DESARROLLO WEB EN ENTORNO CLIENTE

CAPÍTULO 5:

Interacción con el usuario, eventos y formularios

Modelos de gestión de eventos

- Los eventos son unos 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.

Modelos 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');">
```

Modelos de gestión de eventos

- El ejemplo anterior se puede realizar de otro modo llamando a una función. Opción más recomendable si la acción es compleja.

```
<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>
```

Modelos de gestión de eventos

Obteniendo información del evento (objeto event)

- JavaScript permite obtener información sobre el ratón y el teclado mediante un objeto especial llamado event.
- Desafortunadamente, los diferentes navegadores presentan diferencias muy notables en el tratamiento de la información sobre los eventos.
- Internet Explorer considera que este objeto forma parte del objeto window.
- El resto de navegadores lo consideran como el único argumento que tienen las funciones manejadoras de eventos.

Modelos de gestión de eventos

Obteniendo información del evento (objeto event)

- En los navegadores tipo Internet Explorer, el objeto event se obtiene directamente mediante: var evento = window.event;
- Por otra parte, en el resto de navegadores, el objeto event se obtiene *mágicamente* a partir del argumento que el navegador crea automáticamente:

```
function manejadorEventos(elEvento) {
        var evento = elEvento;
}
```

- El siguiente código muestra la forma correcta de obtener el objeto event en cualquier navegador:

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

Modelos de gestión de eventos

```
  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...
```

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.

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`.
  - **Mouseup**. Este evento se produce cuando soltamos un botón del ratón que previamente teníamos pulsado. El manejador de este evento es `onmouseup`.

Modelos de gestión de eventos

- **Eventos del ratón (2):**
  - **Mouseover**. Este evento se produce cuando el puntero del ratón se encuentra fuera de un elemento, y este se desplaza hacia el interior del elemento o alguno de sus hijos. El manejador de este evento es `onmouseover`.
  - **Mouseout**. Este evento se produce cuando el puntero del ratón está dentro de un elemento y este puntero es desplazado fuera del elemento o alguno de sus hijos. El manejador de este evento es `onmouseout`.
  - **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`.

Modelos de gestión de eventos

- **Eventos del ratón (2):**
  - **Mouseenter**. Este evento se produce cuando el puntero del ratón entra dentro de un elemento. El manejador de este evento es `onmouseenter`.
  - **Mouseleave**. Este evento al revés que el anterior se produce cuando el puntero del ratón sale fuera de un elemento. El manejador de este evento es `onmouseleave`.
  - **Contextmenu**. Ocurre al pulsar con el botón derecho del ratón. El manejador de este evento es `oncontextmenu`
  - **wheel**. Ocurre al mover la rueda del ratón. El manejador de este evento es `onwheel`.

Modelos de gestión de eventos

- **Propiedades de los eventos del ratón**:
  - Posición del puntero respecto de la ventana del navegador, que se obtienen mediante las propiedades *clientX* y *clientY*.
  - Posición del puntero del ratón respecto de la pantalla completa del ordenador del usuario mediante las propiedades *screenX* y *screenY*
  - Posición del puntero del ratón respecto de la página mediante las propiedades *pageX* y *pageY (excepto IE)*
  - Nombre del evento, mediante la propiedad *type*.
  - Elemento que origina el evento *target (DOM)* o *srcElement(IE)*.
  - *Button* (solo con mousedown, mouseup,mousover, mousemove,mouseout)
    - 0 – botón izquierdo
    - 1 – botón intermedio o rueda
    - 2 – botón derecho

Modelos de gestión de eventos

- **Orden de ejecución de los eventos del ratón**:
  - Cuando se pulsa un botón del ratón, la secuencia de eventos que se produce es la siguiente: mousedown, mouseup, click
  - La secuencia de eventos necesaria para llegar al doble click llega a ser tan compleja como la siguiente: mousedown, mouseup, click, mousedown, mouseup, click, dblclick.

Modelos de gestión de eventos

- **Eventos del teclado**:
  - **Keydown**. Este evento se produce cuando pulsamos una 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`.
  - **Keyup**. Este evento se produce cuando soltamos una tecla. El manejador de este evento es `onkeyup`.
  - **Keypress**. Este evento se produce si pulsamos una tecla de un carácter alfanumérico (El evento no se produce si pulsamos shift, control, alt, alt gr). En el caso de mantener una tecla pulsada, el evento se produce de forma continuada. El manejador de este evento es `onkeypress`.

Modelos de gestión de eventos

- **Propiedades de los eventos del teclado**:
  - **altKey** → devuelve true si la tecla alt se ha pulsado
  - **code** → devuelve el código asociado a la tecla pulsada
  - **ctrlKey** → devuelve true si la tecla control se ha pulsado
  - **key** → devuelve la tecla pulsada
  - **location** → 0 (localización estándar)
    1 (izquierda)
    2 (derecha)
    3 (teclado numérico)
  - **shiftKey** → devuelve true si la tecla shift se ha pulsado

Modelos de gestión de eventos

- **Orden de ejecución de eventos en el teclado**:
  - Para teclas alfanuméricas: keydown, keypress, keyup
  - Para el resto de teclas: keydown, keyup
- Podemos dejar pulsada la tecla:
  - Para teclas alfanuméricas: Se repiten de forma continua los eventos keydown, keypress.
  - Para el resto de teclas: se repite de forma continuada el evento keydown solamente.

Modelos de gestión de eventos

- **Eventos HTML** :
  - **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`.

Modelos de gestión de eventos

- **Eventos HTML** :
  - **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`.
  - **Beforeunload.** Este evento se produce cuando se pulsa un enlace, refrescamos, cerramos,…El manejador es `onbeforeunload`.
  - **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`.

Modelos de gestión de eventos

- **Eventos FORMS**:
  - **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`.
  - **Reset**. Este evento se produce cuando pulsamos sobre un botón de tipo reset. El manejador es `onreset`.
  - **Invalid**. Este evento se produce cuando el valor de un input no es correcto. El manejador es `oninvalid`. No soportado por Safari

Modelos de gestión de eventos

- **Eventos FORMS**:
  - **Submit**. Este evento se produce cuando pulsamos sobre un botón de tipo submit. El manejador es `onsubmit`.
  - **Focus**. Este evento se produce cuando un elemento obtiene el foco. El manejador es `onfocus`.
  - **Focusin** similar a focus, soporta bubbling. Su manejador es `onfocusin`.
  - **Blur**. Este evento se produce cuando un elemento pierde el foco. El manejador es `onblur`.
  - **Focusout**: similar a blur, soporta bubbling. Su manejador es `onfocusout`.
  - **Input**. Este evento se produce cada vez que se escribe en un input o textarea. El manejador es `oninput`.

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.

Utilización de formularios desde código

- Un formulario web sirve para enviar, tratar y recuperar datos que son enviados y recibidos entre un cliente y un servidor web.
- Cada elemento del formulario almacena un tipo de dato o acciona una de sus funcionalidades.
- Los formularios disponen de una arquitectura, en este contexto están enmarcados en el lenguaje HTML.

Utilización de formularios desde código

- Estructura de un formulario:
  - Los formularios se definen con etiquetas.
  - Dentro de cada etiqueta también podemos acceder a los atributos de la misma.
  - 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`.
  - Más atributos:
    - Enctype: define el tipo de codificación para enviar el formulario al servidor. Se usa cuando permite enviar archivos adjuntos.
    - Accept: indica el tipo de fichero adjunto que acepta el servidor.

Utilización de formularios desde código

- 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>
```

Utilización de formularios desde código

- Elementos de un formulario:
  - El elemento principal del formulario se denomina con la etiqueta `<input>`.
  - Según su funcionalidad, los tipos de input se llaman:
    - Controles de formulario.
    - Campos de formulario.
  - Estos últimos se encargan de guardar los datos que se envían a través del formulario.

Utilización de formularios desde código

- Atributos de la etiqueta `input` (1):
  - **Type**. Indica el tipo de elemento que vamos a definir. De él dependen el resto de parámetros. Los valores posibles que acepta el atributo `type` son:
    - text (cuadro de texto).
    - password (cuadro de contraseña, los caracteres aparece ocultos tras asteriscos).
    - checkbox (casilla de verificación).
    - radio (opción de entre dos o más).
    - submit (botón de envío del formulario).
    - reset (botón de vaciado de campos).
    - file (botón para buscar ficheros).
    - hidden (campo oculto para, el usuario no lo visualiza en el formulario), image (botón de imagen en el formulario).
    - button (botón del formulario).

Utilización de formularios desde código

- Atributos de la etiqueta `input` (2):
  - **Name**. El atributo `name` asigna un nombre al elemento. Si no le asignamos nombre a un elemento, el servidor no podrá identificarlo y por tanto no podrá tener acceso al elemento.
  - **Value**. El atributo `value` inicializa el valor del elemento. Los valores dependerán del tipo de dato, en ocasiones los posibles valores a tomar serán verdadero o falso.
  - **Size**. Este atributo asigna el tamaño inicial del elemento. El tamaño se indica en pixeles. En los campos `text` y `password` hace referencia al número de caracteres.
  - **Maxlength**. Este atributo indica el número máximo de caracteres que pueden contener los elementos `text` y `password`. Es conveniente saber que el tamaño de los campos `text` y `password` es independiente del número de caracteres que acepta el campo.

Utilización de formularios desde código

- Atributos de la etiqueta `input` (3):
  - **Checked**. Este atributo es exclusivo de los elementos `checkbox` y radio. En el definimos que opción por defecto queremos seleccionar.
  - **Disable**. Este atributo hace que el elemento aparezca deshabilitado. En este caso el dato no se envía al servidor.
  - **Readonly**. Este atributo sirve para bloquear el contenido del control, por tanto el valor del elemento no se podrá modificar.
  - **Src**. Este atributo es exclusivo para asignar una URL a una imagen que ha sido establecida como botón del formulario.
  - **Alt**. El atributo `alt`, incluye una pequeña descripción del elemento. Habitualmente y si no lo hemos desactivado cuando posicionamos el ratón (sin pulsar ningún botón) encima del elemento, podemos visualizar la descripción del mismo.

Utilización de formularios desde código

- Tipos de input – Cuadro de texto:
  - Este input muestra un cuadro de texto vacío en el que el usuario puede introducir un texto.
  - Este es uno de los elementos más usados. La forma de indicar que es un campo de texto es: `type="text"`

Utilización de formularios desde código

- Tipos de input – Cuadro de contraseña:
  - El cuadro de contraseña es como el cuadro de texto, con la diferencia que los caracteres que escribe el usuario no se ven en pantalla.
  - En su lugar los navegadores muestran asteriscos o puntos.

```
<input type="password" name="contrasenia" />
```

Utilización de formularios desde código

- Tipos de input – Casilla de verificación:
  - Estos elementos permiten al usuario activar o desactivar la selección de cada una de las casillas de forma individual.

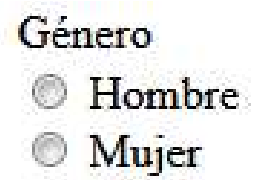
```
<p>Colores favoritos</p>
</br><input name="rojo" type="checkbox" value="ro"/> Rojo
</br><input name="azul" type="checkbox" value="az"/> Azul
</br><input name="verde" type="checkbox" value="ve"/> Verde
```

Colores favoritos
☐ Rojo
☐ Azul
☐ Verde

Utilización de formularios desde código

- Tipos de input – Opción de radio:
  - Este tipo de elemento agrupa una serie de opciones excluyentes entre sí. De esta forma el usuario sólo puede coger una opción de entre todas las que tiene establecidas un grupo de botones radio.

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



Utilización de formularios desde código

- Tipos de input – Botón de envío:
  - Este elemento es el encargado de enviar los datos del formulario al servidor. En este caso el `type` toma el valor `submit`. El valor del atributo `value` se mostrará en este caso en el botón generado.

```
<input type="submit" name="enviar" value="Enviar">
```

Utilización de formularios desde código

- Tipos de input – Botón de reset:
  - Este elemento es un botón que establece el formulario a su estado original.

Utilización de formularios desde código

- Tipos de input – Ficheros adjuntos:
  - Este tipo de input permite adjuntar ficheros adjuntos. El elemento añade de forma automática un cuadro de texto que se dispondrá para almacenar la dirección del fichero adjunto seleccionado.

```
Fichero adjunto
<input type="file" name="fichero"/>c
```


```
<form action="" method="post" enctype="multipart/form-
data"> ... </form>
```

Utilización de formularios desde código

- Tipos de input – Campos ocultos:
  - Los campos ocultos no son visibles en el formulario por el usuario. Estos elementos son útiles para enviar información de forma oculta que no tenga que ser tratada por el usuario.

```
<input type="hidden" name="campoOculto" value="cambiar"/>
```

Utilización de formularios desde código

- Tipos de input – Botón de imagen:
  - Este elemento es una personalización de un botón, cambiando el aspecto por defecto que tienen los botones de un formulario por una imagen.

```
<input type="image" name="enviar" src="imagen_mundo.jpg"/>
```

Utilización de formularios desde código

- Tipos de input – Botón:
  - Existe un elemento botón, al que podemos asociar diferentes funcionalidades. De esta forma no nos tenemos que ceñir los botones de submit o reset que nos ofrecen los formularios.

```
<input type="button" name="opcion" value="Opcion validar"/>
```

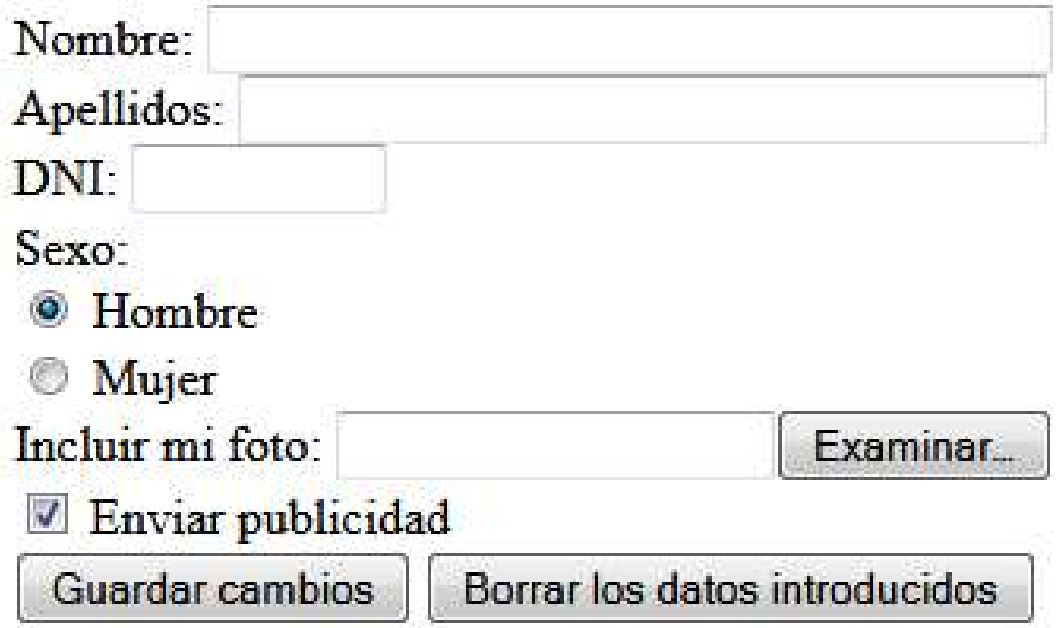
Utilización de formularios desde código

- Ejemplo completo:

```
<form action="pagina.php" method="post"
 enctype="multipart/form-data" /> <br/>
 Nombre:<input type="text" name= "nombre" value="" size="42 maxlength="30”/>
 Apellidos:<input type="text" name="ape" value="” size="40” maxlength="80”/>
 DNI:<input type="text" name="dni" value="" size="10” maxlength="9" />
 Sexo:
 <input type="radio" name="sexo" value="hombre” checked="checked”/>Hombre
 <input type="radio" name="sexo" value="mujer" />Mujer
 Incluir mi foto: <input type="file" name="foto" /> <br/>
 <input name="publ" type="checkbox" value="publicidad” checked="checked"/>
 Enviar publicidad
 <input type="submit" name="enviar" value="Guardar cambios” />
 <input type="reset" name="limpiar" value="Borrar los datos introducidos”/>
</form>
```

Utilización de formularios desde código

- Ejemplo completo:



Modificación de apariencia y comportamiento

- A través de hojas de estilo es posible mejorar notablemente el aspecto de los formularios.
- La visualización de un formulario puede depender de ciertas condiciones que vaya introduciendo el usuario.

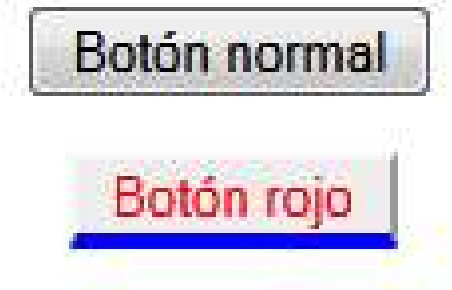
Modificación de la apariencia de un formulario

- 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 HMTL se llama *Cascading Style Sheets* (CSS).

Modificación de la apariencia de un formulario

- Modificar el aspecto de un botón:
  - En ocasiones puede que necesitemos integrar un botón de un formulario en un texto.
  - Color y borde diferente:

```
<input class="azul" type="button" value="Botón azul" />
```



Modificación de la apariencia de un formulario

- Modificar el aspecto de un botón:

```
<html>
  <head>
    <title>Formulario</title>
    <style type="text/css">
      .azul{
        color: red;
        border-bottom: 4px solid blue;
      }
    </style>
  </head>
  <body>  …  </body>
</html>
```

Modificación de la apariencia de un formulario

- Suavizar el aspecto de un campo de texto:
  - Los campos de texto por defecto comienzan a escribir justo al principio del elemento.
  - Desde CSS podemos modificar el punto a partir del que queremos que se escriban los caracteres.

```
<input class="col" type="text" value="texto" /></br></br>

<input class="color" type="text" value="texto" />
```

Modificación de la apariencia de un formulario

- Suavizar el aspecto de un campo de texto:

```
<html>
  <head>
    <title>Formulario</title>
    <style type="text/css">
     .col{ background-color: #FFFF00;}
     .color{ padding: .2em; background-color: #FFFF00;}
    </style>
  </head>
  <body>  …  </body>
</html>
```

Modificación de la apariencia 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>
```

Modificación de la apariencia de un formulario

- Organizar controles de un formulario:

```
<html>
  <head>
    <title>Formulario</title>
   <style type="text/css">
    div { margin: .4em 0;}
    div label { width: 25%; float: left;}
    </style>
  </head>
  <body>  …  </body>
</html>
```

Modificación de la apariencia de un formulario

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

Modificación de la apariencia 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>
```

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.
- Para controlar estas situaciones se pueden usar las validaciones.
- Este tipo de validaciones se suelen realizar llamando a una función que analice si el dato cumple con las restricciones establecidas.

Validación y envío

- Para que el formulario detecte que debe realizar una validación antes de enviar los datos, debemos indicarlo en su estructura.

```
document.form[“formValidado”].onsubmit = validar;

<form action="URL" method="post" name="formValidado”>
   ...
</form>
```

Validación y envío

- Validar un campo como obligatorio:

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

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

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>
```

Validación y envío

- Validar si una fecha es correcta:

```
<script type="text/javascript">
  function validaFecha() {
    var dia = document.getElementById("dia").value;
    var mes = document.getElementById("mes").value;
    var ano = document.getElementById("ano").value;
    fecha = new Date(ano, mes, dia);
    if( !isNaN(fecha) ) {
      return false;
    }
    return true;
  }
</script>
```

Validación y envío

- Validar un checkbox:

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

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

Validación (atributos HTML5)

- Métodos:

checkValidity()

setCustomValidity()

- Propiedades

validity

validationMessage

willValidate

customError

patternMismatch

rangeOverflow

rangeUnderflow

stepMismatch

tooLong

typeMismatch

valueMissing

valid

Se tratan de valores booleanos que toman el valor true si se ha incumplido la restricción indicada a través del atributo html correspondiente

Validación (atributos HTML5)

- Ejemplo:

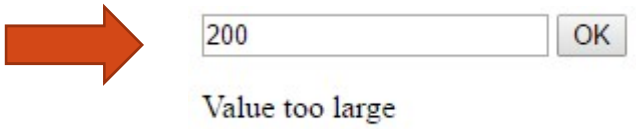
```
<!DOCTYPE html>
<html>
<body>

<p>Enter a number and click OK:</p>
<input id="id1" type="number" max="100">
<button onclick="myFunction()">OK</button>

<p id="demo"></p>

<script>
function myFunction() {
    var txt = "";
    if (document.getElementById("id1").validity.rangeOverflow) {
        txt = "Value too large";
    } else {
        txt = "Input OK";
    }
    document.getElementById("demo").innerHTML = txt;
}
</script>

</body>
</html>
```



Expresiones regulares

- Las expresiones regulares describen un conjunto de elementos que siguen un patrón.
- 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.

Expresiones regulares

- Caracteres especiales (1):
  - **^ 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 seria sería, ^a.
  - **$ Fin de entrada o línea**. Indica que la cadena debe terminar por el elemento precedido al dólar.

Expresiones regulares

- Caracteres especiales (2):
  - *** 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.

Expresiones regulares

- Caracteres especiales (3):
  - **. 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.

Expresiones regulares

- Caracteres especiales (4):
  - **{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.

Expresiones regulares

- Caracteres especiales (5):
  - **[^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.

Expresiones regulares

- Caracteres especiales (6):
  - **\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.

Expresiones regulares

- Caracteres especiales (7):
  - **\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.

Expresiones regulares

- Caracteres especiales (8):
  - **\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.

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.

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+([-
      .]\w+)/.test(valor)) ) {
      return false;
    }
    return true;
  }
</script>
```

Expresiones regulares

- Validar un DNI:

```
<script type="text/javascript">
  function validaDNI(){
    valor = 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(valor)) ){ return false; }
    if(valor.charAt(8) != letras[(valor.substring(0, 8))%23])
    { return false; }
    return true;
}
</script>
```

Expresiones regulares

- Validar un número de teléfono:

```
function validaTelefono(){
  valor = document.getElementById("telefono").value;

  if( !(/^\d{9}$/.test(valor)) ) {
    return false;
  }
  return true;
}
```

Utilización de cookies

- Las cookies surgieron como necesidad ante algunas ausencias tecnológicas del protocolo HTTP.
- Una cookie es un fichero de texto que se almacena en el 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.

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 entrada en el buscador, a través de las *cookies* el servidor reconoce ciertos aspectos que el usuario configuró y conserva el aspecto.

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*.

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.

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, éstas 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.

Utilización de cookies

- **¿Qué guardan las cookies?**

[nombre]=[valor];

expires=[caducidad];

path=[camino];

domain=[dominio];

secure

```
document.cookie="username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
```

Utilización de cookies

nombre	**Nombre** del dato almacenado.(Obligatorio). Números y letras.
valor	**Valor** del dato almacenado. (Obligatorio). Números y letras. En el caso de contener caracteres especiales como “ ” , “ ; ” o espacio en blanco, tienen que reemplazarse por sus valores UNICODE mediante *encodeURIComponent ()*.
expires (Parámetro opcional)	Parámetro de tipo date (y expresada en formato GMT) que indica el tiempo que se conservará la cookie. **Si no se especifica**, la cookie se destruye cuando el usuario sale de la sesión en curso (Esto es cuando se cierran todas las ventanas del navegador y en ese momento se eliminan las cookies).
domain (Parámetro opcional)	Dentro del dominio actual, subdominio para el que la cookie es válida. El valor predeterminado es el subdominio actual. Establecer **domain= .miweb.com** para una cookie que sea válida para cualquier subdominio . Por motivos de seguridad, los navegadores no permiten crear cookies para dominios diferentes al que crea la cookie (**same-origin policy**).
path (Parámetro opcional)	Establece la ruta para la cuál la cookie es válida. Si no se especifica ningún valor, la cookie será **válida para la ruta la página actual**. Para autorizar al dato cookie para todas las carpetas del sitio se debe especificar **“/”**.
secure (Parámetro opcional)	Este atributo indica que la cookie sólo será transmitida a través de un canal seguro con SSL.

Utilización de cookies

- **Limitaciones**
  - Para poder trabajar con la mayoría de los navegadores, es aconsejable no exceder **50 cookies por dominio**, y **4093 bytes por dominio**.
  - Es decir, el tamaño de todas las cookies no debe exceder 4093 bytes.
  - Evitar nombres largos
  - Inyectar una cookie demasiado grande en navegador de usuario puede ocasionar comportamientos anómalos y problemas en las peticiones HTTP

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 cuatro funciones que sirven para:
    - Devolver el valor de una cookie.
    - Escribir una cookie.
    - Borrar una cookie
    - Comprobar si existe un valor para la cookie.

Utilización de cookies

```
function setCookie(name, value, expires, path, domain, secure)
{
  document.cookie = name + "=" + encodeURIComponent(value) +
  ((expires == null) ? "" : "; expires=" +
expires.toGMTString()) +
  ((path == null) ? "" : "; path=" + path) +
  ((domain == null) ? "" : "; domain=" + domain) +
  ((secure == null) ? "" : "; secure");
}
```

Utilización de cookies

```
function getCookie(name){
  var cname = name + "=";
  var dc = document.cookie;
  if (dc.length > 0) {
    begin = dc.indexOf(cname);
    if (begin != -1) {
      begin += cname.length;
      end = dc.indexOf(";", begin);
      if (end == -1) end = dc.length;
        return decodeURIComponent(dc.substring(begin, end));
    }
  }
  return null;
}
```

Utilización de cookies

```
function delCookie (name,path,domain) {
  if (getCookie(name)) {
    document.cookie = name + "=" +
    ((path == null) ? "" : "; path=" + path) +
    ((domain == null) ? "" : "; domain=" + domain) +
    "; expires=Thu, 01-Jan-70 00:00:01 GMT";
  }
}
```