

# **UF1306:**

# **PRUEBAS DE FUNCIONALIDADES Y OPTIMIZACIÓN DE PÁGINAS WEB**

**Manual teórico.**

## INDICE

1.	VALIDACIÓN DE DATOS EN PÁGINAS WEB .....	4
1.1.	Introducción .....	4
1.2.	Funciones De Validación.....	5
1.2.1.	Validar un campo de texto obligatorio .....	7
1.2.2.	Validar un campo de texto con valores numéricos .....	8
1.2.3.	Validar que se ha seleccionado una opción de una lista .....	9
1.2.4.	Validar una dirección de email .....	9
1.2.5.	Validar una fecha .....	10
1.2.6.	Validar un número de DNI .....	11
1.2.7.	Validar un número de teléfono.....	11
1.2.8.	Validar que un checkbox ha sido seleccionado.....	14
1.2.9.	Validar que un radiobutton ha sido seleccionado .....	14
1.3.	Ejemplos de validación de datos en JavaScript.....	15
1.3.1.	Ejemplo Validación De Textos.....	15
1.3.2.	Ejemplo Validación De Números .....	18
1.3.3.	Ejemplo Validación De Contraseña .....	20
2.	Efectos especiales en JavaScript.....	23
2.1.	Introducción .....	23
2.2.	Gestión de tiempo en JavaScript .....	23
2.3.	Efectos especiales con imágenes.....	26
2.3.1.	Obtención y optimización de imágenes .....	26
2.3.2.	El Objeto image .....	27
2.3.3.	Precarga de imágenes.....	29
2.3.4.	Imágenes de sustitución .....	30
2.3.5.	Galería de imágenes .....	31
2.4.	Efectos especiales en textos .....	34
2.5.	Efectos especiales en marcos y capas.....	36
2.6.	Creación de ventanas .....	40
3.	Pruebas y verificación en páginas web.....	43
3.1.	Introducción .....	43
3.2.	Técnicas de verificación.....	43
3.2.1.	Verificaciones fundamentales .....	43
3.2.2.	Verificación HTML.....	44



3.2.3.	Verificación CSS .....	45
3.3.	Herramientas de depuración para navegadores .....	45
3.4.	Compatibilidad en distintos navegadores .....	47
3.4.1.	Identificación del navegador .....	47
3.4.2.	Ejecución de distinto código según el navegador .....	50
4.	Enlaces. ....	51

## 1. VALIDACIÓN DE DATOS EN PÁGINAS WEB

### 1.1. Introducción

Una vez visto, en la unidad formativa anterior, la funcionalidad de JavaScript, pasamos a una de las tareas donde JavaScript es más potente: **la validación de los datos introducidos por el usuario.**

Pensemos en los datos que un usuario introduce mediante un formulario cuando compramos un billete de avión, reservamos o compramos una entrada de teatro, cine..., hacemos la reserva de un hotel, etc, etc, los servidores pueden estar recibiendo, de forma simultánea, cientos de peticiones, por lo que es fundamental que los datos enviados sean correctos y evitar un ida y vuelta de datos inútiles entre cliente y servidor.

Es muy importante destacar que esta validación en el cliente es únicamente para evitar el tráfico de ida y vuelta con datos erróneos, la última validación de los datos enviados debe ser ejecutada en el propio servidor con lenguajes de programación más avanzados como: **PHP, ASP, JSP, PERL.**

La principal utilidad de JavaScript en el manejo de los formularios es la validación de los datos introducidos por los usuarios. Antes de enviar un formulario al servidor, se recomienda validar mediante JavaScript los datos insertados por el usuario. De esta forma, si el usuario ha cometido algún error al rellenar el formulario, se le puede notificar de forma instantánea, sin necesidad de esperar la respuesta del servidor.

Notificar los errores de forma inmediata mediante JavaScript mejora la satisfacción del usuario con la aplicación (lo que técnicamente se conoce como *"mejorar la experiencia de usuario"*) y ayuda a reducir la carga de procesamiento en el servidor.

Normalmente, la validación de un formulario consiste en llamar a una función de validación cuando el usuario pulsa sobre el botón de envío del formulario. En esta función, se comprueban si los valores que ha introducido el usuario cumplen las restricciones impuestas por la aplicación.

Aunque existen tantas posibles comprobaciones como elementos de formulario diferentes, algunas comprobaciones son muy habituales: que se rellene un campo obligatorio, que se seleccione el valor de una lista desplegable, que la

dirección de email indicada sea correcta, que la fecha introducida sea lógica, que se haya introducido un número donde así se requiere, etc.

## 1.2. Funciones De Validación

Normalmente, la validación de un formulario consiste en llamar a una función de validación cuando el usuario pulsa sobre el botón de envío del formulario. En esta función, se comprueban si los valores que ha introducido el usuario cumplen las restricciones impuestas por la aplicación.

Aunque existen tantas posibles comprobaciones como elementos de formulario diferentes, algunas comprobaciones son muy habituales: que se rellene un campo obligatorio, que se seleccione el valor de una lista desplegable, que la dirección de email indicada sea correcta, que la fecha introducida sea lógica, que se haya introducido un número donde así se requiere, etc.

A continuación se muestra el código JavaScript básico necesario para incorporar la validación a un formulario:

---

```
<form action="" method="" id="" name="" onsubmit="return validacion()">
```

```
...
```

```
</form>
```

---

Y el esquema de la función **validacion()** es el siguiente:

---

```
function validacion() {  
    if (condicion que debe cumplir el primer campo del formulario) {  
        // Si no se cumple la condicion...  
        alert('[ERROR] El campo debe tener un valor de...');  
        return false;  
    }  
    else if (condicion que debe cumplir el segundo campo del formulario) {  
        // Si no se cumple la condicion...  
        alert('[ERROR] El campo debe tener un valor de...');  
        return false;  
    }  
}
```

...

```
else if (condicion que debe cumplir el último campo del formulario) {
```

```
    // Si no se cumple la condicion...
```

```
    alert('[ERROR] 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;
```

```
}
```

---

El funcionamiento de esta técnica de validación se basa en el comportamiento del evento **onsubmit** de JavaScript. Al igual que otros eventos como **onclick** y **onkeypress**, el evento '**onsubmit**' varía su comportamiento en función del valor que se devuelve.

Así, si el evento **onsubmit** devuelve el valor **true**, el formulario se envía como lo haría normalmente. Sin embargo, si el evento **onsubmit** devuelve el valor **false**, el formulario no se envía. La clave de esta técnica consiste en comprobar todos y cada uno de los elementos del formulario. En cuando se encuentra un elemento incorrecto, se devuelve el valor **false**. Si no se encuentra ningún error, se devuelve el valor **true**.

Por lo tanto, en primer lugar se define el evento **onsubmit** del formulario como:

```
onsubmit="return validacion()"
```

Como el código JavaScript devuelve el valor resultante de la función **validacion()**, el formulario solamente se enviará al servidor si esa función devuelve **true**. En el caso de que la función **validacion()** devuelva **false**, el formulario permanecerá sin enviarse.

Dentro de la función **validacion()** se comprueban todas las condiciones impuestas por la aplicación. Cuando no se cumple una condición, se devuelve **false** y por tanto el formulario no se envía. Si se llega al final de la

función, todas las condiciones se han cumplido correctamente, por lo que se devuelve **true** y el formulario se envía.

La notificación de los errores cometidos depende del diseño de cada aplicación. En el código del ejemplo anterior simplemente se muestran mensajes mediante la función **alert()** indicando el error producido. Las aplicaciones web mejor diseñadas muestran cada mensaje de error al lado del elemento de formulario correspondiente y también suelen mostrar un mensaje principal indicando que el formulario contiene errores.

Una vez definido el esquema de la función **validacion()**, se debe añadir a esta función el código correspondiente a todas las comprobaciones que se realizan sobre los elementos del formulario. A continuación, se muestran algunas de las validaciones más habituales de los campos de formulario.

### 1.2.1. Validar un campo de texto obligatorio

Se trata de forzar al usuario a introducir un valor en un cuadro de texto o textarea en los que sea obligatorio. La condición en JavaScript se puede indicar como:

---

```
valor = document.getElementById("campo").value;

if( valor == null || valor.length == 0 || /^\\s+$/.test(valor) ) {

    return false;

}
```

---

Para que se dé por completado un campo de texto obligatorio, se comprueba que el valor introducido sea válido, que el número de caracteres introducido sea mayor que cero y que no se hayan introducido sólo 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**, la variable 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.

Por último, la tercera parte de la condición (**/^\\s+\$/.test(valor)**) obliga a que el valor introducido por el usuario no sólo esté formado por espacios en blanco. Esta comprobación se basa en el uso de **"expresiones regulares"**, un recurso habitual en cualquier lenguaje de programación y de gran complejidad. Por lo

tanto, sólo es necesario copiar literalmente esta condición, poniendo especial cuidado en no modificar ningún carácter de la expresión.

### 1.2.2. Validar un campo de texto con valores numéricos

Se trata de obligar al usuario a introducir un valor numérico en un cuadro de texto. La condición JavaScript consiste en:

---

```
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. La ventaja de utilizar la función interna **isNaN()** es que simplifica las comprobaciones, ya que JavaScript se encarga de tener en cuenta los decimales, signos, etc.

A continuación se muestran algunos resultados de la función **isNaN()**:

---

```
isNaN(3);           // false
isNaN("3");         // false
isNaN(3.3545);      // false
isNaN(32323.345);   // false
isNaN(+23.2);       // false
isNaN("-23.2");     // false
isNaN("23a");       // true
isNaN("23.43.54");  // true
```

---



### 1.2.3. Validar que se ha seleccionado una opción de una lista

Se trata de obligar al usuario a seleccionar un elemento de una lista desplegable. El siguiente código JavaScript permite conseguirlo:

---

```
indice = document.getElementById("opciones").selectedIndex;

if( indice == null || indice == 0 ) {

    return false;

}
```

---

```
<select id="opciones" name="opciones">

    <option value="">- Selecciona un valor -</option>

    <option value="1">Primer valor</option>

    <option value="2">Segundo valor</option>

    <option value="3">Tercer valor</option>

</select>
```

---

A partir de la propiedad **selectedIndex**, se comprueba si el índice de la opción seleccionada es válido y además es distinto de cero. La primera opción de la lista (**- Selecciona un valor -**) no es válida, por lo que no se permite el valor **0** para esta propiedad **selectedIndex**.

### 1.2.4. Validar una dirección de email

Se trata de obligar al usuario a introducir una dirección de email con un formato válido. Por tanto, lo que se comprueba es que la dirección *parezca* válida, ya que no se comprueba si se trata de una cuenta de correo electrónico real y operativa. La condición JavaScript consiste en:

---

```
valor = document.getElementById("campo").value;

if( !(/\w+([-+.']\w+)*@\w+([-.\w+)*\.\w+([-.\w+)]/).test(valor)) )
{

    return false;

}
```

---

La comprobación se realiza nuevamente mediante las expresiones regulares, ya que las direcciones de correo electrónico válidas pueden ser muy diferentes. Por otra parte, como el estándar que define el formato de las direcciones de correo electrónico es muy complejo, la expresión regular anterior es una simplificación.

Aunque esta regla valida la mayoría de direcciones de correo electrónico utilizadas por los usuarios, no soporta todos los diferentes formatos válidos de email.

### 1.2.5. Validar una fecha

Las fechas suelen ser los campos de formulario más complicados de validar por la multitud de formas diferentes en las que se pueden introducir. El siguiente código asume que de alguna forma se ha obtenido el año, el mes y el día introducidos por el usuario:

---

```
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 interna de JavaScript que permite construir fechas a partir del año, el mes y el día de la fecha. Es muy importante tener en cuenta que **el número de mes se indica de 0 a 11, siendo 0 el mes de Enero y 11 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.

### 1.2.6. Validar un número de DNI

Se trata de comprobar que el número proporcionado por el usuario se corresponde con un número válido de Documento Nacional de Identidad o DNI. Aunque para cada país o región los requisitos del documento de identidad de las personas pueden variar, a continuación se muestra un ejemplo genérico fácilmente adaptable. La validación no sólo debe comprobar que el número esté formado por ocho cifras y una letra, sino que también es necesario comprobar que la letra indicada es correcta para el número introducido:

---

```
valor = document.getElementById("campo").value;

var letras = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B', 'N', 'J', 'Z', 'S',
, 'Q', 'V', 'H', 'L', 'C', 'K', 'E', 'I'];

if( !(/^\\d{8}[A-Z]$/.test(valor)) ) {

    return false;

}

if(valor.charAt(8) != letras[(valor.substring(0, 8))%23]) {

    return false;

}
```

---

La primera comprobación asegura que el formato del número introducido es el correcto, es decir, que está formado por 8 números seguidos y una letra. Si la letra está al principio de los números, la comprobación sería `/^[A-Z]\\d{8}$/`. Si en vez de ocho números y una letra, se requieren diez números y dos letras, la comprobación sería `/^\\d{10}[A-Z]{2}$/` y así sucesivamente.

La segunda comprobación aplica el algoritmo de cálculo de la letra del DNI y la compara con la letra proporcionada por el usuario. El algoritmo de cada documento de identificación es diferente, por lo que esta parte de la validación se debe adaptar convenientemente.

### 1.2.7. Validar un número de teléfono

Los números de teléfono pueden ser indicados de formas muy diferentes: con prefijo nacional, con prefijo internacional, agrupado por pares, separando los números con guiones, etc.

El siguiente script considera que un número de teléfono está formado por nueve dígitos consecutivos y sin espacios ni guiones entre las cifras:

---

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

---

Una vez más, la condición de JavaScript se basa en el uso de expresiones regulares, que comprueban si el valor indicado es una sucesión de nueve números consecutivos. A continuación se muestran otras expresiones regulares que se pueden utilizar para otros formatos de número de teléfono:

Número	Expresión regular	Formato
900900900	/^\d{9}\$/	9 cifras seguidas
900-900-900	/^\d{3}-\d{3}-\d{3}\$/	9 cifras agrupadas de 3 en 3 y separadas por guiones
900 900900	/^\d{3}\s\d{6}\$/	9 cifras, las 3 primeras separadas por un espacio
900 90 09 00	/^\d{3}\s\d{2}\s\d{2}\s\d{2}\$/	9 cifras, las 3 primeras separadas por un espacio, las siguientes agrupadas de 2 en 2
(900) 900900	/^\(\d{3}\)\s\d{6}\$/	9 cifras, las 3 primeras encerradas por paréntesis y un espacio de separación respecto del resto
+34 900900900	/^\+\d{2,3}\s\d{9}\$/	Prefijo internacional (+ seguido de 2 o 3 cifras), espacio en blanco y 9 cifras consecutivas

### 1.2.8. Validar que un checkbox ha sido seleccionado

Si un elemento de tipo *checkbox* se debe seleccionar de forma obligatoria, JavaScript permite comprobarlo de forma muy sencilla:

---

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

---

Si se trata de comprobar que todos los *checkbox* del formulario han sido seleccionados, es más fácil 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;  
  
        }  
  
    }  
  
}
```

---

### 1.2.9. Validar que un radiobutton ha sido seleccionado

Aunque se trata de un caso similar al de los *checkbox*, la validación de los *radiobutton* presenta una diferencia importante: en general, la comprobación que se realiza es que el usuario haya seleccionado algún *radiobutton* de los que forman un determinado grupo. Mediante JavaScript, es sencillo determinar si se ha seleccionado algún *radiobutton* de un grupo:

```
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** que forman un grupo y comprueba elemento por elemento si ha sido seleccionado. Cuando se encuentra el primer **radiobutton** seleccionado, se sale del bucle y se indica que al menos uno ha sido seleccionado.

### 1.3. Ejemplos de validación de datos en JavaScript

#### 1.3.1. Ejemplo Validación De Textos

En el siguiente ejemplo, la función valida que el campo "alias" tenga una longitud entre 5 y 10 caracteres y que además esté en mayúsculas.

```
<!doctype html>

<html>

<head>

<meta charset="iso-8859-1">

<title>validar texto</title>

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

function validar() {

    var texto=document.getElementById("textoalias").value;

    if (texto.length<5 || texto.length>10){
```

```
    alert ("longitud de la cadena invalida");

    document.getElementById("textoalias").value="";

    return;

}

if (texto!=texto.toUpperCase()) {

    alert ("el texto no está en mayúsculas");

    document.getElementById("textoalias").value="";

    return;

}

}

</script>

<style type="text/css">

#contenedor {

    background-color: #FF9;

    height: auto;

    width: 35%;

    margin-right: auto;

    margin-left: auto;

    padding: 10px;

    border-width: 2px;

    border-style: groove;

    -webkit-border-radius: 16px;

    -moz-border-radius: 16px;

    border-radius: 16px;

}

label {

    color: #903;

}
```



```
.input_text {  
    -webkit-border-radius: 6px;  
    -moz-border-radius: 6px;  
    border-radius: 6px;  
}  
  
</style>  
  
</head>  
  
<body>  
  
<h3 align="center">EJEMPLO ALIAS ENTRE 5 Y 10 CARACTERES Y SOLO  
PERMITIR MAYÚSCULAS</h3>  
  
<div id="contenedor">  
  
<form name="formulariotexto" method="post" action="">  
  
<table width="99%" border="0" cellspacing="0" cellpadding="0">  
  
    <tr>  
  
        <td width="58%"><label for="textfield">ALIAS:</label></td>  
  
        <td width="42%"><input type="text" class="input_text"  
id="textoalias" onblur="validar();"></td>  
  
    </tr>  
  
    <tr>  
  
        <td><label for="textfield2">NOMBRE:</label></td>  
  
        <td><input type="text" class="input_text"  
id="textonombre"></td>  
  
    </tr>  
  
    <tr>  
  
        <td><label for="textfield3">APELLIDOS:</label></td>  
  
        <td><input type="text" class="input_text"  
id="textoapellidos"></td>  
  
    </tr>  
  
</table>
```

```
</form>
```

```
</div>
```

```
</body>
```

```
</html>
```

### 1.3.2. Ejemplo Validación De Números

El siguiente ejemplo muestra un documento con una caja de texto para introducir una edad entre 0 y 120, informando de error si el texto introducido no es un número o bien no está comprendido en el rango válido.

```
<!doctype html>
```

```
<html>
```

```
<head>
```

```
<meta charset="iso-8859-1">
```

```
<title>validar texto</title>
```

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

```
function validar() {
```

```
    var edad=document.getElementById("textoedad").value;
```

```
    if (isNaN(edad)) {
```

```
        alert ("La edad introducida no es un numero");
```

```
        return;
```

```
    }
```

```
    if (edad<0 || edad >120) {
```

```
        alert ("Valor de edad erroneo");
```

```
        return;
```

```
    }
```

```
    alert ("Tu edad es: "+edad);
```

```
}
```

```
</script>
```

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

```
#contenedor {
```

```
    background-color: #FF9;
```

```
    height: auto;
```

```
    width: 35%;
```

```
    margin-right: auto;
```

```
    margin-left: auto;
```

```
    padding: 10px;
```

```
    border-width: 2px;
```

```
    border-style: groove;
```

```
    -webkit-border-radius: 16px;
```

```
    -moz-border-radius: 16px;
```

```
    border-radius: 16px;
```

```
}
```

```
label {
```

```
    color: #903;
```

```
}
```

```
.input_text {
```

```
    -webkit-border-radius: 6px;
```

```
    -moz-border-radius: 6px;
```

```
    border-radius: 6px;
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h3 align="center">EJEMPLO INTRODUCIR EDAD ENTRE 0 Y 120</h3>
```

```
<div id="contenedor">
```

```
<form name="formulariotexto" method="post" action="">
```

```
<table width="99%" border="0" cellspacing="0" cellpadding="0">
```

```
<tr>

    <td width="58%"><label for="textfield2">NOMBRE:</label></td>

    <td width="42%"><input type="text" class="input_text"
id="textonombre"></td>

</tr>

<tr>

    <td><label for="textfield">EDAD:</label></td>

    <td><input type="text" class="input_text" id="textoedad"
onblur="validar();"></td>

</tr>

</table>

</form>

</div>

</body>

</html>
```

### 1.3.3. Ejemplo Validación De Contraseña

Para realizar esta validación tan detallada, se recorrerá el campo contraseña carácter a carácter y se almacenarán en distintos contadores la cantidad de elementos requeridos, dígitos, mayúsculas, etc.

En el siguiente ejemplo muestra la validación de una contraseña.

```
<!doctype html>

<html>

<head>

<meta charset="iso-8859-1">

<title>validar contraseña</title>

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

function validar() {

    var CantMay=0; var CantMin=0; var CantNum=0;

    var passw=document.getElementById("password").value;
```

```
    if (passw.length < 7 || passw.length> 12) {  
        alert ("Contraseña con longitud inválida");  
        return;  
    }  
  
    for (cont=0; cont<passw.length; cont++) {  
        if ( passw.charAt(cont)>="0" &&  
            passw.charAt(cont)<="9") CantNum++;  
        else {  
            if ( passw.charAt(cont)==passw.charAt(cont).toUpperCase())  
                CantMay++;  
            if ( passw.charAt(cont)==passw.charAt(cont).toLowerCase())  
                CantMin++;  
        }  
    }  
  
    if (CantMay==0 || CantMin==0 || CantNum<2)  
        alert ("Contraseña no cumple criterios de seguridad");  
}  
  
</script>  
  
<style type="text/css">  
#contenedor {  
    background-color: #FF9;  
    height: auto;  
    width: 35%;  
    margin-right: auto;  
    margin-left: auto;  
    padding: 10px;  
    border-width: 2px;  
    border-style: groove;
```

```
-webkit-border-radius: 16px;

-moz-border-radius: 16px;

border-radius: 16px;

}

label {

    color: #903;

}

.input_text {

    -webkit-border-radius: 6px;

    -moz-border-radius: 6px;

    border-radius: 6px;

}

</style>

</head>

<body>

<h3 align="center">EJEMPLO INTRODUCIR CONTRASEÑA ENTRE

7 Y 12 CARACTERES,

<BR>CON ALGUNA MAYÚSCULA, ALGUNA MINÚSCULA Y

AL MENOS 2 DÍGITOS NUMÉRICOS </h3>

<div id="contenedor">

<form name="formulariotexto" method="post" action="">

<table width="99%" border="0" cellspacing="0" cellpadding="0">

<tr>

<td width="58%"><label for="textfield2">NOMBRE:</label></td>

<td width="42%"><input type="text" class="input_text"

id="textonombre"></td>

</tr>

<tr>

<td><label for="textfield">CONTRASEÑA:</label></td>
```

```
<td><input type="text" class="input_text" id="password"
onblur="validar();"></td>

</tr>

</table>

</form>

</div>

</body>

</html>
```

## 2. Efectos especiales en JavaScript

### 2.1. Introducción

Junto con la validación de datos, otro de los aspectos más potentes y utilizados de JavaScript es su capacidad para dotar, a las páginas web, de efectos visuales y convertirlas en páginas más atractivas y fáciles de usar.

Los efectos pueden ser muy diversos:

- Galerías de imágenes.
- Imágenes que al interactuar con ellas sufren algún cambio.
- Elementos en movimiento.
- Textos que cambian su apariencia.
- Menús desplegables.
- Etc.

### 2.2. Gestión de tiempo en JavaScript

Los tiempos de ejecución de código (como podemos retardar la ejecución de partes del código o cómo podemos hacer que se ejecute cíclicamente), es un aspecto común a todos los tipos de efectos especiales que vamos a estudiar en los próximos apartados.

Por ejemplo, si se quiere que un texto cambie de color, deberemos indicarle con qué frecuencia queremos que se produzca el cambio (si cada segundo, cada dos segundos, etc...)

Para controlar estas tareas, JavaScript tiene los siguientes métodos del objeto window:

- **setTimeout(función, milisegundos):** La función **setTimeout** permite utilizar un código para que se ejecute dentro de un período de tiempo determinado (o sugerido) por el segundo parámetro en milisegundos. **Ejecuta la función al cabo de un tiempo indicado en milisegundos.**
- **clearTimeout(variable):** detiene la ejecución de una función llamada con el método **Timeout**.
- **setInterval(función, milisegundos):** Es similar a **setTimeout** pero con carácter repetitivo. **Ejecuta la función cada ciertos milisegundos indicados.**
- **clearInterval(variable):** detiene la ejecución de una función llamada con el método **SetInterval**.

**Ejemplo 1: Mensaje de alerta después de 2 segundos.**

---

```
<script text="javascript">
  function ActivarTiempo()
{
  //se activa el método alert después de 2 segundos
  setTimeout("alert('Han pasado 2 segundos!')",2000);
}
</script>

<input type="button" value="Sale un mensaje después 2 segundos" onclick="ActivarTiempo()"/>
```

---

**Ejemplo 2: Hacer un Reloj, utilizando una función que usa el setTimeout para llamar a la misma función.**

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

  function ActivarReloj(){

/*Obteniendo datos del tiempo:
empezamos creando una variable llamada laHora, y a continuación una variable para la hora, los minutos y los segundos, utilizando los métodos getHours(), getMinutes() y getSeconds() de la función reservada new Date()*/

    var laHora = new Date();

    var horario = laHora.getHours();
```



```
var minuterero = laHora.getMinutes();

var segundero = laHora.getSeconds();

// condicional que hace que aparezca un 0 delante de los minutos del 0
al 9

if(minuterero<10)

    minuterero = "0" + minuterero;

// condicional que hace que aparezca un 0 delante de los segundos del
0 al 9

if(segundero<10)

    segundero = "0" + segundero;

//escribiendo sobre el campo de texto la hora actual

document.getElementById('sonLas').value = horario+":"+minuterero
+":"+segundero;

//Desactivando el boton 'Activar Reloj'

document.getElementById('botonActivar').disabled=true;

//Actualizando la hora cada 1 segundo

ahoraSonLas = setTimeout(ActivarReloj,1000);

}

function DetenerReloj(){

    // Deteniendo el setTimeout

    clearTimeout(ahoraSonLas);

    // Volviendo el boton 'Activar Reloj' a la normalidad

    document.getElementById('botonActivar').disabled=false;

}
```

</script>

</br>

<input type="text" id="sonLas"/></br>

<input type="button" onclick="ActivarReloj()" id="botonActivar" value="Activar Reloj"/>

<input type="button" onclick="DetenerReloj()" value="Detener Reloj"/>

**Nota:**

La propiedad "disabled" se usa para desactivar y activar el primer botón.

## 2.3. Efectos especiales con imágenes

### 2.3.1. Obtención y optimización de imágenes

Las imágenes que almacenemos en nuestro servidor tendrán que cumplir las siguientes características:

- **Dimensiones adecuadas:** Procurar que contengan el ancho y alto en píxeles con el que van a ser presentadas en el documento html.
- **Poco peso:** La rapidez con la que se carga una página está relacionada con el peso de sus imágenes en bytes.
- **Permiso del autor:** Si la imagen no es de nuestra propiedad, siempre debemos solicitar permiso al autor para incluirla en nuestro sitio web o bien emplear imágenes libres de derechos. En la web existen diferentes repositorios de imágenes en los que podemos descargar y utilizar imágenes sin coste alguno. Algunos ejemplos:

AllTheFreeStock es un gran repositorio de recursos donde <b>descargar imágenes gratuitas</b> . Son totalmente libres de derechos.	<a href="http://allthefreestock.com/">http://allthefreestock.com/</a>
En Deviant Art podrás encontrar <b>fotografías y dibujos artísticos gratuitos</b> . Suelen ir acompañadas del nombre del autor en la zona inferior. Página web con <b>imágenes</b> muy curiosas	<a href="http://www.deviantart.com/">http://www.deviantart.com/</a>

Europeana ahora cuenta con más de 26.000 <b>imágenes bajo licencia CC0</b> , así como más de 3,6 millones de obras marcadas como <b>dominio público</b> .	<a href="http://www.europeana.eu/">http://www.europeana.eu/</a>
Fotolia es uno de los <b>bancos de imágenes premium</b> más grandes del mercado, además de uno de los más conocidos y recomendables. Tiene una sección de imágenes gratis semanales (el enlace está en el pie de página).	<a href="https://es.fotolia.com/">https://es.fotolia.com/</a>
En este <b>banco de imágenes</b> podrás descargar las <b>fotos gratuitamente</b> , aunque te recuerdan que indiques la autoría con un enlace. Por un ejemplo: <i>Foto cortesía de "autor" at FreeDigitalPhotos.net</i> .	<a href="http://www.freedigitalphotos.net/">http://www.freedigitalphotos.net/</a>
Es la galería de <b>fotografía gratuita</b> más popular. Cuenta con más de 350.000 <b>fotos gratis</b> de gran calidad hechas por más de 30.000 fotógrafos. Puedes usarla para divulgar tus propias fotografías.	<a href="http://www.freeimages.com/">http://www.freeimages.com/</a>

## 2.3.2. El Objeto image

Java Script dispone de un objeto llamado image que permite gestionar las imágenes de cualquier página web. Sus propiedades más importantes son:

- **complete:** contiene el valor verdadero si el navegador ha cargado completamente la imagen. Contiene el valor falso en caso contrario.
- **height/width:** contiene el valor del atributo HTML, con el mismo nombre, que representa el alto/ancho respectivamente en píxeles en el que será mostrada la imagen. Es una propiedad de solo lectura, no puede ser modificada.
- **hspace/vspace:** contiene el valor del atributo HTML, con el mismo nombre, que representa el margen a lo alto/ancho en píxeles entre la imagen y otro contenido del documento. Es una propiedad de solo lectura.

- **lowsrc:** contiene el valor del atributo HTML, con el mismo nombre, que representa la URL, de una versión en baja resolución de la imagen definitiva. El navegador primero cargará la versión de la imagen indicada por este parámetro y luego procederá a cargar la real. Este parámetro es útil cuando las imágenes de la web son muy pesadas.
- **src:** contiene la URL, de la imagen y se corresponde con el atributo SRC de la etiqueta IMG de HTML. Esta propiedad puede ser alterada en cualquier instante y es la fuente principal de los efectos con imágenes. Cuando se cambia una imagen, el alto y el ancho con el que se mostrará será el de la imagen inicial.
- **Y sus eventos fundamentales:**
  - **onLoad():** se produce cuando la carga de la imagen ha finalizado completamente sin error.
  - **onError():** se produce cuando ocurre algún error durante la carga de la imagen.
  - **onAbort():** se produce cuando el usuario cancela la carga de la imagen.

En el siguiente ejemplo, se muestra un mensaje que le dice al usuario que espere mientras se carga la imagen, una vez cargada el texto varia.

```
<!doctype html>
<html>
<head>
<meta charset="iso-8859-1">
<title>Espera mientras se carga la imagen</title>

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

function imagen_cargada(){

document.getElementById("mensaje").innerHTML="Página
totalmente cargada";

}

</script>

</head>

<body>

<span id="mensaje">Espere mientras se carga la
imagen</span>



</body>

</html>
```

---

### 2.3.3. Precarga de imágenes

La precarga de imágenes consiste en la carga de las imágenes del servidor en el momento de la carga de la página (onLoad), aunque la imagen no necesite ser mostrada en ese momento. Es un sistema recomendable cuando se trabaje con imágenes con gran peso, logrando una interacción con el usuario más fluida ya que la imagen ya estará disponible sin esperas.

Para implementar la precarga, en el evento **onLoad** del documento se crea un objeto imagen al que se le asigna el archivo. Luego, en el evento en el que se desea que ya aparezca la imagen, simplemente le asignamos a la imagen real del documento el archivo de la primera imagen a través de su atributo **src**.

Veamos esto en el siguiente ejemplo:

```
<!doctype html>
<html>
<meta charset="iso-8859-1">
<title>Precarga de imágenes</title>
<head>
<script language="javascript" type="text/javascript">
var foto;
function precarga(){
    foto = new Image();
    foto.src = src="http://fondosgratishd.com/wp-
content/uploads/Paisajes-Abstractos.jpg"
}
function ver_foto(){
    document.images[0].src=foto.src;
}
</script>
</head>
<body onLoad="precarga();">
<a href="javascript:ver_foto();">
    Pulse aqui para ver la imagen
</a><br/>

</body>
</html>
```

---

El proceso se podría extender a más imágenes, por ejemplo para un carousel, aplicando esta técnica con una estructura repetitiva.


#### **2.3.4. Imágenes de sustitución**

Una imagen de sustitución es un efecto clásico consistente en sustituir la imagen mostrada cuando el usuario pasa el ratón por encima de ella. También puede mostrar una página distinta cuando el usuario hace clic sobre la misma.

En los nuevos dispositivos táctiles como son los teléfonos móviles y tabletas no existe el concepto de pasar el ratón por encima de un elemento, y dado que cada vez son más los usuarios que los utilizan es aconsejable no abusar de los efectos que emplean este evento.

En el siguiente ejemplo se muestra una imagen de sustitución con las tres opciones: cuando no está el ratón encima, cuando el ratón si está encima y cuando se hace clic sobre la imagen. En esta última situación, la imagen permanece siete décimas de segundo para luego volver a su estado inicial.

Partiríamos, por ejemplo, de estas tres imágenes

		
boton.jpg	botonOK.jpg	botonOVER.jpg

```

<!doctype html>
<html>
<head>
<meta charset="iso-8859-1">
<title>Imagen de sustitución</title>
<script language="javascript" type="text/javascript">
function foto_encima(){
    document.images[0].src="../images/botonOVER.jpg";
}
function foto_origen(){
    document.images[0].src="../images/boton.jpg";
}
function foto_clic() {
    document.images[0].src="../images/botonOK.jpg";
    setTimeout("foto_origen()",700);
}
</script>
</head>
<body>

</body>
</html>

```

### 2.3.5. Galería de imágenes

Las galerías, carrouseles o slideshows son otro tipo de efecto especial consistente en una imagen que va cambiando de contenido cada cierto tiempo.

Existen multitud de galerías de fotos listas para descargar e incorporar en nuestros scripts con gran variedad de funcionalidad, como links,

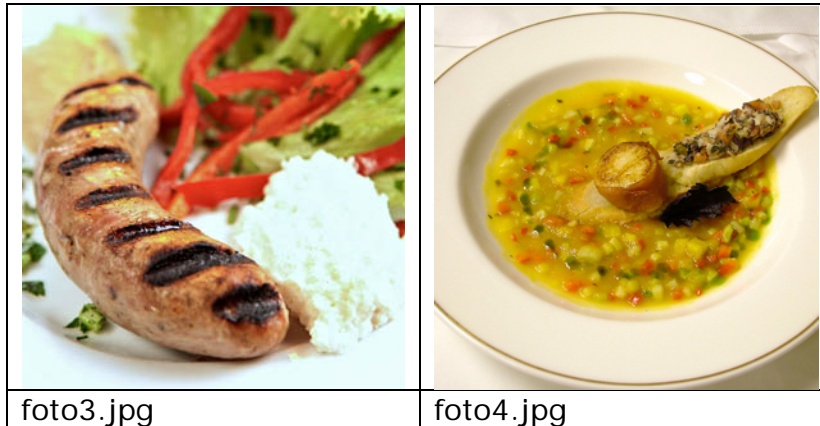
transiciones, etc. Suelen estar basadas en librerías como JQuery o Mootools y son muy sencillas.

Crea una galería de fotos, con un asistente simple e intégralo en cualquier página HTML	<a href="http://www.cincopa.com/es/galeria-de-fotos-para-web">http://www.cincopa.com/es/galeria-de-fotos-para-web</a>
Con menos de 4 KB esta galería responde para su sitio web.	<a href="http://pgwjs.com/pgwslideshow/">http://pgwjs.com/pgwslideshow/</a>
Plugins jquery para carruseles	<a href="http://www.templateMonsterblog.es/2013/01/29/40-plugins-jquery-para-carruseles/">http://www.templateMonsterblog.es/2013/01/29/40-plugins-jquery-para-carruseles/</a>

En el siguiente ejemplo se muestra una galería en su variante más sencilla. Como variable global se mantiene un array con la ruta de las cuatro imágenes que forman la galería. Al cargarse la página se invoca de forma cíclica cada dos segundos a la función que cambia la fotografía asignada a la etiqueta de la imagen.








---

```

<html>
<head>
<meta charset="iso-8859-1">
<title>Carrousel de imágenes</title>
<script language="javascript" type="text/javascript">
var rutaimg = new Array;
rutaimg[0]="../images/foto1.jpg";
rutaimg[1]="../images/foto2.jpg";
rutaimg[2]="../images/foto3.jpg";
rutaimg[3]="../images/foto4.jpg";
var cont=0;
function galeria() {
    setInterval("cambia_foto()",2000);
}
function cambia_foto() {
    cont++;
    if (cont==4)cont=0;
    document.images[0].src=rutaimg[cont];
}
</script>
</head>
<body onload="galeria();">

</body>
</html>

```

---

## 2.4. Efectos especiales en textos

Los efectos especiales aplicables a textos consisten en la modificación de los atributos que los caracterizan.

Los atributos que definen la apariencia de un texto, independientemente de las transformaciones que hagamos con JavaScript, pueden ser de dos tipos: atributos propios HTML como por ejemplo: "font" o bien atributos CSS, como por ejemplo: "background-color". De las dos opciones, la mejor es la segunda, pensando en la claridad de código y facilidad de mantenimiento ya que se separa el formato del documento (CSS) del contenido del mismo (HTML).

Para aplicar los efectos especiales en textos se emplean las instrucciones siguientes:

- **Document.write:** para escribir directamente sobre el documento tanto etiquetas, atributos HTML y CSS, como el texto contenido en las etiquetas.
- **elemento.style.propiedadCSS:** para leer y modificar las propiedades CSS.
- **elemento.innerHTML:** para modificar el texto incluido en la etiqueta HTML.

En el ejemplo siguiente podemos ver la propiedad innerHTML para lograr un efecto en un texto que pasa letra a letra a mayúsculas cada décima de segundo.

---

```
<!doctype html>

<html>

<head>

<meta charset="iso-8859-1">
<title>Cambiando a mayúsculas letra a letra</title>

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

var txt;

var cont=0;

var temp;

function efecto_texto() {

    txt=document.getElementById("txt").innerHTML;

    temp=setInterval("letra_a_mayusculas()",100);

}

function letra_a_mayusculas() {
```

---

```
var letra=txt.charAt(cont).toUpperCase();

txt=txt.substr(0,cont) + letra + txt.substr(cont+1);

document.getElementById("txt").innerHTML=txt;

    cont++;

    if (cont==txt.length) clearInterval(temp);
}

</script>

</head>

<body>

<span id="txt">Texto que pasa a mayusculas poco a poco</span>

<script>efecto_texto();</script>

</body>

</html>
```

---

En el ejemplo siguiente el texto cambia de color de forma intermitente entre azul y rojo, modificando la propiedad style del mismo modo utilizando la función setInterval que prolonga el efecto indefinidamente con una periodicidad de medio segundo.

---

```
<!doctype html>

<html>

<head>

<meta charset="iso-8859-1">
<title>Cambiano el texto de color</title>

<style type="text/css">

#txt {color:blue;font-size:3em;}

</style>

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

function cambio_color() {

    var texto=document.getElementById("txt");

    if (texto.style.color=="blue")
```

---

```
texto.style.color="red";

else

    texto.style.color="blue";

}

</script>

</head>

<body onload="setInterval('cambio_color()', 500)">

<span id="txt">Texto que cambia de color</span>

</body>

</html>
```

---

## 2.5. Efectos especiales en marcos y capas

El uso de marcos (frames en inglés), era una técnica que se utilizaba hasta hace unos años para estructurar las distintas secciones o áreas en las que se dividía una página web. En un sitio web típico, estas áreas se componían de una cabecera, debajo de ella dos o más columnas asimétricas y finalmente un marco a modo de pie de página. Cada una de estas partes o marcos eran archivos HTML independientes. Todos los marcos, a su vez, se reunían en un archivo HTML que actuaba como contenedor.

El uso de marcos generaba muchos problemas, de seguridad, de usabilidad, de posicionamiento en buscadores, etc, por lo que han quedado en desuso.

JavaScript dispone de características que facilitan la gestión de marcos, como pueden ser:

- **window.frames:** array que contiene todos los marcos del documento actual.
- **frame:** objeto con unas características similares al objeto window, por lo que podemos utilizar frame.document de forma similar a la utilizada para window.document.

Actualmente, la técnica habitual para estructurar un área de una página web es el empleo de capas. Una capa (div) corresponderá con un área rectangular del documento con las características que deseemos: dimensiones, posición, disposición con respecto a otros divs, etc. En el interior de un div puede incluirse cualquier elemento HTML, incluso otras capas o divs.

Los efectos especiales que llevará a cabo JavaScript consistirán en la modificación de esos atributos mediante su propiedad **style** como ya hemos visto en el apartado anterior para textos.

Los efectos típicos sobre capas suelen ser:

- **Capas en movimiento:** se modifican los atributos CSS **top** y **left** que definen la posición de la capa en su contexto.
- **Cambio de apariencia de la capa como el color de fondo, tamaño, etc,:** se modificarán los atributos que definen su apariencia.
- **Mostrar/ocultar capas:** se modificarán los atributos CSS **visibility** o **display**.

El siguiente ejemplo muestra un div con un texto que se desplaza verticalmente de arriba abajo indefinidamente por la ventana del navegador:

```
<!doctype html>
```

---

```
<html>
```

```
<head>
```

```
<meta charset="iso-8859-1">
```

```
<title>texto en movimiento, arriba y abajo</title>
```

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

```
#capa1 {
```

```
    background-color:yellow;
```

```
    width:140px;height:100px;
```

```
    position:absolute;
```

```
    left:500px;top:1px;
```

```
    text-align:center;
```

```
}
```

```
</style>
```

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

```
var alto=1;
```

```
var increm=3;
```

```
var baja=1;
```

```
function mueve_capa() {
```

---

```
if (alto<1 || alto > 50) increm=increm*-1; //cambia direccion

alto+=increm;

document.getElementById("capa1").style.top=alto+"px";

}

</script>

</head>

<body onload="setInterval('mueve_capa()', 50)">

<div id="capa1">Texto de la capa en movimiento<br/>

que oscila verticalmente

</div>

Texto de la página

</body>

</html>
```

---

En el siguiente ejemplo vamos a ver un efecto para mostrar y ocultar capas dinámicamente. Es una técnica muy útil para menús desplegados o para mostrar nueva información al usuario por descubrimiento, a medida que él la solicite, logrando reducir la información que aparece inicialmente en el documento.

En el siguiente ejemplo se presentan los títulos de dos ofertas publicadas en nuestra web, con un hipervínculo de más información al lado de cada una. Si el usuario clic en dichos hipervínculos se le mostrará una capa con los detalles de la oferta. Se utilizan los valores **none** y **block** del atributo CSS **display**.

---

```
<!doctype html>

<html>

<head>

<meta charset="iso-8859-1">
<title>Ofertas online</title>

<style type="text/css">

div {

background-color:yellow;
```

---

```
width:100px;height:100px;

display:none;

}

</style>

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

function mostrar1() {

    document.getElementById("oferta1").style.display="block";

    document.getElementById("oferta2").style.display="none";

}

function mostrar2() {

    document.getElementById("oferta2").style.display="block";

    document.getElementById("oferta1").style.display="none";

}

</script>

</head>

<body>

Oferta 1<a href="javascript:mostrar1();">Más info</a> <br/>

<div id="oferta1">

    El precio final de esta oferta son 100 euros

</div><br/>

Oferta 2<a href="javascript:mostrar2();">Más info</a>

<div id="oferta2">

    El precio final de esta oferta son 200 euros

</div>

</body>

</html>
```

---

## 2.6. Creación de ventanas

Para crear nuevas ventanas de forma dinámica a través de un Script, se emplea el método **open** del objeto **window**.

El método open consta de tres parámetros:

- **Primer parámetro:** nombre del archivo HTML que se va a abrir. Será una cadena vacía si es una nueva ventana con contenido que se generará a continuación y no se corresponde con un archivo almacenado en el servidor.
- **Segundo parámetro:** nombre de la ventana.
- **Tercer parámetro:** opciones de presentación de la ventana. La siguiente tabla muestra las opciones disponibles más importantes.

Opción	Significado
<b>fullscreen=[yes/no]</b>	Presenta o no la ventana en pantalla completa
<b>location=[yes/no]</b>	Presenta o no la barra de direcciones en el navegador
<b>toolbar=[yes/no]</b>	Presenta o no la barra de herramientas en el navegador
<b>menubar=[yes/no]</b>	Presenta o no la barra de menús en el navegador
<b>status=[yes/no]</b>	Presenta o no la barra de estado en el navegador
<b>scrollbars=[yes/no]</b>	Presenta o no las barras de desplazamiento en el navegador
<b>resizable=[yes/no]</b>	Permite o no redimensionar la ventana
<b>width/height=[valor]</b>	Establece el ancho/alto de la ventana en píxeles
<b>left/top=[valor]</b>	Establece la posición de la ventana respecto a la pantalla (píxeles a la izquierda y arriba).

Este tercer parámetro va entre comillas y cada opción con su valor separado por comas como se muestra a continuación:

---

```
Window.open("pagina.html", "Nueva ventana", "toolbar=no,
resizable=no, width=400, height=400);
```

---

- **Interactividad entre ventanas**

La gran capacidad de la operación de apertura de ventanas externas es la interactividad que se puede llevar a cabo entre la ventana llamante y la ventana llamada.

Al invocar al método **window.open** este devuelve un identificador de objeto correspondiente a la nueva ventana creada. Sobre ese identificador se podrá ejecutar cualquier operación como la ejecución de los métodos del objeto document contenido en ella.



En el siguiente ejemplo lo podemos ver más claro. Al pulsar el botón se abre una nueva ventana y una vez abierta se escribe en ella el contenido deseado, en este ejemplo la hora y minutos actuales.

---

```
<html>

<head>

<script type="text/javascript">

function reloj () {

    var ventana=window.open("", "Reloj",

        "toolbar=no, resizable=no,width=80, height=50");

    var fec=new Date();

    var hora=fec.getHours()+":"+fec.getMinutes();

    ventana.document.write(hora);

}

</script>

</head>

<body>

<input type="button" onclick="reloj();" value="¿Que hora es?"/>

</body>

</html>
```

---

Acabamos de ver cómo gestionar la ventana llamada desde la llamante pero lo inverso también es posible, es decir, realizar operaciones en la ventana llamada la propiedad **opener**, que hace referencia a la ventana origen. Veamos su funcionamiento con un ejemplo:

```
<html>

<head>

<script type="text/javascript">

function abrir () {

    var
ventana=window.open("ejemploventanasecundaria.html","Nueva
ventana",

                "width=200, height=200");

}

</script>

</head>

<body>

<input type="button" onclick="abrir();" value="Abrir ventana"/>

</body>

</html>
```

---

En el siguiente ejemplo de código que se muestra, corresponde al código de la página **ejemploventanasecundaria.html**, llamada desde el ejemplo anterior. Se puede comprobar que tiene un botón llamado **Cerrar** que al ser pulsado cierra la ventana original desde la que fue abierta esta ventana.

---

```
<html>

<head>

<script type="text/javascript">

function cerrar () {

    window.opener.close();

}

</script>

</head>
```

`<body>`

Esta es la nueva ventana

`<input type="button" onclick="cerrar();" value="Cerrar"/>`

`</body>`

`</html>`

---

### 3. Pruebas y verificación en páginas web

#### 3.1. Introducción

El hecho de que haya diferentes navegadores y distintos tipos de dispositivos de acceso a Internet hace que las aplicaciones web en la fase de pruebas, además de garantizar su correcto funcionamiento, deben cumplir otro requisito que las aplicaciones tradicionales no tienen: que funcionen correctamente en todos esos dispositivos y sobre cualquier navegador.

Conoceremos dos tipos de técnicas independientes:

- Técnicas que garantizan que el software funciona correctamente sin errores
- Técnicas que garantizan que el software puede ser ejecutado sobre cualquier navegador y sobre cualquier dispositivo.

#### 3.2. Técnicas de verificación

##### 3.2.1. Verificaciones fundamentales

La verificación consiste en responder a dos preguntas:

¿Es el producto correcto?, ¿Se está construyendo el producto de forma correcta?. Es decir, por una parte tiene que cumplir los requisitos que precisa el usuario y por otra parte tiene que funcionar correctamente. Para garantizar ambas premisas se hace necesario establecer un entorno de prueba en el que someter a las páginas a multitud de test que verifiquen su funcionamiento. Las tareas a realizar en ese entorno serían:

- Obtener las librerías que se utilizan en el script y ubicarlas en la ruta especificada en el mismo.
- Verificar los plugins y otros complementos necesarios por los navegadores.
- Realizar pruebas individuales a cada script y a cada función desarrollada: en estas pruebas hay que independizar cada script o función del resto de elementos que forman la página como puede ser HTML, CSS, o páginas de servidor como PHP o Java.

- Realizar pruebas de integración de los scripts con el resto de elementos.
- Realizar pruebas alfa: son las primeras pruebas del producto ya integrado y terminado y se realizan por un equipo perteneciente a la propia organización que realiza el proyecto.
- Pruebas beta: se realizan una vez pasadas las pruebas alfa y ya son realizadas por usuarios externos a la organización que no se centran en la corrección del código, sino en el cumplimiento de los requisitos funcionales estipulados por el usuario, justo antes de la comercialización del producto.

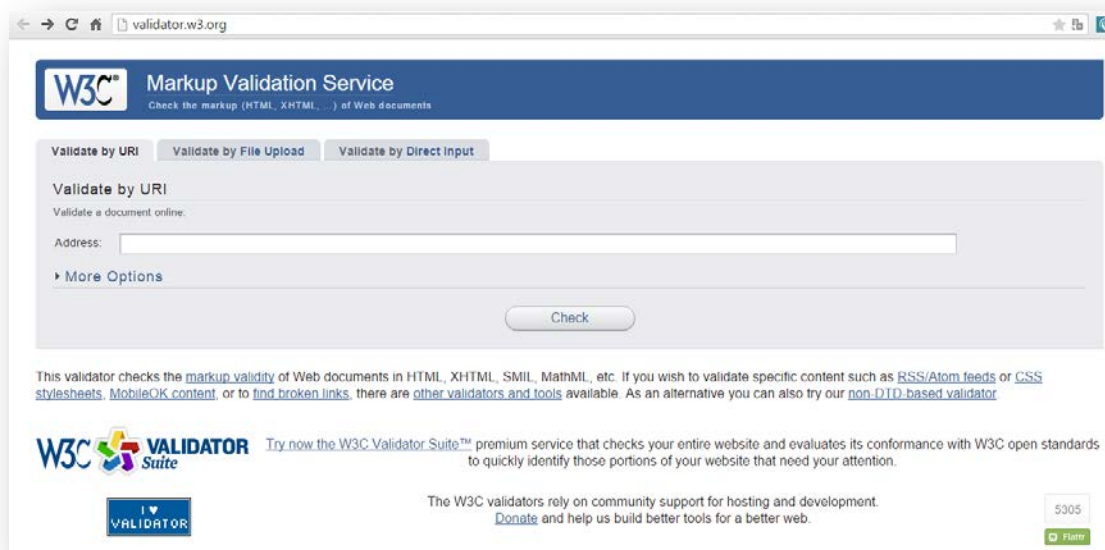
### 3.2.2. Verificación HTML

Para que el comportamiento de todos los navegadores sea idéntico los documentos HTML deben cumplir una serie de especificaciones y normas a los que llamamos **estándares**.

En HTML5 solo existe un tipo de declaración:

<!DOCTYPE html>

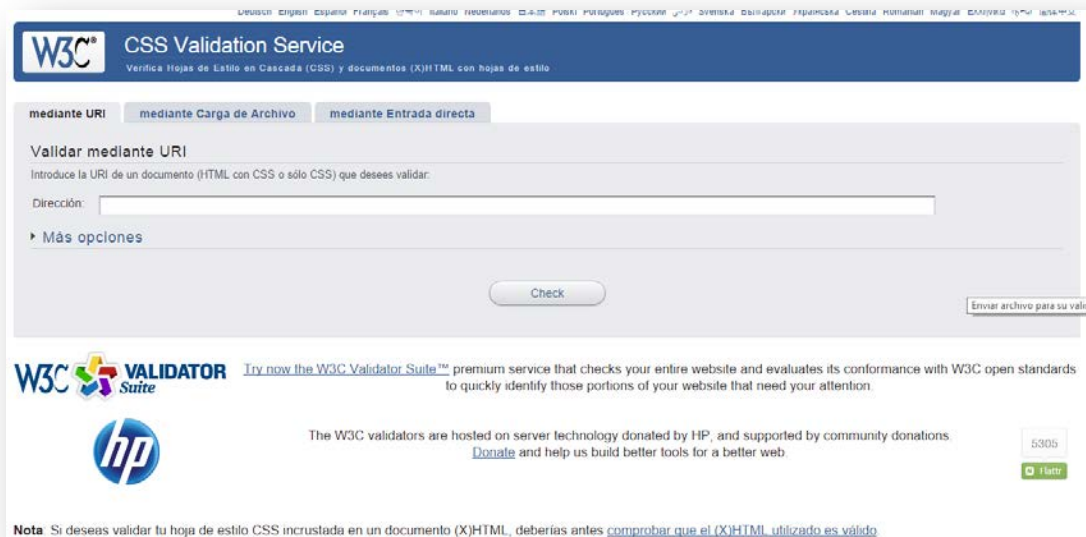
La definición y mantenimiento de este estándar y de los anteriores es realizada por el consorcio W3C y pone a disposición del público en <http://validator.w3.org> una herramienta online de validación de páginas HTML. Permite introducir una URL, subir un archivo o bien introducir directamente el contenido del documento.



## 3.2.3. Verificación CSS

Trabajar con CSS es una forma muy recomendable ya que simplifica los documentos HTML con lo que facilita su desarrollo, prueba y verificación y finalmente, el mantenimiento.

El consorcio W3C ofrece una herramienta online de validación tanto de documentos CSS como de documentos HTML que contienen CSS. La página es la siguiente : <http://jigsaw.w3.org/css-validator/validator.html.es> y dispone de un botón Más opciones que permite especificar la versión de CSS con la que se desea verificar.

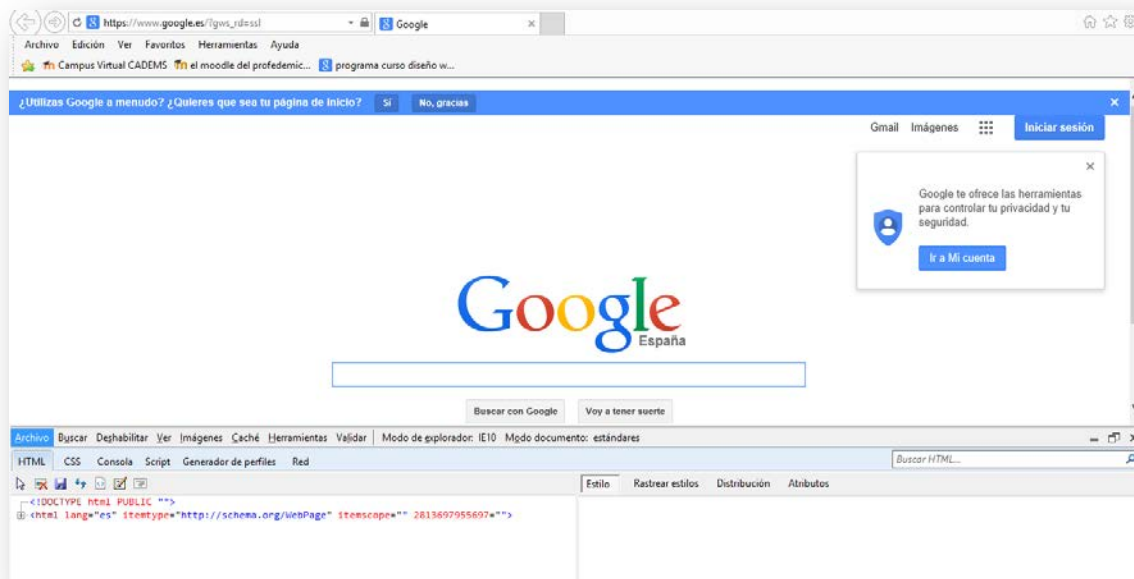


## 3.3. Herramientas de depuración para navegadores

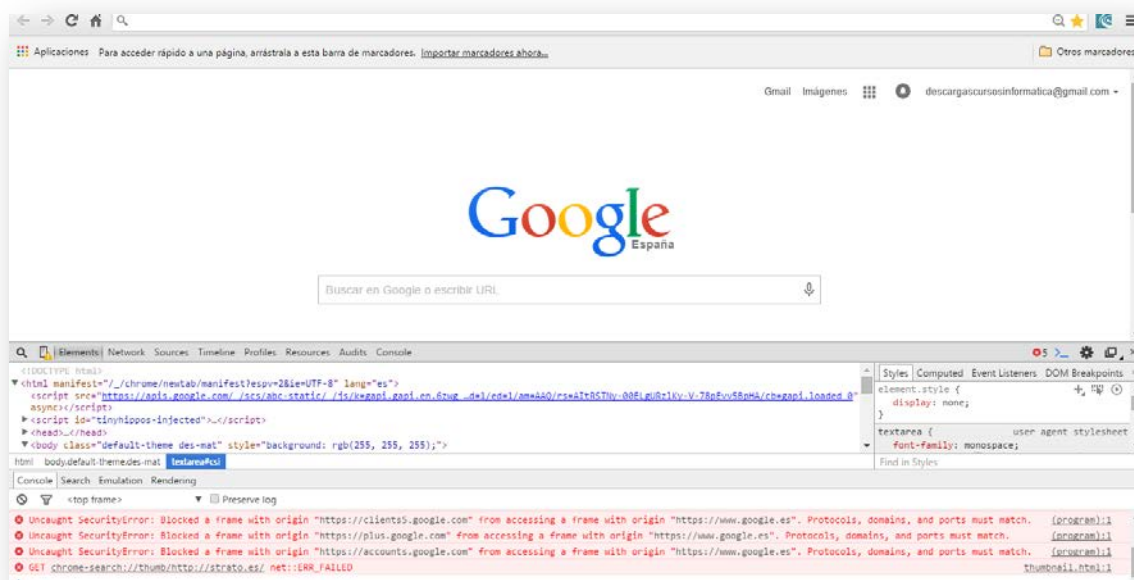
Los navegadores incluyen, en sus últimas versiones, herramientas para desarrolladores que facilitan la depuración y corrección de errores.

- En **Internet Explorer** las funciones de depuración se muestran pulsando la tecla **F12** o bien mediante el menú **Herramientas > Herramientas de Desarrollo**.
- En **Google Chrome** mediante la combinación de teclas **CTRL+MAY+I** o bien desde el menú **Herramientas > Herramientas para desarrolladores**.
- En **Firefox**, el complemento **firebug** se presenta con un icono en forma de cucaracha en blanco y negro.

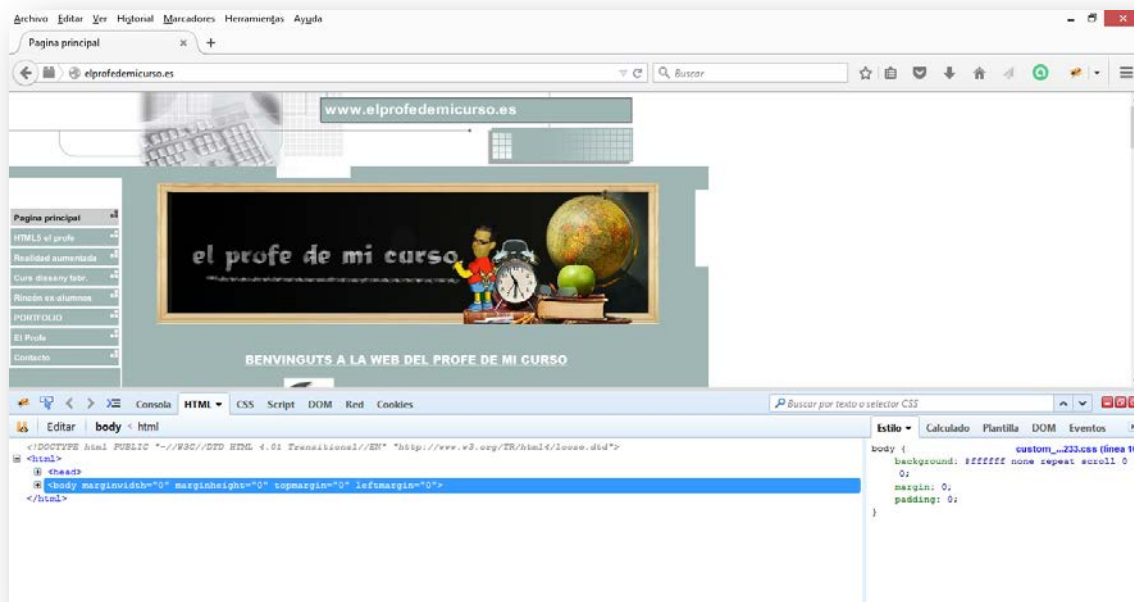
En los tres navegadores el resultado de activar las funciones de depuración es la aparición de una ventana en la parte inferior del navegador en la que se podrá operar para realizar la búsqueda y corrección de errores.



Vista herramientas depuración web en Internet Explorer 10



Vista herramientas depuración web en Google Chrome



Vista complemento firebug de Firefox

### 3.4. Compatibilidad en distintos navegadores

Los navegadores web siguen unos estándares para decidir cómo mostrar las páginas web tanto en su contenido como en su aspecto, pero la realidad actual es que no todos tienen el mismo comportamiento: por una parte hay atributos HTML y CSS que no son aceptados por todos los navegadores y por otra parte la forma de presentar las páginas en pantalla puede ser diferente. Esta problemática se puede resolver mediante JavaScript.

- El primer paso sería identificar el navegador que está utilizando el usuario
- El segundo sería ejecutar el código necesario para adaptar la página a las características de dicho navegador.

#### 3.4.1. Identificación del navegador

Para identificar el navegador junto con su versión desde el que se está visualizando la página JavaScript dispone del objeto **navigator**.

Las propiedades: **userAgent**, **appName**, **appCodeName**, **appVendor**, **platform** y **appVersion** del objeto **navigator** informan de todas las características del navegador. El problema radica en que no todos incluyen la misma información en cada propiedad ni con la misma sintaxis.

En la siguiente tabla se muestra la forma más exacta de identificar los navegadores más populares. En la primera columna se indica el navegador, en la segunda, la propiedad o propiedades donde se puede identificar y la tercera la subcadena que se encontrará en alguna parte de la propiedad y que identificará el navegador.

Navegador	Propiedad	Subcadena a buscar
Chrome	userAgent	Chrome
Firefox	userAgent	Firefox
Internet Explorer	userAgent appName	MSIE Microsoft Internet Explorer
Safari	userAgent vendor	Safari Apple
Konqueror	userAgent vendor	Konqueror KDE

La propiedad **vendor** no es soportada por todos los navegadores, por lo que es mejor no emplearla ya que sobre los navegadores que no lo soportan no se ejecutaría el script. El siguiente ejemplo muestra cómo utilizar los datos de la tabla para identificar el navegador del usuario.

---

```
<html>

<head></head>

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

    function obtener_navegador(){

        if (navigator.userAgent.indexOf("Chrome")!=-1) return
"Chrome";

        if (navigator.userAgent.indexOf("Safari")!=-1) return
"Safari";

        if (navigator.userAgent.indexOf("Konqueror")!=-1) return
"Konqueror";

        if (navigator.userAgent.indexOf("Firefox")!=-1) return
"Firefox";

        if (navigator.appName.indexOf("Explorer")!=-1) return
"Explorer";

        return "Desconocido";

    }

</script>

<body>

Está usted utilizando el navegador web:&nbsp;

<script>document.write(obtener_navegador());</script>
```

---



---

`</body>``</html>`

---

En muchos casos no será necesario identificar el navegador exacto del usuario, solo versiones anteriores de alguno de ellos como pueda ser Internet Explorer ya que no soportan muchas características actuales. La forma de obtener la versión es similar a la vista para el navegador, buscando el número de versión en la propiedad **userAgent**, como se muestra en la siguiente ejemplo.

---

`<html>``<head>``</head>``<script language="javascript" type="text/javascript">`

```
function versionIE() {  
    if (navigator.appName.indexOf("Explorer")!=-1) {  
        var expresion = new RegExp("MSIE ([0-9]{1,}[\.\0-9]{0,})");  
        if (expresion.exec(navigator.userAgent)!= null)  
            return(parseFloat( RegExp.$1 ));  
    }  
    return 0;  
}
```

`</script>``<body>`

La version de IE que está usando es:&nbsp;

`<script>document.write(versionIE());</script>``</body>``</html>`

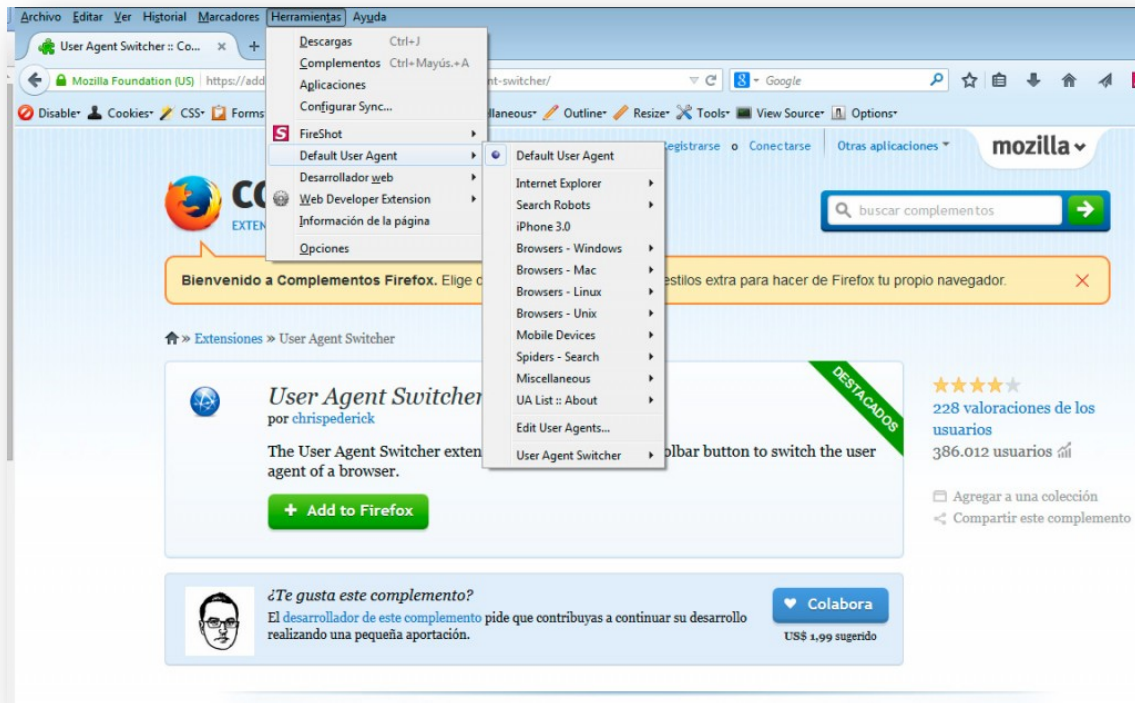
---

El inconveniente de nuevo viene dado por la distinta sintaxis que emplea cada navegador en la propiedad **userAgent**.

La identificación del sistema operativo y dispositivo que está utilizando el usuario no es tan complicada ya que la propiedad **platform** contiene valores más fáciles de procesar: **Win, Mac, Linux, iPhone, iPad, etc.** Quedan fuera de estos

valores otros dispositivos como los que utilizan el sistema operativo Android, que quedaría reflejado en la propiedad userAgent.

Para los desarrolladores, es interesante poder comprobar cómo se muestra una página web desde los distintos navegadores a medida que están desarrollando el aplicativo, para evitar desagradables sorpresas al final del proyecto. Mozilla Firefox ofrece un complemento gratuito llamado **User Agent Switcher** que permite que el navegador se comporte simulando a otros navegadores, tanto de dispositivos de sobremesa como de dispositivos móviles.



### 3.4.2. Ejecución de distinto código según el navegador

El siguiente paso, una vez identificado el navegador y el dispositivo, y comprobado si se trata de una versión antigua que no va a soportar las distintas características de nuestra página web, es la ejecución de un código realizado explícitamente para esas situaciones.

Una primera medida es la redirección de páginas, es decir, ir a una página específica para determinados navegadores/dispositivos. Esto es muy frecuente en el caso de los dispositivos móviles y se podría resolver de esta forma:

```
If ( navigator.userAgent.search(/iPhone|iPod|Android|BlackBerry/)!= -1){
```

```
Document.location.href = http://movil.miweb.com;
```

Otra opción es la de trabajar con media queries o incluso con distintos CSS para adaptarlos al dispositivo de destino.

## 4. Enlaces.

Recurso
Tips en JavaScript para tablets
<a href="http://www.curswebscecot.hol.es/tips">http://www.curswebscecot.hol.es/tips</a>
Manuales JavaScript
<a href="http://librosweb.es/libro/javascript/">http://librosweb.es/libro/javascript/</a>
<a href="http://www.desarrolloweb.com/javascript/">http://www.desarrolloweb.com/javascript/</a>
<a href="http://www.javascriptya.com.ar">http://www.javascriptya.com.ar</a>
Sobre JQuery mobile
<a href="http://www.jtech.ua.es/dadm/restringido/web/sesion05-apuntes.html#Cabeceras">http://www.jtech.ua.es/dadm/restringido/web/sesion05-apuntes.html#Cabeceras</a>
<a href="http://cartografia.cime.es/WebEditor/Pagines/file/tallerJQueryMobile/jquery.html#guia">http://cartografia.cime.es/WebEditor/Pagines/file/tallerJQueryMobile/jquery.html#guia</a>