



UD5: Gestión de eventos y formularios

Antonio Sierra

Basado en el trabajo de Javier G. Pisano

Índice

- ▶ Eventos
- ▶ Tratamiento avanzado de eventos
- ▶ Formularios
- ▶ Formularios HTML5
- ▶ Expresiones regulares
- ▶ Cookies
- ▶ Almacenamiento local



Eventos

UD5: Gestión de eventos y formularios

Eventos

- ▶ Los scripts mostrados hasta ahora se ejecutan secuencialmente
 - ▶ Con las estructuras de control de flujo y funciones cambiamos dicho comportamiento ligeramente.
- ▶ Con Javascript podemos utilizar un modelo de funcionamiento **basado en eventos**.
 - ▶ Los scripts se ejecutan una vez que el usuario haya “hecho algo” (pulsar un botón, mover el ratón, cerrar la ventana..), es decir, se produzca un **evento**.
 - ▶ Puedo asignar una función a cada uno de dichos eventos. Este tipo de funciones se llaman **manejadores de eventos**.

Modelos de eventos

- ▶ La mayor parte de incompatibilidades entre navegadores se produce en el modelo de eventos.
- ▶ Existen 3 modelos diferentes para manejar eventos dependiendo del navegador donde se ejecute la aplicación.
 - ▶ Modelo básico de eventos.
 - ▶ Implementado por todos los navegadores.
 - ▶ Modelo de eventos estándar.
 - ▶ Más poderoso que el básico, lo incluyen todos salvo IE.
 - ▶ Modelo de eventos de Internet Explorer.
 - ▶ Modelo propio.

Principales eventos

Evento	Descripción	Elementos para los que está definido
onblur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
<u>onclick</u>	Pinchar y soltar el ratón	Todos los elementos
ondblclick	Pinchar dos veces seguidas con el ratón	Todos los elementos
onfocus	Seleccionar un elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onkeydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>

Principales eventos

Evento	Descripción	Elementos para los que está definido
onkeypress	Pulsar una tecla	Elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
<u>onload</u>	La página se ha cargado completamente	<body>
onmousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos
onmouseout	El ratón "sale" del elemento (pasa por encima de otro elemento)	Todos los elementos
<u>onmouseover</u>	El ratón "entra" en el elemento (pasa por encima del elemento)	Todos los elementos
<u>onmouseup</u>	Soltar el botón que estaba pulsado en el ratón	Todos los elementos

Principales eventos

Evento	Descripción	Elementos para los que está definido
<u>onmouseup</u>	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
onreset	Inicializar el formulario (borrar todos sus datos)	<code><form></code>
onresize	Se ha modificado el tamaño de la ventana del navegador	<code><body></code>
onselect	Seleccionar un texto	<code><input></code> , <code><textarea></code>
onsubmit	Enviar el formulario	<code><form></code>
onunload	Se abandona la página (por ejemplo al cerrar el navegador)	<code><body></code>

Modelo básico de eventos

- ▶ Cada elemento HTML define su propia lista de posibles eventos que se le pueden asignar.
 - ▶ Un mismo evento (ej: pulsar el ratón) puede estar definido para varios elementos XHTML distintos, y un elemento puede tener asociados varios eventos distintos.
 - ▶ El nombre de cada evento se construye mediante el prefijo on seguido del nombre en inglés de la acción asociada al evento.
 - ▶ Ejemplos: onclick, onmousemove, onkeydown...

Manejadores de eventos

- ▶ Existen varias maneras de indicar los manejadores de eventos:
 - ▶ Como atributos de elementos XHTML.
 - ▶ Como funciones Javascript externas.
 - ▶ Manejadores “semánticos”.
 - ▶ **Registro de eventos avanzado (W3C)**

Manejadores de eventos como atributos XHTML

- ▶ El código se incluye en un atributo del propio elemento XHTML.
 - ▶ Método más sencillo pero menos aconsejable.

```
<input type="button" value="Pinchame!"  
  onclick="alert('Buuu!')" />
```

```
<div onclick="alert('Has pinchado con el ratón');"  
  onmouseover="alert('Has pasado el ratón por encima');">  
  <p>Pulsa sobre el texto o pasa el ratón por encima</p>  
</div>
```

Manejadores de eventos como funciones externas

- ▶ Agrupamos el código Javascript en una función externa y la llamamos desde el elemento XHTML.
 - ▶ La función puede estar definida en el mismo archivo XHTML (etiqueta <script>) o en un archivo externo.

```
<input type="button" value="Pinchame!"  
onclick="muestraMensaje();" />
```

```
function muestraMensaje(){  
    alert('Buuu!');  
}
```

Manejadores de eventos semánticos

- ▶ El problema que presentan los métodos vistos es que “ensucian” el código HTML.
 - ▶ Además de separar contenido (HTML) y presentación (CSS) también se recomienda en la medida de lo posible separar el comportamiento (Javascript).
- ▶ La técnica de los manejadores de eventos semánticos consiste en:
 1. Asignar un identificador único al elemento XHTML (id).
 2. Crear una función Javascript encargada de manejar el evento.
 3. Asignar la función al evento correspondiente (sin paréntesis), utilizando el DOM para acceder al elemento HTML y permitir el enlace.

Manejadores de eventos semánticos: Ejemplo

HTML

1

```
<input id="pinchable" type="button" value="Pinchame">
```

JS

2

```
function muestraMensaje(){  
    alert('Buuu!');  
}
```

3

```
document.getElementById("pinchable").onclick=  
    muestraMensaje;
```

Manejadores de eventos semánticos

- ▶ Como ya vimos, la página debe cargarse correctamente antes de poder utilizar las funciones DOM.
 - ▶ Lo más normal es utilizar el evento `window.onload()`
- ▶ Ejemplo
 - ▶ Aquí usamos una función sin nombre (función anónima):

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

Variable this

- ▶ Podemos usarla desde un manejador de evento para referirnos al elemento XHTML que ha provocado el evento.
- ▶ En los manejadores externos no puede utilizarse directamente, es necesario pasarla como parámetro
- ▶ En los manejadores semánticos puede usarse directamente.

Variable this: Ejemplo

```
<div id="contenidos" style="border: 1px solid silver"
  onmouseover="document.getElementById
    ('contenidos').style.borderColor='black';"
  onmouseout="document.getElementById('contenidos')
    .style.borderColor='silver';">
</div>
```



```
<div id="contenidos" style="border: 1px solid silver"
  onmouseover="this.style.borderColor='black';"
  onmouseout="this.style.borderColor='silver';">
</div>
```

Variable this: Ejemplo

- ▶ En las funciones externas no se puede utilizar la variable, luego es necesario pasarla como parámetro.

```
<div id="contenidos" style="border: 1px solid silver"
    onmouseover="bordeDorado(this);"
    onmouseout="bordeNegro(this);">
</div>
```

```
function bordeDorado(elemento){
    elemento.style.borderColor="silver";
}

function bordeNegro(elemento){
    elemento.style.borderColor="black";
}
```

Registro de eventos avanzado (W3C)

- ▶ El W3C propone una manera de registrar los eventos sobre cualquier objeto usando el método `addEventListener()`.
- ▶ El método recibe 3 argumentos:
 - ▶ Tipo de evento
 - ▶ Función a ejecutar
 - ▶ Valor booleano que indica cuando se captura el evento.

Registro de eventos avanzado: Ejemplo

```
document.getElementById("enlace")
    .addEventListener('click', alertar, false);

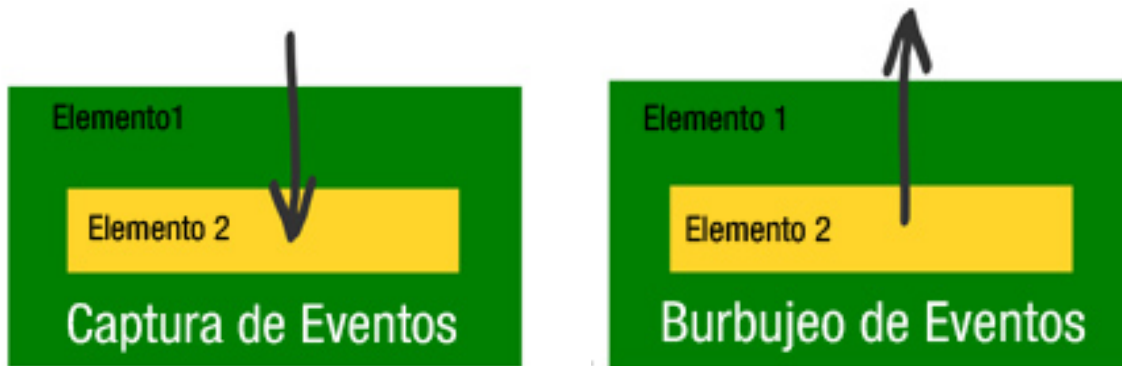
function alertar(){
    alert("Te conectaremos con la página: "
        + this.href);
}
```

► Usando funciones anónimas:

```
element.addEventListener('click', function () {
    this.style.backgroundColor = '#cc0000';
}, false)
```

Registro de eventos avanzados

- ▶ Otras operaciones relacionadas con eventos:
 - ▶ `removeEventListener()` -> Elimina un evento de un elemento
 - ▶ `preventDefault()` -> Cancela la ejecución de un evento.
- ▶ El tercer parámetro indica el modo de transmisión de eventos entre elementos anidados
 - ▶ `true`: Registro el evento en la fase de captura.
 - ▶ `false`: Registro el evento en la fase de burbujeo.



EJERCICIO PROPUESTO



- Implementa un script que muestre por consola el tamaño de la ventana del navegador cada vez que ésta cambie de tamaño



Tratamiento avanzado de eventos

UD6: Modelo de Objetos del Documento

Obteniendo información del evento

- ▶ En muchos casos, los manejadores de eventos requieren de información adicional para procesar sus tareas
 - ▶ Qué tecla he pulsado
 - ▶ Qué botón del ratón he tocado
 - ▶ Qué evento se ha producido
 - ▶ Dónde se ha desencadenado el evento
 - ▶ Etc...
- ▶ Podemos obtener dicha información mediante la clase Event.

Los manejadores de eventos reciben de forma implícita un instancia de un objeto de este tipo

Event

- ▶ Los distintos navegadores presentan diferencias notables en el tratamiento de eventos
 - ▶ Internet Explorer considera que el objeto forma parte del objeto window.
 - ▶ El resto de navegadores lo consideran como el único argumento que tienen las funciones manejadoras de eventos.

```
function manejadorEventos(elEvento){  
    var evento=window.event; /*Internet Explorer */}
```

```
function manejadorEventos(elEvento){  
    var evento=elEvento; /*Resto de navegadores */}
```

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

RECOMENDADO

Información del evento

► Propiedad type (Clase Event)

- Devuelve el tipo de evento producido. Es igual al nombre del evento pero sin el prefijo on.

```
document.getElementById("contenidos").  
    addEventListener("mouseover",resalta);  
document.getElementById("contenidos").  
    addEventListener("mouseout",resalta);
```

```
function resalta(elEvento){  
    var evento= window.event || elEvento;  
    switch(evento.type){  
        case "mouseover":  
            this.className="estilo1";  
            break;  
  
        case "mouseout":  
            this.className="estilo2";  
            break;  
    }  
}
```

Eventos de teclado

- ▶ Cuando un usuario pulsa una tecla se producen tres eventos de teclado:
 - ▶ onkeydown
 - ▶ Pulsar una tecla.
 - ▶ onkeyup
 - ▶ Levantar la tecla
 - ▶ onkeypress
 - ▶ Escribir un carácter
 - ▶ Ciertos caracteres van asociados a combinaciones de varias teclas
 - ▶ Por ejemplo si pulsamos Alt + e (€), solo se desencadena 1 vez el evento onkeypress

Cada evento de teclado tiene un comportamiento ligeramente distinto dependiendo de evento y navegador.

Detectando la tecla pulsada [Clase Event]

- ▶ Propiedad code [Clase Event]
 - ▶ Representa la tecla física en el teclado
- ▶ Propiedad key [Clase Event]
 - ▶ Representa el carácter pulsado.
 - ▶ Si es imprimible, se devuelve el Unicode
 - ▶ Si no es imprimible (control) se devuelve un valor predefinido (cadena)

Eventos de teclado

Ejemplo

```
document.body.addEventListener("keydown", pulsaTecla);
```

```
function pulsaTecla(evento){  
    var e=window.event || evento;  
    var codigo=e.code;  
    var tecla=e.key;  
    console.log("Código: "+codigo+ ". Tecla: "+tecla);  
    if(codigo==="Enter")  
        console.log("Pulsé el Enter");  
    else if(codigo==="Space")  
        console.log("Pulsé el espacio");  
}
```

Detectando la tecla pulsada

- ▶ Las siguientes propiedades de Event permiten comprobar si se ha pulsado alguna tecla adicional
 - ▶ Útiles para implementar por ejemplo atajos de teclado.

altKey	Está presionada la tecla ALT en el momento del evento
ctrlKey	Está presionada la tecla CONTROL en el momento del evento
metaKey	Está presionada la tecla META en el momento del evento
shiftKey	Está presionada la tecla SHIFT en el momento del evento

Eventos del ratón

- ▶ La información más relevante de los eventos de ratón son las coordenadas del puntero.
 - ▶ Origen de coordenadas: Esquina superior izquierda
- ▶ Puedo obtener la posición:
 - ▶ Respecto de la pantalla del ordenador.
 - ▶ Propiedades `evento.screenX` y `evento.screenY`
 - ▶ Respecto de la ventana del navegador
 - ▶ Propiedades `evento.clientX` y `evento.clientY`
 - ▶ Respecto de la página HTML
 - ▶ Útil si se hizo scroll con el ratón.
 - ▶ Propiedades `evento.pageX` y `evento.pageY`

Detectando la tecla pulsada

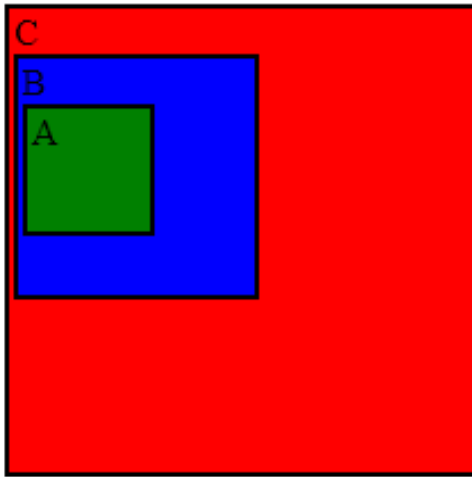
- ▶ Evento **onmousedown**
 - ▶ Propiedad **evento.button**
 - ▶ Distinto valor según la tecla pulsada:
 - ▶ 0: Izquierdo.
 - ▶ 1: Central.
 - ▶ 2: Derecho.

Origen del evento

- ▶ Podemos detectar el origen del evento de 3 modos distintos
 - ▶ **this**
 - ▶ Dentro de un manejador de evento, se refiere al objeto DOM sobre el cual se ha definido el manejador de evento
 - ▶ **currentTarget** (Propiedad de Event)
 - ▶ Referencia al objeto DOM sobre el cual se ha definido el manejador de evento
 - ▶ Es equivalente a this
 - ▶ **target** (Propiedad de Event)
 - ▶ Referencia al objeto DOM que desencadenó el evento.
 - ▶ No tiene por qué ser el mismo

Origen del evento

Ejemplo



```
cajaA.addEventListener("click", pulsa);
cajaB.addEventListener("click", pulsa);
cajaC.addEventListener("click", pulsa);

function pulsa(e){
  console.log("Pulso "+this.id);
  /* Equivalente a
  e.currentTarget.id */
}
```

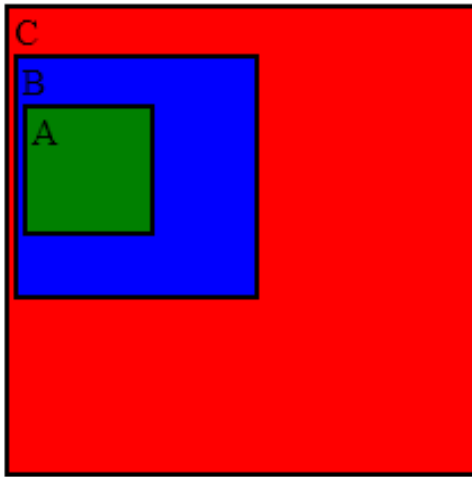
Pulso A

CONSOLA

Pulso A
Pulso B
Pulso C

Origen del evento

Ejemplo



```
cajaA.addEventListener("click",pulsa);
cajaB.addEventListener("click",pulsa);
cajaC.addEventListener("click",pulsa);

function pulsa(e){
    console.log("Pulso "+e.target.id);
}
```

Pulso A

CONSOLA

Pulso A
Pulso A
Pulso A

Elemento tr

Propiedades y métodos

preventDefault()

Cancela la acción por defecto.

Ejemplos:

- No se envía un formulario al pulsar el botón *submit*
- No se sigue la URL de un enlace al pulsar sobre él
- No se marca un radio button al pulsar sobre él

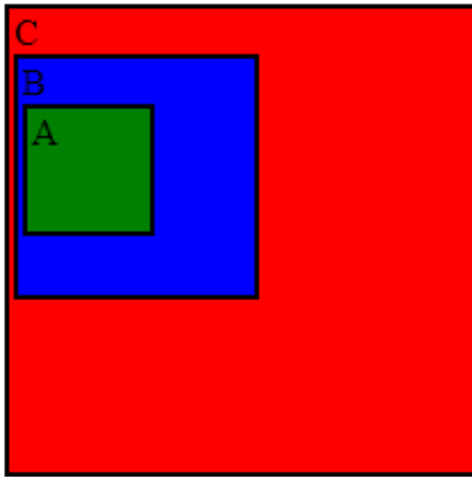
No todos los eventos son cancelables

stopPropagation()

Evita que el evento burbujee.

Origen del evento

Ejemplo



```
cajaA.addEventListener("click",pulsa);
cajaB.addEventListener("click",pulsa);
cajaC.addEventListener("click",pulsa);

function pulsa(e){
  console.log("Pulso "+this.id);
  e.stopPropagation();
}
```

Pulso A

CONSOLA

Pulso A

EJERCICIO PROPUESTO



- ▶ Crear html con un div cuyo contenido sea Contenido del div y un input vacío.
- ▶ Al cargar la página debemos definir 3 procesos de escucha para el elemento input y 2 para el div. En el caso del input definir blur, focus, click y en el caso del div mouseover y mouseout.
- ▶ Los 5 procesos de escucha serán atendidos con el mismo método, que cambiará el className de cada elemento en su caso salvo en el caso del click que únicamente mostraremos mensaje por consola. El className tendrá definido un color para cada caso. Siempre mostraremos un mensaje por consola del evento que ha saltado.

EJERCICIO PROPUESTO



- ▶ Tenemos una página con un botón, un párrafo y una caja de texto.
- ▶ Queremos que al pulsar el botón se muestre en el párrafo el texto de la caja de texto.

EJERCICIO PROPUESTO



- ▶ Dada una página con una página con enlace y un párrafo:
 - ▶ Queremos que al pulsar sobre el enlace el párrafo se oculte/muestre de manera alternativa.
 - ▶ El texto del enlace debería cambiarse (Ocultar/mostrar)
- ▶ Modificar el script para que el método sirva para N conjuntos enlace+ párrafo

EJERCICIO PROPUESTO



- ▶ Tenemos una página HTML (propuesto4.html) con N fotos de perros y gatos, al lado de cada una de las cuales hay una etiqueta, además de haber otra etiqueta general.
- ▶ Queremos que:
 - ▶ Al pulsar sobre cada una de las imágenes la etiqueta muestre el número de pulsaciones sobre su imagen correspondiente.
 - ▶ La etiqueta general muestre el nombre del último animal sobre el cual hemos pinchado.





Formularios

UD5: Gestión de eventos y formularios

Envío y validación de formularios

- ▶ Consiste en comprobar si todos los datos de un formulario han sido introducidos correctamente.
- ▶ Dos métodos (no excluyentes)
 - ▶ Lado servidor (PHP, ASP, JSP..)
 - ▶ Más segura
 - ▶ Lado cliente (JS)
 - ▶ Más fácil y eficiente.

Validación en JS

- ▶ Podemos validar:
 - ▶ A medida que vamos introduciendo datos en el formulario (campo a campo, evento onBlur).
 - ▶ Al pulsar el botón de envío (evento onSubmit)
- ▶ Tipos de validaciones:
 - ▶ Existencia
 - ▶ Tipo de datos
 - ▶ Patrones (email, fecha, DNI)..

Objeto Form

- ▶ Los formularios son el principal medio de introducción de datos en una aplicación Web, y el principal punto de interactividad con el usuario
- ▶ Los formularios y sus controles son objetos del DOM con propiedades únicas.
 - ▶ Se representan mediante un objeto de tipo Form que se corresponde con la etiqueta <form> de HTML
 - ▶ Podemos acceder a los formularios a través de las funciones del DOM

Acceso a formularios

```
<form id="contactar" action="...">...</form>
```

► Métodos de acceso:

```
/* Opción 1 */  
var formulario=document.getElementById("contactar");  
  
/* Opción 2 */  
var formularios=document.getElementsByTagName("form");  
var primerFormulario=formularios[0];  
  
/* Opción 2 (en una línea) */  
var formulario=document.getElementsByTagName("form")[0];  
  
/* Opción 3 */  
var formulario=document.forms[0];
```

Objeto Form: Propiedades y métodos

► Propiedades

action	Atributo action del formulario
length	Número de elementos del formulario
name	Atributo name del formulario
method	Atributo method del formulario
target	Atributo target del formulario
action	Atributo action del formulario
length	Número de elementos del formulario

► Métodos

reset()	Resetea el formulario
submit()	Envía el formulario

Form: Acceso al contenido

- ▶ `form.elements[]`
 - ▶ Es un array que contiene todos los elementos **input** dentro de un formulario en el orden en que aparecen en el código fuente del documento.
 - ▶ Normalmente es más eficaz referenciar a cada elemento individualmente con las propiedades del DOM, pero a veces nos interesa acceder al contenido secuencialmente.

Acceso al contenido: Ejemplo

- Limpiamos todos los campos de texto del formulario:

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

Objeto Form: Objetos contenidos

- ▶ Cada elemento de un formulario también se implementa mediante un objeto con propiedades y métodos sobre los cuales podemos realizar acciones
 - ▶ Validar contenido, marcar o desmarcar opciones, mostrar contenido/ocultarlo..
- ▶ Acceso:
 - ▶ A través del id: `document.getElementById('id')`
 - ▶ A través del nombre: `document.formulario.nombre`
 - ▶ Array `formulario.elements[]`

Acceso al contenido: Ejemplo

► Formulario

```
<form id="contactar" action="contacta.php">  
  <input type="text" id="nombre" name="txtNombre" />  
  <input type="submit" id="enviar" name="btnEnviar"  
    value="Enviar" />  
</form>
```

► Alternativas de acceso al campo de texto:

```
document.getElementById("nombre");  
document.contactar.txtNombre;  
document.elements[0];  
document.forms["contactar"].elements["txtNombre"];  
document.forms["contactar"].txtNombre;
```

Formulario: Entradas de texto

`input (type = radio, password o hidden)`

`textarea`

► Propiedades

defaultValue	Valor por defecto del campo de texto
form	Formulario que contiene el campo de texto
maxLength	Longitud máxima de caracteres permitidos
name	Nombre
readOnly	Atributo de sólo lectura
size	Ancho en caracteres
type	Tipo
value	Valor del campo de texto

► Métodos

submit()	Envía el formulario
-----------------	---------------------

Formulario: Casillas de verificación

```
input (type = checkbox)
```

► Propiedades

checked	Estado del checkbox que indica con un booleano si está marcado
defaultChecked	Valor por defecto del atributo checked
form	Referencia al formulario que contiene el checkbox
name	Atributo name del checkbox
value	Valor del checkbox (atributo value) El value es el texto asociado al objeto que se envía al procesar el formulario

Ejemplo: Acceso a un checkbox

```
<form >  
  <input type="checkbox" id="gusta" name="gusta"  
    value="si"/>Me gusta  
  <input type="submit" value="Enviar" />  
</form>
```

```
// Para que el siguiente código se ejecute debe ser  
// asignado a un evento como veremos posteriormente
```

```
var gusta = document.getElementById("gusta");  
if(gusta.checked){  
  gusta.checked=false;  
} else {  
  gusta.checked=true;  
}
```

Formulario: Botones de radio

```
input (type = radio)
```

- ▶ Comparte propiedades con el checkbox
 - ▶ **name:** Todos los botones del grupo deben coincidir en el valor de este atributo.
 - ▶ Se crea un array con el listado de objetos radio que tienen el mismo **name**, siendo este el nombre del array

Botones de radio: Ejemplo

```
<form name="formulario" action="comida.php">
  <fieldset><legend>¿Qué quieres de comer:</legend>
    <input type="radio" name="comida" id="fabada"
      value="fabada" checked>
    <label for="fabada">Fabada</label>
    <input type="radio" name="comida"
      id="arroz-leche" value="arroz-leche">
    <label for="arroz-leche">Arroz con leche</label>
    <input type="radio" name="comida" id="cachopo"
      value="cachopo">
    <label for="cachopo">Cachopo</label>
  </fieldset></form>
```

```
for (var i=0;i<document.formulario.comida.length; i++){
  if (document.formulario.comida[i].checked){
    alert(document.formulario.comida[i].value);
  }
}
```


Formulario: Lista de selección

`select`

- ▶ Lista desplegable donde seleccionamos una o varias de las opciones
 - ▶ Las opciones se definen con la etiqueta **option**
 - ▶ En JS se representan con un array de objetos **option** llamado **options**
- ▶ `selectedIndex`
 - ▶ Índice (base 0) de la opción seleccionada.
- ▶ Para cada objeto **option**:
 - ▶ **text**: Texto de la selección
 - ▶ **value**: Valor interno de la opción

Lista de selección: Ejemplo

```
<select name="provincias" id="provincias">
  <option value="C">La Coruña</option>
  <option value="LU">Lugo</option>
  <option value="OU">Ourense</option>
  <option value="PO">Pontevedra</option>
</select>
```

```
var provincias = document.getElementById("provincias");
var texto = provincias.options[provincias.selectedIndex].
    text;
var valor = provincias.options[provincias.selectedIndex].
    value;
console.log("Datos de la opción seleccionada:\n\nTexto: “
    + texto + “\nValor: “ + valor);
```

EJERCICIO PROPUESTO



- ▶ Dado el formulario proporcionado (**propuesto5.html**)
 - ▶ Muestra por consola los datos introducidos al pulsar el botón 'Enviar Datos'
 - ▶ Comprueba previamente:
 - ▶ Si se han introducido datos en las cajas de texto y escogido un producto
 - ▶ Las contraseñas coinciden y son mayores de 6 caracteres

Formulario

Nombre	<input type="text"/>
Apellidos	<input type="text"/>
Fecha de nacimiento	<input type="text"/>
DNI	<input type="text"/>
Email	<input type="text"/>
Contraseña	<input type="password"/>
Contraseña (repetición)	<input type="password"/>
Género	<input checked="" type="radio"/> Hombre <input type="radio"/> Mujer
<input type="checkbox"/> Suscribirme al boletín de novedades	
<input type="checkbox"/> Informarme sobre ofertas	
Producto favorito	<input type="text" value="Elige uno..."/>
Comentario	<input type="text" value="Introduce tu comentario..."/>
<input type="button" value="Enviar datos"/>	

EJERCICIO PROPUESTO



- Diseña una clase para guardar objetos de la clase que encapsula el formulario e instancia un objeto de dicha clase cada vez que se pulse el botón de enviar datos.

Formulario

Nombre	<input type="text"/>
Apellidos	<input type="text"/>
Fecha de nacimiento	<input type="text"/>
DNI	<input type="text"/>
Email	<input type="text"/>
Contraseña	<input type="password"/>
Contraseña (repetición)	<input type="password"/>
Género	<input checked="" type="radio"/> Hombre <input type="radio"/> Mujer
<input type="checkbox"/> Suscribirme al boletín de novedades	
<input type="checkbox"/> Informarme sobre ofertas	
Producto favorito	<input type="text" value="Elige uno..."/>
Comentario	<input type="text" value="Introduce tu comentario..."/>
<input type="button" value="Enviar datos"/>	

EJERCICIO PROPUESTO



- ▶ Dado el formulario (propuesto7.html)
 - ▶ Comprueba los campos obligatorios, mostrando el mensaje pertinente en caso de no introducirlos
 - ▶ Almacena el número de intentos, mostrándolos en la capa inferior.

Formulario

Nombre	<input type="text"/>	*
Apellidos	<input type="text"/>	*
Fecha de nacimiento	<input type="text"/>	
DNI	<input type="text"/>	
Email	<input type="text"/>	*
Contraseña	<input type="password"/>	*
Contraseña (repetición)	<input type="password"/>	*
Género	<input checked="" type="radio"/> Hombre <input type="radio"/> Mujer	
	<input type="checkbox"/> Suscribirse al boletín de novedades	
	<input type="checkbox"/> Informarme sobre ofertas	
Producto favorito	<input type="text" value="Elige uno..."/>	*
Comentario	<div><div>Introduce tu comentario...</div><div></div></div>	

Enviar datos

Este es tu primer intento



Formularios HTML5

UD5: Gestión de eventos y formularios

Elementos de formulario

- ▶ Se facilita mucho el desarrollo de formularios
 - ▶ Se añaden dos nuevos métodos que pueden ser usados en la acción del formulario:
 - ▶ update: Cuando queremos modificar recursos en el servidor
 - ▶ delete: Cuando queremos borrar recursos del servidor.
 - ▶ Se añaden 12 nuevos tipos de input y elementos de formulario que mejoran la experiencia de usuario y añaden validaciones (evitando el uso de Javascript).
 - ▶ Si el navegador no los acepta se añade un campo de texto tradicional.
 - ▶ Los dispositivos móviles ofrecen teclados virtuales adaptados al tipo de campo.

Nuevos atributos de input

- ▶ autofocus [booleano]
 - ▶ Define qué elemento tiene el foco al cargar la página.
 - ▶ Solo debe haber un elemento en el formulario con este atributo definido.
- ▶ placeholder [cadena]
 - ▶ Define un texto de ayuda definido en el campo, que desaparece cuando el usuario introduce algún dato.
- ▶ required [booleano]
 - ▶ El navegador no permite el envío de este elemento si está vacío.

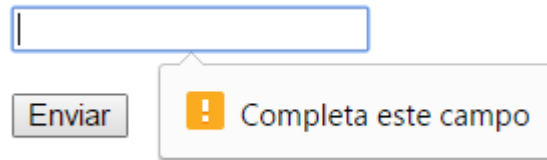


Diagrama de un formulario web que ilustra el atributo required. Se muestra un campo de entrada de texto vacío. Debajo del campo, a la izquierda, hay un botón etiquetado 'Enviar'. A la derecha del botón, se muestra un mensaje de error en un recuadro con un icono de advertencia (un triángulo naranja con un signo de exclamación) y el texto 'Completa este campo'.

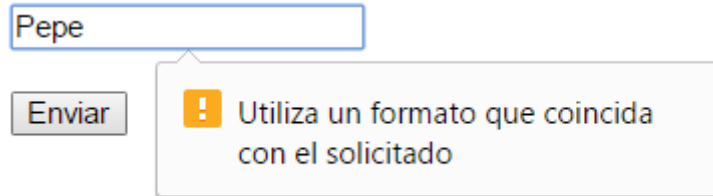
Nuevos atributos de input

- ▶ autocomplete [booleano]
 - ▶ Permite controlar el comportamiento del autocompletado en los campos de texto del formulario (que por defecto están activados)
- ▶ multiple [booleano]
 - ▶ Indica que podemos admitir varios valores (ej: Imágenes, emails...)
- ▶ min, max [numérico]
 - ▶ En campos numéricos indica respectivamente mínimo y máximo valor permitidos.
- ▶ step[numérico]
 - ▶ Si procede, indica el valor en que un campo aumenta/disminuye su valor.

Nuevos atributos de input

- ▶ form [texto]
 - ▶ Si queremos ubicar un elemento fuera de la etiqueta <form>, podemos relacionarlo con el mismo indicando en el atributo form del elemento el id del formulario al que pertenece.
- ▶ pattern [texto]
 - ▶ Permite introducir una expresión regular (usando sintaxis Javascript) que el valor del campo debe cumplir
 - ▶ Ejemplo: Un número seguido de tres letras mayúsculas:

```
<input type="text" pattern="[0-9][A-Z]{3}" />
```



Pepe

Enviar

Utiliza un formato que coincida con el solicitado

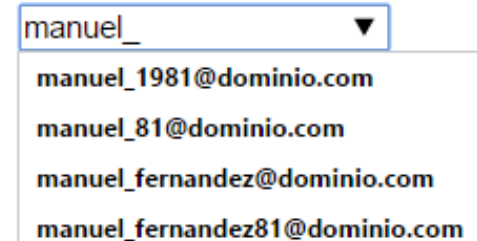
El formulario muestra un campo de texto con el valor "Pepe" y un botón "Enviar". Debajo del botón, hay un mensaje de error que indica: "Utiliza un formato que coincida con el solicitado".

Nuevos atributos de input

► list[texto]

- Se usa en campos de texto en conjunción con un elemento de tipo <datalist>
- Las opciones definidas en el datalist se muestran como una lista desplegable:

```
<input type="text" list="sugerencias" />  
<datalist id="sugerencias">  
  <option>manuel_1981@dominio.com</option>  
  <option>manuel_81@dominio.com</option>  
  <option>manuel_fernandez@dominio.com</option>  
  <option>manuel_fernandez81@dominio.com</option>  
</datalist>
```



manuel_ ▼

- manuel_1981@dominio.com
- manuel_81@dominio.com
- manuel_fernandez@dominio.com
- manuel_fernandez81@dominio.com

Tipo email

```
<input type="email"/>
```

- ▶ Indica al navegador que no debemos permitir el envío de un formulario si no hemos introducido una dirección válida.
 - ▶ No comprueba si la dirección existe



Enviar

! Incluye un signo "@" en la dirección de correo electrónico. La dirección "xx" no incluye el signo "@".

Tipo url

```
<input type="url"/>
```

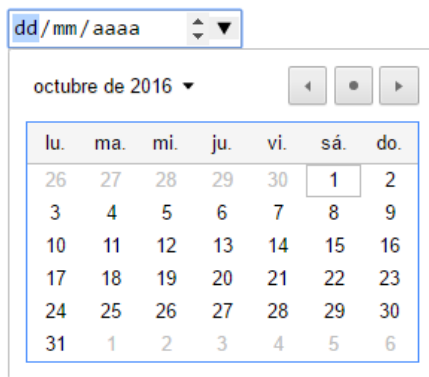
- ▶ No permito el envío del formulario si no introducimos una URL correcta
 - ▶ Ojo, esto incluye el prefijo http:// [algunos navegadores lo añaden]



Tipo fecha

```
<input type="date"/>
```

- ▶ Se proporciona una interfaz de usuario para introducir y validar una fecha.
 - ▶ La fecha se envía al servidor en formato ISO para representación de fechas
 - ▶ Permite unificar el código de tratamiento de fechas (habitualmente a criterio del programador)



Tipo hora

```
<input type="time"/>
```

- ▶ Se proporciona una interfaz de usuario para la introducción de una hora en formato 24h y validarla
 - ▶ La fecha se envía al servidor en formato ISO para representación de fechas



A screenshot of a web browser's time input field. It displays '23:--' where the minutes are masked with dashes. To the right of the dashes is a blue square button with a white 'X' (clear) and a vertical spinner button.



A modal dialog titled 'Establecer hora' (Set time). It features a 3x2 grid of buttons for selecting hours and minutes. The first row shows '7' and '08', the second row shows '8' and '09', and the third row shows '9' and '10'. Each button is underlined. At the bottom, there are three buttons: 'ELIMINAR' (Delete), 'CANCELAR' (Cancel), and 'ESTABLECER' (Set).

Tipo mes

```
<input type="month"/>
```

- ▶ Permite escoger un mes de un año determinado
 - ▶ Internamente el mes se almacena como un número del 1 al 12.
- ▶ Existe un tipo week para escoger la semana del año
 - ▶ Muy usado en países anglosajones.

Establecer mes

sept.	2015
oct.	2016
nov.	2017

ELIMINAR CANCELAR ESTABLECER

Tipo rango

```
<input type="range"/>
```

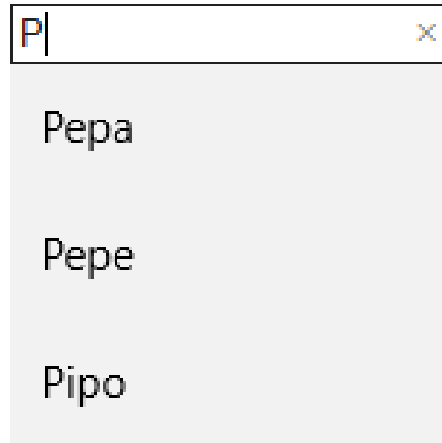
- ▶ Permite añadir de manera sencilla un deslizador
 - ▶ Se suele combinar con los atributos max, min y step



Tipo búsqueda

```
<input type="search"/>
```

- ▶ Diferencia estética con una caja de texto corriente.
 - ▶ Añade un historial de búsquedas



P

Pepa

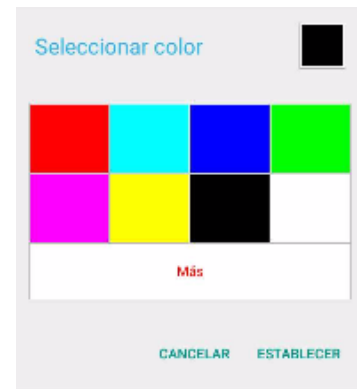
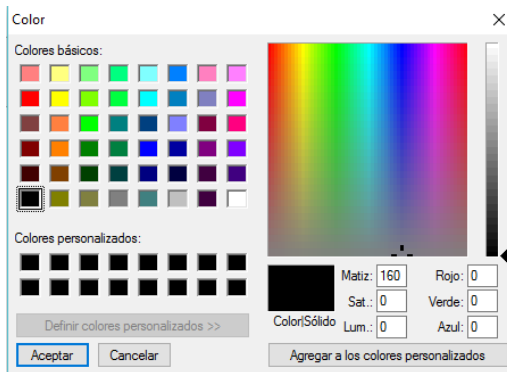
Pepe

Pipo

Tipo color

```
<input type="color"/>
```

- Muestra un cuadro de diálogo para escoger un color.



Barra de progreso

► Etiqueta <progress>

- Usada para representar un avance o progreso en la ejecución de una tarea que lleva tiempo.
 - No tiene animación, habitualmente usaremos JavaScript para animarla.
 - Atributos:
 - max (por defecto 1.0)
 - value (entre 0 y max 1.0). Si no lo especifico, la barra muestra una animación dinámica.



EJERCICIO PROPUESTO



- Modifica el formulario proporcionado (**propuesto7.html**) para añadir los elementos HTML5 pertinentes

Formulario

Nombre	<input type="text"/>	*
Apellidos	<input type="text"/>	*
Fecha de nacimiento	<input type="text"/>	
DNI	<input type="text"/>	
Email	<input type="text"/>	*
Contraseña	<input type="password"/>	*
Contraseña (repetición)	<input type="password"/>	*
Género	<input checked="" type="radio"/> Hombre <input type="radio"/> Mujer	
	<input type="checkbox"/> Suscribirse al boletín de novedades	
	<input type="checkbox"/> Informarme sobre ofertas	
Producto favorito	<input type="text" value="Elige uno..."/>	*
Comentario	<div><div>Introduce tu comentario...</div><div></div></div>	

Enviar datos

Este es tu primer intento



Expresiones regulares

UD5: Gestión de eventos y formularios

Expresiones regulares

- ▶ Para validar expresiones usando los métodos del objeto String en ocasiones necesitamos una lógica de programación de gran complejidad.
- ▶ Las expresiones regulares son patrones de búsqueda que facilitan la validación de datos
- ▶ **Buscan texto que coincida con el patrón especificado**
 - ▶ El texto se pasa usando el método test(texto)
- ▶ Se gestionan a través del objeto **RegExp**

Expresiones regulares

- ▶ Creación de un literal de tipo RegExp:
 - ▶ `var expresion=/expresión regular/`
- ▶ La expresión regular está compuesta de:
 - ▶ Caracteres normales
 - ▶ Caracteres en combinación con caracteres especiales (+,[,],*..)
- ▶ El carácter de escape (\) se usa:
 - ▶ Para indicar que un carácter normal se usa como carácter especial (d,d,w...)
 - ▶ Para indicar que un carácter especial debe tratarse literalmente

Expresiones regulares

Carácter	Coincidencias	Ejemplo de patrón	Ejemplo de cadena
^	Al inicio de una cadena	^Esto	Esto es...
\$	Al final de una cadena	final\$	Es el final
*	Coincide 0 o más veces	se*	se see seeee
?	Coincide 0 o una vez	ap?	an apio apple
+	Coincide 1 o una más veces	ap+	apple And
{n}	Coincide exactamente n veces	ap{2}	apple apabullante

Expresiones regulares

Carácter	Coincidencias	Ejemplo de patrón	Ejemplo de cadena
{n,}	Coincide n o más veces	ap{2,}	apple appple apabullante
{n,m}	Coincide de n a m veces	ap{2,4}	apple apppppppple
.	Cualquier carácter excepto salto de línea	a.e	Cualquier carácter excepto salto de línea entre la a y la e
[..]	Cualquier carácter entre corchetes	a[px]e	ape axe ale

Expresiones regulares

Carácter	Coincidencias	Ejemplo de patrón	Ejemplo de cadena
[^..]	Cualquier carácter salvo los que están entre corchetes	a[^px]e	ale axe ape
\d	Dígitos del 0 al 9	\d{3}	456 4335 hola
\D	Cualquier carácter que no sea un dígito	\D{2,4}	se pepe 5no
\s	Espacio en blanco		

Expresiones regulares: Ejemplos

- ▶ Número de la seguridad social americano (formato AAA-GG-SS):

```
d{3}-?\d{2}-?\d{3}$
```

- ▶ Dirección de email según definición oficial:

```
/^[a-zA-Z0-9.!#$%&'*/+=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$/
```

- ▶ Número de DNI (sin validar letra)

```
\d{8}[A-Z]$
```

Expresiones regulares con RegExp

- ▶ RegExp es tanto un literal como un objeto JS que se puede crear usando un constructor:

```
var expresion = new RegExp("Texto de la Expresión Regular");
```

- ▶ Es más eficiente usar el literal si la expresión no cambia (se compila al ejecutar el script).

Objeto RegExp: Propiedades

► Propiedades

global	Especifica que se utilice el modificador “g”
ignoreCase	Especifica que se utilice el modificador “i”
lastIndex	El índice donde comenzar la siguiente búsqueda
multiline	Especifica si el modificador “m” es utilizado.
source	El texto de la expresión regular RegExp

Objeto RegExp: Métodos

► Métodos

compile()	Compila una expresión regular
exec()	Busca una coincidencia en una cadena, devolviendo la primera coincidencia
test()	Busca una coincidencia en una cadena, devolviendo true o false.

Expresiones regulares y validación:

Ejemplos

- Comprobar si una subcadena está contenida en otra. Dados los datos de cadenas en datos. Definir el patrón y mostrar por consola aquellas que cumplan lo especificado en el comentario.

```
var datos = new Array();  
// Imprimir cadenas que contengan "Blog" cualquier caracter  
(.) 0 o más veces (*) y a continuación "Goog"  
  
datos[0] = "El Blogger de Google";  
datos[1] = "El blogger de Google";  
datos[2] = "BloggerGoogle";  
datos[3] = "Google Blogger";
```


Expresiones regulares y validación: Ejemplos

- ▶ Comprobar un número de la seguridad social americano
 - ▶ Formato: 8 dígitos separados por guiones: AAA-GG-SSS.
 - ▶ En la expresión regular ponemos los guiones como opcionales.

```
function comprobar(numero) {  
    var patron = /^\\d{3}-?\\d{2}-?\\d{3}$/;  
    return ( numero.match(patron));  
}
```

Expresiones regulares y validación: Ejemplos

- Validar un campo de texto obligatorio

```
if( valor === null  
    || valor.length === 0 || /^\\s+$/.test(valor)){  
    return false;  
}
```

- Validar un campo de texto con valores numéricos

```
var valor = document.getElementById("campo").value;  
if(valor.trim()==="" || isNaN(valor)){  
    return false;  
}
```

Expresiones regulares y validación: Ejemplos

- Validar que se ha seleccionado una opción de una lista

```
var indice =  
    document.getElementById("opciones").selectedIndex;  
if( indice == null || indice == 0 ){  
    return false;  
}
```

- Validar una dirección de email

```
var valor = document.getElementById("campo").value;  
if(!(/^[a-zA-Z0-9.!#$%&'*/+=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-  
zA-Z0-9-]{0,61}[a-zA-Z0-9])?\.[a-zA-Z0-9](?:[a-zA-Z0-9-  
{0,61})*[a-zA-Z0-9]$/.test(valor))){  
    return false;  
}
```

Expresiones regulares y validación: Ejemplos

► Validar una fecha

```
var ano = document.getElementById("ano").value;
var mes = document.getElementById("mes").value;
var dia = document.getElementById("dia").value;
valor = new Date(ano, mes, dia);
if( !isNaN(valor)){
    return false;
}
```

Expresiones regulares y validación: Ejemplos

► Validar un número de DNI

```
var valor = document.getElementById("campo").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.substr(0, 8))%23]){
    return false;
}
```

Expresiones regulares y validación: Ejemplos

► Validar un número de teléfono

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

► Validar si se ha seleccionado un checkbox

```
var elemento = document.getElementById("campo");  
if( !elemento.checked ){  
    return false;  
}
```

Expresiones regulares y validación: Ejemplos

- Validar si se ha seleccionado un radio button

```
var opciones = document.getElementsByName("opciones");
var seleccionado = false;
for (var i=0; ((i<opciones.length) && (!seleccionado)); i++) {
    if (opciones[i].checked){
        seleccionado = true;
    }
}

if (!seleccionado){
    return false;
}
```

EJERCICIO PROPUESTO



► Valida mediante expresiones regulares:

- Fecha de nacimiento en formato DD/MM/AAAA ó DD/MM/AA.
- Email correcto
- NIF correcto

Formulario

Nombre	<input type="text"/>	*
Apellidos	<input type="text"/>	*
Fecha de nacimiento	<input type="text"/>	
DNI	<input type="text"/>	
Email	<input type="text"/>	*
Contraseña	<input type="password"/>	*
Contraseña (repetición)	<input type="password"/>	*
Género	<input checked="" type="radio"/> Hombre <input type="radio"/> Mujer	
	<input type="checkbox"/> Suscribirse al boletín de novedades	
	<input type="checkbox"/> Informarme sobre ofertas	
Producto favorito	<input type="text" value="Elige uno..."/>	*
Comentario	<input type="text" value="Introduce tu comentario..."/>	

Enviar datos

Este es tu primer intento



Cookies

UD5: Gestión de eventos y formularios

Cookies

- ▶ Son **ficheros de texto** que se almacenan en determinada carpeta en el navegador
 - ▶ Mozilla. En el perfil del navegador.
 - ▶ [windows]\Application Data\Mozilla\Profiles\[profilename]\;
 - ▶ IES: Archivos temporales de Internet.
 - ▶ Chrome
 - ▶ [user] \ Local Settings \ Application Data \ Google \ Chrome \ User Data \ Default.

Cookies

- ▶ Son específicas al dominio
 - ▶ Si un dominio crea una cookie, otro dominio no puede acceder a ella a través del navegador.
- ▶ Tienen fecha de caducidad.
- ▶ Formato:
 - ▶ Dominio
 - ▶ Trayectoria de URLs que pueden acceder a la cookie.
 - ▶ Fecha de caducidad
 - ▶ **Nombre de una entrada de datos**
 - ▶ **Cadena de texto asociada a la entrada de datos**

Cookies: Grabar una cookie

- ▶ Mediante asignación simple con la propiedad `document.cookie`.
 - ▶ Sintaxis ([] → Opcional)

```
document.cookie = "nombreCookie = datosCookie  
[; expires=horaformatoGMT]  
[; path=ruta]  
[; domain=nombreDominio]  
[; secure] "
```

Grabar una cookie: Ejemplo

```
document.cookie="username=manolo";
```

- ▶ Cada cookie debe tener nombre y texto asignado (aunque sea vacío)
- ▶ Si no existen una cookie con el nombre la creará automáticamente.
- ▶ Si ya existe una cookie con el nombre se reemplaza.

Grabar una cookie: Otros parámetros

- ▶ **expires.** Formato GMT:
 - ▶ expires=Thu, 01-Jan-15 00:00:01 GMT;
 - ▶ Si no se establece la cookie dura el tiempo de la sesión (hasta cierre de navegador).
- ▶ **path.** Ruta actual de nuestra Web
- ▶ **domain.** Si no se pone es el dominio de la página que creó la cookie.
- ▶ **secure.** Nuestra cookie será accesible por cualquier programa en el dominio.

Cookies: Recuperar una cookie

- ▶ Accedemos a document.cookie e imprimimos su valor.
- ▶ No podemos tratar las cookies como objetos:
 - ▶ Debemos obtener la cadena de texto de la misma y extraer los datos necesarios de su contenido.

Cookies: Recuperar una cookie

```
function getCookie(nomCookie) {  
    var n;  
    var nombre;  
    var valor;  
    var resultado = null;    // si no se encuentra = nulo  
    var cook=document.cookie.split(";"); // pares de valores  
    // revisamos todos los pares  
    for (var i=0; i<cook.length; i++) {  
        n = cook[i].split("="); // separamos nombre/valor  
        nombre=n[0];  
        valor =n[1];  
        // si es el buscado  
        if (nombre.trim()==nomCookie.trim()){  
            resultado = valor;// devolvemos su valor  
        }  
    }  
    return resultado;  
}
```


Estableciendo Cookies

```
/*  
    Indicariamos el nombre y valor de la cookie mediante  
    strings  
    La cducidad sería un entero con el número de días que  
    queremos darle de caducidad  
*/  
function setCookie(name,value,caducidad)  
{  
    var now = new Date();  
    var then = now;  
    then.setDate(then.getDate() + caducidad);  
    console.log(then);  
    document.cookie=name+"="+value  
    +";expires="+then.toGMTString()+";path="/";  
}
```

Comprobaciones previas

- ▶ Antes de trabajar con cookies debemos comprobar si están activas en el navegador
 - ▶ Por ejemplo: Podemos intentar guardar un valor y luego recuperarlo

```
function haycookies(){  
    document.cookie="micookie=hay";  
    return ( getCookie("micookie") == "hay" );  
}
```

EJERCICIO PROPUESTO



- ▶ Guarda los datos del formulario en las cookies
- ▶ Carga los datos desde las cookies (si existen) al cargar la página.

Formulario

Nombre	<input type="text"/>	*
Apellidos	<input type="text"/>	*
Fecha de nacimiento	<input type="text"/>	
DNI	<input type="text"/>	
Email	<input type="text"/>	*
Contraseña	<input type="password"/>	*
Contraseña (repetición)	<input type="password"/>	*
Género	<input checked="" type="radio"/> Hombre <input type="radio"/> Mujer	
	<input type="checkbox"/> Suscribirse al boletín de novedades	
	<input type="checkbox"/> Informarme sobre ofertas	
Producto favorito	<input type="text" value="Elige uno..."/>	*
Comentario	<div><div>Introduce tu comentario...</div><div></div></div>	

Enviar datos

Este es tu primer intento



Almacenamiento local

UD5: Gestión de eventos y formularios

Cookies: Limitaciones

- ▶ La cookie viaja en las cabeceras de cada petición HTTP.
- ▶ Tamaño limitado (4KB) → Permite guardar muy poca información
 - ▶ Habitualmente un identificador de sesión que sirve para recuperar información adicional del servidor.

Almacenamiento local (API HTML5)

- ▶ HTML5 dispone de 3 tecnologías que permiten que las aplicaciones almacenen datos en el cliente:
 - ▶ **WebStorage**. Permite almacenar parejas de clave/valor
 - ▶ **Web SQL Database**. Sistema de almacenamiento basado en SQL
 - ▶ No va a ser mantenido en el futuro, aunque su uso está extendido.
 - ▶ **IndexedDB**. Sistema de almacenamiento basado en objetos.

WebStorage

- ▶ Permite almacenar una cantidad de datos mucho mayor que las cookies (hasta 5-10M MB)
 - ▶ Más potente.
- ▶ No tiene caducidad
- ▶ No transmite los datos en cada petición HTTP.
 - ▶ Mejor rendimiento
 - ▶ Mejor seguridad.

localStorage y sessionStorage

- ▶ Son dos implementaciones de la interfaz **Storage** que permiten el almacenamiento local.
 - ▶ **sessionStorage** actúa sobre el ámbito de la sesión de la ventana o pestaña.
 - ▶ **localStorage** permite que los datos perduren indefinidamente.

localStorage y sessionStorage

getItem(key)	Devuelve el valor correspondiente a la clave key.
setItem(key,value)	Almacena el value referenciado por la cadena key.
removeItem(key)	Elimina el par clave/valor con clave igual a key.
length	Atributo que contiene el número de elementos par/valor almacenados.
key(i)	Devuelve la clave del elemento que está en la posición i.
clear()	Elimina todos los elementos.

Comprobando la compatibilidad

- Práctica conveniente pues versiones obsoletas no lo permiten (debemos buscar una alternativa)

```
function compruebaCompatibilidad(){  
    var info=document.getElementById("info");  
    if(window.sessionStorage && window.localStorage){  
        console.info("Tu navegador acepta almacenamiento local");  
    }  
    else{  
        console.info("Tu navegador NO acepta almacenamiento local");  
    }  
}
```

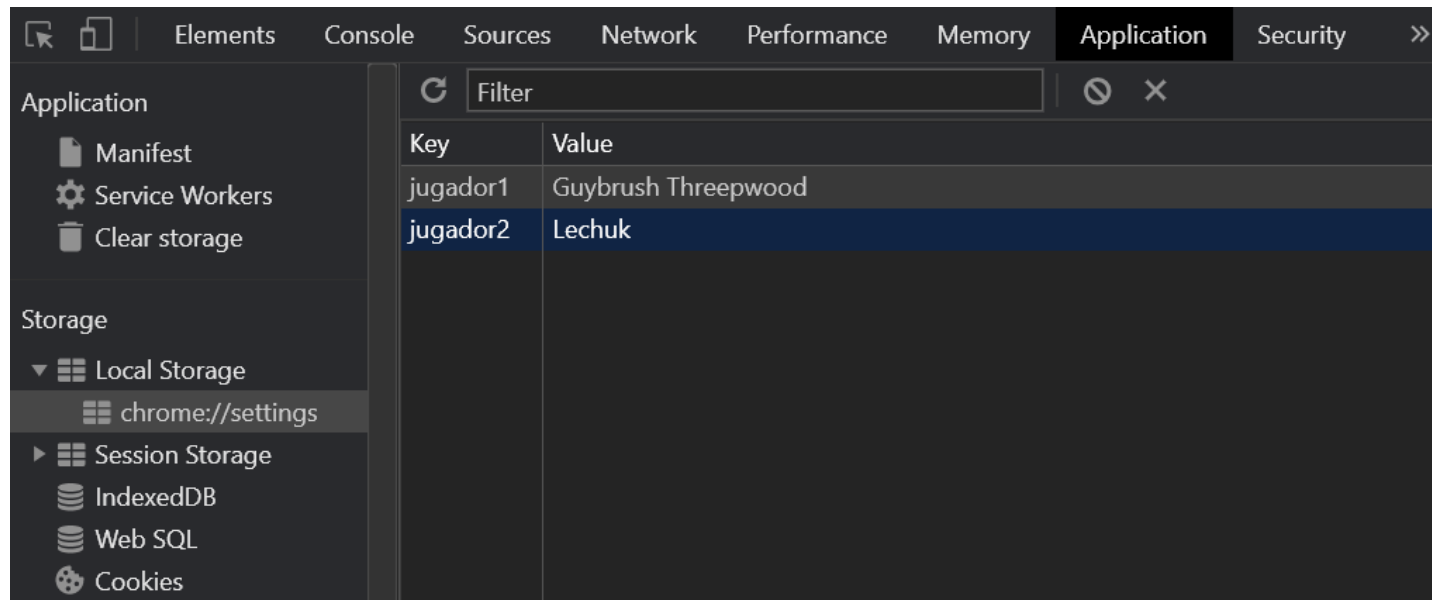
Guardando datos

- La clave identifica de manera unívoca el valor que guardo.

```
function guardaDatos(clave,valor){  
    localStorage.setItem(clave,valor);  
}  
  
guardaDatos("jugador1","Guybrush Threepwood");  
guardaDatos("jugador2","Lechuk");
```

Guardando datos: Comprobación

► En Chrome: Pestaña *Application*



Guardando datos: Ejemplo

► ¿Qué valor se guarda?

```
guardaDatos("jugador1","Fermín");  
guardaDatos("jugador2","Pachín");
```

Guardando datos

```
function recuperaDatos(clave){  
    return localStorage.getItem(clave);  
}  
  
var info = document.getElementById("salida");  
info.innerHTML=recuperaDatos("jugador1");
```

Borrando datos

```
function eliminar(clave){  
    localStorage.removeItem(clave);  
}  
  
eliminar("continuaciones");
```

Recuperando todos los datos

```
guardaDatos("jugador", "Guybrush Threepwood");
guardaDatos("continuaciones", "3");
guardaDatos("tiempoJuego", "6:03");

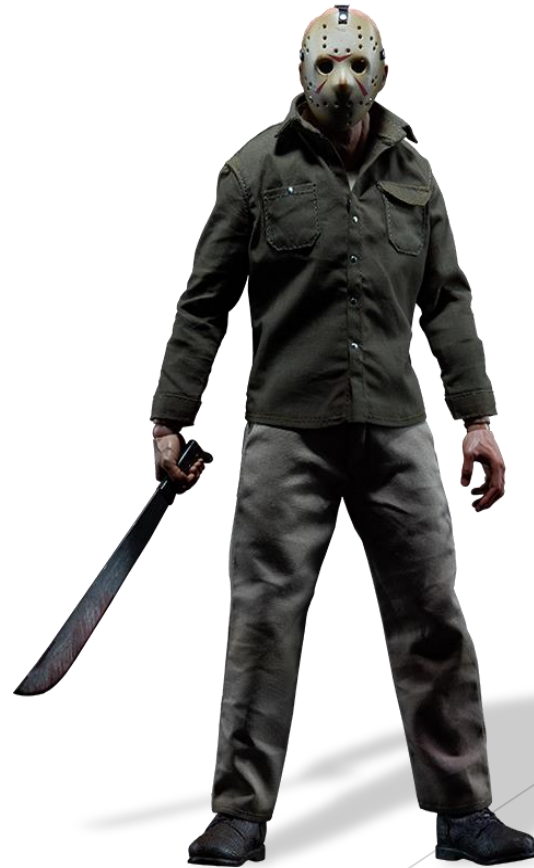
// Llamamos a la función definida a continuación
muestraTodosLosDatos();

function muestraTodosLosDatos(){
    var info=document.getElementById("info");
    info.innerHTML="";

    for(var i=0; i<localStorage.length;i++){
        var clave=localStorage.key(i);
        var contenido=localStorage.getItem(clave);
        info.innerHTML+=clave+": "+contenido+"<br/>"
    }
}
```


Almacenando objetos

- ▶ Como vimos en el ejemplo anterior, en ocasiones nos puede interesar almacenar objetos en lugar de un único String.
- ▶ Tenemos que hacer uso de JSON



JSON (JavaScript nOtationN)

- ▶ Es un formato de intercambio de datos que sigue una sintaxis muy parecida al modo en que almacenamos objetos en Javascript.
- ▶ Intenta reemplazar a XML, pues es mucho más sencillo de procesar.
- ▶ Es aceptado de manera nativa por Javascript

```
{  "result": [{
    "actor": "Vivien Leigh",
    "title": "Gone with the Wind",
    "director": "Victor Fleming",
    "description": "Going with the wind"  },
  {
    "actor": "Michael J Fox",
    "title": "Back to the Future",
    "director": "Robert Zemeckis",
    "description": "Going back to the future"  }
]}
```

Serializando objetos con JSON

- ▶ **Serializar:** Convertir un objeto a una representación que puedo almacenar fácilmente de manera persistente.
- ▶ Entre otros, JSON proporciona un par de métodos que permiten representar un objeto como cadena y viceversa.

JSON.stringify(objeto)	Devuelve una representación como String del objeto que le paso.
JSON.parse(cadena)	Devuelve un objeto de acorde con la representación en cadena que le paso.

Serializando objetos con JSON

- ▶ Las propiedades se representan entre comillas
 - ▶ El resto de tipos de datos con su representación “habitual”.
- ▶ No podemos serializar objetos con métodos
- ▶ Podemos serializar objetos anidados (proprios o predefinidos)
 - ▶ Ejemplo: Arrays.

Convirtiendo a JSON: Ejemplo

```
var jugador={  
  nombre:"Zangief",  
  porcentajeVida:100,  
  nivel:3,  
  tiempo:"1:00",  
  trucos:false,  
}  
  
var jugadorJSON = JSON.stringify(jugador);  
alert(jugadorJSON);
```

Esta página dice

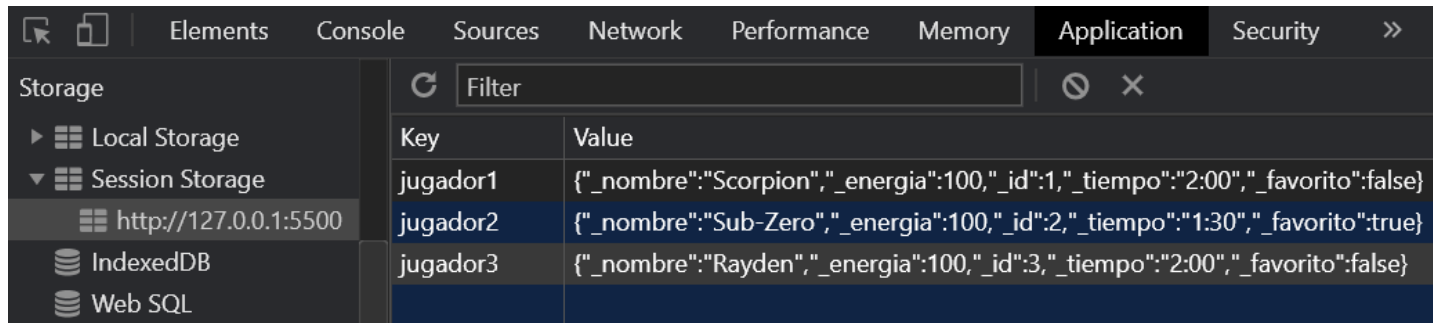
```
{"nombre":"Zangief","porcentajeVida":100,"nivel":3,"tiempo":"1:00","tru  
cos":fals  
e}
```

Aceptar

Serializando objetos: Ejemplo

```
var jugador1=new Jugador("Scorpion",100,1,"2:00",false);
var jugador2=new Jugador("Sub-Zero",100,2,"1:30",true);
var jugador3=new Jugador("Rayden",100,3,"2:00",false);

sessionStorage.setItem("jugador"+jugador1.id,
                        JSON.stringify(jugador1));
sessionStorage.setItem("jugador"+jugador2.id,
                        JSON.stringify(jugador2));
sessionStorage.setItem("jugador"+jugador3.id,
                        JSON.stringify(jugador3));
```



Storage	
Local Storage	
Session Storage	
http://127.0.0.1:5500	
IndexedDB	
Web SQL	

Key	Value
jugador1	{"_nombre":"Scorpion","_energia":100,"_id":1,"_tiempo":"2:00","_favorito":false}
jugador2	{"_nombre":"Sub-Zero","_energia":100,"_id":2,"_tiempo":"1:30","_favorito":true}
jugador3	{"_nombre":"Rayden","_energia":100,"_id":3,"_tiempo":"2:00","_favorito":false}

Recuperando objetos serializados

```
for(var i=1;i<sessionStorage.length;i++){  
    var jugadorJSON=sessionStorage.getItem("jugador"+i);  
    var jugador=JSON.parse(jugadorJSON);  
    alert(jugador._nombre);  
}
```

127.0.0.1:5500 dice

Scorpion

Aceptar