



XUNTA DE GALICIA
CONSELLERÍA DE EDUCACIÓN
E ORDENACIÓN UNIVERSITARIA



www.xunta.gal

Ciclo superior:

Diseño de aplicaciones web

MÓDULO:

Desarrollo web en entorno cliente

Material realizado por Rafael Veiga para el MEC

Adaptación y ampliación realizada por Beatriz Buyo

ÍNDICES

Índices	2
Índice de Contenidos	2
Índice de Tablas	3
Índice de Imágenes	3
Índice de Para Saber Más	3
Índice de Ejemplos	4
Índice de Recomendación	5
Índice de Ejercicios	5
Unidad didáctica 5	6
Mapa conceptual	44
Anexo.- Licencias de recursos.	45
GLOSARIO	46

ÍNDICE DE CONTENIDOS

1. El objeto form.	7
1.1 Formas de selección del objeto form.	8
1.2 El formulario como objeto y contenedor.	9
1.3 Acceso a propiedades y métodos del formulario.	10
1.4 Colección form.elements	11
2. Objetos relacionados con formularios.	12
2.1 Objetos input de tipo texto y otros.	13
2.2 Objeto input de tipo checkbox.	16
2.3 Objeto input de tipo radio.	17
2.4 Objeto select.	18
2.5 Objeto textarea.	21
2.6 Pasando objetos a las funciones usando this.	22
3. Eventos.	23
3.1 Modelo de registro de eventos en línea.	24
3.2 Modelo de registro de eventos tradicional.	26
3.3 Modelo de registro avanzado de eventos según W3C.	28
3.4 Modelo de registro de eventos según Microsoft.	30
3.5 Orden de disparo de los eventos.	32
4. Envío y validación de formularios.	34
5. Almacenamiento de datos en el lado del cliente	36
5.1 Las cookies.	37
5.2 Gestión y uso de cookies.	39
5.3 localStorage y sessionStorage.	40

ÍNDICE DE TABLAS

Tabla 1 Propiedades del objeto form	10
Tabla 2 Métodos del objeto form	10
Tabla 3 Colecciones del objeto form	10
Tabla 4 Propiedades de la colección elements	11
Tabla 5 Métodos de la colección elements	11
Tabla 6 Propiedades de los objetos input	14
Tabla 7 Métodos de los objetos input	14
Tabla 8 Propiedades y métodos de los objetos Input	15
Tabla 9 Propiedades del objeto select	19
Tabla 10 Colecciones del objeto select	19
Tabla 11 Métodos del objeto select	19
Tabla 12 Propiedades del objeto option	19
Tabla 13 Métodos del objeto option	20
Tabla 14 Propiedades del objeto Textarea	21
Tabla 15 Métodos del objeto textarea	21
Tabla 16 Listado de eventos	30
Tabla 17 Propiedad del objeto Storage	42
Tabla 18 Métodos del objeto Storage	42
Tabla 19 Eventos del objeto Storage	42
Tabla 20 Propiedades del objeto creado por el evento Storage	42

ÍNDICE DE IMÁGENES

Ilustración 1 Form en la jerarquía de objetos del DOM (árbol de nivel 0)	9
Ilustración 2 Árbol de nivel 0 para el formulario del ejemplo	9
Ilustración 3 Form en la jerarquía de objetos del DOM (árbol de nivel 2)	9
Ilustración 4 Tipos de objetos Input que soporta el HTML5 a 29/12/2017	13
Ilustración 5 Consentimiento informado de Cookies	36
Ilustración 6 Una posible interacción entre un navegador web y un servidor, en la que el servidor envía una cookie con la página y el navegador la devuelve cuando el usuario solicita otra página	37
Ilustración 7 Soporte de Web Storage de las diferentes versiones de los navegadores	40

ÍNDICE DE PARA SABER MÁS

PARA SABER MÁS 1 Ejemplo de uso del objeto select	20
PARA SABER MÁS 2 Documentación completa del objeto forms	20

ÍNDICE DE EJEMPLOS

Ejemplo 1 Método <code>getElementById</code>	8
Ejemplo 2 Método <code>getElementsByTagName</code>	8
Ejemplo 3 Método <code>document.forms</code>	8
Ejemplo 4 Limpieza de los <code>input type text</code> de la Colección <code>elements</code> de un formulario	11
Ejemplo 5 Referencias a un objeto del formulario	12
Ejemplo 6 Sintaxis general de las referencias a los objetos del formulario	12
Ejemplo 7 Formulario con un <code>checkbox</code>	16
Ejemplo 8 Propiedad <code>checked</code> (ver ejemplo en la carpeta Ejemplo casilla)	16
Ejemplo 9 Botones de opción	17
Ejemplo 10 Propiedad <code>selectedIndex</code> del objeto <code>select</code>	18
Ejemplo 11 Propiedad <code>text</code> y <code>value</code> del objeto <code>select</code>	18
Ejemplo 12 Uso de la palabra <code>this</code>	22
Ejemplo 13 Uso de la palabra reservada <code>this</code>	22
Ejemplo 14 Detectar el navegador usado	23
Ejemplo 15 Registro de eventos en línea	24
Ejemplo 16 Registro de eventos en línea. Otro modo	24
Ejemplo 17 Anulación de la acción por defecto de un evento en línea	24
Ejemplo 18 Anulación de la acción por defecto de un evento en línea usando <code>confirm()</code>	25
Ejemplo 19 Registro de eventos tradicional	26
Ejemplo 20 Anulación de un gestor de eventos tradicional	26
Ejemplo 21 Disparo del evento tradicional de forma manual	26
Ejemplo 22 Uso de <code>this</code> en el gestor de eventos en línea	26
Ejemplo 23 Uso de <code>this</code> en el modelo de gestor de eventos tradicional	26
Ejemplo 24 Programando un evento multifunción	27
Ejemplo 25 Sintaxis del método <code>addEventListener</code>	28
Ejemplo 26 Método <code>addEventListener</code>	28
Ejemplo 27 Método <code>addEventListener</code> para añadir múltiples eventos	28
Ejemplo 28 Método <code>addEventListener</code> usando función anónima	28
Ejemplo 29 Sintaxis del método <code>removeEventListener</code>	29
Ejemplo 30 Sintaxis del método <code>attachEvent()</code>	30
Ejemplo 31 Sintaxis del método <code>detachEvent()</code>	30
Ejemplo 32 Comprobación de si admite el <code>Web Storage</code>	41
Ejemplo 33 Almacenar un nombre con <code>localStorage</code>	42
Ejemplo 34 Visualizar un nombre almacenado con <code>localStorage</code>	43

ÍNDICE DE RECOMENDACIÓN

<i>Recomendación 1 Propagación de eventos</i>	33
<i>Recomendación 2 Formulario moderno</i>	35
<i>Recomendación 3 LSSI y guía de uso de cookies</i>	36
<i>Recomendación 4 Gestión de cookies con w3schools</i>	38
<i>Recomendación 5 Cookies según Wikipedia</i>	38
<i>Recomendación 6 Política de Cookies de diferentes sitios web</i>	38
<i>Recomendación 7 Política de Cookies y descarga de código de ejemplo</i>	38
<i>Recomendación 8 Haz tu propio consentimiento informado</i>	38
<i>Recomendación 9 Especificación de Web Storage (recomendación del W3C desde el 19/04/2016)</i>	41
<i>Recomendación 10 Uso de sessionStorage en W3Schools</i>	41
<i>Recomendación 11 Uso de localStorage en W3Schools</i>	41
<i>Recomendación 12 API WebStorage</i>	42

ÍNDICE DE EJERCICIOS

<i>Ejercicio 1 Operaciones con formularios que incluyen label, select, radio, textbox (resuelto)</i>	31
<i>Ejercicio 2 Generar combinaciones de juegos de azar con eventos (resuelto)</i>	31
<i>Ejercicio 3 Validación de formulario a la antigua usanza usando expresiones regulares (resuelto)</i>	35

UNIDAD DIDÁCTICA 5

GESTIÓN DE EVENTOS Y FORMULARIOS EN JAVASCRIPT.

1. EL OBJETO FORM.

La mayor parte de interactividad entre una página web y el usuario tiene lugar en el momento en que un usuario usa un formulario. Es en ellos donde nos vamos a encontrar con los campos de texto, los botones, las casillas de verificación, las listas desplegables, etc., en los que el usuario introducirá los datos, que luego se enviarán al servidor para su procesamiento.

En este apartado verás cómo identificar un formulario y sus objetos, cómo modificarlos, cómo examinar las entradas de usuario, cómo validar los datos y cómo enviar el formulario al terminar todo el proceso.

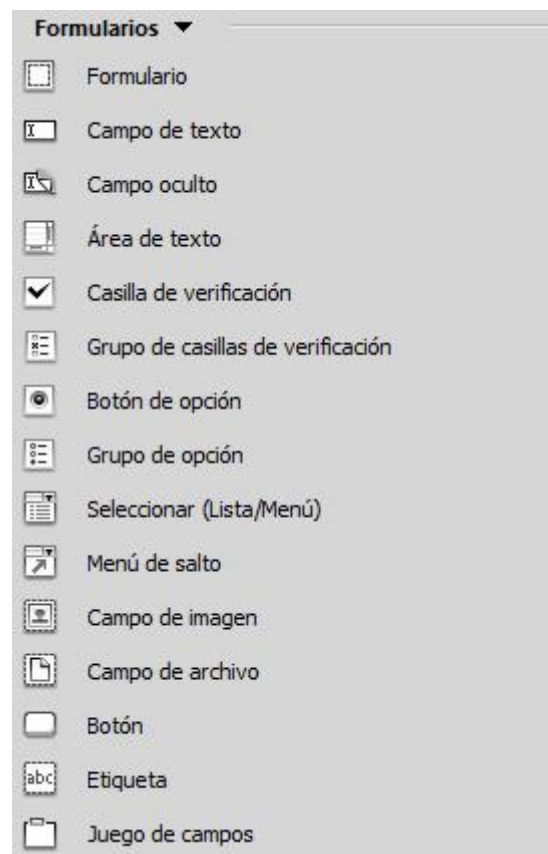
Los formularios y sus controles son objetos del DOM que tienen propiedades únicas que otros objetos no poseen. Por ejemplo, un formulario tiene una propiedad `action`, que le indica al navegador a dónde tiene que enviar los datos introducidos por el usuario cuando este envía el formulario (usando un botón enviar). Un control `select` posee una propiedad llamada `selectedIndex` que indica la opción del control que ha sido seleccionada por el usuario. Y mucho más....

JavaScript añade a los formularios dos características muy interesantes:

- JavaScript permite examinar y validar las entradas de usuario directamente, en el lado del cliente.
- JavaScript permite enviar mensajes al usuario de forma instantánea con información relativa a los datos que ha introducido.

El objeto `form` de JavaScript, es una propiedad del objeto `document`. Se corresponderá con la etiqueta `<form>` del HTML. Un formulario podrá ser enviado llamando al método `submit` de JavaScript, o bien haciendo clic en el botón `submit` (etiqueta `<input type="submit" ...>`) del formulario.

En la imagen de la derecha puedes ver un ejemplo de una lista de los objetos utilizados al trabajar con formularios. En este caso, se muestra una captura de la aplicación



1.1 FORMAS DE SELECCIÓN DEL OBJETO FORM.

```
...
<div id="menulateral">
  <form id="contactar" name="contactar" action="...">
    ...
  </form>
</div>
...
```

Existen varias formas de acceder al objeto **Form** dentro del documento:

A través del método `getElementById` del DOM

Esta forma nos permite acceder a un objeto a través de su atributo ID. Debes recordar que los nombres asignados a la propiedad `id` de un objeto del DOM deben ser únicos dentro de un documento HTML.

```
var formulario=document.getElementById("contactar");
```

EJEMPLO 1 MÉTODO GETELEMENTBYID

A través del método `getElementsByTagName` del DOM

Esta forma nos permite acceder a un objeto a través de la etiqueta HTML que queramos. Por ejemplo para acceder a los objetos con etiqueta `form` haremos:

```
var formularios = document.getElementsByTagName("form");
var primerFormulario = formularios[0]; // primer formulario del documento

// también se podría hacer lo anterior en una única línea
var primerFormulario = document.getElementsByTagName("form")[0];
```

EJEMPLO 2 MÉTODO GETELEMENTBYTAGNAME

Este método crea una colección de objetos con dicha etiqueta (ya que puede haber más de uno) y empleamos los índices para acceder a los elementos de la colección. En el caso de que el HTML contenga un único formulario la colección tendrá un único objeto al que se accede con el índice 0.

A través de la colección `forms[]` del objeto `document`.

Esta forma nos permite acceder a la colección de objetos formulario de un documento. Dicha colección es un array, que contiene una referencia de todos los formularios que tenemos en nuestro documento.

```
var formularios = document.forms; //referencia a todos los formularios del documento
var primerFormulario = formularios[0]; //primer formulario del documento
//también se podría hacer lo anterior en una única línea
var primerFormulario = document.forms[0];
//también se podría acceder a algún formulario a través de su nombre
var formularioContacto = formularios["contactar"];
```

EJEMPLO 3 MÉTODO DOCUMENT.FORMS

1.2 EL FORMULARIO COMO OBJETO Y CONTENEDOR.

Debido a que el DOM ha ido evolucionando con las nuevas versiones de JavaScript, nos encontramos con que el objeto `form` está dentro de dos árboles al mismo tiempo. En las nuevas definiciones del DOM, se especifica que `form` es el padre de todos sus nodos hijos, incluidos objetos y textos, mientras que en las versiones antiguas `form` sólo era padre de sus objetos (`input`, `select`, `button` y elementos `textarea`).

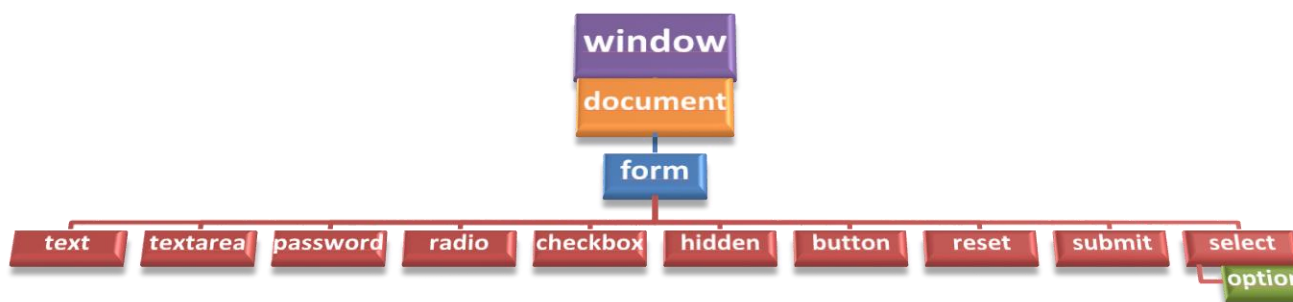


ILUSTRACIÓN 1 FORM EN LA JERARQUÍA DE OBJETOS DEL DOM (ÁRBOL DE NIVEL 0)

Supongamos el siguiente trozo de código HTML:

```
<form action="buscar.php" name="elFormulario" id="miFormulario" method="post">
  <p>
    <label for="busqueda">Buscar por:</label>
    <input id="busqueda" name="busqueda" type="text" value="">
    <input id="submit" type="submit" value="Buscar">
  </p>
</form>
```

El árbol de nivel 0 que representa dicho formulario sería el mostrado en la figura de la derecha, mientras que el árbol que representa el nivel 2 para el mismo formulario sería el que muestra la imagen de abajo. Estos árboles pueden ser útiles para diferentes propósitos. El árbol del DOM de nivel 2, se puede utilizar para leer y escribir en todo el documento con un nivel muy fino de *granularidad*. El árbol del DOM de nivel 0, hace muchísimo más fácil leer y escribir los controles del formulario.

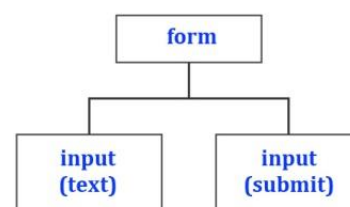
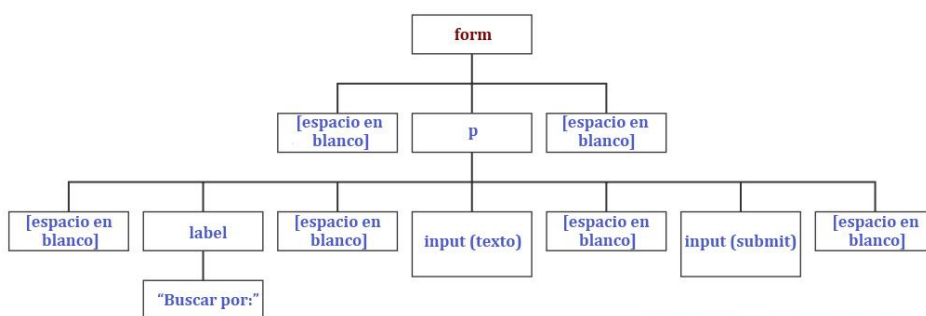


ILUSTRACIÓN 2 ÁRBOL DE NIVEL 0 PARA EL FORMULARIO DEL EJEMPLO



Árbol de Nivel 2 del DOM en JavaScript

ILUSTRACIÓN 3 FORM EN LA JERARQUÍA DE OBJETOS DEL DOM (ÁRBOL DE NIVEL 2)

1.3 ACCESO A PROPIEDADES Y MÉTODOS DEL FORMULARIO.

Los formularios pueden ser creados usando las etiquetas HTML o utilizando JavaScript y los métodos del DOM. En cualquier caso, se pueden asignar atributos como **name**, **action**, **target** y **enctype**. Cada uno de estos atributos es una propiedad del objeto **form**, a la que podemos acceder utilizando su nombre en minúsculas, por ejemplo:

```
var paginaDestino = objetoFormulario.action; //obtenemos el valor del atributo action
```

Para modificar una de estas propiedades lo haremos mediante asignaciones, por ejemplo:

```
objetoFormulario.action = "http://www.educacion.gob.es/recepcion.php";
```

Podemos usar referencias a objetos para hacer lo mismo:

```
var paginaDestino = document.getElementById("id").action; //o también
document.forms[0].action = "http://www.educacion.gob.es/recepcion.php";
```

Propiedades del objeto form	
Propiedad	Descripción
acceptCharset	Establece o retorna el valor del atributo accept-charset . Lista (delimitada por espacios) de codificación de caracteres aceptada por el servidor.
action	Establece o retorna el valor del atributo action . La URI del programa que procesa el envío por medio del formulario.
autocomplete (*)	Establece o retorna el valor del atributo autocomplete . Sus valores pueden ser: off y on. Para que los controles del formulario puedan ser completados de forma automática,
enctype encoding (alias)	Establece o retorna el valor del atributo enctype : (como se codificarán los datos a enviar al servidor) <ul style="list-style-type: none">• application/x-www-form-urlencoded (es el valor por defecto, los espacios se sustituyen por + y los caracteres especiales a sus valores ASCII hexadecimales)• multipart/form-data (se usa cuando hay algún input type="file")• text/plain (solo se cambian los espacios por +)
length	Retorna el número de elementos en un formulario. Solo objetos de formulario (el label no)
method	Establece o retorna el valor del atributo method . (Sus valores pueden ser post o get)
name	Establece o retorna el valor del atributo name .
noValidate (*)	Establece o retorna el valor del atributo noValidate . (Sus valores son true o false) Si es true el navegador no realizará ningún tipo de validación sobre los campos del formulario: ignora los required, los type="email", etc.
target	Establece o retorna el valor del atributo target . (Sus valores pueden ser _blank, _top,...) Indica dónde se visualizará la respuesta dada por el servidor al enviar los datos del formulario.

TABLA 1 PROPIEDADES DEL OBJETO FORM

Métodos del objeto form	
Método	Descripción
reset()	Inicializa todos los objetos del formulario con sus valores por defecto.
submit()	Envía los datos del formulario a la URI especificada en la propiedad action.

TABLA 2 MÉTODOS DEL OBJETO FORM

Colecciones del objeto form	
Colección	Descripción
elements	Colección de elementos de un formulario. (No se tienen en cuenta los label)

TABLA 3 COLECCIONES DEL OBJETO FORM

Las propiedades marcadas con un () han sido añadidas con el HTML5.*

1.4 COLECCIÓN FORM.ELEMENTS

elements es una colección que contiene todos los objetos **input** que existen en el formulario. Esta colección es un array de objetos de formulario ordenados por su aparición en el código fuente del documento HTML.

Al igual que todos los demás objetos de JavaScript, **elements** tiene sus propiedades y métodos:

Propiedad de la colección elements	
Propiedad	Descripción
length	Retorna el número de elementos en un formulario.

TABLA 4 PROPIEDADES DE LA COLECCIÓN ELEMENTS

Métodos de la colección elements	
Método	Descripción
[índice] item(índice)	Retorna el elemento de la colección con dicho índice .
namedItem(id)	Retorna el elemento de la colección con dicho id .

TABLA 5 MÉTODOS DE LA COLECCIÓN ELEMENTS

Generalmente, es mucho más eficaz y rápido referenciar a un elemento individual usando su identificador (atributo **id**). En el ejemplo siguiente, empleamos la colección **elements** para hacer un bucle que recorra un formulario y poner en blanco aquellos campos (objetos **input**) que sean de tipo texto:

```
var miFormulario = document.getElementById("contactar");
if (miFormulario){
    // Si existe ese formulario continúa
    for (var i=0; i< miFormulario.elements.length; i++) {
        if (miFormulario.elements[i].type == "text") {
            miFormulario.elements[i].value = "";
        }
    }
}
```

EJEMPLO 4 LIMPIEZA DE LOS INPUT TYPE TEXT DE LA COLECCIÓN ELEMENTS DE UN FORMULARIO

2. OBJETOS RELACIONADOS CON FORMULARIOS.

Para poder trabajar con los objetos de un formulario es necesario tener una referencia al objeto. Eso se puede conseguir a través del atributo `id` del objeto, o bien con el nombre asignado a la etiqueta, empleando para ello los métodos del DOM nivel 2. También se puede hacer usando la sintaxis del DOM nivel 0 construyendo el camino que comienza por `document`, luego el formulario y finalmente el control.

Lo forma más cómoda es identificar cada uno de los objetos con un atributo `id` (que es único) al que accederemos con el método `getElementById` del objeto `document`.

Vemos un ejemplo sencillo de un formulario acompañado de una serie de referencias a sus campos de entrada (todas ellas válidas):

```
<form name="formularioBusqueda" action="cgi-bin/buscar.pl">
  <p>
    <input type="text" id="entrada" name="cEntrada">
    <input type="submit" id="enviar" name="enviar" value="Buscar...">
  </p>
</form>

//referencia al objeto <input type="text" id="entrada" name="cEntrada"> del
formulario
document.getElementById("entrada");
document.formularioBusqueda.cEntrada;
document.formularioBusqueda.elements[0];
document.forms["formularioBusqueda"].elements["cEntrada"];
document.forms["formularioBusqueda"].cEntrada;
```

EJEMPLO 5 REFERENCIAS A UN OBJETO DEL FORMULARIO

A la vista del ejemplo anterior podemos generalizar la siguiente sintaxis:

```
//Usando el índice numérico correspondiente del array de elementos
document.forms[i].elements[k]
//Usando el nombre del elemento (atributo name)
document.nombreFormulario.nombreElemento;
//Usando como índice el nombre del elemento (atributo name)
document.forms["nombreFormulario"].elements["nombreElemento"];
//Usando una mezcla de los tres sistemas anteriores
document.nombreFormulario.elements[k];
document.forms[i].nombreElemento; //etcétera...
```

EJEMPLO 6 SINTAXIS GENERAL DE LAS REFERENCIAS A LOS OBJETOS DEL FORMULARIO

Aunque muchos de los controles de un formulario tienen propiedades en común, algunas propiedades son únicas a un control en particular. Por ejemplo: un objeto `select` tiene una propiedad que permite conocer la opción que está actualmente seleccionada, lo mismo ocurre con los objetos tipo `checkbox` o con los botones de tipo `radio`. En los siguientes apartados se estudiarán todos estos objetos.

2.1 OBJETOS INPUT DE TIPO TEXTO Y OTROS.

Hasta la llegada del HTML5 había cuatro elementos de tipo texto dentro de la jerarquía de objetos de un formulario:

- `text`
- `password`
- `hidden`
- `textarea`

Todos los elementos, a excepción del tipo `hidden`, se muestran en la página permitiendo a los usuarios introducir texto.

Para poder usar estos objetos dentro de nuestros scripts será suficiente con asignar un atributo `id` a cada uno de los elementos. Te recomiendo que asignes a cada objeto un atributo `id` único que coincida con el `name` del objeto.

Cuando se envían los datos de un formulario a un programa en el lado del servidor, lo que en realidad se envía son los atributos `name` junto con los valores (contenido del atributo `value`) de cada elemento. Sin lugar a dudas, la propiedad más utilizada en un elemento de tipo texto es, por lo tanto, `value`. Un script podrá recuperar y ajustar el contenido de la propiedad `value` en cualquier momento.

El contenido de la propiedad `value` es siempre una cadena de texto, y quizás sea necesario realizar conversiones numéricas para realizar operaciones matemáticas con esos textos.

En este tipo de objetos, los eventos (cuya introducción verás más adelante y se desarrollan en el tema siguiente) se podrán **disparar** de múltiples formas como, por ejemplo:

- Al poner el foco en un campo (situar el cursor dentro de ese campo).
- Al modificar el texto (al cambiar el contenido de un texto).
- Al sacar el foco de un campo (cuando el cursor sale de él), etc.

Con la llegada de HTML5 se han incorporado nuevos tipos de objetos input. En la Ilustración 4 mostrada debajo de este párrafo se pueden ver todos los tipos de objetos input soportados por HTML5 el 29/12/2017:

1 <code><input> button</code>	7 <code><input> email</code>	13 <code><input> password</code>	19 <code><input> text</code>
2 <code><input> checkbox</code>	8 <code><input> file</code>	14 <code><input> radio</code>	20 <code><input> time</code>
3 <code><input> color</code>	9 <code><input> hidden</code>	15 <code><input> range</code>	21 <code><input> url</code>
4 <code><input> date</code>	10 <code><input> image</code>	16 <code><input> reset</code>	22 <code><input> week</code>
5 <code><input> datetime</code>	11 <code><input> month</code>	17 <code><input> search</code>	
6 <code><input> datetime-local</code>	12 <code><input> number</code>	18 <code><input> submit</code>	

ILUSTRACIÓN 4 TIPOS DE OBJETOS INPUT QUE SOPORTA EL HTML5 A 29/12/2017

En las tablas 6 y 7 se hace referencia a las propiedades y métodos de estos 22 tipos de objetos `input`. Las propiedades marcadas con un (*) han sido añadidas con el HTML5.

Hay propiedades que son comunes a distintos tipos de objetos `input`. En cada propiedad se ha dejado un enlace a la documentación proporcionada por w3schools sobre esta propiedad aplicada a un tipo de objeto concreto (así se puede comprobar cómo funciona). En algunas propiedades se deja un enlace a la documentación del W3C (en inglés).

A continuación, en la tabla 8, se indica qué tipo de objeto tiene cada propiedad y cada método. Los números de las columnas hacen referencia al tipo de objeto según la numeración reflejada en la Ilustración 4 de esta misma página. En la Ilustración 4 se deja un enlace a la documentación de cada tipo de objeto proporcionada por la página de w3schools para cada tipo de objeto `input`. (El enlace está en el área que ocupa el número y el tipo de objeto.)

Propiedades de los objetos input	
Propiedad	Descripción
accept	Establece o retorna el valor del atributo accept del botón de upload de un fichero.
alt	Establece o retorna el valor del atributo alt de un objeto input image .
autocomplete (*)	Establece o retorna el valor del atributo autocomplete del campo de texto.
autofocus (*)	Establece o retorna si un campo de texto obtendrá el foco de forma automática cuando se carga la página.
checked	Establece o retorna el estado de un checkbox.
defaultChecked	Retorna el valor por defecto del atributo checked.
defaultValue	Establece o retorna el valor por defecto del objeto.
disabled	Establece o retorna si el objeto está o no desactivado.
files	Retorna un objeto FileList que representa el/los ficheros seleccionados.
form	Retorna la referencia al formulario que contiene ese objeto.
formAction (*)	Establece o retorna el valor del atributo formaction .
formEnctype (*)	Establece o retorna el valor del atributo formenctype .
formNoValidate (*)	Establece o retorna si el formulario será validado antes de su envío. (para el input type="submit")
formTarget (*)	Establece o retorna el valor del atributo formtarget .
height (*)	Establece o retorna el valor del atributo height . (para el input type="image")
list Otro enlace	Retorna una referencia al datalist que contiene la lista de opciones a mostrar en un input type="text".
indeterminate (*)	Establece o retorna el estado indeterminate del checkbox. 
max	Establece o retorna el valor del atributo max . (para los inputs de fecha, tiempo y numéricos)
min	Establece o retorna el valor del atributo min .
maxLength	Establece o retorna la longitud máxima de caracteres permitidos en el campo de tipo texto.
multiple (*)	Establece o retorna si el usuario tiene permitido introducir más de una dirección en un input type="email".
name	Establece o retorna el valor del atributo name .
pattern (*)	Establece o retorna el valor del atributo pattern .
placeholder (*)	Establece o retorna el valor del atributo placeholder . (texto visualizado que será sustituido por lo que introduzca el usuario)
readOnly	Establece o retorna si un campo es o no de sólo lectura.
required (*)	Establece o retorna si un campo de texto debe tener obligatoriamente contenido antes de hacer el envío del formulario.
size	Establece o retorna el valor del atributo size .
src	Establece o retorna el valor del atributo src de un objeto input image .
step	Establece o retorna el valor del atributo step .
type	Retorna cuál es el tipo de elemento.
value	Establece o retorna el valor del atributo value .
width (*)	Establece o retorna el valor del atributo width de un objeto input image .

TABLA 6 PROPIEDADES DE LOS OBJETOS INPUT

Métodos del objeto input de tipo texto	
Método	Descripción
select()	Selecciona el contenido de un campo.
stepDown()	Decrementa el valor del campo en el número especificado.
stepUp()	Incrementa el valor del campo en el número especificado.

TABLA 7 MÉTODOS DE LOS OBJETOS INPUT

Tabla resumen de propiedades de los objetos input																						
Propiedad	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
accept								x														
alt										x												
autocomplete			x	x	x	x	x				x	x	x		x		x		x	x	x	x
autofocus	x	x	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x	x
checked		x												x								
defaultChecked		x												x								
defaultValue	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
disabled	x	x	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x	x
files								x														
form	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
formAction										x								x				
formEnctype										x								x				
formNoValidate										x								x				
formTarget										x								x				
height										x												
list			x	x	x	x	x				x	x			x		x		x	x	x	x
indeterminate		x																				
max				x	x	x					x	x			x					x		x
min				x	x	x					x	x			x					x		x
maxLength							x						x				x		x		x	
multiple							x	x														
name	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
pattern							x						x				x		x		x	
placeholder							x					x	x				x		x		x	
readOnly				x	x	x	x				x	x	x				x		x	x	x	x
required		x		x	x	x	x	x			x	x	x				x		x	x	x	x
size							x						x				x		x		x	
src										x												
step				x	x	x					x	x			x					x		x
type	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
width										x												
Método	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
select()		x	x	x	x	x	x				x	x	x	x	x		x		x	x	x	x
stepDown()				x	x	x					x	x			x					x		x
stepUp()				x	x	x					x	x			x					x		x

TABLA 8 PROPIEDADES Y MÉTODOS DE LOS OBJETOS INPUT

2.2 OBJETO INPUT DE TIPO CHECKBOX.

Un **checkbox** es también un objeto muy utilizado en los formularios en el que algunas de sus propiedades no son muy intuitivas.

En los botones de un formulario la propiedad **value** contiene el texto que se muestra en el botón.

Los objetos de texto que vimos en el apartado anterior contienen (en la propiedad **value**) el valor que se muestra dentro de la caja inicialmente y este valor se modifica con el texto escrito por el usuario.

En el caso de un **checkbox**, la propiedad **value** es un texto que está asociado al objeto. Este texto no es visible en la página y su finalidad es la de asociar un valor concreto a cada opción seleccionada. Dichos valores serán los que se enviarán al servidor con los demás valores del formulario.

```
<form>
  <label for="cantidad">Si desea recibir 20 Kg marque esta opción: </label>
  <input type="checkbox" id="cantidad" name="cantidad" value="20 Kg">
</form>
```

EJEMPLO 7 FORMULARIO CON UN CHECKBOX

Si chequeamos la casilla del ejemplo anterior (**checkbox**) y enviamos el formulario, el navegador enviará el par **name=value** (cantidad="20 Kg"). **Si el checkbox no está marcado, entonces este campo no será enviado con el formulario.** El texto del **label** se muestra en la pantalla, pero no se envía al servidor.

Para saber si un campo de tipo **checkbox** está o no marcado, disponemos de la propiedad **checked**. Esta propiedad contiene un valor booleano: **true** si el campo está marcado y **false** si no lo está. Esta propiedad es de lectura y escritura, es decir, podemos cambiar su valor dentro de un script, tal y como podrás comprobar si ejecutas el ejemplo 8 (en el que se han añadido eventos en línea y código JavaScript incrustado por comodidad).

Además de la propiedad **checked** y las marcadas con un (+) en la tabla 3, este objeto tiene la propiedad **defaultChecked** que devuelve el valor por defecto establecido en el atributo **checked** del HTML.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <title>DWE05 - Trabajando con un objeto checkbox</title>
    <script type="text/javascript">
      function marcar() {
        document.getElementById("verano").checked=true;
      }
      function desmarcar() {
        document.getElementById("verano").checked=false;
      }
    </script>
  </head>
  <body>
    <form action="" method="post">
      <input type="checkbox" id="verano" name="verano" value="Sí">
      <label for="verano">¿Te gusta el verano?</label>
      <input type="submit">
    </form>
    <button onclick="marcar()">SÍ</button>
    <button onclick="desmarcar()">NO</button>
  </body>
</html>
```

EJEMPLO 8 PROPIEDAD CHECKED (VER EJEMPLO EN LA CARPETA EJEMPLO CASILLA)

2.3 OBJETO INPUT DE TIPO RADIO.

Trabajar con objetos de tipo **radio** con JavaScript requiere un poco más de trabajo.

Para que el navegador pueda gestionar un grupo de objetos de tipo **radio**, debemos asignar el mismo atributo **name** a cada uno de los botones del grupo.

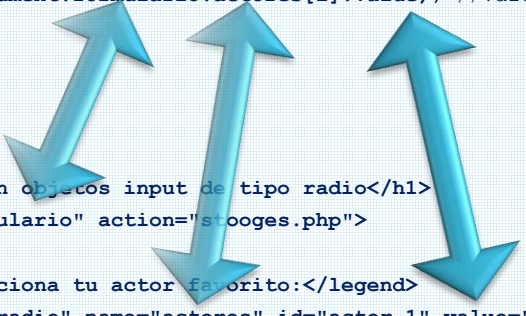
En un formulario podemos tener más de un grupo de botones de tipo **radio**, pero los miembros de cada grupo tienen que tener el mismo atributo **name**.

Cuando asignamos el mismo **name** a varios elementos en un formulario, el navegador lo que hace es crear un array con la lista de los objetos con ese nombre. El valor del atributo **name** será el nombre del array.

Hay propiedades que se pueden aplicar al grupo en su conjunto y hay propiedades que se pueden aplicar a cada elemento del grupo empleando un índice para hacer referencia a cada elemento dentro del array.

Por ejemplo: podemos saber cuántos botones hay en un grupo **radio** consultando la propiedad **length** de ese grupo y podemos acceder a la propiedad **checked** de un botón a través de la posición que ocupa ese botón en particular en el array (en el ejemplo se han utilizado eventos en línea y código JavaScript incrustado por comodidad).

```
...
<script type="text/javascript">
  function mostrarDatos(){
    for (var i=0;i<document.formulario.actores.length; i++) {
      if (document.formulario.actores[i].checked) { //saber si un botón está seleccionado
        alert(document.formulario.actores[i].value); //valor asociado al botón
      }
    }
  }
</script>
...
<h1>Trabajando con objetos input de tipo radio</h1>
<form name="formulario" action="stoooges.php">
  <fieldset>
    <legend>Selecciona tu actor favorito:</legend>
    <input type="radio" name="actores" id="actor-1" value="BW" checked="checked">
    <label for="actor-1">Bruce Willis</label>
    <input type="radio" name="actores" id="actor-2" value="JC">
    <label for="actor-2">Jim Carrey</label>
    <input type="radio" name="actores" id="actor-3" value="MG">
    <label for="actor-3">Mel Gibson</label>
    <input type="button" id="ver" name="ver" value="Consultar" onclick="mostrarDatos()">
  </fieldset>
</form>
...
```



EJEMPLO 9 BOTONES DE OPCIÓN

Además de la propiedad **checked**, los objetos de este tipo (al igual que ocurría con los de tipo **checkbox**), disponen de la propiedad **defaultChecked** que devuelve el valor por defecto establecido en el atributo **checked** del HTML.

2.4 OBJETO SELECT.

Uno de los controles más complejos que podrás encontrar en los formularios es el objeto `select`. Un objeto `select` está compuesto realmente de un array de objetos `option`. El objeto `select` se suele mostrar como una lista desplegable en la que puedes seleccionar una de las opciones. También existe la opción de realizar selecciones múltiples; todo depende de cómo se defina el objeto en el documento.

Vamos a ver cómo gestionar una lista que permita realizar una selección simple.

Algunas propiedades pertenecen al objeto `select` en general, mientras que otras, sólo se pueden aplicar a sus opciones. Podemos detectar la opción seleccionada por el usuario usando la propiedad `selectedIndex` propia de este objeto, lo cual se puede hacer de las siguientes formas:

```
objetoFormulario.nombreCampoSelect.selectedIndex;  
document.getElementById("idObjetoSelect").selectedIndex;
```

EJEMPLO 10 PROPIEDAD SELECTEDINDEX DEL OBJETO SELECT

El valor devuelto por la propiedad `selectedIndex` es el **índice** de la opción actualmente seleccionada. **Ten en cuenta que los índices comienzan siempre en la posición 0.**

Las opciones tienen dos propiedades: `text` y `value`, que permiten acceder al texto visible en la lista de selección (`text`) y al valor interno asociado a cada opción (`value`). El ejemplo siguiente muestra cómo acceder a estas dos propiedades (se ha puesto el script incrustado en el HTML y un evento en línea para una mejor visualización del ejemplo).

```
<!DOCTYPE html>  
<html lang="es">  
  <head>  
    <meta http-equiv="content-type" content="text/html; charset=utf-8">  
    <title>DWE05 - Trabajando con un objeto Select</title>  
    <script type="text/javascript">  
      function ver() {  
        var objProvincias=document.getElementById("provincias");  
        var texto=objProvincias.options[objProvincias.selectedIndex].text;  
        var valor=objProvincias.options[objProvincias.selectedIndex].value;  
        alert("Datos opción seleccionada:\n\nTexto: "+texto+"\nValor: "+valor);  
      }  
    </script>  
  </head>  
  <body>  
    <h1>Trabajando con un objeto Select</h1>  
    <form id="formulario" action="pagina.php">  
      <p>  
        <label for="provincias">Seleccione provincia: </label>  
        <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>  
      </p>  
      Selecciona una opción y pulsa el botón.  
      <input type="button" name="boton" value="Consultar opción" onclick="ver()" />  
    </form>  
  </body>  
</html>
```

EJEMPLO 11 PROPIEDAD TEXT Y VALUE DEL OBJETO SELECT

Como ya mencionamos antes, el objeto `select` es uno de los objetos más complejos que podemos manejar con JavaScript. Dispone de propiedades (algunas de ellas son colecciones con sus propias propiedades y métodos) y de métodos que nos permiten añadir y/o borrar opciones de la lista.

Propiedades del objeto select	
Propiedad	Descripción
<code>autofocus</code> (*)	Establece o retorna si el objeto <code>select</code> recibirá el foco cuando se cargue la página.
<code>disabled</code>	Establece o retorna si el objeto <code>select</code> está o no desactivado.
<code>form</code> (*)	Devuelve la referencia al formulario que contiene el objeto <code>select</code> .
<code>length</code>	Devuelve el número de opciones que hay en el objeto <code>select</code> .
<code>multiple</code>	Establece o retorna el valor del atributo <code>multiple</code> de un objeto <code>select</code> .
<code>name</code>	Establece o retorna el valor del atributo <code>name</code> de un objeto <code>select</code> .
<code>required</code> (*)	Establece o retorna el valor del atributo <code>required</code> de un objeto <code>select</code> (que indica si es obligatorio elegir al menos una opción).
<code>selectedIndex</code>	Establece o retorna el índice del elemento seleccionado en un objeto <code>select</code> .
<code>size</code>	Establece o retorna el número de elementos visibles en un objeto <code>select</code> . El valor por defecto es 0.
<code>type</code>	Devuelve el tipo de selección que se puede realizar (<code>select-one</code> o <code>select-multiple</code>).
<code>value</code>	Establece o retorna el valor del elemento seleccionado en un objeto <code>select</code> .

TABLA 9 PROPIEDADES DEL OBJETO SELECT

Colecciones del objeto select	
Colección	Descripción
<code>labels</code>	Devuelve el conjunto de elementos <code><label></code> asociados a un control
<code>options</code>	Devuelve el conjunto de opciones (elementos <code><option></code>).
<code>selectedOptions</code>	Devuelve el conjunto de opciones seleccionadas. (cuando el objeto <code>select</code> tiene el atributo <code>multiple</code>)

TABLA 10 COLECCIONES DEL OBJETO SELECT

Métodos del objeto select y su colección options	
Método	Descripción
<code>add(opción[, índice])</code>	Añade una <code>opción</code> a la lista en la posición indicada por <code>índice</code> . Si se omite <code>índice</code> se añade al final.
<code>item(índice)</code> <code>[índice]</code>	Accede directamente a la opción que ocupa la posición indicada por <code>índice</code> .
<code>namedItem("id")</code> <code>["id"]</code>	Accede directamente a la opción con un identificador con el valor especificado por <code>id</code> .
<code>remove(índice)</code>	Elimina la opción de la posición <code>índice</code> en la lista de opciones del objeto <code><select></code> .

TABLA 11 MÉTODOS DEL OBJETO SELECT

Propiedades del objeto <option>	
Propiedad	Descripción
<code>defaultSelected</code>	Establece o retorna si una determinada opción está seleccionada por defecto.
<code>disabled</code>	Establece o retorna si la opción está o no desactivado.
<code>form</code>	Devuelve la referencia al formulario que contiene esa opción.
<code>index</code>	Establece o retorna el valor del índice de una opción de la lista.
<code>label</code>	Establece o retorna el valor de la etiqueta <code>label</code> asociada a un objeto <code>option</code> . Si no hay etiqueta devuelve lo mismo que la propiedad <code>text</code> . Si hay un atributo <code>label</code> asociado a una opción lo que se verá en la lista desplegable será el valor asignado a <code>label</code> .
<code>selected</code>	Establece o retorna el valor del atributo <code>selected</code> de un objeto <code>option</code> .
<code>text</code>	Establece o retorna el texto contenido en un objeto <code>option</code> .
<code>value</code>	Establece o retorna el contenido del atributo <code>value</code> de un objeto <code>option</code> . En caso de no existir devuelve lo mismo que la propiedad <code>text</code> .

TABLA 12 PROPIEDADES DEL OBJETO OPTION

Métodos del objeto option	
Método	Descripción
Option()	<p>Es un constructor de nuevas opciones para el objeto <code>select</code>. Añade un objeto <code>option</code>. Sintaxis: <code>new Option([text[,value[,defaultSelected[,selected]]]])</code> donde:</p> <ul style="list-style-type: none"> • <code>text</code> = "texto" "" • <code>value</code> = "valor" nada (le asigna lo que haya en text) • <code>defaultSelected</code> = true false nada (si se omite = false) • <code>selected</code> = true false nada (si se omite = false)

TABLA 13 MÉTODOS DEL OBJETO OPTION

PARA SABER MÁS

Desde este otro enlace accederás directamente a los ejemplos del uso de este objeto dentro del tutorial de JavaScript de InformaticaPC.com.

[InformaticaPC.com - Objeto Select](#)

PARA SABER MÁS 1 EJEMPLO DE USO DEL OBJETO SELECT

PARA SABER MÁS

Desde este otro enlace accederás a toda la documentación actualizada el 15 de enero de 2021 sobre todo el tema relacionado con los formularios.

Son muchos los temas que se quedan sin ver y que os pueden resultar interesantes: Las restricciones de validación es un tema muy extenso que se desarrolla a partir del punto 4.10.20.

Está en inglés.

En el punto 4.10.7 está el objeto `select`. Una de sus propiedades `length` es de lectura y escritura con lo que se pueden eliminar opciones con solo reducir el valor de esta propiedad.

[HTML Living Standard](#)

PARA SABER MÁS 2 DOCUMENTACIÓN COMPLETA DEL OBJETO FORMS

2.5 OBJETO TEXTAREA.

El objeto textarea representa el elemento <textarea> de un documento HTML.

Se puede acceder a él usando cualquiera de los métodos del documento getElement...

Propiedades de los objetos input	
Propiedad	Descripción
autofocus (*)	Establece o retorna si un campo textarea el foco de forma automática cuando se carga la página.
cols	Establece o retorna el valor del atributo cols de un campo textarea. N° de columnas visibles.
defaultValue	Establece o retorna el valor por defecto del objeto.
disabled	Establece o retorna si el objeto está o no desactivado.
form	Retorna la referencia al formulario que contiene ese objeto.
maxLength (*)	Establece o retorna la longitud máxima de caracteres permitidos en el campo textarea (valor del atributo maxlength).
name	Establece o retorna el valor del atributo name .
placeholder (*)	Establece o retorna el valor del atributo placeholder .
readOnly	Establece o retorna si un campo es o no de sólo lectura.
required (*)	Establece o retorna si un campo de texto debe tener obligatoriamente contenido antes de hacer el envío del formulario.
rows	Establece o retorna el valor del atributo rows . (n° de filas visibles)
type	Retorna cuál es el tipo de elemento.
value	Establece o retorna el valor del atributo value .
wrap (*)	Establece o retorna el valor del atributo wrap de un objeto textarea . Solo afecta a los datos que son enviados con submit. Tiene dos modos: soft – El texto se envía tal cual hard – El texto se envía con saltos de línea adicionales en función del valor del atributo cols.

TABLA 14 PROPIEDADES DEL OBJETO TEXTAREA

Métodos del objeto textarea	
Método	Descripción
select()	Selecciona el contenido del textarea .

TABLA 15 MÉTODOS DEL OBJETO TEXTAREA

2.6 PASANDO OBJETOS A LAS FUNCIONES USANDO `this`.

En los ejemplos que hemos visto anteriormente, cuando un gestor de eventos (`onclick`, `onblur`,...) llama a una función, es la función la que se encarga de buscar el objeto sobre el cuál va a trabajar. En JavaScript disponemos de un método que nos permite llamar a una función, pasándole directamente la referencia del objeto, sin tener que usar variables globales o referenciar al objeto al comienzo de cada función.

Para conseguir hacerlo necesitamos usar la palabra reservada `this`. Por ejemplo, si tenemos un botón y queremos que al hacer clic sobre él se realice alguna operación, programamos una función para y llamamos a dicha función con el evento `onclick` dentro del HTML de definición del botón. Si dentro de esa función usamos la palabra `this` estaremos haciendo referencia al objeto en el cuál hemos hecho clic, que en este caso será el botón. El uso de `this` nos permite evitar usar variables globales y hacer scripts más genéricos.

En el siguiente ejemplo, cada vez que hagamos clic en alguno de los objetos del formulario, llamaremos a la función `ver()` pasándole `this` (referencia al propio objeto) como parámetro. La función `ver()` recibe ese parámetro y lo almacena en la variable `objeto`, accediendo a todas sus propiedades. En el siguiente apartado veremos los eventos y se mostrará otra forma de usar `this`. En este ejemplo se utiliza el modelo de eventos en línea (incrustando el evento en el HTML). También para una mejor visualización se ha desarrollado el ejemplo incrustando el código JavaScript en el código HTML.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8">
    <title>DWEC05 - Uso de la palabra reservada this</title>
    <script type="text/javascript">
      function ver(objeto){
        var atrName=objeto.name;
        var atrId=objeto.id;
        var atrValue=objeto.value;
        var atrType=objeto.type;
        var mensaje="Datos del objeto:\n\n";
        mensaje+="  name : " + atrName + "\n";
        mensaje+="  id   : " + atrId + "\n";
        mensaje+="  value: " + atrValue + "\n";
        mensaje+="  type : " + atrType;
        alert(mensaje);
      }
    </script>
  </head>
  <body>
    <h1>Trabajando con la palabra reservada this</h1>
    <form id="formulario" action="pagina.php">
      <p>
        <label for="nom">Nombre: </label>
        <input type="text" name="nom" id="nom" value="Ana" onclick="ver(this)"/>
        <label for="apel">Apellidos: </label>
        <input type="text" name="ape" id="ape" value="Paz" onclick="ver(this)"/>
        <label for="eda">Edad: </label>
        <input type="text" name="eda" id="eda" value="55" onclick="ver(this)"/>
        <label for="pais">Pais: </label>
        <input type="radio" name="pais" id="p1" value="ES" onclick="ver(this)"/>
        <label for="p1">España</label>
        <input type="radio" name="pais" id="p2" value="FR" onclick="ver(this)"/>
        <label for="p2">Francia</label>
      </p>
      Haga click en cada uno de los campos para ver más información.
    </form>
  </body>
</html>
```

EJEMPLO 12 USO DE LA PALABRA THIS

3. EVENTOS.

Hay que tener en cuenta que, sin eventos prácticamente no hay scripts. En casi todas las páginas web que incorporan JavaScript, suele haber eventos programados que disparan la ejecución de dichos scripts. La razón es muy simple, JavaScript fue diseñado para añadir interactividad a las páginas: el usuario realiza algo y la página reacciona. Por lo tanto, JavaScript necesita detectar de alguna forma las acciones del usuario para saber cuándo reaccionar. También necesita conocer las funciones a ejecutar al producirse esas acciones.

Cuando el usuario hace algo se produce un evento. También hay algunos eventos que no están relacionados directamente con acciones de usuario: por ejemplo, el evento `load` de un documento, que se produce automáticamente cuando un documento ha sido cargado.

Todo el tema de gestión de eventos se popularizó a raíz de la versión 2 de Netscape que soportaba algunos eventos. Los eventos `mouseover` y `mouseout` se hicieron muy famosos a raíz de su utilización para hacer el efecto de sustitución de una imagen por otra al pasar el ratón por encima. El resto de los navegadores, incluido el Internet Explorer, tuvieron que adaptarse a la forma en la que Netscape 2 y 3 gestionaban los eventos.

Aunque hoy en día la técnica de gestión de eventos varía con el objetivo de independizar el código de JavaScript de la estructura HTML, los navegadores todavía son compatibles con las técnicas utilizadas por Netscape.

Anteriormente a las versiones 4 de los navegadores, los eventos (interacciones del usuario con el sistema), eran capturados preferentemente por gestores de eventos definidos como atributos en las etiquetas HTML (modelo de eventos en línea). Por ejemplo, cuando un usuario hacía clic en un botón, se disparaba el evento `onclick` que se había programado en la etiqueta HTML. Ese evento hacía una llamada a una función en la que se realizaban las operaciones programadas por el usuario.

Aunque todo ese modelo de gestión de eventos sigue funcionando, los navegadores modernos incorporan un modelo de gestión de eventos que proporciona la información sobre cómo ocurre un evento. Esta información es accesible usando JavaScript, permitiendo programar respuestas más inteligentes a las interacciones del usuario con los objetos del documento.

Incompatibilidades entre navegadores

Muchas veces, lo que se hacía en un principio antes de programar cualquier evento, era detectar qué navegador estábamos utilizando, para saber si nuestro navegador soportaba o no, los métodos y propiedades que queríamos usar. Por ejemplo:

```
if (Netscape) {  
    //utilizar modelo Netscape  
} else if (Explorer) {  
    //utilizar modelo Microsoft  
}
```

EJEMPLO 14 DETECTAR EL NAVEGADOR USADO

Hoy en día, debido a las diferencias que hay entre todos los navegadores actuales, no se recomiendan ni esto, ni el modelo de detección basado en DHTML. Por lo tanto, hay que intentar usar modelos de detección de eventos estándar y que sean los navegadores los que tengan que adaptarse a ese modelo.

3.1 MODELO DE REGISTRO DE EVENTOS EN LÍNEA.

Aunque a estas alturas, este modelo de eventos ya los has visto en multitud de ejemplos, en el modelo de registro de eventos en línea (estandarizado por Netscape), el evento es añadido como un atributo más a la etiqueta HTML del objeto sobre el cual se quiere controlar un determinado suceso, como en el siguiente ejemplo en el que al hacer clic en el enlace, se llama al gestor de eventos `onClick` y se ejecuta el script, que contiene en este caso una alerta de JavaScript.

```
<a href="pagina.html" onClick="alert('Has pulsado en el enlace')">Pulsa aquí</a>
```

EJEMPLO 15 REGISTRO DE EVENTOS EN LÍNEA

Aunque quizás estés más familiarizado/a con la segunda forma de hacerlo donde hacemos lo mismo, pero llamando a una función:

```
<a href="pagina.html" onClick="alertar()">Pulsa aquí</a>
//siendo la función alertar() almacena en un archivo JavaScript
function alertar(){
    alert("Has pulsado en el enlace");
}
```

EJEMPLO 16 REGISTRO DE EVENTOS EN LÍNEA. OTRO MODO

La mezcla de minúsculas y mayúsculas en los nombres de evento (`onClick`, `onMouseOver`) es sólo por tradición. Recuerda que HTML no es sensible al uso de mayúsculas y/o minúsculas. En cambio, en XHTML, los atributos sí que tendrían que ir obligatoriamente en minúsculas: `onclick` y `onmouseover`.

No es recomendable usar el modelo de registro de eventos en línea.

Aunque lo has visto en los ejemplos que hemos utilizado hasta ahora, tiene el problema de que estamos mezclando la estructura de la página web (contenido) con la programación de la misma (comportamiento).

Hoy en día se intenta separar la programación en JavaScript de una página de su estructura HTML, al igual que intentamos separar lo que es contenido de lo que es comportamiento.

En el ejemplo anterior, cuando haces clic en el enlace se mostrará la alerta y a continuación te conectará con la página `pagina.html`. En ese momento desaparecerán de memoria los objetos que estaban en un principio, cuando se originó el evento. Esto puede ser un problema ya que si, por ejemplo, la función a la que llamamos tiene que realizar varias tareas, estas tendrían que hacerse antes de que nos conecte con la nueva página.

Este modo de funcionamiento ha sido un principio muy importante en la gestión de eventos. Si un evento genera la ejecución de un script y además también se genera la acción por defecto para ese objeto entonces:

- El script se ejecutará primero.
- La acción por defecto se ejecutará después.

A veces es interesante bloquear o evitar que se ejecute la acción por defecto. Por ejemplo, en el caso anterior podríamos evitar que nos conecte con la nueva página. Cuando programamos un gestor de eventos, ese gestor podrá devolver un valor booleano `true` o `false` usando la instrucción `return true` o `return false`. `false` quiere decir "no ejecute la acción por defecto". Por lo tanto, nuestro ejemplo quedará del siguiente modo:

```
<a href="pagina.html" onclick="alertar();return false">Pulsa aquí</a>
```

EJEMPLO 17 ANULACIÓN DE LA ACCIÓN POR DEFECTO DE UN EVENTO EN LÍNEA

De esa forma, cada vez que pulsemos en el enlace realizará la llamada a la función `alertar()` y cuando termine ejecutará la instrucción `return false`, que le indicará al navegador que no ejecute la acción por defecto asignada a ese objeto (en este caso la acción por defecto de un hipervínculo es conectarnos con la página especificada en el atributo `href`).

También sería posible hacer que nos preguntara si queremos o no ir a la página de destino. Eso podríamos hacerlo sustituyendo el script del evento por una llamada a una función que solicite una confirmación usando la función global `confirm` (ya conocida).

```
<a href="pagina.html" onclick="confirmar()">Pulsa aquí</a>
//siendo la función confirmar()
function confirmar(){
    return confirm("¿Deseas cambiar a la página indicada?");
}
```

EJEMPLO 18 ANULACIÓN DE LA ACCIÓN POR DEFECTO DE UN EVENTO EN LÍNEA USANDO CONFIRM()

3.2 MODELO DE REGISTRO DE EVENTOS TRADICIONAL.

En los navegadores antiguos, el modelo utilizado era el modelo en línea. Con la llegada de DHTML, el modelo se extendió para ser más flexible. En este nuevo modelo el evento pasó a ser una propiedad del elemento, de modo que el siguiente código de JavaScript es aceptado por los navegadores modernos.

```
elemento.onclick = hacerAlgo;  
// cuando el usuario haga click en el objeto, se llamará a la función hacerAlgo()
```

EJEMPLO 19 REGISTRO DE EVENTOS TRADICIONAL

Esta forma de registro no fue estandarizada por el W3C, pero todavía es válida hoy en día debido a que fue ampliamente utilizada por Netscape y Microsoft. La ventaja de este modelo es que podemos asignar un evento a un objeto desde JavaScript, con lo que ya estamos separando el código de la estructura. Fíjate que aquí los nombres de los eventos sí que van siempre en minúsculas.

Para eliminar un gestor de eventos de un elemento u objeto, le asignamos `null`.

```
elemento.onclick = null;  
// cuando el usuario haga click en el objeto, no se llama a ninguna función
```

EJEMPLO 20 ANULACIÓN DE UN GESTOR DE EVENTOS TRADICIONAL

Otra gran ventaja es que, como el gestor de eventos es una función, podremos realizar una llamada directa a ese gestor, con lo que estamos disparando el evento de forma manual. Por ejemplo:

```
elemento.onclick();  
// disparamos el evento click de forma manual. Ejecuta la función hacerAlgo
```

EJEMPLO 21 DISPARO DEL EVENTO TRADICIONAL DE FORMA MANUAL

Fíjate que en el primer ejemplo no usamos paréntesis en la función `hacerAlgo`. El método `onclick` espera que se le asigne una función completa. Si hiciéramos `element.onclick = hacerAlgo()`; la función sería ejecutada y el resultado devuelto por esa función sería asignado a la propiedad `onclick` del elemento. Pero esto no es lo que queremos que haga, queremos que se ejecute la función cuando se dispare el evento.

En el modelo en línea podíamos utilizar la palabra reservada `this` cuando programamos el gestor de eventos, tal y como se indica en el siguiente ejemplo:

```
<a href="pagina.html" id="mienlace" onClick="alertar(this)">Pulsa aquí</a>  
//siendo la función alertar en el archivo JavaScript  
function alertar(objeto) {  
    alert("Te conectaremos con la página: "+objeto.href);  
}
```

EJEMPLO 22 USO DE THIS EN EL GESTOR DE EVENTOS EN LÍNEA

```
<a href="pagina.html" id="mienlace">Pulsa aquí</a>  
//siendo el script almacenado en el archivo JavaScript  
document.getElementById("mienlace").onclick = alertar; //programamos el evento  
function alertar() {  
    alert("Te conectaremos con la página: "+this.href);  
}
```

EJEMPLO 23 USO DE THIS EN EL MODELO DE GESTOR DE EVENTOS TRADICIONAL

Fíjate que estamos usando `this` dentro de la función `alertar`, sin pasar ningún objeto como argumento (tal y como hacíamos en el modelo en línea). En el modelo tradicional, el `this` que está dentro de la función, hace referencia al objeto donde hemos programado el evento y el causante de la ejecución del código asociado a dicho evento.

En el modelo tradicional es importante que el hipervínculo (o cualquier objeto) sea declarado (conocido) antes de la asignación de cualquier evento (en el ejemplo `onclick`) ya que, si no se conoce todavía el objeto, no se le podrá asignar el evento. La solución es programar la asignación de eventos a los objetos en una función que se ejecute en el momento en que el documento esté completamente cargado programando el evento `onload` del objeto `window`. Por ejemplo, con `window.onload=asignarEventos`.

El código que se ejecuta cuando se produce un evento puede llamar a su vez a otras funciones definidas en el código. Esto lo podemos hacer de la siguiente forma:

```
elemento.onclick = function ( ) {  
    llamada1 ( );  
    llamada2 ( );  
};
```

EJEMPLO 24 PROGRAMANDO UN EVENTO MULTIFUNCIÓN

Recuerda que después de asignar una función a una propiedad de un objeto hay que poner un punto y coma después de la llave de cierre. Estamos haciendo una instrucción de asignación.

3.3 MODELO DE REGISTRO AVANZADO DE EVENTOS SEGÚN W3C.

El W3C, en su especificación del DOM de nivel 2, pone especial atención a los problemas del modelo tradicional de registro de eventos y ofrece una forma sencilla de registrar los eventos que queramos sobre un objeto determinado. La clave para poder hacer todo eso está en el método `addEventListener()`.

Este método tiene tres argumentos:

- El tipo de evento.
- La función a ejecutar.
- Un valor booleano (true o false).

El último argumento se utiliza para indicar en qué momento se deben ejecutar las acciones asociadas al evento:

- En la fase de captura (true).
- En la fase de propagación (false). (burbujeo=bubble)

En el apartado 3.5 veremos en detalle la diferencia entre estas dos fases.

```
elemento.addEventListener('evento', función, false|true);
```

EJEMPLO 25 SINTAXIS DEL MÉTODO ADDEVENTLISTENER

Por ejemplo, para registrar la función `alertar()` de los ejemplos anteriores, haríamos:

```
document.getElementById("mienlace").addEventListener('click',alertar,false);
function alertar(){
    alert("Te conectaremos con la página: "+this.href);
}
```

EJEMPLO 26 MÉTODO ADDEVENTLISTENER

La ventaja de este método, es que podemos añadir tantos eventos como queramos. Por ejemplo:

```
document.getElementById("mienlace").addEventListener('click',alertar,false);
document.getElementById("mienlace").addEventListener('click',avisar,false);
document.getElementById("mienlace").addEventListener('click',chequear,false);
```

EJEMPLO 27 MÉTODO ADDEVENTLISTENER PARA AÑADIR MÚLTIPLES EVENTOS

Por lo tanto, cuando hagamos clic en "`mienlace`" se disparará la llamada a las tres funciones. Por cierto, el W3C no indica el orden de disparo, por lo que no sabemos cuál de las tres funciones se ejecutará primero. Fíjate también, que el nombre de los eventos en el método `addEventListener` no comienza con "on".

En este modelo podemos usar funciones anónimas (sin nombre de función), tal y como se muestra en el siguiente ejemplo:

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

EJEMPLO 28 MÉTODO ADDEVENTLISTENER USANDO FUNCIÓN ANÓNIMA

En este modelo se puede usar la palabra `this` de la misma forma que en el modelo tradicional.

¿Qué eventos han sido registrados?

Uno de los problemas de la implementación del modelo de registro del W3C es que no podemos saber con antelación los eventos que hemos registrado en un elemento.

En el modelo tradicional si hacemos `alert(elemento.onclick)` ; nos devuelve:

- `undefined` - si no hay funciones registradas para ese evento, o bien,
- el nombre de la función que hemos registrado para ese evento.

En este modelo no podemos hacer esto.

El W3C, en el reciente nivel 3 del DOM, introdujo un método llamado `addEventListener` que almacena una lista de las funciones que han sido registradas en un elemento. Tienes que tener cuidado con este método, porque no está soportado por todos los navegadores.

Para eliminar un evento de un elemento, usaremos el método `removeEventListener()` , cuya sintaxis se muestra en el siguiente ejemplo:

```
elemento.removeEventListener('evento', función, false|true);
```

EJEMPLO 29 SINTAXIS DEL MÉTODO REMOVEEVENTLISTENER

Sin embargo, si lo que queremos es cancelar el comportamiento por defecto de un determinado objeto (como en los ejemplos 17 y 18, este modelo nos proporciona el método `preventDefault()` .

3.4 MODELO DE REGISTRO DE EVENTOS SEGÚN MICROSOFT.

Microsoft también ha desarrollado un modelo de registro de eventos. Es similar al utilizado por el W3C, pero tiene algunas modificaciones importantes.

Para registrar un evento, tenemos que enlazarlo con `attachEvent()`:

```
elemento.attachEvent('onclick', hacerAlgo);
```

EJEMPLO 30 SINTAXIS DEL MÉTODO ATTACHEVENT() Si se necesita dos gestores para el mismo evento:

```
elemento.attachEvent('onclick', hacerUnaCosa);  
elemento.attachEvent('onclick', hacerOtraCosa);
```

Para

eliminar un evento, se hará con `detachEvent()`:

```
elemento.detachEvent('onclick', hacerAlgo);
```

EJEMPLO 31 SINTAXIS DEL MÉTODO DETACHEVENT()

Comparando este modelo con el del W3C vemos dos diferencias importantes:

- Los eventos siempre se propagan hacia arriba, no hay fase de captura (no hay tercer argumento para seleccionar el modo).
- La función que gestiona el evento está referenciada, no copiada, con lo que la palabra reservada `this` siempre hará referencia al objeto window y no se podrá utilizar para conocer el elemento que lanzó el evento.

Eventos	
Evento	Se produce cuando...
<code>onblur</code>	Un elemento pierde el foco.
<code>onchange</code>	El contenido de un campo cambia.
<code>onclick</code>	Se hace clic con el ratón en un objeto.
<code>ondblclick</code>	Se hace doble clic con el ratón sobre un objeto.
<code>onerror</code>	Hay algún error al cargar un documento o una imagen.
<code>onfocus</code>	Un elemento tiene el foco.
<code>onkeydown</code>	Se presiona una tecla del teclado.
<code>onkeypress</code>	Se presiona una tecla o se mantiene presionada.
<code>onkeyup</code>	Cuando soltamos una tecla.
<code>onload</code>	Una página o imagen terminaron de cargarse.
<code>onmousedown</code>	Se presiona un botón del ratón.
<code>onmousemove</code>	Se mueve el ratón sobre un elemento.
<code>onmouseout</code>	Movemos el ratón fuera de un elemento.
<code>onmouseover</code>	El ratón se mueve sobre un elemento.
<code>onmouseup</code>	Se libera un botón del ratón.
<code>onresize</code>	Se redimensiona una ventana o marco.
<code>onselect</code>	Se selecciona un texto.
<code>onunload</code>	El usuario abandona una página.

TABLA 16 LISTADO DE EVENTOS

EJERCICIO RESUELTO

Descarga el [siguiente ejercicio](#). En él se muestra un formulario para añadir y borrar elementos en una lista de selección múltiple. El formulario dispone de:

1. Una lista de selección múltiple,
2. Una caja de texto donde se escribe el dato a añadir,
3. Unos botones de opción donde se elige el tipo de operación a realizar:
 - a. Añadir el elemento escrito en la caja a la lista.
 - b. Borrar los elementos que estén seleccionados en la lista (permite selección múltiple).
4. Un botón que ejecuta la operación escogida.

El archivo.js gestiona el formulario de la siguiente forma:

1. Los nombres de las frutas de la lista tienen siempre la inicial en mayúscula.
2. Tiene en cuenta los acentos a la hora de no repetir frutas en el caso de que el usuario escriba la fruta sin acentuar o ponga el acento donde no debe.
3. No es sensible al uso de mayúsculas por parte del usuario al introducir una nueva fruta.
4. Cuando el usuario se posiciona en la caja de texto se activa la opción añadir y se selecciona automáticamente lo que esté escrito en ella.
5. Cuando el usuario selecciona una o más frutas de la lista se activa la opción borrar.
6. Hay mensajes que informan al usuario en todo momento de lo que sucede con la interacción que está realizando.

Puedes ampliar el ejercicio para que la lista de frutas esté siempre ordenada insertando la fruta a añadir en la posición correcta. La lista inicial está ordenada.

*(**recuerda que para que funcione el enlace tienes que descargar el recurso en la misma carpeta que este documento)*

EJERCICIO 1 OPERACIONES CON FORMULARIOS QUE INCLUYEN LABEL, SELECT, RADIO, TEXTBOX (RESUELTO)

EJERCICIO RESUELTO

Descarga el siguiente [ejercicio](#).

En él se muestra un formulario que genera combinaciones de juegos de azar.

Te permite mediante la selección del tipo de juego, general el número de combinaciones que quieras.

*(**recuerda que para que funcione el enlace tienes que descargar el recurso en la misma carpeta que este documento)*

EJERCICIO 2 GENERAR COMBINACIONES DE JUEGOS DE AZAR CON EVENTOS (RESUELTO)

3.5 ORDEN DE DISPARO DE LOS EVENTOS.

Si has leído con atención el enlace recomendado en el apartado anterior y has comprobado el funcionamiento de sus ejemplos, te resultará ahora mucho más fácil el comprender el orden de disparo de los eventos. Si no es así, deberás leer este apartado con atención para comprender el funcionamiento de lo que hemos llamado en apartados anteriores fase de captura y fase de propagación hacia arriba.

En el modelo DOM, los objetos del HTML se asemejan a un árbol que partiendo de la raíz (objeto `document`) tal y como veíamos en la [Ilustración 1](#) de esta misma unidad.

Dicha ilustración hacía referencia a la jerarquía de los objetos dentro del DOM. Pero, en un documento HTML podemos tener un objeto que está contenido dentro de otro, este último a su vez dentro de otro, y un largo etcétera.

Imagina que tenemos un elemento contenido dentro de otro elemento y que tenemos programado el mismo tipo de evento para ambos (por ejemplo, el evento clic). ¿Cuál de ellos se disparará primero?

Sorprendentemente, esto va a depender del tipo de navegador que tengamos.

Si el usuario posiciona el ratón sobre la zona central que abarca el Elemento 2 y realiza un clic con el ratón, lanzará el evento clic en ambos elementos. ¿Cuál de ellos se disparará primero?, ¿cuál es el orden en el que se van a ejecutar las acciones asociadas a estos eventos?

Tenemos dos **Modelos propuestos** por Netscape y Microsoft en sus orígenes:

- Netscape dijo que el evento en el Elemento1 tendrá lugar primero. Es lo que se conoce como "captura de eventos".
- Microsoft dijo que el evento en el Elemento2 tendrá preferencia. Es lo que se conoce como bubbling "propagación *hacia* arriba de eventos".

Los dos modelos son claramente opuestos. Internet Explorer sólo soporta el segundo modelo. Mozilla, Opera 7 y Konqueror soportan los dos modelos. Y las versiones antiguas de Opera e iCab no soportan ninguno.

Modelo W3C

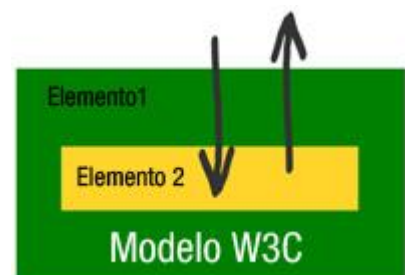
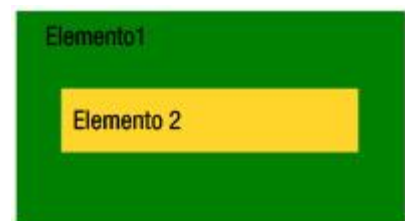
W3C decidió tomar una posición intermedia. Así supone que, cuando se produce un evento primero se producirá la fase de captura hasta llegar al elemento de destino, y luego se producirá la fase de propagación hacia arriba en el modelo del DOM. Este modelo es el estándar, que todos los navegadores deberían seguir para ser compatibles entre sí.

Si usas el método `addEventListener` podrás decidir cuando quieres que se registre el evento: en la fase de captura o en la fase de burbujeo. Para ello tendrás que dar el valor `true` o `false`, respectivamente, al tercer parámetro. Por ejemplo:

```
elemento1.addEventListener('click',hacerAlgo1,true);  
//elemento1 representa la caja verde  
elemento2.addEventListener('click',hacerAlgo2,false);  
//elemento2 representa la caja amarilla
```

Si el usuario hace clic en el `elemento2` ocurrirá lo siguiente:

1. El evento clic comenzará en la fase de captura.
2. El gestor de eventos comprueba si hay algún ancestro del `elemento2` en el árbol del DOM que tenga programado un evento `onclick` para la fase de captura (`true`).



3. El gestor de eventos encuentra un `elemento1.hacerAlgo1()` que ejecutará primero, pues su tercer argumento tiene un valor `true`.
4. El evento viajará hacia el destino (`elemento2`), pero no encontrará más eventos para la fase de captura. Entonces el evento pasa a la fase de propagación, y ejecuta `hacerAlgo2()`, el cual está programado para esta fase (con un valor `false` en su tercer argumento).
5. El evento viaja hacia arriba de nuevo y chequea si algún ancestro tiene programado ese evento para la fase de propagación. Este no será el caso, por lo que no hará nada más.
6. Para detener la propagación del evento en la fase de propagación, disponemos del método `stopPropagation()`. En la fase de captura es imposible detener la propagación.

RECOMENDACIÓN

En la siguiente página encontrarás una explicación más detallada de las diferencias que hay entre el modelo de Microsoft y el del W3C, acompañado de un ejemplo que puedes probar.

[Propagación de eventos en JavaScript. Blog de OpenAlfa](#)

En el siguiente ejemplo hay un enlace a una carpeta comprimida con los archivos del ejemplo anterior. Para que funcione el enlace tienes que descargar el material complementario en la misma carpeta que este documento.

[Ejemplo anterior](#) para comprobación de funcionamiento

RECOMENDACIÓN 1 PROPAGACIÓN DE EVENTOS

4. ENVÍO Y VALIDACIÓN DE FORMULARIOS.

La validación de un formulario es un proceso que consiste en comprobar que todos los datos han sido introducidos correctamente. Por ejemplo, si tu formulario contiene un campo de texto en el que hay que escribir un e-mail, sería interesante comprobar si ese e-mail está escrito correctamente, antes de pasar al siguiente campo.

Hay dos métodos principales de validación de formularios:

- en el lado del servidor (usando scripts CGI, PHP, ASP, etc.) y
- en el lado del cliente (generalmente usando JavaScript).

La **validación en el lado del servidor** es más segura, pero a veces también es más complicada de programar, mientras que la **validación en el lado del cliente** es más fácil y más rápida de hacer (el navegador no tiene que conectarse al servidor para validar el formulario, por lo que el usuario recibirá información al instante sobre posibles errores o fallos encontrados).

La idea general que se persigue al validar un formulario, es que solo se envíen datos correctos al servidor.

Podremos realizar la validación de los datos a medida que se van metiendo en el formulario, campo a campo, o al final, cuando se pulse el botón de envío del formulario.

JavaScript añade a tus formularios dos características muy interesantes:

- JavaScript te permite examinar y validar las entradas de usuario directamente, en el lado del cliente.
- JavaScript te permite dar mensajes instantáneos, con información de la entrada del usuario.

Las validaciones que se suelen hacer con los datos de un formulario son:

- Comprobar cuando existe o no un valor. Hay campos que consideramos necesarios y que el usuario debe rellenar (por ejemplo: un campo de número de artículos en una compra on line) y otros que los consideramos optativos (por ejemplo: un campo de comentarios adicionales en el pedido de una compra on line que puede dejarse en blanco).
- Comprobar que un dato contenga datos de un determinado tipo (por ejemplo: el campo de número de artículos debe contener una cantidad numérica y no un conjunto de letras).
- Comprobar que un dato se adapte a un determinado patrón (por ejemplo: un e-mail, una fecha, un número de teléfono, un NIF, etc.).

JavaScript también se puede utilizar para modificar dinámicamente los elementos de un formulario, basándose en los datos introducidos por el usuario (por ejemplo: cubrir un campo de selección con una lista de nombres de ciudades después de haber seleccionado la provincia en otro cuadro de selección).

Una parte muy importante que no debes olvidar al usar JavaScript con los formularios, es la posibilidad de que el usuario desactive JavaScript en su navegador, por lo que envío de datos desde un formulario no debería ser dependiente de JavaScript.

Acuérdate de que JavaScript está para mejorar, no para reemplazar.

Pero, cuando los datos que va a enviar un formulario se van a utilizar para realizar alguna gestión con algún programa de servidor, los datos tendrán que ser validados en el lado del servidor, independientemente de que se haya programado la validación del formulario en el lado del cliente para mejorar la experiencia del usuario a la hora de introducir los datos, ya que no se puede garantizar realmente que se hayan validado los datos en el lado del cliente.

Ejemplo sencillo de validación de un formulario

Como se comentó anteriormente, el objetivo de validar un formulario es comprobar antes de su envío que todos los campos poseen valores correctos, evitando así que el usuario tenga que comenzar de nuevo el formulario cuando no se puede validar correctamente en el servidor. Esto no evita que también tengamos que hacer validación en el servidor, ya que el usuario puede desactivar JavaScript en su navegador, con lo que la validación que hemos programado en JavaScript no tendría efecto.

La llegada del HTML5 con sus nuevos objetos Input con y sus nuevos atributos ha facilitado mucho la tarea de la validación de formularios. Con los nuevos niveles de CSS la apariencia que toman los distintos objetos en función de la interacción del usuario hace que el diseño visual de la interacción no recaiga sobre JavaScript.

Ambos elementos, HTML5 y CSS han ahorrado muchas horas de desarrollo de la interacción con JavaScript y de modificar el estilo de los elementos de la página en función de esta interacción.

Aun así, hay muchas aplicaciones web que hay que mantener, habrá muchos ordenadores que no tengan actualizados sus navegadores y no reconozcan esas aplicaciones web que están desarrolladas con su última tecnología. ¿Hay algo de malo en saber cómo se hacían antes las cosas, cuando sólo contábamos con esos 4 objetos de tipo texto?

EJERCICIO RESUELTO

Descarga el siguiente [ejercicio](#).

En él se encuentra el código fuente con comentarios de todas las funciones utilizadas para la validación de un formulario a la antigua usanza. Para ello se utilizan también expresiones regulares que verás en el apartado siguiente.

Estas validaciones con expresiones regulares no garantizan la corrección de los datos. El NIF puede ser inválido, aunque su formato sea correcto. Lo mismo ocurre con la fecha de nacimiento y la hora de la cita.

Las expresiones regulares facilitan muchas cosas, pero no lo hacen todo.

*(**recuerda que para que funcione el enlace tienes que descargar el recurso en la misma carpeta que este documento)*

EJERCICIO 3 VALIDACIÓN DE FORMULARIO A LA ANTIGUA USANZA USANDO EXPRESIONES REGULARES (RESUELTO)

RECOMENDACIÓN

Intenta hacer el mismo ejemplo, pero aprovechando las características del HTML5 con todos los objetos input de que dispone y con sus atributos **pattern**.

RECOMENDACIÓN 2 FORMULARIO MODERNO

5. ALMACENAMIENTO DE DATOS EN EL LADO DEL CLIENTE

¿Alguna vez al entrar en alguna página te has encontrado con algo parecido a lo que se ve al pie de esta imagen? Seguro que sí.



ILUSTRACIÓN 5 CONSENTIMIENTO INFORMADO DE COOKIES

¿Sabías que es obligatorio incluir en tu página un consentimiento informado cuando quieres almacenar datos en el equipo local?

¿Sabías que hay una ley que así lo estipula y que las sanciones van desde 30.000€ para una infracción leve hasta 600.000€ para una infracción grave? Se trata de la Ley 34/2002, de 11 de julio, de Servicios de la Sociedad de la Información y Comercio Electrónico, conocida como LSSI.

RECOMENDACIÓN

En el primer enlace podrás acceder a toda la información sobre la LSSI desde la página del Ministerio de Energía, Turismo y Agenda Digital.

[LSSI](#)

En el segundo enlace podrás descargar una guía sobre el uso de las cookies desarrollada por la Agencia Española de Protección de Datos.

[Guía de uso de cookies](#)

Recuerda lo que dice nuestro Código Civil: "el desconocimiento de la ley no exime de su cumplimiento".

RECOMENDACIÓN 3 LSSI Y GUÍA DE USO DE COOKIES

5.1 LAS COOKIES.

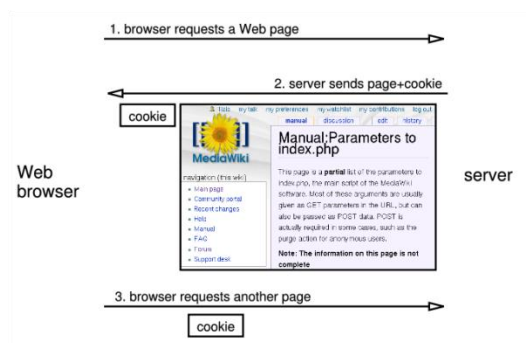


ILUSTRACIÓN 6 UNA POSIBLE INTERACCIÓN ENTRE UN NAVEGADOR WEB Y UN SERVIDOR, EN LA QUE EL SERVIDOR ENVÍA UNA COOKIE CON LA PÁGINA Y EL NAVEGADOR LA DEVUELVE CUANDO EL USUARIO SOLICITA OTRA PÁGINA

Permitir que algún programa pueda leer y escribir en el disco duro puede dar que pensar en un principio, pero el mecanismo de las cookies es algo más seguro, ya que no abre tu disco duro al mundo para que cualquiera pueda ver su contenido o modificarlo. Este mecanismo de las cookies proporciona acceso a un fichero de texto (en Internet Explorer) o a un fichero especial (en otros navegadores distintos a Internet Explorer), que está situado en un directorio especial del disco duro.

En Windows 7:

```
C:\Users\[user_name]\AppData\Roaming\Mozilla\Firefox\Profiles\[profile_name].default\cookies.sqlite
```

```
C:\Users\[user_name]\AppData\Local\Microsoft\Windows\Temporary Internet Files
```

En Windows XP:

```
C:\Documents and Settings\[user_name]\Application Data\Mozilla\Firefox\Profiles\[profile_name].default\cookies.sqlite
```

Los párrafos anteriores son ejemplos de localizaciones de las cookies para diferentes versiones del sistema operativo y navegador usado.

Un fichero de cookies es un fichero cuyo formato de almacenamiento de los datos dependerá del navegador. La estructura que tenga este fichero te dará igual ya que para acceder a las cookies lo vas a hacer a través de la propiedad `document.cookie`.

Formato de un registro de cookie

Entre todos los campos que se almacenarán en una cookie tenemos los siguientes (no necesariamente en el mismo orden):

- Dominio del servidor que creó la cookie.
- Información de si es necesaria una conexión http segura para acceder a la cookie.
- Trayectoria de las URL que podrán acceder a la cookie.
- Fecha de caducidad de la cookie.
- Nombre de una entrada de datos.
- Cadena de texto asociada a ese nombre de entrada de datos.

Las cookies son específicas del dominio. En otras palabras, si un dominio crea una cookie, otro dominio no podrá acceder a ella a través del navegador. La razón de ello es que muchas veces podremos almacenar datos importantes como usuarios/contraseñas en las cookies, y no queremos que otros dominios puedan consultarlos. La mayor parte de las veces, cuando almacenamos datos de este tipo, estarán encriptados dentro de la cookie.

Las cookies tienen una fecha de caducidad, ya que algunos navegadores tienen limitado el número máximo de cookies que pueden almacenar (1000 en Firefox). Será el propio navegador el encargado de borrar las cookies caducadas.

RECOMENDACIÓN

En los siguientes enlaces podrás obtener más información sobre el tratamiento de las cookies con JavaScript.

[W3Schools](#)

RECOMENDACIÓN 4 GESTIÓN DE COOKIES CON W3SCHOOLS

RECOMENDACIÓN

En el siguiente enlace podrás obtener más información sobre el tema de las cookies, de la mano de Wikipedia.

[Cookie \(Wikipedia\)](#)

RECOMENDACIÓN 5 COOKIES SEGÚN WIKIPEDIA

La eliminación o desactivación de las cookies no debe limitar la funcionalidad del sitio.

RECOMENDACIÓN

En los siguientes enlaces puedes ver cómo han redactado la "Política de cookies" algunos sitios web.

[inouthostel](#)

[coenfeba](#)

[casa de la estilografía](#)

[e-escola](#)

RECOMENDACIÓN 6 POLÍTICA DE COOKIES DE DIFERENTES SITIOS WEB

RECOMENDACIÓN

En el siguiente enlace puedes obtener más información sobre la Política de cookies desde el que se puede descargar un código a insertar en las páginas web que desarrolles y que empleen el almacenamiento local.

<http://politicadecookies.com>

RECOMENDACIÓN 7 POLÍTICA DE COOKIES Y DESCARGA DE CÓDIGO DE EJEMPLO

El código al que da acceso el último enlace recomendado, tiene el JavaScript incrustado en el HTML y utiliza eventos en línea y en el texto hace referencia a otra ley.

RECOMENDACIÓN

Ahora que conoces el modelo de eventos del W3C, deberías tener preparado tu propio código de consentimiento informado adaptado a este modelo y en tu propio archivo js para poder enlazarlo cuando lo necesites en cualquiera de tus aplicaciones web.

En el siguiente enlace hay un ejemplo de un uso de cookies con el código descargado del enlace de la recomendación anterior modificado para tener separado el JavaScript del HTML y utilizando el modelo de eventos del W3C.

[Ejemplo de cookies con consentimiento informado](#)

RECOMENDACIÓN 8 HAZ TU PROPIO CONSENTIMIENTO INFORMADO

5.2 GESTIÓN Y USO DE COOKIES.

Grabar una cookie

Para grabar datos en un fichero de cookie, podemos utilizar una asignación simple con la propiedad `document.cookie`. Pero tendrás que tener mucha precaución con el formato de datos para que la cookie sea grabada correctamente. Aquí te muestro la sintaxis de cómo se asignaría un valor a una cookie (los campos opcionales van entre corchetes; en cursiva irán las posiciones para escribir nuestros propios datos):

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

Cada cookie deberá tener un nombre y un texto asignado (aunque sea cadena vacía ""). Por ejemplo si quieres almacenar la cadena "Martin" para una cookie "usuario", haríamos:

```
document.cookie = "usuario=Martin";
```

Si el navegador ve que no tenemos ninguna cookie con este nombre, la creará automáticamente; si la cookie ya existe, entonces la reemplazará. Se pueden omitir el resto de parámetros de la cookie; en ese caso el navegador usará valores por defecto. Para cookies temporales generalmente sólo necesitaremos escribir nombre=valor. Es decir estas cookies durarán solamente el tiempo de la sesión. Si por ejemplo cerramos el navegador y lo volvemos a abrir la cookie desaparece.

```
document.cookie="contador=0";  
// Almacena contador=0 en la cookie sin ningún otro contenido a mayores.
```

La fecha de caducidad asignada a la propiedad `expires`, tendrá que ir en formato GMT. Por ejemplo:

```
expires=Thu, 01-Jan-70 00:00:01 GMT;
```

El `path` será la ruta actual de nuestra página web.

Si no se pone el dominio (`domain`), se asignará por defecto el dominio de la página que creó la cookie.

Si se omite el valor `secure`, nuestra cookie será accesible por cualquier programa en nuestro dominio que se ajuste a las propiedades de `domain` y `path`.

Recuperar información de una cookie

Para recuperar los datos de una cookie tendremos que acceder a la propiedad `document.cookie` e imprimir su valor. Los resultados nos llegarán en forma de cadena de texto. Por ejemplo, una cadena de texto `document.cookie` podría tener el siguiente aspecto:

```
usuario=Martin; password=OjYgdjUA
```

En otras palabras, no podremos tratar las cookies como objetos. En su lugar, deberemos pasar la cadena de texto de la cookie y extraer los datos necesarios de su contenido.

5.3 LOCALSTORAGE Y SESSIONSTORAGE.

El almacenamiento web ha cobrado una mayor relevancia con la nueva especificación de HTML5.

Ha sido estandarizado por el World Wide Web Consortium (W3C) y actualmente hay una especificación de una API para la persistencia de datos almacenados y pares de datos (clave:valor) en el ordenador del cliente, recomendada con fecha de 19 de abril de 2016.

A diferencia de lo que ocurría con las cookies, con estos objetos no hay problemas con las caducidades, se evita el tráfico por Internet ya que los datos almacenados no se envían con cada petición y permite el almacenamiento de mayor cantidad de información.

Esta funcionalidad nueva del HTML5 se llama Web Storage y entre otras cosas nos da grandes posibilidades de trabajo con páginas y aplicaciones trabajando offline (sin acceso a Internet).

Hay dos tipos de Web Storage e implican el trabajo con dos objetos distintos:

- **Local Storage:** Los datos almacenados no tienen fecha de caducidad, permanecerán indefinidamente disponibles en el ordenador del cliente. (Se utiliza el objeto localStorage)
- **Session Storage:** Los datos almacenados sólo estarán disponibles durante la sesión de navegación, cuando se cierre, desaparecerán del ordenador del cliente. (Se utiliza el objeto sessionStorage)

Cuando viste el objeto window de alto nivel, ya viste que estos objetos (localStorage y sessionStorage) eran dos de sus propiedades. Es por ello que no hay que utilizar la palabra window cuando se utilizan.

Desde la perspectiva del código asociado a la página web, localStorage y sessionStorage se comportan de la misma manera. Sólo cambia la disponibilidad temporal de la información.

Ambos están asociados a un dominio de Internet y no a una página concreta. La información es guardada por dominio web (incluye todas las páginas del dominio). Dado que está asociado a un determinado ordenador y usuario, si se accede desde otro ordenador o desde el mismo ordenador, pero como diferente usuario, los anteriores datos de localStorage no estarán disponibles.

Aunque HTML5 no fija nada acerca de la máxima capacidad, la mayor parte de los navegadores permiten almacenar entre 5 MB y 10 MB de información; incluyendo texto y multimedia. La recomendación es de 5 MB. El tamaño máximo para las cookies era de 4 KB.

Los datos se almacenan siempre como texto por lo que habrá que utilizar funciones de conversión para acceder a datos de otro tipo –números, fechas, ...

Los datos se almacenan en formato de parejas de nombre:valor, en dónde el nombre es el de la variable almacenada y el valor es el contenido textual que se guarda.






API					
Web Storage	4.0	8.0	3.5	4.0	11.5

ILUSTRACIÓN 7 SOPORTE DE WEB STORAGE DE LAS DIFERENTES VERSIONES DE LOS NAVEGADORES

Antes de usar Web Storage conviene comprobar si el navegador lo reconoce.

Para ello basta con incorporar una simple condicional a nuestro JavaScript.


```

if (typeof(Storage) !== "undefined") {
    //Soporta localStorage/sessionStorage. Pondríamos aquí el código
} else {
    //No soporta el almacenamiento local con este tipo de objetos.
    //Pondríamos un aviso al usuario si es que el funcionamiento de nuestra
    aplicación depende de este almacenamiento
}

```

EJEMPLO 32 COMPROBACIÓN DE SI ADMITE EL WEB STORAGE

sessionStorage

Este objeto se instancia por sesión y ventana, por lo que dos pestañas del navegador abiertas al mismo tiempo y para un mismo sitio web pueden tener información distinta.

Al cerrar la pestaña se pierde la información.

RECOMENDACIÓN

En el siguiente enlace está la 2ª edición de la especificación de Web Storage publicada por el W3C como Recomendación desde el 19 abril 2016.

[Web Storage](#)

RECOMENDACIÓN 9 ESPECIFICACIÓN DE WEB STORAGE (RECOMENDACIÓN DEL W3C DESDE EL 19/04/2016)

RECOMENDACIÓN

Prueba el ejemplo del enlace siguiente. Verás que se abre una pestaña en el navegador y, cada vez que haces clic en el botón "Click me!", se incrementa el valor de un contador.

Si abres el mismo ejemplo en diferentes pestañas podrás comprobar que los contadores son independientes en cada una de ellas.

Pero si en cualquiera de ellas pulsas el botón Run que recarga la página verás que mantiene el valor del contador, aunque se haya recargado.

[sessionStorage](#)

RECOMENDACIÓN 10 USO DE SESSIONSTORAGE EN W3SCHOOLS

localStorage

Este mecanismo está orientado a las aplicaciones que necesitan mantener gran cantidad de información del usuario entre diferentes sesiones.

El W3C ha definido este objeto para poder acceder al área de almacenamiento local del dominio.

Puede ser accedido cada vez que se visita un dominio concreto (no valen los subdominios) y todas las sesiones abiertas sobre el mismo dominio ven la misma información.

RECOMENDACIÓN

Prueba ahora el mismo ejemplo del enlace, pero con localStorage. Verás que, aunque abras varias pestañas con este enlace, el valor se incrementa y se sigue incrementando en las diferentes pestañas. Y, aunque las cierres todas, si vuelves a abrir una y le das de nuevo a "Click me!", el contador mostrará el valor incrementado.

[localStorage](#)

RECOMENDACIÓN 11 USO DE LOCALSTORAGE EN W3SCHOOLS

En la especificación oficial se indica que, cualquier cambio en el almacén de datos, debe disparar un evento de tipo `StorageEvent`, de forma que cualquier ventana con acceso al almacén pueda responder al mismo. Los dos objetos descritos, `localStorage` y `sessionStorage`, poseen una interfaz heredada del objeto `Storage` donde se describen sus propiedades, métodos y eventos:

Propiedades del objeto Storage	
Propiedad	Descripción
<code>length</code>	Obtiene el número de elementos almacenados. Cada elemento es un par (clave:valor)

TABLA 17 PROPIEDAD DEL OBJETO STORAGE

Métodos del objeto Storage	
Método	Descripción
<code>clear()</code>	Borro todos los datos almacenados en esa sesión.
<code>getItem(clave)</code>	Obtiene el dato almacenado con esa clave.
<code>key(índice)</code>	Devuelve la clave que ocupa la posición indicada por índice.
<code>removeItem(clave posicion)</code>	Borra el elemento almacenado con dicha clave o que ocupa dicha posición.
<code>setItem(clave,valor)</code>	Almacena un elemento en la lista proporcionando la clave y el valor asociado a dicha clave.

TABLA 18 MÉTODOS DEL OBJETO STORAGE

Eventos del objeto Storage	
Evento	Descripción
<code>Storage()</code>	Se desencadena cuando hay cualquier cambio en el área de almacenamiento (sesión o local). Se crea un objeto para ese evento con sus propiedades.

TABLA 19 EVENTOS DEL OBJETO STORAGE

Propiedades del objeto creado por el evento Storage	
Propiedad	Descripción
<code>key</code>	Contiene la clave que ha sido agregada, modificada o eliminada.
<code>newValue</code>	El nuevo valor asignado a una determinada clave. Null si fue eliminada.
<code>oldValue</code>	El valor anterior a la modificación o borrado. Null si es una clave añadida.
<code>storageArea</code>	Representa el objeto <code>sessionStorage</code> o <code>localStorage</code> afectado.
<code>url</code>	El dominio asociado al objeto cambiado.

TABLA 20 PROPIEDADES DEL OBJETO CREADO POR EL EVENTO STORAGE

RECOMENDACIÓN

En el siguiente enlace, tienes un enlace a la última actualización de la API de WEB Storage (no es normativa).

[WebStorage](#)

RECOMENDACIÓN 12 API WEBSTORAGE

EJEMPLOS

```
if (typeof(Storage) !== "undefined") {
    //almacenamos el nombre de un usuario que está almacenado en una variable name
    localStorage.setItem("nombre",name); //o bien localStorage.nombre=name;
    //también localStorage["nombre"]=name
} else {
    //No soporta el almacenamiento local con este tipo de objetos - alertaríamos
}
```

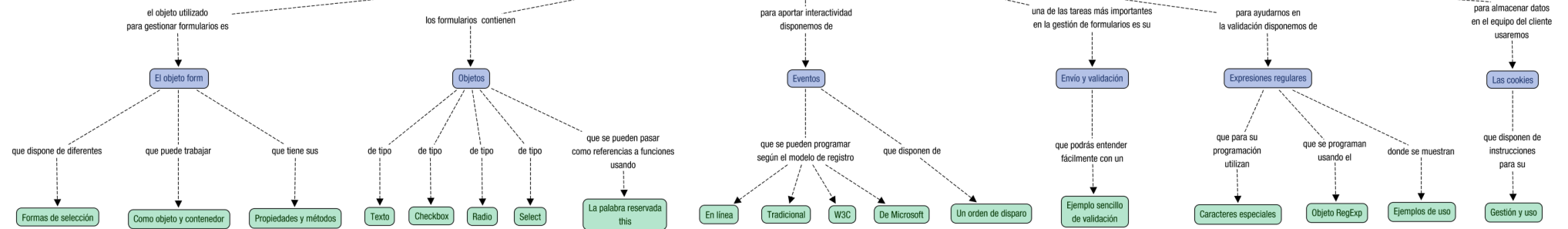
EJEMPLO 33 ALMACENAR UN NOMBRE CON LOCALSTORAGE

```
if (typeof(Storage) !== "undefined") {  
    //visualizamos el nombre de un usuario que está almacenado en una variable name  
    alert(localStorage.getItem("nombre"));  
    //también alert(localStorage["nombre"]); o alert(localStorage.nombre);  
} else {  
    //No soporta el almacenamiento local con este tipo de objetos - alertaríamos  
}
```




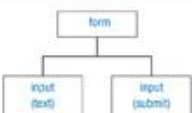




EJEMPLO 34 VISUALIZAR UN NOMBRE ALMACENADO CON LOCALSTORAGE

MAPA CONCEPTUAL

GESTIÓN DE EVENTOS Y FORMULARIOS EN JAVASCRIPT



ANEXO.- LICENCIAS DE RECURSOS.

Recurso	Datos del recurso	Recurso	Datos del recurso
	Autoría: Rafael Veiga Cid. Licencia: Copyright (derecho de cita). Procedencia: Captura de pantalla de Dreamweaver CS5.5, propiedad de Adobe Systems Incorporated.		Autoría: Beatriz Eugenia Buyo Pérez. Licencia: CC BY 2.0. Procedencia: SamrtArt
	Autoría: raveiga. Licencia: CC BY 2.0. Procedencia: http://www.flickr.com/photos/raveiga/5732922077/sizes/l/in/photostream/		Autoría: raveiga. Licencia: CC BY 2.0. Procedencia: http://www.flickr.com/photos/raveiga/5733502264/sizes/z/in/photostream/
	Autoría: Jurok. Licencia: CC BY-SA 3.0 Procedencia: http://upload.wikimedia.org/wikipedia/commons/a/ac/Cookieeee.png		Autoría: DonkeyHotey. Licencia: CC BY 2.0. Procedencia: http://www.flickr.com/photos/donkeyhotey/5639011991/sizes/z/in/photostream/
	Autoría: Beatriz Eugenia Buyo Pérez Licencia: CC BY 2.0 Procedencia: Captura de pantalla		Autoría: Beatriz Eugenia Buyo Pérez Licencia: CC BY 2.0 Procedencia: Captura de pantalla

Disparar. En términos de programación indicamos que un evento se dispara o es disparado, cuando ese evento es ejecutado o es lanzado. Por lo tanto lo podemos traducir por ejecutar o lanzar.

Encriptar. Es la acción de proteger la información para que no pueda ser leída sin una clave. También se puede decir cifrar la información.

Granularidad. Se refiere a la especificidad con la que se define un nivel de detalle.

Índice. Cuando hablamos de un índice en un array, nos estamos refiriendo a la posición de ese array.

Literal. Cuando hablamos de expresiones regulares, un literal, hace referencia a una cadena o estructura, que contiene caracteres especiales y símbolos que conforman el patrón de la expresión regular.