

# **DESARROLLO WEB EN ENTORNO CLIENTE**

**UNIDAD 4 (parte 1/2):  
Interacción con el usuario, eventos y  
formularios**

# 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.

# Modelos de gestión de eventos

- Para poder controlar un evento se necesita un “manejador” (handler).
- 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:

```

```

# Modelos de gestión de eventos

- El ejemplo anterior se puede realizar de otro modo llamando a una función:

```
<html>
  <head>
    <title>Página de Evento del ratón</title>
    <script>
      function funcion() {
        alert("Click en imagen");
      }
    </script>
  </head>
  <body>
    
  </body>
</html>
```

# 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`.
  - **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`.

# 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`.

# 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`.
  - **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`.



# 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`.

# 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`.

# 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`.

## Modelos de gestión de eventos

- Ejercicio: Crea un script donde al pulsar con el botón izquierdo del ratón sobre una imagen, aparezca una ventana emergente con un texto.
- Ejercicio: Crea un script donde al pulsar una tecla dentro de un campo de entrada, aparezca una ventana emergente con un texto.
- Ejercicio: Crea tres campos de entrada, cuando el último obtenga el foco, debe aparecer una ventana emergente con un texto.

# 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.

# Modelos de gestión de eventos

- Objeto **window.event**:

- Normalmente, los manejadores de eventos requieren información adicional para procesar sus tareas. Por ejemplo, si una función se encarga de procesar el evento *onclick*, quizás necesite saber en qué posición estaba el ratón en el momento de pinchar el botón. El caso más habitual en el que es necesario conocer información adicional sobre el evento es el de los eventos asociados al teclado para conocer la tecla que se ha pulsado.
- JS permite obtener información sobre el ratón y el teclado mediante un objeto especial llamado **event**. Desafortunadamente, los distintos navegadores presentan diferencias en el tratamiento de la información sobre los eventos, sobre todo los relacionados con el teclado.

# Ejercicios

- Realiza todos los ejercicios del bloque 'HojaDeEjercicios 4.1'.

# La hamburguesería del vecino

1. Análisis de la estructura de clases, código e información adjunta.
2. Reformulación, creación de una App web que te permita:
  - a. Página de bienvenida (puede incluir un pequeño texto de ayuda que facilite la utilización de la app)
  - b. Registro de pedido del cliente, cesta (elementos, cantidad, precio total)
  - c. Expedición de número de orden de pedido y recibo en una página nueva.
  - d. Información de los ingredientes del producto (obligatorio en el producto principal)
  - e. Registro de información adicional para la comanda (cuadro de texto donde pueda haber observación importantes, cambio de ingredientes, alergias, etc)
  - f. Posibilidad de guardar pedido para realizar un pedido rápido previo.
  - g. Claridad en la interfaz

**¡Recuerda!:** Es el vecino el que elige la hamburguesería y es la hamburguesería la que quiere que sean los vecinos la hamburguesería.



# La hamburguesería del vecino

ID	Criterio	0->1	1->2	2->3	3->4
1	Aspecto funcional	No cumple objetivo. No puedes realizar el pedido y obtener precio de lo elegido.	Cumple objetivo.	Cumple objetivo guiando correctamente pero con dudas o en algún paso con información incompleta.	La interfaz con el usuario es comprensible/ guía bien y se entiende bien hasta el final/ tienes control en todo momento
2	Aspecto estético	No utiliza los recursos especificados	Utiliza los recursos, pero no los explota y entorpece la comprensión.	Utiliza otros recursos, no se ciñe al enunciado, pero no entorpece la comprensión.	Utilizas los recursos de los que dispone la consola/html plano/ para comunicar como interfaz.
3	Documentación	No entrega documentación alguna	No entrega toda la documentación. Faltan cosas.	Entrega toda la documentación, pero hay errores.	Documentación entregada en tiempo y forma. Documento zip con documento recogiendo los cambios en las clases o elemento del la app más relevantes.
4	Informe/Estructura de Clases	No se emplean las clases, no se analiza el trabajo recibido	Se recogen las clases recibidas, pero no se adaptan/evalúan correctamente para la aplicación actual	Se crean nuevas clases pero no se utilizan de manera correcta. Hay redundancia en las mismas de una manera muy evidente afectando a la eficiencia.	Implementa un modelo basado en clases, adaptando o creando nuevas, y explota su utilización
5	Formato código	No comenta, no está estructurado ni con formato	Está estructurado pero no comentado	Tiene deficiencias en cuanto a comentarios. Hay partes relevantes que no están comentadas y afectan a la comprensión del código.	Bien estructurado, comentado y tabulado. Es comprensible.
6	Bases de JS	No utiliza bien el lenguaje. Errores con tipos, estructura de flujo mal empleadas, confusión y funciones sin sentido.	Utiliza el lenguaje, pero hay algún error grave.	Emplea funciones, tipos y estructuras de flujo.	Utiliza bien las variables y las estructuras de control de flujo.
7	Página bienvenida	No existe elemento		Existe página de bienvenida, pero no está bien implementada (no es clara, no cumple función...)	Existe página de bienvenida, que representa lo que tiene que ser y anticipa el empleo de la aplicación de manera clara.
8	Expedición de número orden	No existe elemento		Existe un elemento similar pero con fallos al mostrar, definir...	Existe el elemento y cumple la función de manera clara.
9	Recibo de pedido	No existe elemento		Existe un elemento similar pero con fallos al mostrar, definir...	Existe el elemento y cumple la función de manera clara.
10	Registro de información adicional	No existe elemento		Existe un elemento similar pero con fallos (descontextualizado, oculto...)	Existe el elemento y cumple la función de manera clara.
11	Guarda y recuperación de pedido anterior	No guarda ni recupera.		La app permite guardar y recuperar datos del pedido del usuario pero no persiste entre sesiones.	La app permite guardar y recuperar datos del pedido del usuario en el momento que se quiere y persiste entre sesiones de trabajo con la app.

# 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.
  - La etiqueta principal es `<form>` `</form>`.
  - Para que el formulario 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`.

# 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, limpia texto al pulsar).
    - `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).
    - `Date` (muestra un calendario)
    - `Color` (añade una barra de color y el interfaz para configurarlo)

# 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 qué 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"`:

```
<input type="text" name="nombre"/>
```

# 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
```

Género

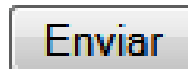
☐ Hombre

☐ 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.

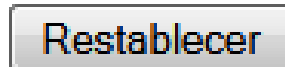
```
<input type="submit" name="enviar" value="Enviar"/>
```



# Utilización de formularios desde código

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

```
<input type="reset" name="Restablecer"/>
```



# 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"/>
```

Fichero adjunto

Examinar...

# 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"/><br/>
Apellidos: <input type="text" name="apellidos" value="" size="40"
           maxlength="80"/><br/>
DNI: <input type="text" name="dni" value="" size="10" maxlength="9"/><br/>
Sexo:<br/>
<input type="radio" name="sexo" value="hombre" checked="checked"/>Hombre<br/>
<input type="radio" name="sexo" value="mujer"/>Mujer<br/>
Incluir mi foto:<input type="file" name="foto"/><br/>
<input name="publicidad" type="checkbox" value="publicidad" checked="checked"/>
  Enviar publicidad<br/>
<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:

Nombre:

Apellidos:

DNI:

Sexo:

☒ Hombre

☐ Mujer

Incluir mi foto:  No se ha seleccionado ningún archivo.

☒ Enviar publicidad

# Modificación de la apariencia y comportamiento de un formulario

- 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.

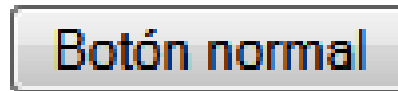
# Modificación de la apariencia de un formulario

- 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 HTML 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

- Ejercicio: Modifica el botón "Guardar cambios" del formulario anterior para que su texto aparezca en color rojo y su borde inferior quede resaltado azul.
- Ejercicio: Modifica el campo de texto "Nombre" del formulario anterior para que quede en amarillo y su borde resaltado.

# Modificación de la apariencia de un formulario

- Organizar controles de un formulario:



Formulario

Nombre

Apellidos

```
<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="Dar me de alta"/>
</fieldset>
```

# Modificación de la apariencia de un formulario

- Organizar controles de un formulario con estilos:

```
<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 del comportamiento 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 del comportamiento de un formulario

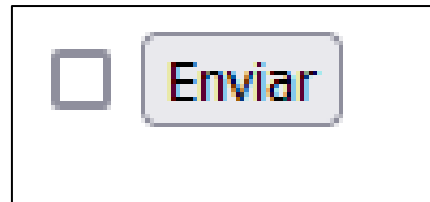
```
<script>
function enviar() {
    if (formulario.alta.checked == true) {
        formulario.action = "alta.html";
    }

    if (formulario.alta.checked == false) {
        formulario.action = "baja.html";
    }

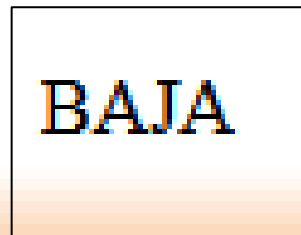
    formulario.submit() // Se envía el formulario.
}
</script>
```

# Modificación del comportamiento de un formulario

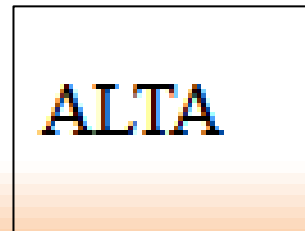
- Ejercicio: Crea un formulario con una casilla de verificación y un botón. Si se pulsa sin que esté seleccionada la casilla, el formulario se envía a la página "baja.html", en cambio, si está seleccionada, a "alta.html".



A rectangular box containing a small square checkbox on the left and a button labeled "Enviar" on the right. The button has a light blue gradient and a thin border.



A rectangular box with a light orange gradient background and a thin black border. The word "BAJA" is centered in a bold, blue, serif font.



A rectangular box with a light orange gradient background and a thin black border. The word "ALTA" is centered in a bold, blue, serif font.



# 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.

```
<form action="URL" method="post" name="formValidado"  
  onsubmit="return validacion() ">  
  ...  
</form>
```

Ver ejemplo...

# Validación y envío

- Validar un campo como obligatorio:

```
<script>
  function validaCampo() {
    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>
  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>
    function validaFecha() {
        var dia = document.getElementById("dia").value;
        var mes = document.getElementById("mes").value;
        var anio = document.getElementById("anio").value;

        fecha = new Date(anio, mes, dia);

        if ( isNaN(fecha) ) {
            alert("La fecha no es correcta");
            return false;
        }

        alert("La fecha introducida es: " + fecha);
        return true;
    }
</script>
```

# Validación y envío

- Validar una casilla de verificación:

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

    if ( !elemento.checked ) {
      alert("La casilla no ha sido validada");
      return false;
    }

    return true;
  }
</script>
```

# Validación y envío

- Ejercicio: Crea un formulario con las validaciones vistas en este apartado.

Nombre:

Teléfono:

Día:  Mes:  Año:

Validación: ☐

# 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.
- El método **test()** ejecuta la búsqueda de una ocurrencia entre una expresión regular y una cadena especificada. Devuelve true o false.



# 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 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>
  function validaEmail() {
    valor = document.getElementById("email").value;

    if ( !(/^(.+\.+\.+)$/.test(valor)) ) {
      return false;
    }

    return true;
  }
</script>
```

- El método test () ejecuta la búsqueda entre una expresión regular (patrón) y una determinada cadena.

# Expresiones regulares

- Validar un DNI:

```
<script>
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-zA-Z]$/).test(valor)) ) { return false; }

    if ( valor.charAt(8).toUpperCase() !=
        letras[(valor.substring(0,8))%23] )
    { return false; }

    return true;
}
</script>
```

# Expresiones regulares

- Validar un número de teléfono nacional:

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

# Expresiones regulares

- Ejercicio: Crea un formulario con las validaciones vistas en este apartado.

Email:

DNI:

Teléfono:

# Ejercicios

- Realiza todos los ejercicios del bloque 'Ejercicios 4.2'.