <img id="imagen" src="images/red_light.png" />
</body>
</html>
```

Información sobre tecla pulsada (*which*)

Ejemplo: El siguiente ejemplo muestra el código necesario para que cuando al pulsar la tecla "," en una caja de texto se añada en su lugar un ".":

```
    <script>
        $().ready(function () {
            // Cuando se envia el formulario
            $("#valor").keydown(function (e) {
                if (e.which == 188) {
                    e.preventDefault();
                    $(this).val(function (i, v) { return v + "." });
                }
            });
        });
    </script>
```

En este caso se captura el evento *keydown* que se produce cuando el elemento tiene el foco y el usuario pulsa una tecla. La propiedad *which* del *objeto origen del evento* (***e***) devuelve un valor numérico que identifica la tecla pulsada antes de añadirla a la caja de texto. Si el valor es 188 (ASCII correspondiente a la coma), se cancela el evento con *preventDefault()* para evitar que se añada la coma, y se añade un punto con ***$().val()***.

Información sobre posición del puntero de ratón (*pageX, pageY*)

Ejemplo: La siguiente página muestra una capa en color negro sobre la que al moverse el puntero del ratón se muestran sus coordenadas en una caja de texto. Para ello se captura el evento ***mousemove*** sobre la capa para obtener las coordenadas del ratón con cada movimiento:

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title></title>
    <style>
        .capa {width:500px; height:500px; background-color: #000000; }
    </style>
    <script src="Scripts/jquery-3.0.0.js"></script>
     <script>
         $().ready(function () {
             // Cuando se mueve el ratón sobre la capa
             $(".capa").mousemove(function (e) {
             // Se muestra sus coordenadas en la caja de texto
                 $("#posicion").val(e.pageX + "," + e.pageY);
             });
         });
     </script>
</head>
<body>
    <div class="capa"></div>
    <input type="text" id="posicion" />
</body>
</html>
```

En este caso se han empleado las propiedades *pageX* y *pageY* para obtener las coordenadas del ratón en relación a la página en el momento en que se produce el evento ***mousemove***. Este se evento se produce al moverse el ratón sobre la capa.

Información de eventos de teclado

Existen tres tipos de eventos de teclado básicos: **keyup**, **keydown**, **keypress**. En cualquiera de los tres casos, el objeto *event* permite obtener la tecla pulsada mediante las propiedad ***keyCode*** y ***charCode***. No obstante; existen ciertas diferencias en función del navegador empleado:

- Evento **keydown**:
  - Mismo comportamiento en todos los navegadores:
    - Propiedad keyCode: código interno de la tecla
    - Propiedad charCode: no definido

- Evento **keypress**:
  - Internet Explorer:
    - Propiedad keyCode: el código del carácter de la tecla que se ha pulsado
    - Propiedad charCode: no definido
  - Resto de navegadores:
    - Propiedad keyCode: para las teclas normales, no definido. Para las teclas especiales, el código interno de la tecla.
    - Propiedad charCode: para las teclas normales, el código del carácter de la tecla que se ha pulsado. Para las teclas especiales, 0.

- Evento **keyup**:
  - Mismo comportamiento en todos los navegadores:
    - Propiedad keyCode: código interno de la tecla
    - Propiedad charCode: no definido

Información de eventos de ratón

La información más relevante dada por los eventos relacionados con el ratón: *click*, *mouseover*, *mouseout*, *mouseup*, *mouseout*, *mousemove*, es la relativa a la posición del puntero del ratón en el momento de darse el evento. Estas coordenadas pueden obtenerse en base a tres puntos de referencia distintos.

- Con respecto a la pantalla → La coordenada (0,0) se corresponde con el margen superior izquierdo de la pantalla del dispositivo u ordenador del usuario:
  - Propiedades: **screenX**, **screenY**

- Con respecto a la ventana del navegador → La coordenada (0,0) se corresponde con el margen superior izquierdo de la ventana del navegador del usuario.
  - Propiedades: **clientX**, **clientY**

- Con respecto al origen de la página → La coordenada (0,0) se corresponde con el margen superior izquierdo de la página. Estas coordenadas son distintas a las indicadas por las propiedades *clientX* y *clientY* cuando el usuario se desplaza por la página empleando las barras de desplazamiento.
  - Propiedades: **pageX** y **pageY**

Cancelación de eventos

En algunas ocasiones resulta necesario cancelar un evento para evitar que realice su acción predeterminada en un elemento. Para cancelar la acción predeterminada de un evento debe llamarse al método ***preventDefault()*** del objeto *event*.

Ejemplo 1: La siguiente página muestra una caja de texto en la que no se permite la introducción de más de 10 caracteres.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <script src="Scripts/jquery-3.1.0.js"></script>
    <script>
        $(function () {
            $("#texto").on("keypress", function (ev) {
                if ($(this).val().length >= 10) {
                    ev.preventDefault();
                }
            });
        });
    </script>
</head>
<body>
    <input type="text" id="texto">
</body>
</html>
```

El código registra el evento "***keypress***" de la caja de texto. Cada vez que se pulsa una tecla teniendo la caja de texto seleccionada la *función manejadora* comprueba la longitud del valor de la caja de texto y si es igual o mayor de 10 caracteres cancela el evento llamando al método ***preventDefault()***. De este modo se evita que se añada el carácter excediendo la longitud deseada.

Ejemplo 2: Supóngase una página provista de un formulario con tres cajas de texto valor1, valor2 y valor3, en las que el usuario debe introducir tres valores numéricos. Cuando el usuario pulsa el botón "ENVIAR" debe comprobarse los valores introducidos de modo que si algunas de las cajas de texto está vacía o no contiene un valor numérico válido no debe mostrarse un mensaje de error.

En el código mostrado se registra el evento "*submit*" del formulario a la función *validar().* Esta función comprueba si cada una de las tres cajas de texto tiene un valor y es un número. Si alguna de las comprobaciones falla se asigna un color rojo al fondo de la caja de texto y se cancela el emvio mediante el método ***preventDefault()***.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
        <meta charset="utf-8" />
    <script src="scripts/jquery-3.1.0.js"></script>
    <script>
        function validar() {
            var ok = true;
            // Obtencion de todos los campos input con name='valores'
            $("input[name='valores']").each(function () {
                // Por cada elemento -> Obtengo valor inscrito.
                var valor = $(this).val();
                console.log(valor);
                // Comprobacion de cadena vacía y valor numérico.
                if (valor.trim().length == 0 || isNaN(valor)) {
                    // ERROR -> Ausencia de valor o no numérico.
                    console.log("FALLO");
                    $(this).css("background-color", "#ff0000");
                    ok = false;
                } else {
                    // OK -> Valor válido.
                    console.log("OK");
                    $(this).css("background-color", "#00ffff");
                }
            });
            return ok;
        }
        $(function () {
            // Registro de evento de submit del formulario
            $("#formulario").submit(function (ev) {
                // Llamada a funcion validadora
                if (!validar()) {
                    // Cancelación del evento de submit.
                    ev.preventDefault();
                }
            });
        })
    </script>
</head>
<<body>
    <form id="formulario" action="servidor.php" method="get">
        valor1: <input id="va" type="text" name="valores"><br>
        valor2: <input id="vb" type="text" name="valores"><br>
        valor3: <input id="vc" type="text" name="valores"><br>
        <input id="envio" type="submit" value="enviar">
    </form>
</body>
</html>
```

Ejercicios

1. Crea una página dotada de una serie de botones con los nombres “Mercurio”, “Venus”, “Tierra”,”Marte”, y una imagen. Cada vez que el usuario pulse uno de los botones deberá mostrarse la imagen correspondiente al planeta indicado. Puedes descargar las imágenes de Google.

2. Crea una página provista de una imagen y cuatro botones: “*Arriba*”, “*Abajo*”, “*Izquierda*”, “*Derecha*”. Cada botón al ser pulsado debe desplazar la imagen 1 pixel en la dirección correspondiente. Para ello debes crear cuatro funciones de javascript: *MoverArriba*(), *MoverAbajo*(), *MoverIzquierda*(), y *MoverDerecha*() cada una de las cuales modificar la posición de la imagen 1 píxel en la dirección correspondiente.

3. Modifica el ejercicio anterior para emplear únicamente una función de Javascript llamada Mover. Esta función deberá recibir como único parámetro un valor que indique la dirección del movimiento de la imagen. La función debe evitar que la imagen salga de los límites visibles de la pantalla.

Para conocer los límites de la pantalla puede emplearse las propiedades *width* y *height* del objeto ***screen***.

4. Tomando como punto de partida la página ‘index.html’, implementar las siguientes operaciones:

- Establece el valor de la caja de texto igual al del elemento <*label*>
- Añadir la clase de estilo “*hint”* a la caja de texto. (La clase se encuentra ya declarada en el archivo “*css/style.css*”).
- Elimina el elemento <*label*>.
- Registra el evento *“focus”* a la caja de texto para eliminar el texto de sugerencia y la clase de estilo “*hint*” cuando ésta tenga el foco.
- Registra el evento *“blur”* a la caja de texto para restaurar el texto de sugerencia y la clase de estilo “*hint*” si no se ha insertado ningún valor al perder el foco.

5. Tomando como punto de partida la página 'index.html', implementar las siguientes operaciones para simular un sistema de navegación por pestañas:

- Oculta todas las capas con la clase de estilo *"module"*.
- Crea una lista desordenada antes de la primera capa con estilo "module". Esta servirá de marco para las pestañas.
- Recorre las capas con estilo "module" empleando $().each() y crea un elemento para la lista desordenada por cada capa con el texto presente en su encabezado <h2>. Estas son las pestañas.
- Registr el evento "click" a cada elemento de la lista de modo que se realicen las siguientes operaciones:
  - Oculte todos las capas con estilo "module" mostrando únicamente la correspondiente al elemento.
  - Elimina la clase "current" de todos los elementos de la lista añadiéndolo al elemento seleccionado.
- Mostrar la primera pestaña (simulando la selección de la primera pestaña)