

INFORMÁTICA Y
COMUNICACIONES

UF1306. PRUEBAS DE FUNCIONALIDADES Y OPTIMIZACIÓN DE PÁGINAS WEB

Contenidos basados en los Certificados de profesionalidad

**UF1306. PRUEBAS DE FUNCIONALIDADES Y OPTIMIZACIÓN
DE PÁGINAS WEB**



Certia
editorial

DATOS DEL AUTOR

Ariel Santiago Castro Álvarez es técnico superior en Desarrollo de Aplicaciones Informáticas por el Instituto Politécnico de Vigo, y obtuvo el Certificado de Aptitud Pedagógica en la Universidad de Santiago de Compostela.

Ha dedicado gran parte de su vida profesional a la impartición de cursos sobre lenguajes de programación, desarrollo de páginas web, diseño gráfico, diseño y posicionamiento de blogs, *e-commerce*, *social media marketing*, ofimática, etc. En los últimos años ha elaborado numerosos materiales didácticos para la enseñanza de las TIC a todos los niveles.

El autor participa asiduamente en acciones formativas sobre distintas tecnologías de la información y la comunicación con el fin de mantenerse al día en estas materias, siempre en evolución. En la actualidad desarrolla su actividad profesional dentro del ámbito de las redes informáticas.

FICHA

Pruebas de funcionalidades y
optimización de páginas web. Informática
y comunicaciones

1ª Edición

Certia Editorial, Pontevedra, 2015

Autor: Ariel Santiago Castro Álvarez

Formato: 170 x 240 mm • 133 páginas.

PRUEBAS DE FUNCIONALIDADES Y OPTIMIZACIÓN DE PÁGINAS WEB. INFORMÁTICA
Y COMUNICACIONES.

No está permitida la reproducción total o parcial de este libro, ni su tratamiento informático, ni la transmisión de ninguna forma o por cualquier medio, ya sea electrónico, mecánico, por fotocopia, por registro u otros métodos, sin el permiso previo y por escrito de los titulares del Copyright.

Derechos reservados 2015, respecto a la primera edición en español, por Certia Editorial.

ISBN: 978-84-16019-79-3

Depósito legal: PO 235-2015

Impreso en España - Printed in Spain

Certia Editorial ha incorporado en la elaboración de este material didáctico citas y referencias de obras divulgadas y ha cumplido todos los requisitos establecidos por la Ley de Propiedad Intelectual. Por los posibles errores y omisiones, se excusa previamente y está dispuesta a introducir las correcciones pertinentes en próximas ediciones y reimpresiones.

Fuente fotografía portada: MorgueFile, autoriza a copiar, distribuir, comunicar públicamente la obra y adaptar el trabajo.

Familia profesional: INFORMÁTICA Y COMUNICACIONES

Área profesional: Desarrollo

FICHA DE CERTIFICADO DE PROFESIONALIDAD (IFCD0110) CONFECCIÓN Y PUBLICACIÓN DE
PÁGINAS WEB (RD 1531/2011, de 31 de octubre modificado por el RD 628/2013, de 2 de agosto)

Correspondencia con el Catálogo Modular de Formación Profesional				
H. Q	Módulos certificado	H.CP	Unidades formativas	Horas
210	MF0950_2: Construcción de páginas web	210	UF1302: Creación de páginas web con el lenguaje de marcas	80
			UF1303: Elaboración de hojas de estilo	70
			UF1304: Elaboración de plantillas y formularios	60
210	MF0951_2: Integración de componentes soft	180	UF1305: Programación con lenguajes de guión en páginas web	90
90	MF0952_2: Publicación de páginas web	90	UF1306: Pruebas de funcionalidades y optimización de páginas web	90
	MP0278: Módulo de prácticas profesionales no laborales	80		90
510	Duración horas totales certificado de profesionalidad	560	Duración horas módulos formativos	480

ÍNDICE

• INTRODUCCIÓN	11
• UNIDAD DIDÁCTICA 1. Validaciones de datos en páginas web	13
1.1. Funciones de validación.....	16
1.1.1. Descripción de las funciones.....	16
1.1.2. Utilidad de las funciones.....	18
1.1.3. Implementación de las funciones.....	18
1.1.4. Validaciones alfanuméricas, numéricas y de fecha	19
1.1.5. Definición de validaciones.....	22
1.1.6. Código de validación	24
1.1.7. Ejecución del código de validación.....	25
1.2. Verificar formularios.....	26
1.2.1. Identificación de los datos.....	26
1.2.2. Implementación del código de verificación	29
1.2.3. Comprobación de los datos introducidos por el usuario.....	31
RESUMEN.....	37
ACTIVIDADES	39
EVALUACIÓN	41
• UNIDAD DIDÁCTICA 2. Efectos especiales en páginas web	43

2.1. Trabajar con imágenes: imágenes de sustitución e imágenes múltiples	46
2.1.1. Selección de imágenes.....	46
2.1.2. Optimización de imágenes	47
2.1.3. Implementación de código con varias imágenes.....	51
2.2. Trabajar con textos: efectos estéticos y de movimiento.....	55
2.2.1. Creación de textos mejorados y con movimiento.....	56
2.2.2. Implementación de efectos	58
2.2.3. Adecuación de los efectos a la página web	59
2.3. Trabajar con marcos	60
2.3.1. Dónde utilizar los marcos.....	61
2.3.2. Limitaciones de los marcos	66
2.3.3. Alternativas a los marcos	67
2.4. Trabajar con ventanas.....	70
2.4.1. Creación de varias ventanas.....	70
2.4.2. Interactividad entre varias ventanas	74
2.5. Otros efectos	76
2.5.1. Efectos con HTML	77
2.5.2. Efectos con CSS.....	80
2.5.3. Efectos con capas	83
RESUMEN.....	87
ACTIVIDADES	89

EVALUACIÓN	91
 • UNIDAD DIDÁCTICA 3. Pruebas y verificación en páginas web	93
3.1. Técnicas de verificación	95
3.1.1. Fundamentales.....	96
3.1.2. Técnicas HTML	98
3.1.3. Técnicas CSS.....	99
3.2. Herramientas de depuración para distintos navegadores	102
3.2.1. Utilidades para HTML.....	104
3.2.2. Utilidades para JavaScript	104
3.2.3. Utilidades para CSS.....	106
3.2.4. Utilidades para el DOM.....	107
3.3. Verificación de la compatibilidad de <i>scripts</i>	108
3.3.1. Parámetros para distintos navegadores.....	110
3.3.2. Creación de código alternativo para diversos navegadores ..	111
 RESUMEN.....	115
ACTIVIDADES	117
EVALUACIÓN	119
 • RESUMEN FINAL.....	123
 • EVALUACIÓN FINAL	125

• BIBLIOGRAFÍA/WEBGRAFÍA	127
--------------------------------	-----

INTRODUCCIÓN

Este manual se dirige a un lector con conocimientos previos de HTML, CSS y JavaScript. Presenta una serie de cuestiones que se deben tener en cuenta a la hora de desarrollar cualquier proyecto para la web, desde la verificación de los datos introducidos por el usuario a través de un formulario, hasta la corrección técnica y funcional de cada página y del sitio web en su conjunto, pasando por la implementación óptima de distintos efectos especiales en imágenes, textos y otros elementos.

En el primer capítulo se describen algunas funciones de validación junto con las posibilidades actuales a la hora de verificar los datos de un formulario del lado del cliente. Se introduce también el concepto de *plugin* como extensión que permite añadir funcionalidades a un proyecto facilitando el trabajo a los desarrolladores.

El aspecto más visual y estético de un sitio web es abordado en la segunda parte. Conoceremos distintas técnicas para hacer uso de imágenes, textos, animaciones, etc., y veremos qué tipo de pruebas debemos realizar para comprobar el nivel de optimización con respecto al usuario final.

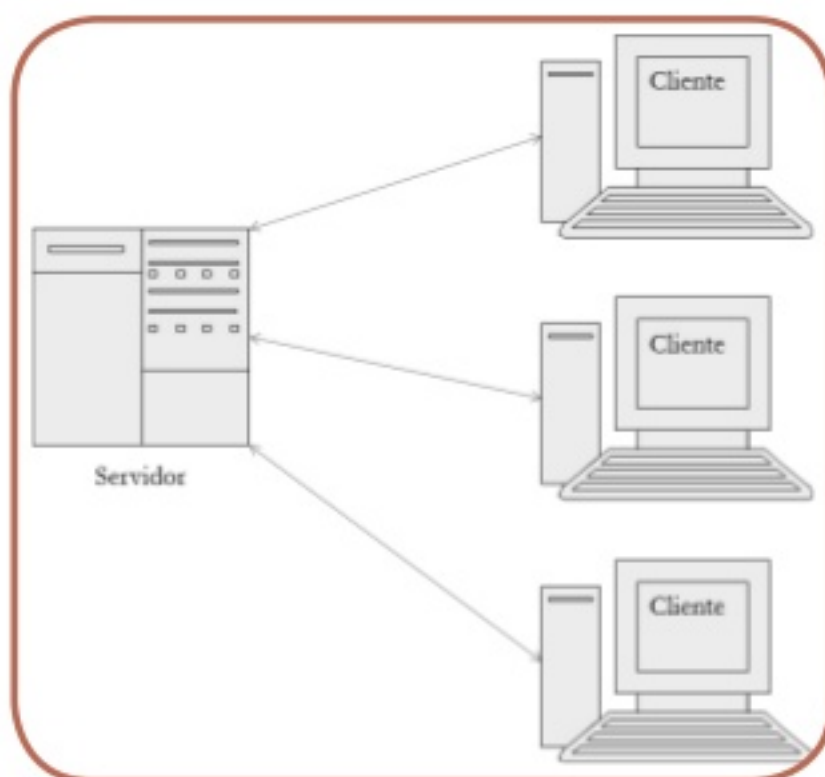
Finalmente, se hará un repaso de las principales herramientas de depuración y se recomendarán algunas utilidades para la detección y corrección de los errores de nuestro código. Veremos además cómo verificar y resolver el problema de la compatibilidad con los distintos estándares y dispositivos que pueblan el panorama actual.

VALIDACIONES DE DATOS EN PÁGINAS WEB

- **CONTENIDO**
 - 1.1. Funciones de validación
 - 1.2. Verificar formularios
- **RESUMEN**
- **ACTIVIDADES**
- **EVALUACIÓN**

Como usuarios de internet interactuamos continuamente con formularios de registro en los que se nos pide que cubramos una serie de campos obligatorios con distintos tipos de datos (nuestro *e-mail*, fecha de nacimiento, etc.) o, una vez ya registrados, con formularios de *login*, de acceso a nuestro correo o a cualquier otro servicio por medio de nuestro usuario y contraseña. JavaScript nació precisamente para mejorar nuestra experiencia como usuarios a la hora de utilizar este tipo de aplicaciones.

Obviamente la validez de la información aportada por nosotros a través de los formularios debe comprobarse antes de ser almacenada en una base de datos. Esta validación, proceso a través del cual verificamos que son correctos cada uno de los campos (la fecha, la dirección de correo, la contraseña, el nombre, etc.), es posible realizarla en el *servidor*, el ordenador remoto donde se almacenarán los datos, o en el *cliente*, el ordenador de origen que está manejando el usuario.



Con JavaScript, que como ya sabemos, es interpretado y ejecutado en el navegador del usuario, nos ahorramos el tiempo que tarda la información en llegar hasta el servidor y volver con el resultado, pues la validación tiene lugar en el ordenador cliente. Además liberamos al servidor, el cual puede estar recibiendo miles de peticiones a la vez, de la tarea de ocuparse de cada una de

ellas. De esta forma, si cometemos algún error al rellenar un formulario, se nos puede notificar instantáneamente sin necesidad de esperar la respuesta de un ordenador localizado a miles de kilómetros.

1.1. Funciones de validación

La validación de un formulario consiste generalmente en la invocación de una función cuando el usuario pulsa sobre el botón de envío o cuando sale de un campo y pasa a otro. Dicha función comprueba si los valores introducidos cumplen las restricciones impuestas por la aplicación.

1.1.1. Descripción de las funciones

Las funciones son bloques de código identificados mediante un nombre. Una vez definida, una función puede ser invocada escribiendo su nombre seguido de dos paréntesis. Cuando el intérprete de JavaScript llega a una línea de código donde se ha llamado a una función, ejecuta el bloque de instrucciones contenido en la misma, de modo que es posible utilizarla varias veces en distintos contextos.



Las funciones en JavaScript se definen mediante la palabra reservada «*function*», seguida del nombre de la función. Su definición formal es la siguiente:

```
function nombre_funcion() {  
// instrucciones a ejecutar cuando se llame a la función  
}
```

La estructura básica de una función genérica de validación de un formulario sería la siguiente:

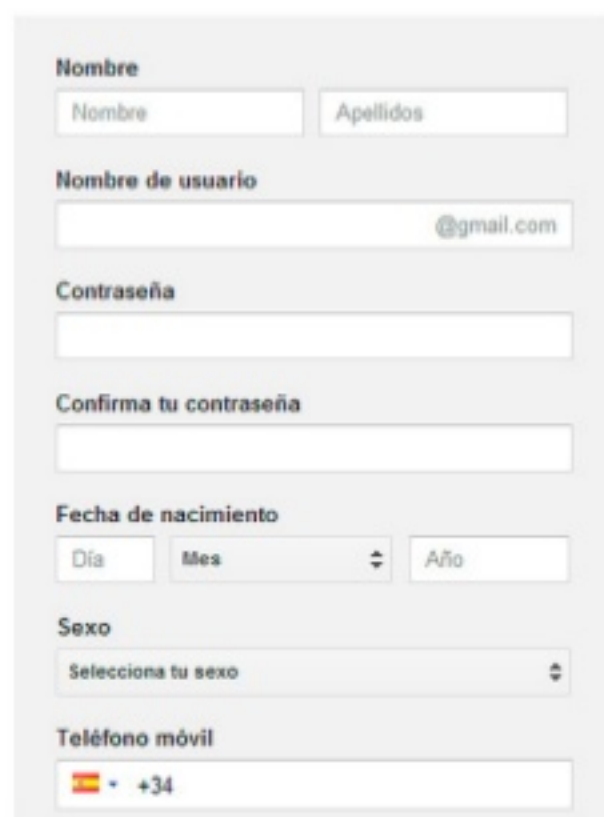
```
function validacion() {  
    if (condición que debe cumplir el primer campo del formulario) {  
        // Si no se cumple la condición...  
        alert('El campo debe tener un valor de...');  
        return false;  
    }  
    else if (condición que debe cumplir el segundo campo del formulario) {  
        // Si no se cumple la condición...  
        alert('El campo debe tener un valor de...');  
        return false;  
    }  
    ...  
    else if (condición que debe cumplir el último campo del formulario) {  
        // Si no se cumple la condición...  
        alert('El campo debe tener un valor de...');  
        return false;  
    }  
    // Si el script ha llegado a este punto, todas las condiciones  
    // se han cumplido, por lo que se devuelve el valor true  
    return true;  
}
```

En este caso la función gestiona cada uno de los campos mediante estructuras de decisión encadenadas (*if else if*). Entre los paréntesis de cada condición escribiríamos una expresión que de ser verdadera mostraría al usuario un mensaje de advertencia informando de que el campo no ha sido cumplimentado

correctamente. Al mismo tiempo, con la sentencia *return* hacemos que la función devuelva un valor *false* que, como veremos, impedirá el envío del formulario.

1.1.2. Utilidad de las funciones

Existen diferentes funciones de validación según el tipo de dato y los requisitos exigidos. Por ejemplo, podemos comprobar si un campo ha sido cubierto o está en blanco, o si una dirección de correo tiene la forma adecuada (es decir, no debe contener espacios en blanco y ha de incluir el carácter @ en medio de dos cadenas de texto y terminar con un punto y otra cadena).



Formulario de registro con los siguientes campos:

- Nombre:** Campos para "Nombre" y "Apellidos".
- Nombre de usuario:** Campo de texto con "@gmail.com" como sugerencia.
- Contraseña:** Campo de texto.
- Confirma tu contraseña:** Campo de texto.
- Fecha de nacimiento:** Campos para "Día", "Mes" (con flecha de selección) y "Año".
- Sexo:** Selector con la opción "Selecciona tu sexo".
- Teléfono móvil:** Campo de texto con un selector de bandera (+34).

En el caso de las fechas, comprobaríamos, por ejemplo, si es una fecha real, contando con que no tenemos más de 12 meses y estos no sobrepasan los 31 días, e incluso si el año en cuestión es bisiesto o no.

1.1.3. Implementación de las funciones

Veamos un ejemplo sencillo donde implementar una función de validación. Partimos de un formulario con un campo de texto (`type="text"`). Al producirse el evento *submit*, es decir, en el momento de pulsar el botón de envío, se invoca a la función. Dentro de la función se comprueba en la condición del *if* si el valor

del campo *nombre* está vacío. De ser así informa al usuario e impide el envío de los datos.

```
<html>

<head>

<title> Ejemplo sencillo de implementación</title>

<script>

    function validar(formulario){

        if (formulario.nombre.value == ""){

            alert("escribe tu nombre para poder enviarlo al servidor");

            return(false);

        }

    }

</script>

</head>

<body>

<h2>Ejemplo muy sencillo de validación</h2>

<form action="recibir.php" onSubmit="return validar(this)">

Escribe tu nombre para enviarlo al servidor:

<input name="nombre" type="text"><input value="enviar" type="submit">

</form>

</body>

</html>
```

1.1.4. Validaciones alfanuméricas, numéricas y de fecha

Las funciones de validación, como estamos viendo, tienen la responsabilidad de contrastar la correcta inserción de la información en los campos de un formulario. Los datos que se espera recibir pueden ser cadenas de caracteres

alfanuméricos (números, letras y otros símbolos), números (únicamente dígitos) o fechas (restringidas al formato de una fecha real del calendario).

Veamos el código de una función que, utilizando una expresión regular, (será explicada en el próximo apartado), obliga al usuario a no dejar en blanco un campo de texto. La condición en JavaScript se puede indicar como sigue:

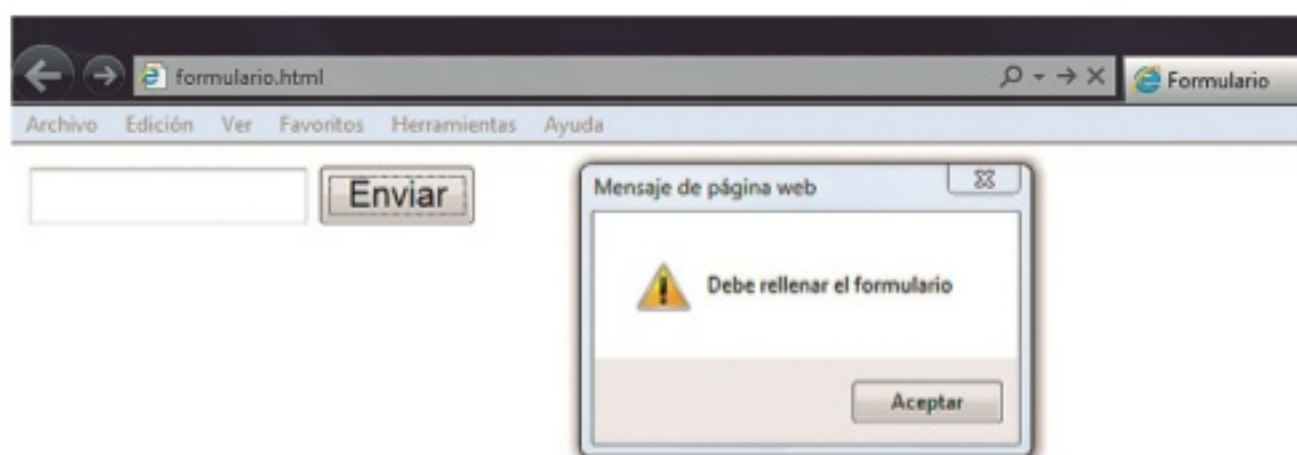
```
if( valor == null || valor.length == 0 || /^s+$/ .test(valor) ) {  
    return false;  
}
```

Para verificar si un campo de texto obligatorio ha sido completado, se comprueba que el valor introducido sea válido, que el número de caracteres sea mayor que cero y que no se hayan insertado solo espacios en blanco.

La palabra reservada *null* es un valor especial que se utiliza para indicar «ningún valor». Si el valor de una variable es *null*, esta no contiene ningún valor de tipo objeto, *array*, numérico, cadena de texto o booleano.

La segunda parte de la condición obliga a que el texto introducido tenga una longitud superior a cero caracteres, esto es, que no sea un texto vacío.

Finalmente, la tercera parte de la condición (*/^s+\$/ .test(valor)*) obliga a que el valor introducido por el usuario no solo esté formado por espacios en blanco. Esta comprobación se basa en el uso de *expresiones regulares*, un recurso muy común en cualquier lenguaje de programación.



Ahora comprobaremos si el usuario ha introducido un valor numérico en un cuadro de texto. En este ejemplo no se hace uso de expresiones regulares sino la

función nativa *isNaN* (*is not a Number*), que devuelve *true* si su argumento no es un número y *false* si lo es:

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

Si el contenido de la variable *valor* no es un número válido no se cumple la condición del *if*. La ventaja de utilizar la función *isNaN()* es que simplifica las comprobaciones, ya que JavaScript se encarga de tener en cuenta los decimales, signos, etc.



Las *fechas* suelen ser los campos de formulario más complicados de validar por la gran variedad de formatos. Recogemos el año, mes y día insertados por el usuario y contrastamos con el objeto *Date*:

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

La función *Date (ano, mes, dia)* es una función nativa de JavaScript que permite construir fechas a partir del año, el mes y el día de la fecha. El número de mes se

indica de 0 a 11, de modo que 0 es el mes de enero y 11 es el mes de diciembre. Los días del mes siguen una numeración diferente, ya que el mínimo permitido es 1 y el máximo 31.

La validación consiste en intentar construir una fecha con los datos proporcionados por el usuario. Si los datos del usuario no son correctos, la fecha no se puede construir correctamente y por tanto la validación del formulario no será correcta.

1.1.5. Definición de validaciones

Una técnica muy poderosa para definir reglas de validación se basa en el uso de expresiones regulares o *regex*, que son secuencias de caracteres que forman un patrón o conjunto de reglas de búsqueda o de contraste. JavaScript incluye de la posibilidad de crear objetos de *regex* con distintos métodos para manejarlos.

Una expresión regular se forma por una sucesión de reglas que determinan paso a paso lo que se busca en un texto. ¹Veamos un ejemplo.

En este caso veremos dos formas de generar la misma expresión regular en JS, y a continuación el uso de la función *test* para contrastarla con una cadena.

```
var expresionRegular1 = new RegExp("[a-z]", "i");  
var expresionRegular2 = /[a-z]/i;  
var cadena="Texto de la a a la z";  
alert(expresionRegular1.test(cadena));  
alert(expresionRegular2.test(cadena));
```

La ejecución de este código generará dos mensajes con el valor *true*, puesto que el resultado de comparar la cadena de texto con el patrón arroja dicho resultado. Esto se debe a que en este caso el patrón *[a-z]* permite cualquier carácter entre la 'a' y la 'z', y el modificador *i*, (también llamado *bandera*), indica que pueden estar en mayúsculas o minúsculas.

¹TERRADILLOS, P., (2013). Introducción a las expresiones regulares [Internet], Fernetjs. Disponible en: <http://fernetjs.com/2013/07/introduccion-a-expresiones-regulares/> [Fecha de acceso: 15/06/2014].

En la tabla siguiente se especifican los caracteres utilizados en la confección de expresiones regulares.

Carácter	Significado	Ejemplo
\	Indica que el siguiente carácter debe ser considerado como especial. También se utiliza como carácter de escape para los caracteres especiales.	/\n/ encuentra un salto de línea. Si se desea buscar el carácter '\', utilizaremos /\
^	Encuentra el comienzo de una línea.	/a/ encuentra una 'a' en cualquier lugar, pero /^a/ halla una coincidencia con el carácter 'a' si se encuentra al comienzo de una línea.
\$	Se utiliza para encontrar el final de una línea.	
*	Encuentra coincidencia del carácter que le precede cuando aparece 0 o más veces en la cadena.	/hola*/ encuentra coincidencias en las cadenas "hol", "hola" y "holaaaa".
+	Igual que el '*', pero cuando el carácter aparece 1 o más veces.	
?	Como el '+', pero cuando el carácter aparece 0 o 1 vez.	
.	Sirve para encontrar cualquier carácter que no sea de nueva línea.	/sa/ encontrará coincidencias en las cadenas "casa", "cosa" y "cesa", pero no en "asa".
(x)	Encuentra coincidencias con 'x', y recuerda el patrón para su posterior utilización.	
x y	Encuentra coincidencia si aparece el carácter 'x' o el 'y'.	
{n}	Encuentra coincidencia si hay exactamente n apariciones del carácter que precede.	Por ejemplo, /a{3}/ encuentra coincidencia en "holaaa", pero no en "holas".
{n,}	Halla coincidencia si hay al menos n apariciones del carácter que precede.	
{n,m}	Encuentra coincidencia si hay como mínimo n y como máximo m apariciones del carácter precedente.	
[xyz]	Representa un conjunto de caracteres individuales.	/[aeiou]/ encuentra coincidencias con cualquier vocal. Si se utiliza el carácter '-' se pueden definir rangos. Por ejemplo, /[0-3]/ encuentra coincidencias si aparecen en la cadena los caracteres '0', '1', '2' o '3'.
[^xyz]	Encuentra coincidencias con aquellos caracteres que no aparezcan en el conjunto. Al igual que en el caso anterior, con el '-' se pueden definir rangos.	
[\\b]	Encuentra coincidencia con el carácter de retroceso.	
\\b	Encuentra coincidencias con los límites de las palabras.	Por ejemplo, /\bola/ encuentra la cadena "ola" en "Viene una ola", pero no en "Viene una cola".

1.1.6. Código de validación

El código de validación es un conjunto de sentencias escritas en un lenguaje de programación cuya finalidad consiste en examinar uno o más datos introducidos por el usuario y comprobar su adecuación a un criterio preestablecido por el programador.

Imaginemos una aplicación para gestionar un concesionario de automóviles en la cual el operador debe cubrir un campo de texto para insertar las matrículas de los mismos. Las matrículas han de tener un formato determinado, ya que de no ser así la aplicación no permitirá almacenar los datos del vehículo. Por ejemplo, estos podrían ser patrones para matrículas válidas: AB-0000-CD y 0000-ABC. Las expresiones regulares con las reglas que debería cumplir cualquier cadena de caracteres para ajustarse a estos patrones serían:

```
var viejaMatricula = /\b[A-Z]{1,2}[-]\d{4}[-][A-Z]{1,2}\b/;
```

```
var nuevaMatricula = /\d{4}[-][A-Z]{3}\b/
```



En otro formulario quizás necesitemos cumplimentar un campo con la dirección de correo electrónico de un cliente. El código de validación en JavaScript podría ser el siguiente:

```
valor = document.getElementById("campo").value;  
if( !( /^[_a-zA-Z0-9-]+\.[_a-zA-Z0-9-]+)*@([_a-zA-Z0-9-]+\.)*[a-zA-Z0-9-]{2,200}\.[a-zA-Z]{2,6}$/.test(valor) ) {  
    return false;  
}
```

De este modo, al menos comprobaríamos si la dirección parece válida,

aunque, para verificar que una dirección es real y pertenece al usuario, deberíamos enviar un mensaje pidiendo su confirmación.

1.1.7. Ejecución del código de validación

El código de validación se ejecuta en el momento en el que es invocada una función de validación. Esta llamada a la función tiene lugar al producirse un evento asociado a la misma, como por ejemplo «*submit*», que en su caso consiste en la pulsación del botón de envío del formulario. A dicha función se le denomina «manejador del evento». Si la función de validación devuelve el valor *true* el formulario se envía, y si en cambio es *false* el formulario no se envía.

Según la estructura básica de una función de validación vista anteriormente, el código va examinando todos y cada uno de los campos y al encontrar uno incorrecto devuelve el valor *false*. En caso contrario, es decir, cuando todo está bien, devuelve el valor *true*.

Una forma de asignar la función de validación como manejador del evento «*onsubmit*» sería de este modo:

```
<form... onsubmit="return validacion()">  
...  
</form>
```

Cuando se ejecuta el código de validación, además de permitir o impedir el envío de los datos insertados en un formulario, debe informar al usuario de lo que está pasando. Del diseño de cada aplicación depende la forma y el momento en el que serán notificados al usuario los errores cometidos. Lo habitual y más recomendable es mostrar cada mensaje junto al campo correspondiente, junto con algún tipo de indicación general de formulario erróneo.

The image shows a web form for password creation. On the left, a light gray box contains a message: "Seguridad de la contraseña: Demasiado corta" (Password security: Too short), followed by a red progress bar and instructions: "Usa ocho caracteres como mínimo. No uses una contraseña de otro sitio ni un nombre demasiado común, como el nombre de tu mascota. ¿Por qué?" (Use at least eight characters. Do not use a password from another site or a too common name, like the name of your pet. Why?). On the right, the form has two input fields. The first is labeled "Contraseña" (Password) and has a red border. The second is labeled "Confirma tu contraseña" (Confirm your password) and also has a red border. Below the second field, a red message says "No puedes dejar este campo en blanco." (You cannot leave this field blank.).

1.2. Verificar formularios

Sabemos que un formulario cuenta con distintos tipos de campos. Hemos visto algunos ejemplos de validación, todos relativos a campos de tipo texto, pero contamos también con otros como *radiobutton*, *checkbox* o *select* también de ser verificados.

1.2.1. Identificación de los datos

Si un elemento de tipo *checkbox* ha de ser seleccionado de forma obligatoria, JavaScript nos permite comprobarlo fácilmente:

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

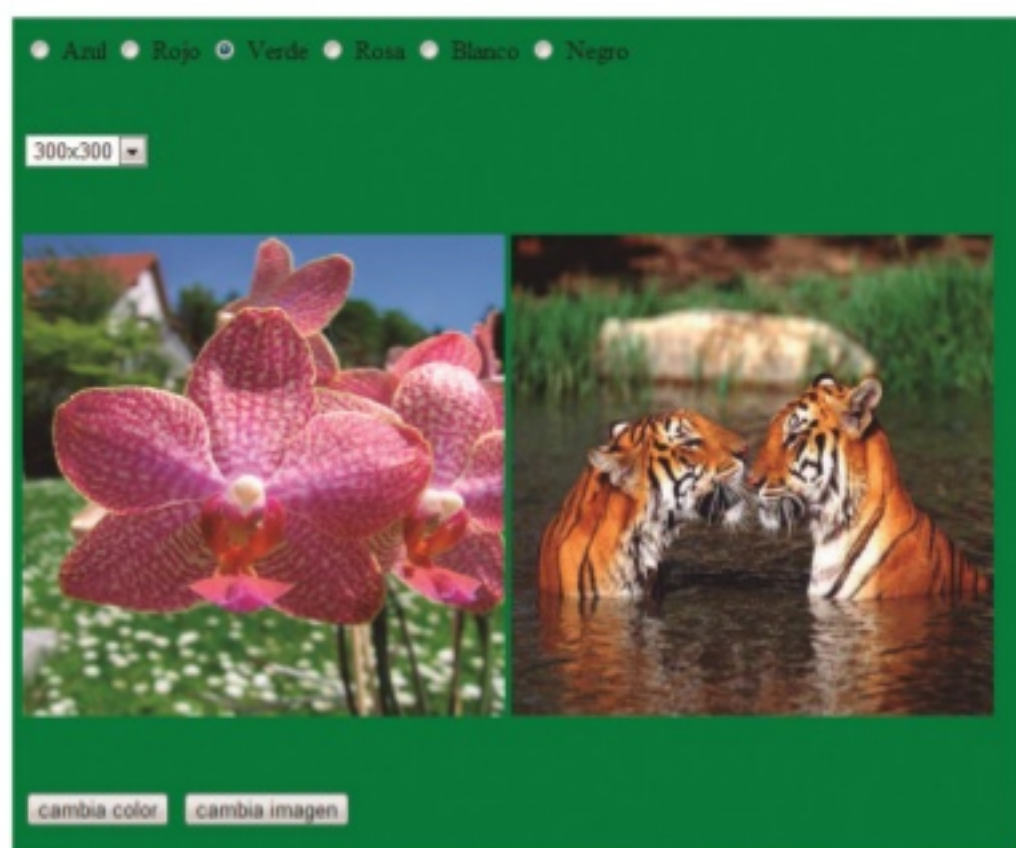
☐ Azul ☐ Rojo ☐ Verde ☐ Rosa ☐ Blanco ☐ Negro

100x100 ▼



Si se trata de comprobar que todos los *checkbox* del formulario han sido seleccionados, lo recomendable sería utilizar un bucle:

```
formulario = document.getElementById("formulario");  
for(var i=0; i<formulario.elements.length; i++) {  
    var elemento = formulario.elements[i];  
    if(elemento.type == "checkbox") {  
        if(!elemento.checked) {  
            return false;  
        }  
    }  
}
```



La validación de los *radiobutton*, aunque similar a la de los *checkbox*, presenta una diferencia relevante: la comprobación que se realiza es que el usuario haya seleccionado alguno de los *radiobutton* que forman un determinado grupo. Mediante JS es sencillo determinar si se ha seleccionado algún *radiobutton*:

```

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

```

El anterior ejemplo recorre todos los *radiobutton* del grupo *opciones* y, mediante un bucle *for*, comprueba elemento por elemento si ha sido seleccionado. Al toparse con el primer *radiobutton* seleccionado sale del bucle y devuelve *true*. Si al terminar el bucle la variable *seleccionado* sigue siendo *false*, esto significará que ninguna de las opciones ha sido elegida.

Asimismo, es muy probable que necesitemos verificar que al menos una opción de un campo *select* ha sido seleccionada. Imaginemos este ejemplo:

```

<form name="fvalida"...>
<select name="tam">
<option value="100">100x100</option>
<option value="200">200x200</option>
<option value="300">300x300</option>
</select>
...
</form>

```

Por último, para comprobar si se ha escogido alguno de los valores:

```

if (document.fvalida.tam.selectedIndex==0){

```

```

    alert("Debe seleccionar un motivo de su contacto.")

    document.fvalida.interes.focus()

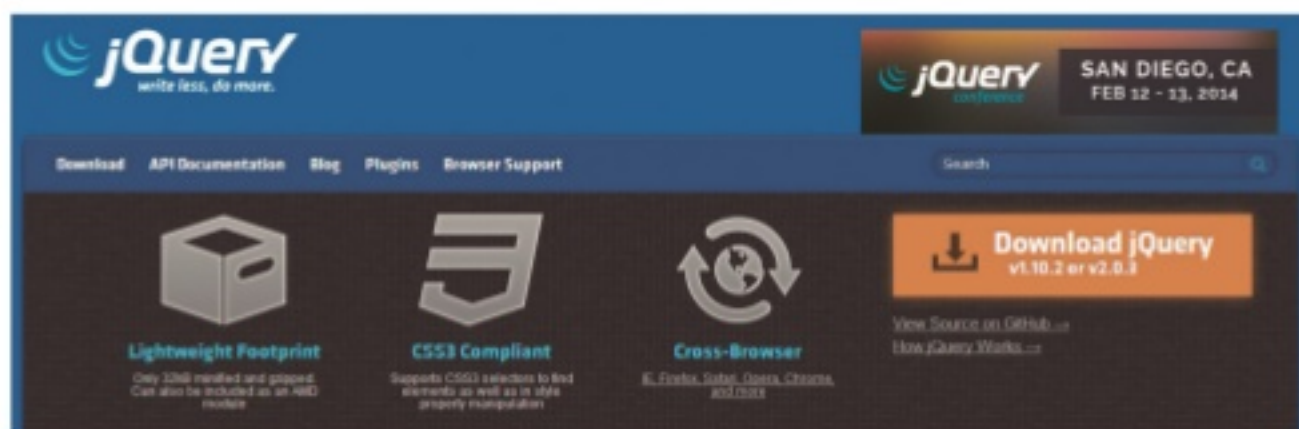
    return 0;
}

```

1.2.2. Implementación del código de verificación

El código JS responsable de la verificación puede ser añadido en cualquier parte del documento HTML dentro de la etiqueta `<script/>`. En el caso de utilizar librerías externas, programadas por nosotros o por terceros, enlazaremos los archivos JS donde se encuentran las funciones necesarias para extender las capacidades de nuestra aplicación.

Existen muchas librerías, *plugins* o extensiones gratuitas y libres de derechos programadas en JavaScript para facilitar el trabajo a los desarrolladores. Una de las más conocidas es *jQuery*, de la que existe una ingente cantidad de documentación en la red.²



En el documento HTML, normalmente en la cabecera, `<head>`, se enlazan los archivos indicando la ruta a los mismos en el atributo «src».

<!-- Agregamos una versión de JQuery -->

`<script type="text/javascript" src="lib/jquery-1.9.0.js"></script>`

² ÁLVAREZ, M.A., (2009). Manual de jQuery, [Internet]. DesarrolloWeb. Disponible en: <http://www.desarrolloweb.com/manuales/manual-jquery.html> [Fecha de acceso: 30/11/2014].

<!-- Agregamos unplugin para validar el formulario -->

<script type="text/javascript" src="dist/jquery.validate.min.js"></script>

<script type="text/javascript" src="dist/additional-methods.min.js"></script>

Posteriormente, con la función *ready()* de JQuery nos aseguramos de que el cuerpo del documento, donde se encuentra el formulario, ha sido descargado completamente antes de invocar a las funciones que se ocuparán de verificarlo:

```
$(document).ready(function()  
{...}
```

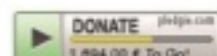
Veámoslo con un ejemplo. Vamos a describir brevemente cómo implementar *jQuery Validation Plugin* para comprobar los datos introducidos por el usuario.

Primero accedemos a la página web desde la que descargar los archivos que enlazaremos: <http://jqueryvalidation.org/>.

jQuery Validation Plugin

This jQuery plugin makes simple clientside form validation easy, whilst still offering plenty of customization options. It makes a good choice if you're building something new from scratch, but also when you're trying to integrate something into an existing application with lots of existing markup. The plugin comes bundled with a useful set of validation methods, including URL and email validation, while providing an API to write your own methods. All bundled methods come with default error messages in english and translations into 37 other languages.

If you want to support the development of this plugin, [please donate to the ongoing pledgie.org campaign](#).



The plugin is written and maintained by Jörn Zaefferer, a member of the [jQuery team](#), lead developer on the jQuery UI team and maintainer of QUnit. It was started back in the early days of jQuery in 2006, and updated and improved since then.

Current version: 1.11.1

License: MIT

Files:

[Download](#)

RECENT POSTS

[jQuery.validator.addClassRules\(\)](#)
[.validate\(\)](#)
[.valid\(\)](#)
[url_method](#)
[:unchecked Selector](#)

RECENT COMMENTS

ARCHIVES

[May 2013](#)

CATEGORIES

[Methods](#)
[Plugin Methods](#)
[Selectors](#)
[Validator](#)

Una vez descargado el paquete, descomprimos los archivos y accedemos

a la carpeta *demo*. Podemos ver una demostración abriendo el archivo *index.html*.

Nombre	Tipo	Tamaño
demo	Carpeta de archivos	
dist	Carpeta de archivos	
lib	Carpeta de archivos	
localization	Carpeta de archivos	
test	Carpeta de archivos	
changelog.txt	Documento de tex...	28 KB
package.json	Archivo JSON	1 KB
README.md	Archivo MD	2 KB

1.2.3. Comprobación de los datos introducidos por el usuario

En la demo *jQuery Validation Plugin* observamos que el *e-mail* se valida con un campo HTML5³. Sin necesidad de utilizar JS muestra un mensaje apropiado al valor requerido:

```
<input id="email" type="email" name="email" required />
```

Default submitHandler is set to display an alert into of submitting the form

Please provide your name, email address (won't be published) and a comment

Name (required, at least 2 characters)	<input type="text" value="Ariel Santiago Castro"/>
E-Mail (required)	<input type="text" value="arielsantiagoc"/>
URL (optional)	<input type="text" value="w.infosofia"/>
Your comment (required)	<input type="text" value="Un comen"/>
<input type="button" value="Submit"/>	

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

En el archivo *README.md* se explica cómo incluir el *plugin* en nuestros proyectos. Debemos asegurarnos de que enlazamos bien las librerías *jQuery*. Tenemos toda la documentación en <http://jqueryvalidation.org/documentation>.

³ MARTÍNEZ ORTIZ, T.C., (2012), Validación de formularios nativos con HTML5, Html5Facil[Internet]. Disponible en: <http://html5facil.com/tutoriales/validacion-de-formularios-nativos-con-html5> [Fecha de acceso: 30/11/2013].

Veamos un ejemplo como el de la imagen:



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//
EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
```

```
<title> Validar formularios con JQuery</title>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

```
<style type="text/css">
```

```
body { background-color:lightyellow;font-family:Arial;font-size:10pt; }
```

```
label { width:200px; }
```

```
input[type="text"] { width:200px;margin:2px; }
```

```
    #btnEnviar { margin-top:20px; }
```

```
</style>
```

```
<!-- Agregamos JQuery -->
```

```
<script type="text/javascript" src="lib/jquery-1.9.0.js"></script>
```

```
<!-- Agregamos el plugin para validar el formulario -->
```

```
<script type="text/javascript" src="dist/jquery.validate.min.js"></script>
```

```
<script type="text/javascript" src="dist/additional-methods.min.js"></script>
```

```
<script type="text/javascript">
```

```

$(document).ready(function()
{
    $("#form1").validate({
    rules: {
        "txtNombre": { required:true, lettersonly:true },
        "txtApellidos": { required:true, lettersonly:true },
        "txtFNac": { dateITA:true },
        "txtNIF": { rangelength:[9,9] }, // Formato
        "txtCPostal": { required:true, digits:true, rangelength:[5,5] },
        "txtMail": { required:true, email:true },
        "txtBlog": { url:true }
            },
            messages: {
                "txtNombre": { required:"Introduce el nombre", lettersonly:"Solo se admiten letras en el nombre"},
                "txtApellidos": { required:"Introduce los apellidos", lettersonly:"Solo se admiten letras en los apellidos"},
                "txtFNac": { dateITA:"<img src='error.gif' alt='' /> Formato de fecha no válido" },
                "txtNIF": "Formato de NIF incorrecto",
                "txtCPostal": { required:"Escribe un código postal válido", digits:"Solo pueden haber números en el código postal", rangelength:"El código postal debe contener cinco dígitos" },
                "txtMail": { required:"Introduce tu E-Mail", email:"La dirección E-Mail no tiene formato correcto" },
                "txtBlog": "URL no válida"
            },
            submitHandler: function(form){
                alert("Los datos son correctos");
            }
    });
}

```

```

    }

    });

    });
</script>
</head>
<body>
<form name="form1" id="form1" method="post" action="#">
<label for="txtNombre">Nombre (*):</label>
<input type="text" name="txtNombre" id="txtNombre" size="20" /><br />
<label for="txtApellidos">Apellidos (*):</label>
<input type="text" name="txtApellidos" id="txtApellidos" size="40" /><br />
<label for="txtFNac">F/nacimiento (dd/mm/yyy):</label>
<input type="text" name="txtFNac" id="txtFNac" /><br />
<label for="txtNIF">N.I.F.:</label>
<input type="text" name="txtNIF" id="txtNIF" /><br />
<label for="txtCPostal">Código postal (*):</label>
<input type="text" name="txtCPostal" id="txtCPostal" /><br />
<label for="txtMail">E-Mail (*):</label>
<input type="text" name="txtMail" id="txtMail" /><br />
<label for="txtBlog">Blog personal:</label>
<input type="text" name="txtBlog" id="txtBlog" size="40" /><br />
<input type="submit" name="btnEnviar" id="btnEnviar" value="Enviar ">>
/>

</form>
</body>
</html>

```

Nombre (*): Introduce el nombre

Apellidos (*): Introduce los apellidos

F/nacimiento (dd/mm/yyyy):

N.I.F.:

Código postal (*): Escribe un código postal válido

E-Mail (*): Introduce tu E-Mail

Blog personal:

En la última imagen se observa el formulario y los mensajes de validación que advierten al usuario de los campos obligatorios o de los errores de formato del *e-mail*, de la fecha y del código postal.

RESUMEN

En este capítulo hemos tratado de una de las principales tareas para las que se utiliza el lenguaje de programación JavaScript: la validación de datos del lado del cliente en la gestión de formularios.

Se han visto algunos ejemplos de funciones de validación para diferentes tipos de campos de formulario y la posibilidad de extender las capacidades de nuestros proyectos utilizando *plugins* o bibliotecas especialmente creados para tales fines, describiendo su instalación y configuración con un ejemplo concreto.

ACTIVIDADES

- 1) Prueba los ejemplos de este tema elaborando formularios sencillos y poniendo a prueba las funciones descritas para cada tipo de campo.
- 2) Consulta la documentación del *pluginjQuery Validation* y elabora un ejemplo de implementación. Añade comentarios explicativos al código y recógelos posteriormente como parte de la documentación de tu propio proyecto.

EVALUACIÓN

Crea un formulario como el de la imagen y programa la función o funciones que validen todos los campos. Ninguno puede quedar en blanco y todos deben ajustarse al tipo de dato que contienen. Informa al usuario de los posibles errores en cuanto salga de un campo para entrar en otro y también cuando pulse el botón Validar. Elabora un documento describiendo tu trabajo.

Nombre:	<input type="text"/>
Primer apellido:	<input type="text"/>
Email:	<input type="text"/>
Tel. Móvil:	<input type="text"/>
Dirección:	<input type="text"/>
Población:	<input type="text"/>
Provincia:	<input type="text" value="Selecciona..."/>
Estado civil:	<input type="radio"/> Soltero <input type="radio"/> Casado
Cada cuanto te conectas?:	<input type="checkbox"/> Todos los días <input type="checkbox"/> Un par de veces por semana <input type="checkbox"/> Un par de veces por mes <input type="checkbox"/> Un par de veces al año
<input type="button" value="Validar"/>	