estudios abiertos
SEAS
GRUPO SANVALERO

3

UNIDAD DIDACTICA 3

Programación Web

3. Avanzando en JavaScript

ÍNDICE

OBJETIVOS

- En esta unidad vamos a profundizar en los conceptos de JavaScript y realizar scripts más complejos y con funcionalidades más aplicables al entorno de desarrollo web.
- Nos centraremos en profundizar en los formularios, conociendo las propiedades y funciones que nos ofrece JavaScript para operar con ellos:
  - Veremos como obtener el valor de los campos de un formulario, de los cuadros de texto y textareas, de los RadioButton, de los CheckBox y de las listas desplegables.
  - Veremos como establecer el foco en un elemento determinado dentro de un formulario.
  - Aprenderemos a evitar el envío duplicado de un formulario a un servidor deshabilitando el botón de envío.
  - Veremos como limitar los caracteres en un textarea, definiendo el máximo permitido a ingresar.
  - Sabremos como acotar un caracter determinado en un cuadro de texto, permitiendo la introducción de unos determinados
- La validación será la principal de las utilidades para las que usaremos el lenguaje JavaScript:
  - Obligaremos a rellenar los campos que sean necesarios en un formulario.
  - Comprobaremos cadenas de texto para ver si se han introducido únicamente números.
  - Veremos como obligar a seleccionar un valor de una lista desplegable.
  - Comprobaremos:
    - Una dirección de email, el formato de una fecha un DNI y NIE, un número de teléfono.
    - Si se ha seleccionado un checkbox o un radiobutton.
- Aprenderemos a realizar e introducir en nuestras páginas web relojes, un calendario, realizar tooltips, menús desplegables y galerías de imágenes.
- Veremos como detectar errores en nuestros scripts, dependiendo de que navegador usemos para realizar las pruebas de nuestros scripts.

INTRODUCCIÓN

Podríamos decir que una de los principales motivos por los que se inventó el lenguaje de programación JavaScript fue la necesidad de validar directamente en el navegador del usuario, los datos de los formularios. Por ello, la programación de aplicaciones que contienen los formularios web siempre ha sido una de las tareas fundamentales de JavaScript. De esta forma, evitaremos recargar la página cuando el usuario cometa errores al rellenar los formularios.

También debemos decir que con la aparición de las aplicaciones AJAX, JavaScript ha sido relevado únicamente a las comunicaciones asíncronas con los servidores y a la manipulación dinámica de las aplicaciones.

Aun así, saber manejar los formularios va a ser un requerimiento imprescindible de cualquier programador de lenguaje JavaScript.

3.1. Propiedades de los formularios

Como podrás imaginar, JavaScript nos proporciona numerosas propiedades y funciones que nos van a facilitar la programación de las aplicaciones que van a manejar los formularios.

Debemos tener en cuenta que, cuando se carga una página web, el navegador va a crear automáticamente un array llamado forms y que va a contener la referencia a todos los formularios de la página.

Vamos a utilizar el objeto document, para acceder a esos elementos, y document.forms será el array que contendrá todos los formularios de la página. Si hacemos memoria de cómo debemos acceder a los elementos de un array, accederemos a estos mediante la siguiente instrucción, y accederemos al primer formulario de la página:

```
document.forms[0];
```

Además de ese array de formularios, se va a crear automáticamente un array llamado elements por cada uno de los formularios de la página. Este Array contendrá las referencias a todos los elementos (listas desplegables, botones, cuadros de texto, etc.) de ese formulario. Y usando la misma sintaxis de los arrays, con la siguiente instrucción obtendremos el primer elemento del primer formulario de la página:

```
document.forms[0].elements[0];
```

Veamos un ejemplo de cómo acceder directamente al último elemento del primer formulario de la página:

```
document.forms[0].elements[document.forms[0].elements.length - 1];
```

Aunque esta es la forma de acceder a los formularios de una forma sencilla y rápida, nos podemos encontrar con un grave problema, si cambia el diseño de la página y en el código HTML se cambia el orden o se añaden nuevos formularios. Si sucede esto "el primer formulario de la página" podría ser otro diferente al que espera la aplicación.

Por este motivo, evitaremos el acceso a los formularios de una página mediante el array document.forms.

Para eso, accederemos a los formularios de la página mediante su atributo name, o a través de su atributo id. Con el objeto Document, podemos acceder directamente a cualquier formulario mediante su nombre. Imaginemos la siguiente página:

EJEMPLO

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <script type="text/javascript">
        var formularioPrincipal = document.formulario;
        var primerElemento = document.formulario.
elemento;
        var formularioSecundario = document.otro_
formulario;
    </script>
</head>
<body>
    <form name="formulario">
    <input type="text" name="elemento" />
    </form>
    <form name="otro_formulario">
    ...
    </form>
</body>
</html>
```

De esta forma cuando accedamos a los formularios de la página, el script funcionará correctamente aunque se reordenen o se añadan nuevos formularios a la página. En el ejemplo anterior, vemos que también podremos acceder a los elementos de los formularios directamente mediante su atributo name:

```
var primerElemento = document.formulario.elemento;
```

Utilizando las funciones DOM de acceso directo a los nodos también podemos acceder a los formularios y a sus elementos. Veamos mediante un ejemplo, el uso de la función document.getElementById() para acceder de forma directa a un formulario y a uno de sus elementos:

EJEMPLO

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <script type="text/javascript">
        var formularioPrincipal = document.
        getElementById("formulario");
        var primerElemento = document.
        getElementById("elemento");
    </script>
</head>
<body>
    <form name="formulario" id="formulario">
    <input type="text" name="elemento" id="elemento" />
    </form>
</body>
</html>
```

Independientemente del método utilizado para obtener la referencia a un elemento de formulario, cada elemento dispone de las siguientes propiedades útiles para el desarrollo de las aplicaciones:

- **type:** nos indicará el tipo de elemento que se trata.
- Para los elementos de tipo `<input>` (text, button, checkbox, etc.) coincide con el valor de su atributo *type*.
- Para las listas desplegables normales (elemento `<select>`) su valor es select-one, lo que permite diferenciarlas de las listas que permiten seleccionar varios elementos a la vez y cuyo tipo es select-multiple.
- En los elementos de tipo `<textarea>`, el valor de *type* es textarea.
- **form:** va a ser una referencia directa al formulario al que pertenece el elemento. Por lo tanto, para acceder a un elemento del formulario, podemos utilizar `document.getElementById("id_del_elemento").form;`
- **name:** obtendrá el valor del atributo `name` de XHTML. Solamente se puede leer su valor, por lo que no se puede modificar.
- **value:** nos va a permitir leer y modificar el valor del atributo `value` de XHTML.
- Para los campos de texto (`<input type="text">` y `<textarea>`) obtendrá el texto que ha escrito el usuario.
- Para los botones obtiene el texto que se muestra en el botón.
- Para los elementos checkbox y radiobutton no nos será muy útil, como se verá más adelante.

Y ya por último, en este capítulo vamos a ver los eventos más utilizados en el manejo de los formularios:

- **onclick:** evento que se produce cuando se pincha con el ratón sobre un elemento. Se puede utilizar con cualquiera de los tipos de botones que permite definir XHTML
  - `<input type="button">`
  - `<input type="submit">`
  - `<input type="image">`
- **onchange:** este evento se produce cuando el usuario cambia el valor de un elemento de texto ya sea de tipo `<input type="text">` o `<textarea>`. Este evento también se desencadenará cuando el usuario seleccione una opción en una lista desplegable `<select>`. Sin embargo, el evento sólo se produce si después de realizar el cambio, el foco cambia de campo.

- onfocus: evento que se produce cuando el usuario selecciona un elemento del formulario.
- onblur: este evento es un evento complementario de onfocus, ya que se producirá cuando el usuario ha deseleccionado un elemento por haber seleccionado otro elemento del formulario, o cuando pierda el foco.

3.2. Utilidades básicas para formularios

En este capitulo vamos a ver todas las utilidades que nos proporciona JavaScript para poder realizar bien nuestros scripts.

3.2.1. Obtener el valor de los campos de formulario

Como veremos a partir de aquí, casi todas las técnicas JavaScript relacionadas con los formularios nos van a servir para leer y/o modificar el valor de los campos del formulario. Vamos a mostrar cómo obtener el valor de los campos de formulario más utilizados.

3.2.1.1. Cuadro de texto y textarea

Mediante la propiedad value obtendremos directamente el valor del texto mostrado por estos elementos.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <script type="text/javascript">
        var valor = document.getElementById("texto").value;
        var valor = document.getElementById("parrafo").value;
    </script>
</head>
<body>
    <input type="text" id="texto" />
    <textarea id="parrafo"></textarea>
</body>
</html>
```

3.2.1.2. Radiobutton

En los radiobuttons, lo importante es obtener cual de ellos se ha seleccionado, no el valor del atributo `value` de alguno de ellos. Para poder saber cual es el que está seleccionado, usaremos la propiedad `checked`, que nos devolverá true para el seleccionado y false en cualquier otro caso. Veámoslo con el siguiente ejemplo:

```
<body>
    <input type="radio" value="si" name="pregunta" id="pregunta_si" />
    SI
    <input type="radio" value="no" name="pregunta" id="pregunta_no" />
    NO
    <input type="radio" value="nsnc" name="pregunta" id="pregunta_nsnc" />
    NS/NC
</body>
```

Con el siguiente código insertado entre las etiquetas de <script> podremos determinar si cada radiobutton ha sido seleccionado o no:

```
    <script type="text/javascript">
        var elementos = document.getElementsByName("pregunta");
        for (var i = 0; i < elementos.length; i++) {
            alert(" Elemento: " + elementos[i].value + "\n
Seleccionado: " + elementos[i].checked);
        }
    </script>
```

3.2.1.3. Checkbox

El funcionamiento de estos elementos son muy similares a los radiobutton, salvo que debemos comprobar cada checkbox de forma independiente del resto. Y esto ocurre ya que los grupos de radiobutton son mutuamente excluyentes y sólo se puede seleccionar uno de ellos cada vez. En cambio, los checkbox se pueden seleccionar de forma independiente respecto de los demás checkbox.

Veámoslo mejor con un ejemplo: imaginemos que disponemos de los siguientes checkbox (con la propiedad checked es posible comprobar si cada checkbox ha sido seleccionado):

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <script type="text/javascript">
        var elemento = document.getElementById("condiciones");
        alert(" Elemento: " + elemento.value + "\n Seleccionado: " +
        elemento.checked);

        elemento = document.getElementById("privacidad");
        alert(" Elemento: " + elemento.value + "\n Seleccionado: " +
        elemento.checked);
    </script>
</head>
<body>
    <input type="checkbox" value="condiciones" name="condiciones"
     id="condiciones" />
    He leído y acepto las condiciones
    <input type="checkbox" value="privacidad" name="privacidad"
     id="privacidad" />
    He leído la política de privacidad
</body>
</html>
```

3.2.1.4. Select

Los select son las listas desplegables en los que más difícil va a resultarnos obtener su valor. Con un ejemplo lo veremos mejor:

```
<body>
    <select id="opciones" name="opciones">
        <option value="1">Primer valor</option>
        <option value="2">Segundo valor</option>
        <option value="3">Tercer valor</option>
        <option value="4">Cuarto valor</option>
    </select>
</body>
```

Lo que necesitamos es obtener el valor del atributo `value` de la opción `<option>` seleccionada por el usuario. Para obtenerlo se deben realizar una serie de pasos. Además, para obtener el valor seleccionado deben utilizarse las siguientes propiedades:

- `Options`: es un array creado automáticamente por el navegador para cada lista desplegable y es el que contiene la referencia a todas las opciones de esa lista. Para obtener la primera opción de una lista lo haremos de la siguiente forma:

```
document.getElementById("id_de_la_lista").options[0].
```

- `selectedIndex`: cuando el usuario selecciona una opción, el navegador actualiza automáticamente el valor de esta propiedad, que guarda el índice de la opción seleccionada. El índice hace referencia al array options creado automáticamente por el navegador para cada lista.

En el siguiente ejemplo, vemos como podemos acceder a estos checkbox:

```
<script type="text/javascript">
  // Obtenemos la referencia a la lista
  var lista = document.getElementById("opciones");
  // Obtenemos el índice de la opción que se ha seleccionado
  var indiceSeleccionado = lista.selectedIndex;
  // Con el índice y el array "options", obtenemos la opción seleccionada
  var opcionSeleccionada = lista.options[indiceSeleccionado];
  // Obtener el valor y el texto de la opción seleccionada
  var textoSeleccionado = opcionSeleccionada.text;
  var valorSeleccionado = opcionSeleccionada.value;
  alert("Opción seleccionada: " + textoSeleccionado + "\n Valor
 de la opción: " + valorSeleccionado);
</script>
```

Como habréis apreciado, para obtener el valor del atributo value correspondiente a la opción seleccionada por el usuario, es necesario realizar varios pasos. De todas formas, podemos abreviar todos los pasos necesarios en una única instrucción:

```
var lista = document.getElementById("opciones");

// Obtenemos el valor de la opción seleccionada
var valorSeleccionado = lista.options[lista.selectedIndex].value;
```

```
// Obtenemos el texto que muestra la opción seleccionada
var valorSeleccionado = lista.options[lista.selectedIndex].text;
```

No confundir el valor de la propiedad *selectedIndex* con el valor correspondiente a la propiedad *value* de la opción seleccionada.

En el anterior ejemplo, la primera opción tiene un value igual a 1. Sin embargo, si se selecciona esta opción, el valor de selectedIndex será 0, ya que es la primera opción del array options (y los arrays empiezan a contar los elementos en el número 0).

3.2.2. Establecer el foco en un elemento

En programación, se dice que un elemento tiene el foco del programa, cuando está seleccionado y se puede modificar alguna de sus propiedades o se puede escribir directamente en él.

Para que lo veáis más claro, cuando un cuadro de texto del formulario tiene el foco, se podrá escribir directamente en él sin tener que pinchar con el ratón en el interior del cuadro. Lo mismo ocurre si una lista desplegable tiene el foco, el usuario podrá seleccionar una opción directamente subiendo y bajando con las flechas del teclado.

También debéis saber que, si pulsamos sobre la tecla TABULADOR sobre una página web, los diferentes elementos irán obteniendo el foco del navegador.

Debéis asignar automáticamente el foco al primer elemento del formulario cuando se carga la página, por ejemplo, en una página con un formulario de registro o en una página de búsqueda,

Para asignar dicho foco a un elemento de XHTML, utilizaremos la función focus(). Vamos a entenderlo mejor con un ejemplo, en el que asignaremos el foco al elemento del formulario que tenga como id = “primero”:

```
<html xmlns=”http://www.w3.org/1999/xhtml”>
<head>
    <title></title>
    <script type=”text/javascript”>
        document.getElementById(“primero”).focus();
     </script>
</head>
<body>
    <form id=”formulario” action=”#”>
    <input type=”text” id=”primero” />
    </form>
</body>
</html>
```

Variando un poco el código anterior, podemos asignar automáticamente el foco del programa al primer elemento del primer formulario de la página, sin que tengamos que definir el id del formulario y de los elementos:

```
<script type="text/javascript">
     if (document.forms.length > 0) {
        if (document.forms[0].elements.length > 0) {
             document.forms[0].elements[0].focus();
         }
     }
</script>
```

Lo que hemos hecho en el código anterior, es comprobar primero que existe al menos un formulario en la página mediante el tamaño del array forms. Si es así, y su tamaño es mayor que 0, utilizará este primer formulario. De la misma manera, comprobamos si el formulario tiene al menos un elemento `if (document.forms[0].elements.length > 0)`. Si es así, vamos a establecer el foco del navegador al primer elemento del primer formulario con la instrucción `document.forms[0].elements[0].focus();`

Y para dejar el código completamente correcto, deberemos añadir una comprobación adicional, que el campo de formulario que se selecciona no sea de tipo hidden:

```
<script type="text/javascript">
    if (document.forms.length > 0) {
      for (var i = 0; i < document.forms[0].elements.length; i++) {
            var campo = document.forms[0].elements[i];
            if (campo.type != "hidden") {
                  campo.focus();
                break;
             }
         }
     }
</script>
```

3.2.3. Evitar el envío duplicado de un formulario

Como usuarios de páginas web, habréis visto que en muchas páginas existe la posibilidad de que el usuario pulse dos veces seguidas sobre el botón "Enviar". Puede ocurrir, que la conexión del usuario sea demasiado lenta o la respuesta del servidor se haga esperar, y por este motivo, el formulario original sigue mostrándose en el navegador y el usuario vuelve a pinchar sobre el botón de "Enviar".

Este problema no es tan grave, salvo que los formularios impliquen transacciones económicas.

Por este motivo, lo que debemos hacer es deshabilitar el botón de envío después de la primera pulsación. Veamos con un ejemplo como realizarlos:

```
<body>
    <form id="formulario" action="#">
    ...
    <input type="button" value="Enviar" onclick="this.
     disabled=true; this.value='Enviando...'; this.form.submit()" />
    </form>
```

```
</body>
```

Podéis observar, que cuando pulsemos sobre el botón de envío del formulario, se desencadenará el evento `onclick` sobre el botón y por tanto, ejecutaremos el código JavaScript que contiene el atributo `onclick`:

- Lo primero, deshabilitaremos el botón con la instrucción `this.disabled=true;` Esta será la única instrucción que necesitaremos si sólo queremos deshabilitar un botón.
- A continuación, cambiamos el mensaje del botón. Escribiremos: `'Enviando...'`
- Y por último, enviaremos el formulario con la función `submit()` en la siguiente instrucción: `this.form.submit()`

Si os habéis fijado, el botón del ejemplo anterior lo hemos definido mediante un botón de tipo `<input type="button" />`, ya que el código JavaScript mostrado no funciona correctamente con un botón de tipo `<input type="submit" />`. Si utilizásemos un botón de tipo submit, este se deshabilitaría antes de enviar el formulario y, por tanto, no se enviaría.

3.2.4. Limitar el tamaño de caracteres de un textarea

Cuando usamos HTML e introducimos un campo de formulario de tipo textarea, no podemos limitar el número máximo de caracteres que se pueden introducir. Por ello, JavaScript nos va a permitir añadir esta característica de una manera muy sencilla.

Acordaros de puntos anteriores que con algunos eventos (como *onkeypress, onclick y onsubmit*) podemos evitar el comportamiento normal modificando completamente el comportamiento habitual de este, si nos devuelve el valor false.

- Si en el evento onkeypress se devuelve el valor false, la tecla pulsada por el usuario no se tiene en cuenta.
- Si en el evento onclick se devuelve false de un elemento como un enlace, el navegador no carga la página indicada por el enlace.

Si un evento devuelve el valor true, su comportamiento es el habitual:

```
<textarea onkeypress="return true;"></textarea>
```

En el `textarea` del ejemplo de arriba, el usuario puede escribir cualquier caracter, ya que el evento `onkeypress` devuelve `true` y por tanto, su comportamiento es el normal y la tecla pulsada se transforma en un carácter dentro del `textarea`.

Por ejemplo, en el siguiente código:

```
<textarea onkeypress="return false;"></textarea>
```

Como el evento `onkeypress` nos va a devolver `false`, no se ejecutará el comportamiento por defecto del evento, y con esto quiero decir, que la tecla presionada no se transformará en ningún caracter dentro del `textarea`. No importa la tecla pulsada ni las veces que se pulse dicha tecla, ese `textarea` no permitirá escribir ningún caracter.

De esta manera, nos va a resultar sencillo limitar el número de caracteres que se pueden escribir en un elemento de tipo `textarea`: primero comprobamos si se ha llegado al máximo número de caracteres permitido y en caso de ser así, evitaremos el comportamiento habitual del evento y por tanto, los caracteres adicionales no se añadirán al `textarea`:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <script type="text/javascript">
        function limita(maximoCaracteres) {
            var elemento = document.getElementById("texto");
            if (elemento.value.length >= maximoCaracteres) {
                return false;
            }
            else {
                return true;
            }
        }
    </script>
</head>
<body>
    <textarea id="texto" onkeypress="return limita(10);" rows="5"
     cols="3"></textarea>
</body>
</html>
```

En este ejemplo, cada vez que pulsemos una tecla compararemos el número total de caracteres, con el máximo número permitido, que en este caso serán 10. Si el número de caracteres es igual o mayor que el límite, evitaremos el comportamiento por defecto de `onkeypress`, devolviendo el valor `false` y por tanto, la tecla no se añade.

3.2.5. Restringir los caracteres permitidos en un cuadro de texto

Muchas veces, tendremos que permitir el uso de determinados caracteres, como por ejemplo, en un cuadro en el que se solicita un teléfono, puede ser útil no permitir introducir caracteres no numéricos.

Al igual nos ocurrirá en el caso contrario, e impedir que el usuario introduzca números en un cuadro de texto. Para realizarlo, lo podemos hacer usando el evento onkeypress y otras sentencias JavaScript:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <script type="text/javascript">
        function permite(elEvento, permitidos) {
            // Variables que definen los caracteres permitidos
            var numeros = "0123456789";
```

```
            var caracteres = "
            abcdefghijklmnñopqrstuvwxyzABCDEFGHIJKLMNÑOPQRSTUVWXYZ";
            var numeros_caracteres = numeros + caracteres;
            var teclas_especiales = [8, 37, 39, 46];
            // 8 = BackSpace, 46 = Supr, 37 = flecha izquierda, 39 =
               flecha derecha
            // Seleccionar los caracteres a partir del parámetro de
               la función
            switch (permitidos) {
                case 'num':
                    permitidos = numeros;
                    break;
                case 'car':
                    permitidos = caracteres;
                    break;
                case 'num_car':
                     permitidos = numeros_caracteres;
                    break;
            }

            // Obtener la tecla pulsada
            var evento = elEvento || window.event;
            var codigoCaracter = evento.charCode || evento.keyCode;
            var caracter = String.fromCharCode(codigoCaracter);

            // Comprobar si la tecla pulsada es alguna de las teclas
               especiales
             // (teclas de borrado y flechas horizontales)
            var tecla_especial = false;
            for (var i in teclas_especiales) {
                if (codigoCaracter == teclas_especiales[i]) {
                    tecla_especial = true;
                    break;
                }
             }

            // Comprobar si la tecla pulsada se encuentra en los
               caracteres permitidos
             // o si es una tecla especial
            return permitidos.indexOf(caracter) != -1 || tecla_especial;
        }
    </script>
</head>
<body>
    // Sólo números
    <input type="text" id="text1" onkeypress="return permite(event, 'num')" />
    <br />
    // Sólo letras
    <input type="text" id="text2" onkeypress="return permite(event, 'car')" />
    <br />
    // Sólo letras o números
    <input type="text" id="text3" onkeypress="return permite(event, 'num_car')"
/>
</body>
</html>
```

Nuevamente, vamos a basar el funcionamiento del script en permitir el comportamiento habitual del evento `onkeypress`. Lo que haremos será comprobar cada vez que se pulsa una tecla, si este caracter está dentro de los permitidos de cada elemento `<input>`.

El funcionamiento del script anterior se basa en permitir o impedir el comportamiento habitual del evento `onkeypress`. Cuando se pulsa una tecla, se comprueba si el carácter de esa tecla se encuentra dentro de los caracteres permitidos para ese elemento `<input>`:

Devolverá `true` y por tanto el comportamiento de `onkeypress` es el habitual y la tecla se escribirá, si el caracter se encuentra dentro de los caracteres permitidos

- Devolverá `false` y, por tanto, se impidirá el comportamiento normal de `onkeypress` y la tecla no llega a escribirse en el input, cuando el caracter no se encuentre dentro de los caracteres permitidos.

Además, el script anterior siempre permite la pulsación de algunas teclas especiales. En concreto, las teclas BackSpace y Supr para borrar caracteres y las teclas Flecha Izquierda y Flecha Derecha para moverse en el cuadro de texto siempre se puden pulsar independientemente del tipo de caracteres permitidos.

3.3. Validación

La validación de los datos introducidos por los usuarios va a ser la principal utilidad de JavaScript en el manejo de los formularios. Recomendamos siempre validar mediante JavaScript los datos insertados por el usuario, antes de ser enviados al servidor. De esta manera, se le puede notificar de forma instantánea, sin necesidad de esperar la respuesta del servidor, si el usuario ha cometido algún error al rellenar el formulario.

De esta manera, podemos mejorar la satisfacción del usuario con la aplicación y poder reducir la carga de procesamiento en el servidor.

Lo que haremos será lo siguiente. Llamaremos a una función de validación que comprobará si los valores introducidos por el usuario cumplen todas las normas impuestas por la aplicación.

Piensa que pueden existir tantas comprobaciones como elementos en un formulario, y algunas de ellas son muy habituales:

- Rellenar un campo obligatorio.
- Seleccionar el valor de una lista desplegable.
- Comprobar que la dirección de email indicada sea correcta.
- Comprobar que la fecha introducida sea lógica.
- Comprobar que se haya introducido un número donde se requiere, etc.

Vamos a ver el código JavaScript básico para introducir la validación a un formulario:

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

Y el esquema de la función validacion() sería el siguiente:

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

Usaremos el comportamiento del evento `onsubmit` de JavaScript para realizar la validación de los campos. Tal y como hemos visto con otros eventos como `onclick` y `onkeypress`, `onsubmit` variará su comportamiento en función del valor que se devuelva:

- Si devuelve el valor `true`, el formulario se envía como lo haría normalmente.
- Si devuelve el valor `false`, el formulario no se envía.

Debemos realizar esta comprobación en todos y cada uno de los elementos del formulario. Cuando encuentre un elemento incorrecto, devolverá false, y si no encuentra ningún error, devolverá true.

Por lo tanto, primero definiremos el evento `onsubmit` del formulario:

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

El formulario se enviará únicamente al servidor si el valor resultante de la función validación() es `true`. En el caso de que devuelva `false`, el formulario no se enviará.

La función validacion() comprobará todas las condiciones insertadas en ella. En el momento que no se cumpla una condición, se devolverá `false` y el formulario no se enviará. Si llegamos al final de la función y todas las condiciones se han cumplido correctamente, devolverá `true` y el formulario se enviará.

La forma de notificar los errores dependerá del diseño realizado en la aplicación. En el caso expuesto anteriormente, mostramos los mensajes mediante la función alert(), con la que indicamos el error producido. Un buen diseño es mostrar cada mensaje de error al lado del elemento del formulario, y mostrar un mensaje principal indicando que el formulario contiene errores y que se deben modificar para poder enviar el formulario.

Una vez que tenemos definido todo el esquema de la función, tendremos que añadir el código correspondiente a todas las comprobaciones realizadas sobre los elementos del formulario.

En los siguientes capítulos vamos a ver las validaciones más habituales en los campos de un formulario.

3.3.1. Campo de texto obligatorio

Estos son los campos en los que el usuario debe introducir obligatoriamente un valor. Puede ser en un campo de texto o un textarea. La condición la podríamos indicar de la siguiente manera:

```
function validacion() {
     valor = document.getElementById("campo").value;
    if (valor == null || valor.length == 0 || /^\s+$/.test(valor))
{
        return false;
     }
 }
```

Comprobamos 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.

Con la segunda condición, vamos a comprobar que el texto introducido tenga una longitud superior a cero caracteres, es decir, que no esté vacío el campo.

Y con `/^\s+$/.test(valor)` que es la tercera parte de la condición obligaremos a que el valor introducido 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 pero que por su gran complejidad no se van a estudiar. Sólo será necesario copiar literalmente esta condición, poniendo especial cuidado en no modificar ningún caracter de la expresión. Para más información se puede consultar: *http://www.regular-expressions.info/*

3.3.2. Campo de texto con valores numéricos

Son los campos de los formularios en los que el usuario debe introducir un valor numérico en un cuadro de texto. La condición JavaScript consiste en:

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

Si el contenido de la variable valor no es un número válido, no se cumplirá la condición y devolverá `false`. Utilizando la función interna `isNaN()` simplificamos las comprobaciones, ya que JavaScript se encarga de tener en cuenta los decimales, signos, etc.

Vemos a continuación 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
```

3.3.3. Opción de una lista seleccionada

Comprobamos que el usuario ha seleccionado una opción de una lista despegable. El siguiente código JavaScript permite conseguirlo:

```
function validacion() {
    indice = document.getElementById("opciones").selectedIndex;
    if (indice == null || indice == 0) {
        return false;
    }
}
...
<form action="" method="" id="" name="" onsubmit="return validacion()">
<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>
</form>
```

Comprobaremos con la propiedad selectedIndex, si el índice de la opción seleccionada es válido y además es distinto de cero. Podéis observar que la primera opción de la lista no es válida, por lo que no se permite el valor 0 para esta propiedad `selectedIndex`.

3.3.4. Dirección de email

Comprobaremos que el usuario ha introducido una dirección de email con un formato válido. No se comprueba si se trata de una cuenta de correo electrónico real y operativa, lo que se comprueba es que la dirección parezca válida, ya que la condición JavaScript consiste en:

```
function validacion() {
     valor = document.getElementById("campo").value;
    if (!(/\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)/.test(valor))) {
        return false;
     }
}
```

Las direcciones de correo electrónico válidas pueden ser muy diferentes, se manera que de nuevo realizamos la comprobación mediante las expresiones regulares.

También debéis saber que la expresión regular anterior es una simplificación, ya que el estándar que define el formato de las direcciones de correo electrónico es muy complejo.

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

La expresión regular correcta sería la siguiente, aunque desaconsejamos su uso:

```
(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-
9!#$%&'*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\
x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\
x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-
9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:25[0-
5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4]
[0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\
x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\
x0c\x0e-\x7f])+)\])
```

3.3.5. Fecha

Las fechas serán los campos de formulario más complicados de validar por las formas diferentes en las que se pueden introducir.

En el siguiente código vamos a suponer que de alguna forma se ha obtenido el año, el mes y el día introducidos por el usuario:

```
function validacion() {
    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;
     }
}
```

Usaremos la función interna de JavaScript Date(ano, mes, dia) para construir fechas a partir del año, el mes y el día de la fecha. Debemos tener en cuenta que:

- El número de mes se indica de 0 a 11, siendo 0 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.

Intentaremos construir una fecha con los datos proporcionados por el usuario. Si los datos no son correctos, la fecha no se construirá correctamente y, por tanto, la validación del formulario no será correcta.

3.3.6. Número de DNI o NIE

Con esta validación comprobaremos que el número proporcionado por el usuario se corresponde con un número válido de DNI. Para cada país o región pueden variar los requisitos del documento de identidad de las personas aunque vamos a ver un ejemplo genérico fácilmente adaptable. Además de comprobar que el número esté formado por ocho cifras y una letra, comprobaremos que la letra indicada es correcta para el número introducido:

```
function validacion() {
     valor = document.getElementById("campo").value;
    var letrasDNI = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F',
    'P', 'D', 'X', 'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H',
    'L', 'C', 'K', 'E', 'T'];
    // comprobamos que tenga formato correcto
    if (!/^[X-Z0-9]{1}[0-9]{7}[A-Z]$/.test(valor)) {
        return false;
     }
    // comprobamos caracter inicial
    if (/^[XYZ]{1}/.test(valor)) {
        var pos = str_replace(['X', 'Y', 'Z'], ['0', '1',
        '2'], valor).substring(0, 8) % 23;
        if (valor.charAt(8) != letrasDNI.substring(pos, pos + 1)) {
            return false;
          }
    } else if (valor.charAt(8) != letrasDNI[(valor.
      substring(0, 8)) % 23]) {
        return false;
     }
 }
```

Las validaciones las realizaremos usando las expresiones regulares.

Con el primer if nos aseguramos que el formato del número introducido es el correcto, 8 números seguidos y una letra, para el caso de los DNI, y una letra X, Y o Z, seguido de 7 números y una letra, para el caso de los NIE.

La segunda comprobación consistirá en ver si empieza por una de las letras anteriormente indicadas, aplica el algoritmo de cálculo de la letra del DNI sustituyendo previamente el valor de:

- X por 0.
- Y por 1.
- Z por 2.

Y la compara con la letra proporcionada por el usuario.

Y en la tercera comprobación, será el caso de un DNI, aplica el algoritmo y la compara con la letra proporcionada por el usuario.

3.3.7. Número de teléfono

Para poder realizar la validación de un número de teléfono, deberemos soportar que se pueda introducir de diferentes maneras:

- Que sea fijo nacional o móvil.
- Con prefijo internacional.
- Agrupado por pares.
- Separando los números con guiones, etc.

Nosotros vamos a realizar el ejemplo suponiendo que un número de teléfono está formado por nueve dígitos consecutivos y sin espacios y sin guiones entre las cifras, únicamente permitiendo insertar un móvil o un fijo:

```
function validacion() {
     valor = document.getElementById("campo").value;
    if (!(/^[89]\d{8}$/.test(valor)) && !(/^[6]\d{8}$/.
    test(valor)) {
        return false;
     }
 }
```

De nuevo, hemos usado las expresiones regulares, dentro de la condición de JavaScript, para comprobar si el valor indicado es una sucesión de nueve números consecutivos correctos.

3.3.8. Checkbox seleccionado

Mediante JavaScript podemos comprobar si un elemento de tipo checkbox ha sido seleccionado o no, de una forma muy sencilla:

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

Si lo que queremos es validar que todos los checkbox de un formulario han sido seleccionados, podemos usar un bucle:

```
function validacion() {
     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;
            }
        }
    }
}
```

3.3.9. Radiobutton seleccionado

Para realizar la validación, en este caso se realiza comprobando los radiobuttons que forman parte de un grupo determinado. La forma de hacerlo mediante JavaScript será la siguiente:

```
<script type="text/javascript">
    function validacion() {
        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;
        }
    }
</script>
```

Lo que hemos hecho con este ejemplo es recorrer los radiobutton que forman un grupo y comprobar cada uno de los elementos que lo componen, y ver si alguno ha sido seleccionado. En cuanto encontremos el primero seleccionado, saldrá del bucle y se indicará que al menos uno ha sido seleccionado.

3.4. Otras utilidades

En estos capítulos sucesivos, vamos a descubrir muchas otras utilidades que es necesario conocer para desarrollar aplicaciones completas. Estudiaremos algunas de las utilidades básicas más comunes que podemos crear con JavaScript.

3.4.1. Relojes, contadores e intervalos de tiempo

Si os habéis fijado, muchas páginas web tienen integrado un reloj con la hora actual. Esa hora debe de actualizarse cada segundo, por lo que no podemos mostrarla directamente en la página generada por el servidor, ya que debería actualizarse y recargarse cada segundo, por lo que no sería operativa nunca. Vamos a desarrollarlo mediante JavaScript de una manera muy sencilla, utilizando para ello utilizaremos el objeto Date() para actualizar el reloj cada segundo.

Una vez que está creado ese objeto de tipo fecha, será posible manipularlo para obtener información o realizar cálculos con las fechas. Vamos a crear el objeto Date() sin necesidad de pasarle ningún parámetro y así obtendremos la fecha y hora actuales:

```
var fechaHora = new Date();
```

Ahora vamos a construir un reloj muy básico que no actualiza su contenido, es decir, es estático:

```
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
   <title>EV4_UD3</title>
   <script type="text/javascript">
        function reloj() {
           var fechaHora = new Date();
            document.getElementById("reloj").innerHTML = fechaHora;
        }
    </script>
</head>
<body onload="reloj()" >
    <div id="reloj">
    </div>
</body>
</html>
```

Lo único que nos va a mostrar será:

```
Sun Jul 29 2012 11:15:55 GMT+0200 (Hora de verano romance).
```

Como podemos observar, este reloj no es muy práctico, ya que ni se actualiza cada segundo mostrándonos la hora actual, y además nos muestra demasiada información.

El objeto Date() nos proporciona funciones para obtener información sobre la fecha y la hora. Más concretamente, vamos a ver las funciones que nos devuelven la hora, los minutos y los segundos, y de esta manera mostrar un reloj que únicamente nos muestre las horas, minutos y segundos:

```
        function reloj() {
            var fechaHora = new Date();

            var horas = fechaHora.getHours();
            var minutos = fechaHora.getMinutes();
            var segundos = fechaHora.getSeconds();

            document.getElementById("reloj").innerHTML = horas + ':'
+ minutos + ':' + segundos;
        }
```

De esta forma, hemos conseguido que únicamente nos muestre la hora, minutos y segundos:

```
11:27:5
```

Si nos fijamos un poco, las horas, minutos y segundo que no tengan más que un dígito, no le va a insertar automáticamente el 0, por lo que debemos forzarlo nosotros con el código. Insertaremos antes de la línea que imprime en el div lo siguiente:

```
            if (horas < 10) { horas = '0' + horas; }
            if (minutos < 10) { minutos = '0' + minutos; }
            if (segundos < 10) { segundos = '0' + segundos; }
```

Ahora el reloj ya muestra correctamente los dígitos con un cero delante:

```
11:32:09
```

El formato del reloj va a ser de 24h, si se desea cambiar a formato de 12h y con la notación "am/pm", deberemos realizar una pequeña conversión de nuestro código:

```
    <script type="text/javascript">
        function reloj() {
            var fechaHora = new Date();

            var horas = fechaHora.getHours();
            var minutos = fechaHora.getMinutes();
            var segundos = fechaHora.getSeconds();

            var sufijo = ' am';
            if (horas > 12) {
                horas = horas - 12;
                sufijo = ' pm';
             }

            if (horas < 10) { horas = '0' + horas; }
            if (minutos < 10) { minutos = '0' + minutos; }
            if (segundos < 10) { segundos = '0' + segundos; }

           document.getElementById("reloj").innerHTML = horas + ':' +
           minutos + ':' + segundos+sufijo;
        }
```

La hora mostrada quedará de la siguiente forma:

```
11:47:29 am
```

Para dejar nuestro reloj completamente operativo, sólo falta que se actualice su valor cada segundo. Para hacerlo usaremos unas funciones especiales de JavaScript que ejecutarán determinadas instrucciones cuando ha transcurrido un determinado espacio de tiempo.

Una función que nos puede resultar útil en otros proyectos es setTimeout()

La función `setTimeout()` permite ejecutar una función una vez que haya transcurrido un periodo de tiempo indicado.

La definición de la función la haremos de la siguiente manera:

```
setTimeout(nombreFuncion, milisegundos);
```

Para realizar un uso correcto de esta función debemos tener en cuenta lo siguiente:

- La función que se va a ejecutar se debe indicar mediante su nombre sin paréntesis.
- El tiempo que debe transcurrir hasta que se ejecute se indica en milisegundos.

De esta forma, indicaremos que se ejecute nuestra función reloj() un segundo después de que se cargue la página:

```
setTimeout(reloj, 1000);
```

Pero para poder realizar nuestro cometido de mostrar nuestro reloj, y que cada segundo se actualice, usaremos la función setInterval().

La función `setInterval()` permite ejecutar una función programada infinitas veces de forma periódica con un lapso de tiempo entre ejecuciones de tantos milisegundos como se hayan establecido.

La definición de esta función es idéntica a la función setTimeout(), y debemos tener en cuenta las mismas normas en su uso. Veamos como quedará finalizado nuestro script:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <title>Ejemplo_Reloj</title>
    <script type="text/javascript">
        function reloj() {
            var fechaHora = new Date();
            var horas = fechaHora.getHours();
            var minutos = fechaHora.getMinutes();
```

```
            var segundos = fechaHora.getSeconds();
            var sufijo = ' am';
            if (horas > 12) {
                 horas = horas - 12;
                 sufijo = ' pm';
             }

            if (horas < 10) { horas = '0' + horas; }
            if (minutos < 10) { minutos = '0' + minutos; }
            if (segundos < 10) { segundos = '0' + segundos; }
            document.getElementById("reloj").innerHTML = horas + ':'
            + minutos + ':' + segundos + sufijo;
        }

        window.onload = function () {
             setInterval(reloj, 1000);
         }
    </script>
</head>
<body>
    <div id="reloj">
    </div>
</body>
</html>
```

Si os fijáis, en este caso hemos cambiado `<body onload="reloj()">` por la función window.onload = `function` (), que nos asegura que una vez se cargue la página por completo, se ejecutará el código definido en la misma.

3.4.2. Calendario

Lo más complicado de obtener en los formularios los más complicado son los datos de las fechas, ya que existen diferentes formatos para poder insertarlas: *dd/mm/aaaa, dd/mm/aa, mm/dd/aaaa*, etc.

Lo normal es que el usuario la tenga que insertar en un cuadro de texto, indicándole el formato que debe seguir, o tener en el formulario un cuadro de texto para el día, una lista desplegable para el mes y otro cuadro de texto para el año.

En cualquier caso, lo más cómodo para el usuario es que la aplicación incluya un calendario en el que se pueda indicar la fecha pinchando sobre el día elegido:

Figura 3.78. Calendario creado con JavaScript .

Este método va a ser el menos propenso a cometer errores, ya que no es posible introducir fechas inválidas si el calendario está bien construido, ni es posible insertar las fechas en un formato incorrecto.

Debéis saber que, realizar un calendario completo en JavaScript, es una tarea muy complicada y no se va a estudiar completo su desarrollo. Pero por suerte, existen scripts gratuitos que permiten su uso libre incluso en aplicaciones comerciales.

De entre los muchos calendarios disponibles, el desarrollado por la empresa DynArch es uno de los más completos, que se puede acceder desde su página web oficial y que se puede descargar gratuitamente desde:

http://www.dynarch.com/projects/calendar/

Accediendo a la sección "Download it". El archivo descargado incluye todos los scripts necesarios, su documentación, ejemplos de uso, diferentes estilos CSS para el calendario, etc.

Vamos a explicar a continuación como introducirlos en cualquier página web:

Enlazar los archivos JavaScript y CSS requeridos:

Debemos descomprimir el archivo descargado, y guardar las carpetas que contienen los archivos JavaScript y CSS. La estructura de los archivos debe de quedar como se ve en la siguiente imagen:

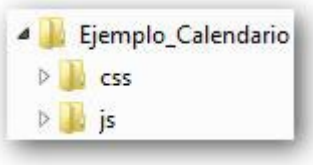

Figura 3.79. Estructura de la página de ejemplo.

Después en un archivo Ejemlo_Calendario.html, debemos enlazarlos desde la cabecera de la página HTML:

```
<head>
    <title>Ejemplo_Calendario</title>
    <script src="js/jscal2.js" type="text/javascript"></script>
    <script src="js/lang/es.js" type="text/javascript"></script>
    <link href="css/jscal2.css" rel="stylesheet" type="text/css" />
    <link href="css/border-radius.css" rel="stylesheet" type="text/css" />
    <link href="css/win2k/win2k.css" rel="stylesheet" type="text/css" />
    <!--<link href="css/matrix/matrix.css" rel="stylesheet"
         type="text/css" />
    <link href="css/steel/steel.css" rel="stylesheet" type="text/css" />-->
</head>
```

Para poner el calendario en español hemos añadimos la siguiente línea:

```
<script src="js/lang/es.js" type="text/javascript"></script>
```

El resto son las referencias a las diferentes utilidades necesarias para su funcionamiento. Si queréis cambiar el estilo del calendario, podéis descomentar alguna de las líneas comentadas y comentar las demás:

```
<link href="css/win2k/win2k.css" rel="stylesheet" type="text/css" />
<!--<link href="css/matrix/matrix.css" rel="stylesheet" type="text/css" />
<link href="css/steel/steel.css" rel="stylesheet" type="text/css" />-->
```

Añadir el código XHTML necesario para el elemento que va a mostrar el calendario

```
<input size="30" id="Fecha" readonly="readonly"/>
<button id="boton">
    ...</button>
```

En este caso, el calendario está formado por dos elementos:

- Un cuadro de texto llamado `"Fecha"` y que almacena el valor seleccionado en el calendario por el usuario. Se le ha asignado un atributo `readonly="readonly"`, para que el usuario no pueda modificar su valor.
- Podríamos insertar una imagen o icono que se utiliza para activar el calendario. Cuando el usuario pincha con el ratón sobre el boton, se mostrará el calendario de JavaScript.

Configurar el calendario

```
<script type="text/javascript">
   window.onload = function () {
        Calendar.setup({
            inputField: "Fecha",
            trigger: "boton",
            onSelect: function () { this.hide() },
            showTime: 12,
            dateFormat: "%d/%m/%Y"
       });
    }
```

Con este código inicializamos y preparamos el calendario. Lo que haremos es establecer el valor de alguna de sus propiedades, siendo las propiedades básicas imprescindibles:

- `inputField`: es el atributo id del elemento en el que se mostrará la fecha seleccionada, en este ejemplo sería `"Fecha"`.
- `dateFormat`: formato en el que se mostrará la fecha seleccionada (en este caso se ha optado por el formato dd / mm / aaaa).
- `trigger`: será el atributo id del elemento que se pulsa para mostrar el calendario de selección de fecha. En este ejemplo, el id del button.

Podéis descargaros el archivo completo de la plataforma *Ejemplo_Calendario.zip.*

3.4.3. Tooltip

Para los que no lo sepan todavía, los tooltips son pequeños recuadros de información que aparecen al posicionar el ratón sobre un elemento. Se usan para ofrecer información adicional sobre un elemento seleccionado o para mostrar al usuario pequeños mensajes de ayuda.

Al igual que ocurre con el calendario, realizar un tooltip completo mediante JavaScript es algo muy complejo, ya que en función de la posición del ratón se debe mostrar el mensaje correctamente.

También existen scripts que ya están preparados para generar cualquier tipo de tooltip. La librería que se va a utilizar se denomina overLIB, y se puede descargar desde su página web principal:

http://www.bosrup.com/web/overlib/?Download.

La descarga incluye todos los scripts necesarios para el funcionamiento del sistema de los tooltips, y si se quiere su documentación, se puede consultar en:

http://www.bosrup.com/web/overlib/?Documentation

La referencia de todos los comandos disponibles se puede consultar en:

http://www.bosrup.com/web/overlib/?Command_Reference

Los pasos necesarios para incluir un tooltip básico en cualquier página web son los siguientes:

1. Enlazar los archivos JavaScript requeridos. Para eso añadimos la siguiente línea a nuestra página:

```
<script src="js/overlib.js" type="text/javascript"><!-- overLIB
(c) Erik Bosrup --></script>
```

   Descomprimimos el archivo descargado, y guardamos los archivos JavaScript en el directorio js/ y se enlaza directamente desde la cabecera de la página HTML. Los tooltips sólo requieren que se enlace un único archivo JavaScript (overlib.js).

   Debemos introducir la línea tal cual, inclusive el comentario que incluye el código XHTML, ya que es el aviso de copyright de la librería, y es obligatorio incluirlo para poder usarla.

2. Definimos el texto que activa el tooltip y su contenido:

```
<body>
    <p>
Esto es un texto de prueba, en el cual, cuando pasemos
<a href="#" onmouseover="return overlib('deberias estar viendo este
mensaje');"onmouseout="return nd();">el raton por encima</a>. Una
vez que apartes el raton del enlace, dejaras de ver el
<a href="#" onmouseover="return overlib('Este es otro enlace de
prueba.');" onmouseout="return nd();"> ToolTip</p>
</body>
</html>
```

Para que funcione, los tooltip se deben incluir como enlaces de tipo <a> para los que se definen dos eventos:

- *Onmouseover:* cuando el ratón se pasa por encima del enlace anterior, se muestra el tooltip con el aspecto por defecto.
- *Onmouseout:* cuando se retira el ratón del enlace, el tooltip desaparece.

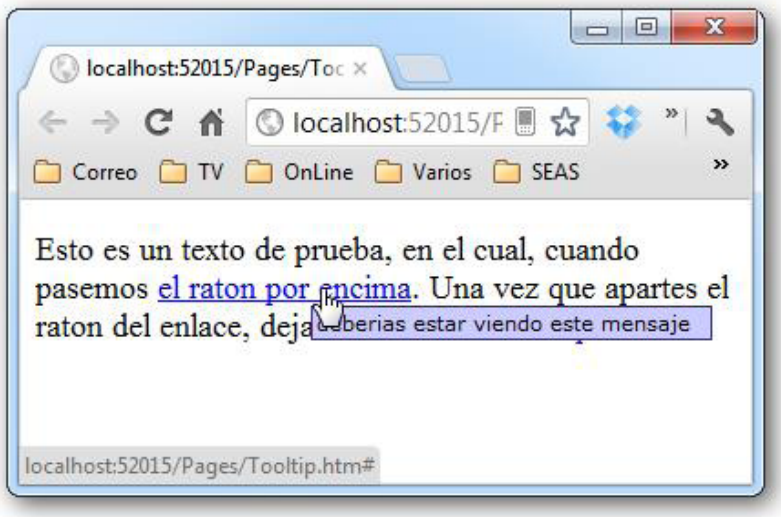

Figura 3.80. Imagen del tooltip cuando pasas por encima del enlace.

Esta librería nos permite configurarla a nuestro antojo, tanto el aspecto como el comportamiento de cada tooltip. Estas opciones se deben indicar en cada tooltip y los incluiremos como parámetros al final de la llamada a la función overlib():

```
<a href="#" onmouseover="return overlib('Este es otro enlace de
prueba.', CENTER);"onmouseout="return nd();">
```

Con el parámetro CENTER indicamos al tooltip que se muestre centrado respecto de la posición del ratón:

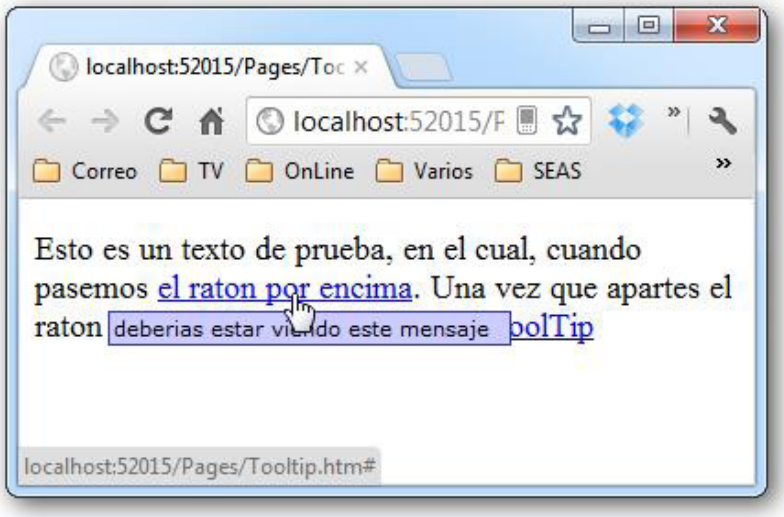

Figura 3.81. Imagen del tooltip centrado sobre la posición del ratón.

Otra opción que podemos controlar será la anchura del tooltip. Por defecto es de 200px, pero la opción WIDTH permite modificarla al tamaño que nosotros deseemos:

```
<a href="#" onmouseover="return overlib('deberias estar viendo este mensaje', CENTER, WIDTH, 100);"onmouseout="return nd();">
```

Lo indicaremos poniendo el nuevo tamaño detrás del parámetro WIDTH. El nuevo aspecto del tooltip es el siguiente:

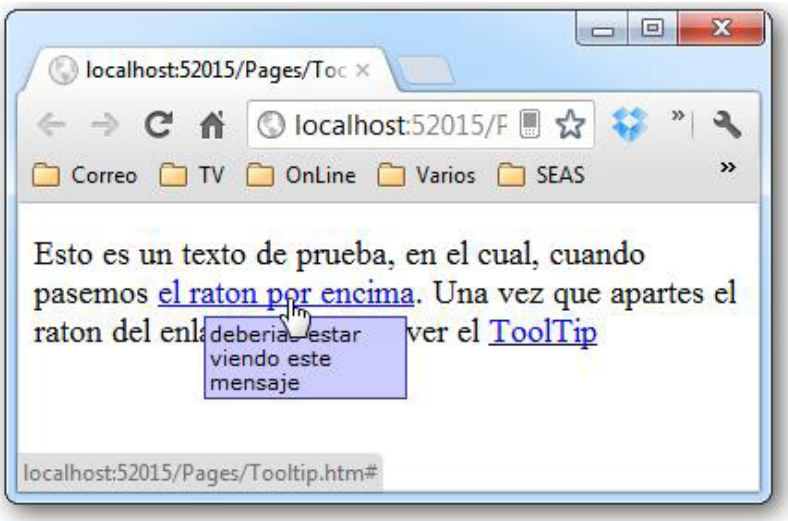

Figura 3.82. Imagen del Tooltip con la nueva anchura.

Recomendamos no usar esto cuando el número de parámetros empieza a ser muy numeroso. Disponemos de una función para configurar todos los tooltips de la página. Realizaremos una configuración global llamando a la función overlib_pagedefaults() con todos los parámetros de configuración deseados. Lo haremos de la siguiente forma:

```
<script type="text/javascript">
    overlib_pagedefaults(CENTER, WIDTH, 100);
</script>
```

Nos podemos descargar el ejemplo completo totalmente funcional desde la plataforma: *Ejemplo_Tooltip.zip*

3.4.4. Menú desplegable

Los menús desplegables son los que disponen de varios niveles jerárquicos, que se despliegan a medida que el usuario pasa el puntero del ratón por encima de cada elemento.

En capítulos siguientes, veremos como realizar menús desplegables horizontales y verticales de cualquier nivel de profundidad con las hojas de estilos CSS, pero debemos tener en cuenta, que los navegadores de la familia Internet Explorer versión 6.0 y anteriores no disponen del suficiente soporte de CSS como para crear este tipo de menús.

La única manera de crear un menú desplegable y asegurarnos de que funcione de manera correcta en cualquier navegador es realizarlo con JavaScript. Al igual que ocurre con los Tooltips y los calendarios, no es una tarea fácil hacerlo, por lo tanto, usaremos componentes externos al igual que en los casos anteriores. En este caso usaremos una librería de desarrollo de Yahoo.

La Yahoo UI Library, traducida como "Librería de componentes de interfaz de usuario de Yahoo" es un conjunto de utilidades y controles en JavaScript para el desarrollo rápido y sencillo de aplicaciones web complejas.

Dicha librería está dividida en módulos y componentes relacionados con:

- CSS.
- DOM.
- Eventos.
- AJAX, etc.

Dispone de las siguientes utilidades:

- Calendarios.
- Tooltips.
- Cuadros para autocompletar el texto.
- Árboles jerárquicos, etc.

Además incluye un completo módulo de menús para realizarlos: verticales, horizontales, estáticos, desplegables, menús de aplicación, menús contextuales, menús realizados con XHTML o con JavaScript, etc.

La librería YUI se puede descargar gratuitamente en:

http://yuilibrary.com/

De aquí nos podemos descargar todos los archivos necesarios, documentación y ejemplos. De esta manera podemos usar dichos archivos desde nuestro servidor, aunque recomendamos enlazar los archivos CSS y JavaScript desde sus propios servidores.

De esta manera, los archivos requeridos se descargan directamente desde sus servidores, que siempre van a ser mucho más rápidos y fiables que cualquier servidor particular. Como el usuario final de la página, se descarga los archivos desde los servidores de Yahoo, nos estamos ahorrando mucho ancho de banda y la aplicación carga mucho más rápido.

Veamos que pasos debemos seguir para cargarlos en nuestra página:

1. Debemos enlazar los siguientes archivos:

   ```
   <script        type="text/javascript"src="http://yui.yahooapis.
   com/3.5.1/build/yui/yui-min.js"></script>
   ```

   A continuación, aplicamos el estilo a todo el documento incluyendo en la etiqueta `<body>` el nombre de la clase:

   ```
   <body class="yui3-skin-sam">
   ```

2. Configurando el código XHTML de los elementos del menú::

   Necesitamos definir primero dos `<div>`

   ```
   <div id="menu_horizontal" class="yui3-menu yui3-menu-horizontal">
   <div class="yui3-menu-content">
       </div>
     </div>
   ```

Que contienen los estilos aplicados a nivel de clase. Después un elemento

```
<ul class="first-of-type">

</ul>
```

Que será el que contenga todos los elementos del menú. Dentro de este elemento, irán los ítems del menú:

```
<li class="yui3-menuitem"><a class="yui3-menuitem-content"
 href="menu.htm">Principal</a></li>
<li><a class="yui3-menu-label" href="#menu1">Menu 1</a>
</li>
```

Insertaremos tantos como elementos principales deseemos tener en él.

De momento, la estructura tiene que quedar de esta manera:

```
<div id="menu_horizontal" class="yui3-menu yui3-menu-horizontal">
    <div class="yui3-menu-content">
        <ul class="first-of-type">
           <li class="yui3-menuitem"><a class="yui3-menuitem-
            content"href="menu.htm">
              Principal</a></li>
           <li><a class="yui3-menu-label" href="#menu1">Menu 1</a>
              //aquí insertaremos los submenus
           </li>
        </ul>
    </div>
</div>
```

Veamos ahora como insertar los submenús dentro de cada menú. Necesitaremos crear dos que tendrán unos nuevos estilos definidos a nivel de clase:

```
<li><a class="yui3-menu-label" href="#menu1">Menu 1</a>
    <div id="menu1" class="yui3-menu">
     <div class="yui3-menu-content">
     </div>
     </div>
     </li>
```

Y posteriormente, volver a definir un nuevo elemento <ul>, para poder insertar los subitems:

```
<div id="menu_horizontal" class="yui3-menu yui3-menu-horizontal">
    <div class="yui3-menu-content">
       <ul class="first-of-type">
          <li class="yui3-menuitem"><a class="yui3-menuitem-content"
           href="menu.htm">
              Principal</a></li>
           <li><a class="yui3-menu-label" href="#menu1">Menu 1</a>
             <div id="menu1" class="yui3-menu">
                <div class="yui3-menu-content">
                   <ul>
                      //insertamos aquí los subitems
                   </ul>
```

```
                    </div>
                </div>
            </li>
        </ul>
    </div>
</div>
```

Dentro de estos, insertaremos los necesarios:

```
<li class="yui3-menuitem"><a class="yui3-menuitem-content"
href="#menu1">SubMenu1</a></li>
<li class="yui3-menuitem"><a class="yui3-menuitem-content"
href="#menu1">SubMenu2</a></li>
```

Por último, si queremos tener un subitem de Segundo nivel, a su vez con nuevos subitems de tercer nivel, los definiremos de la siguiente manera:

```
<li><a class="yui3-menu-label" href="#SubMenu1_2_1">SubMenu3</a>
    <div id="SubMenu3_1" class="yui3-menu">
        <div class="yui3-menu-content">
            <ul>
                <li class="yui3-menuitem"><a class="yui3-menuitem-
                 content" href="#menu1">SubMenu3_1</a></li>
                <li class="yui3-menuitem"><a class="yui3-menuitem-
                 content" href="#menu1">SubMenu3_2</a></li>
            </ul>
            <ul>
                <li class="yui3-menuitem"><a class="yui3-menuitem-
                 content" href="#menu1">SubMenu3_3</a></li>
                <li class="yui3-menuitem"><a class="yui3-menuitem-
                 content" href="#menu1">SubMenu3_4</a></li>
                <li class="yui3-menuitem"><a class="yui3-menuitem-
                 content" href="#menu1">SubMenu3_5</a></li>
            </ul>
            <ul>
                <li class="yui3-menuitem"><a class="yui3-menuitem-
                 content" href="#menu1">SubMenu3_6</a></li>
            </ul>
        </div>
    </div>
</li>
```

3. Construimos el menú y lo mostramos en la página:

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

    //  llamamos al metodo "use", y le pasamos como parametro
"node-menunav".

    // Esto cargará el script y el CSS para el Plugin del
nodo MenuNav

    // y todos las dependencias necesarias

YUI().use('node-menunav', function (Y) {
```

```
      // Recuperaramos la instancia de nodo que representa el
menú raíz

              // ()<div id="menu_horizontal">) y llamamos al
método del "plugin" y

                  // le pasamos una referencia al Plugin de
NodeMenuNav.</div>

      var menu = Y.one("#menu_horizontal");

      menu.plug(Y.Plugin.NodeMenuNav);

         });

  </script>
```

Veamos el código completo de la página:

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

<html lang="en" class="yui3-loading">

<head>

    <meta charset="utf-8">

   <title>Ejemplo_Menu</title>

    <script  type="text/javascript" src="http://yui.yahooapis.
com/3.5.1/build/yui/yui-min.js"></script>

  <script type="text/javascript">

     //  llamamos al metodo "use", y le pasamos como parametro
"node-menunav".

        // Esto cargará el script y el CSS para el Plugin del
nodo MenuNav

        // y todos las dependencias necesarias

     YUI().use('node-menunav', function (Y) {

      // Recuperaramos la instancia de nodo que representa el
menú raíz

              // ()<div id="menu_horizontal">) y llamamos al
método del "plugin" y

                  // le pasamos una referencia al Plugin de
NodeMenuNav.</div>
```

```
        var menu = Y.one("#menu_horizontal");
       menu.plug(Y.Plugin.NodeMenuNav);

     });

   </script>
</head>
<body class="yui3-skin-sam">
  <h1>
    Cabecera</h1>
     <div id="menu_horizontal"  class="yui3-menu  yui3-menu-
horizontal">
    <div class="yui3-menu-content">
      <ul class="first-of-type">
          <li class="yui3-menuitem"><a class="yui3-menuitem-
content" href="menu.htm">Principal</a></li>
       <li><a class="yui3-menu-label" href="#menu1">Menu 1</a>
         <div id="menu1" class="yui3-menu">
           <div class="yui3-menu-content">
             <ul>
                  <li class="yui3-menuitem"><a class="yui3-
menuitem-content" href="#menu1">SubMenu1</a></li>
                  <li class="yui3-menuitem"><a class="yui3-
menuitem-content" href="#menu1">SubMenu2</a></li>
                            <li><a  class="yui3-menu-label"
href="#SubMenu1_2_1">SubMenu3</a>
                <div id="SubMenu3_1" class="yui3-menu">
                 <div class="yui3-menu-content">
                   <ul>
                  <li class="yui3-menuitem"><a class="yui3-
```

```
menuitem-content" href="#menu1">SubMenu3_1</a></li>
                    <li class="yui3-menuitem"><a class="yui3-
menuitem-content" href="#menu1">SubMenu3_2</a></li>
                    </ul>
                                        <ul>
                    <li class="yui3-menuitem"><a class="yui3-
menuitem-content" href="#menu1">SubMenu3_3</a></li>
                    <li class="yui3-menuitem"><a class="yui3-
menuitem-content" href="#menu1">SubMenu3_4</a></li>
                    <li class="yui3-menuitem"><a class="yui3-
menuitem-content" href="#menu1">SubMenu3_5</a></li>
                    </ul>
                    <ul>
                    <li class="yui3-menuitem"><a class="yui3-
menuitem-content" href="#menu1">SubMenu3_6</a></li>
                    </ul>
                  </div>
                </div>
              </li>
                  <li class="yui3-menuitem"><a class="yui3-
menuitem-content" href="#menu1">SubMenu4</a>
              </li>
                                  <li class="yui3-menuitem"><a
class="yui3-menuitem-content" href="#menu1">SubMenu5</a></li>
                  <li class="yui3-menuitem"><a class="yui3-
menuitem-content" href="#menu1">SubMenu6</a></li>
                  <li class="yui3-menuitem"><a class="yui3-
menuitem-content" href="#menu1">SubMenu7</a></li>
            </ul>
          </div>
        </div>
      </li>
```

```
      </ul>
    </div>
  </div>
</body>
</html>
```

Así nos quedaría el menú. Podemos ir ampliándolo lo que nosotros necesitemos, dependiendo de los requerimientos de la pagina.

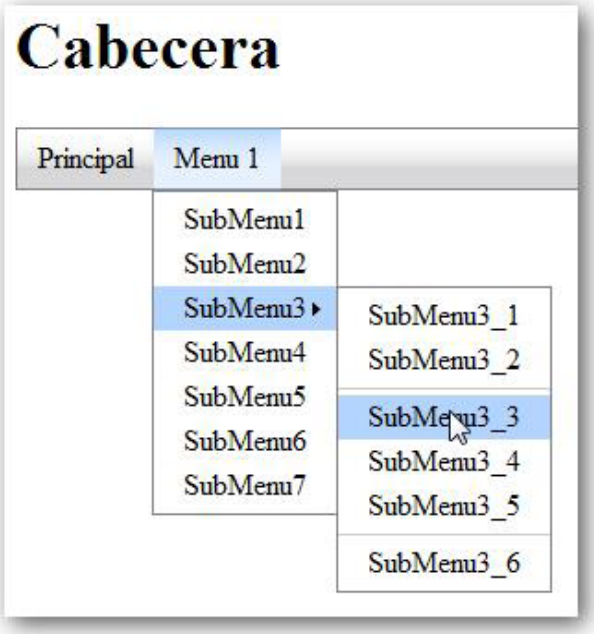

Figura 3.83. Imagen del menú realizado con JavaScript

Nos podemos descargar el ejemplo completo desde la plataforma, el archivo es *Ejemplo_menu.zip.*

3.4.5. Galerías de imágenes (Lightbox)

En este capítulo vamos a ver como crear una galería de imágenes (lightbox), con las que mostrar productos y servicios. Va a ser una serie de miniaturas de imágenes que se ampliarán al pinchar sobre cada una de ellas.

JavaScript ha supuesto una revolución en la creación de las galerías de imágenes y ha permitido mejorar la experiencia de navegación del usuario.

INFO

La creación de galerías de imágenes se conoce como *Lightbox* y fue creada originalmente por Lokesh Dhakar. *Lightbox* es muy sencilla de utilizar, funciona correctamente en todos los navegadores y no ensucia la página con código JavaScript.

Desde esta dirección:

http://lokeshdhakar.com/projects/lightbox2/

Podemos descargar gratuitamente, y también se puede ver una demostración de las galerías de imágenes construidas con Lightbox.

La descarga incluye:

- Completa documentación.
- Archivos CSS.
- El código fuente del script.
- Imágenes de prueba.
- Todos los archivos JavaScript externos necesarios.

Vamos a ver los pasos necesarios para incluir Lightbox en una página web:

Lo primero, debemos saber que Lightbox 2 utiliza el framework de jQuery. Tenemos que cargar jQuery y los archivos de JavaScript necesarios para que funcione correctamente y en el orden correcto. Todo esto lo entenderemos mejor cuando estudiemos la unidad de jQuery.

```
<head>
    <title>Ejemplo_LightBox</title>
    <script src="js/jquery-1.7.2.min.js" type="text/javascript"></
script>
    <script src="js/lightbox.js" type="text/javascript"></script>
</head>
```

Después, incluimos la referencia al archivo CSS de LightBox.

```
   <link href="css/lightbox.css" rel="stylesheet" type="text/css" />
```

Debemos asegurarnos que, en el archive lightbox.css, la ruta de las imágenes coincide con la que tenemos nosotros las imágenes almacenadas:

```
.lb-prev:hover {

  background: url(../images/prev.png) left 48% no-repeat;
```

```
}

.lb-next:hover {

  background: url(../images/next.png) right 48% no-repeat;

}
```

Para activar el script, debemos añadir el atributo `rel="lightbox[nombreAlbum]"`

Insertándole el nombre del álbum para que cargue todas las fotos en el script. Si no lo ponemos, las cargará de forma individual.

```
<a                          href="images/examples/image-5.jpg"
rel="lightbox[nombreAlbum]" title="descripcion">
```

Como opcional podemos usar el atributo title para insertar una descripción. Además, no hay límite de imágines para poder insertar en la página.

Veamos como quedaría el código XHTML completo:

```
<!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>Ejemplo_LightBox</title>
    <script src="js/jquery-1.7.2.min.js" type="text/javascript"></
script>
    <script src="js/lightbox.js" type="text/javascript"></script>
    <link href="css/lightbox.css" rel="stylesheet" type="text/css"
/>
</head>
<body>
    <div>
        <div>
            <div class="set">
                <div>
                    <a href="images/examples/image-3.jpg"
rel="lightbox[nombreAlbum]" title="Aqui podemos poner la
descripción de la imagen">
                        <img src="images/examples/thumb-3.jpg"
alt="Plantas: imagen 1 de 4" /></a>
                </div>
                <div>
                    <a href="images/examples/image-4.jpg"
rel="lightbox[nombreAlbum]" title="Sera la que aparezca debajo de
la misma cuando la mostremos">
                        <img src="images/examples/thumb-4.jpg"
alt="Plantas: imagen 2 de 4" /></a>
                </div>
                <div>
                    <a href="images/examples/image-5.jpg"
rel="lightbox[nombreAlbum]" title="Planta 3">
                        <img src="images/examples/thumb-5.jpg"
alt="Plantas: imagen 3 de 4" /></a>
                </div>
```

```
                <div>
                    <a href="images/examples/image-6.jpg"
rel="lightbox[nombreAlbum]" title="Para cerrar debes de pulsar
sobre la x inferior">
                      <img src="images/examples/thumb-6.jpg"
alt="Plantas: imagen 4 de 4" /></a>
                </div>
            </div>
        </div>
    </div>
</body>

</html>
```

Os podéis descargar el ejemplo completo, para modificarlo a vuestro gusto en la plataforma: *Ejemplo_Lightbox.zip*

3.5. Detección y corrección de errores

Como podréis haber observado hasta el momento, la mayoría de los errores en el código, no se pueden detectar hasta que se ejecutan los scripts. Esto ocurre ya que JavaScript es un lenguaje de programación interpretado.

Estos errores que se producen durante la ejecución, pueden se depurados por el mismo navegador.

Solucionar los errores de un script se denomina *"depurar el script"* o *"debugear el script"*, término que viene de la palabra inglesa *"debug"*, que significa *"eliminar los errores de una aplicación"*.

Actualmente, la mayoría de los navegadores traen las herramientas necesarias para poder detectar y corregir los errores de nuestros scripts. En los siguientes capítulos vamos a mostrar las herramientas que proporcionan dichos navegadores.

3.5.1. Internet Explorer

Lo normal es que el navegador tenga desactivada la notificación de errores de JavaScript. Debemos activarla para que nos muestre los mensajes de aviso sobre los errores de JavaScript.

Para activar debemos acceder Herramientas > Opciones, pestaña Opciones Avanzadas y deshabilitamos la opción "Deshabilitar la depuración de scripts (Internet Explorer)", como se muestra en la siguiente imagen:

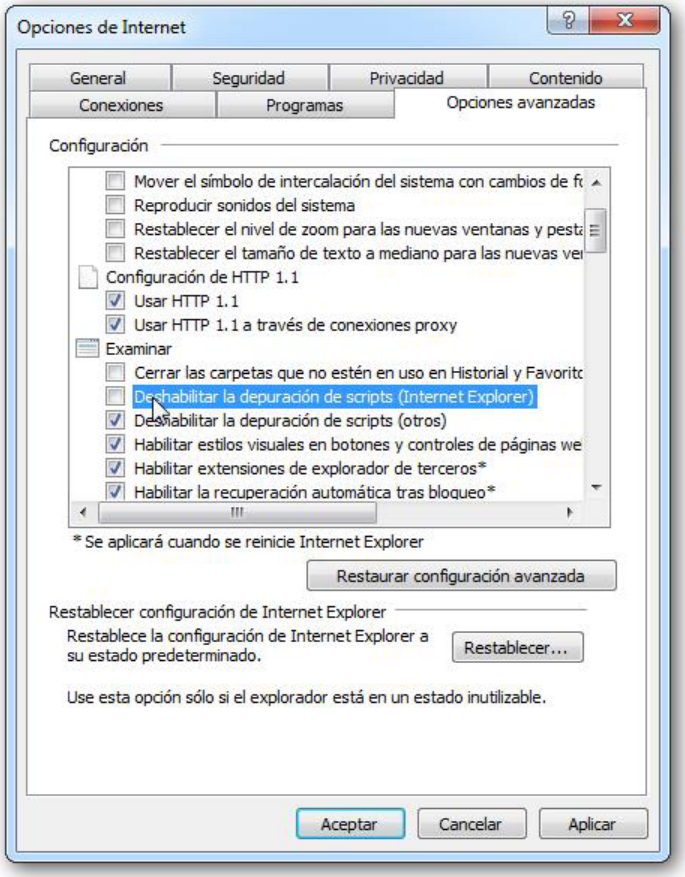

Figura 3.84. Opciones avanzadas del Internet Explorer.

Después de haber desactivado esta opción, se mostrará un mensaje de error cada vez que se produzcan errores de JavaScript en una página. Además, nos saldrá la opción de poder depurar el error, en el depurador que lleva integrado en las últimas versiones de IE.

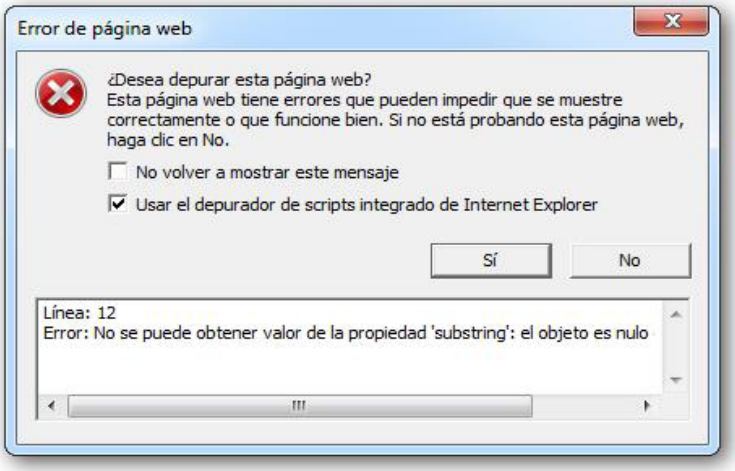

Figura 3.85. Ventana de alerta que muestra que nuestro script contiene errores.

Si pulsamos sobre “Sí”, se abrirá el depurador incorporado, mostrándonos donde esta el error:

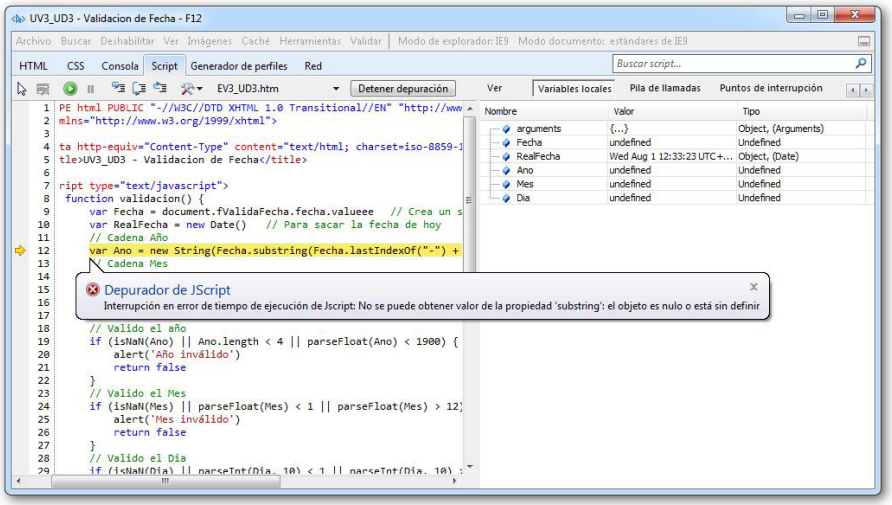

Figura 3.86. Imagen del depurador de errores de JavaScript del IE.

Hasta las últimas versiones de IE, era el navegador que menos información proporcionaba a los desarrolladores. Además de esto, no siempre indicaba correctamente la posición del posible error. El tipo de mensajes y la falta de precisión que tenía sobre el lugar en el que se ha producido realmente el error, hacían que depurar en Internet Explorer fuese una tarea excesivamente complicada.

3.5.2. Mozilla Firefox

Con este navegador, disponemos de la “Consola de errores”, a la que podemos acceder mediante el menú o con el atajo de teclado “Ctrl + Mayús + J”:

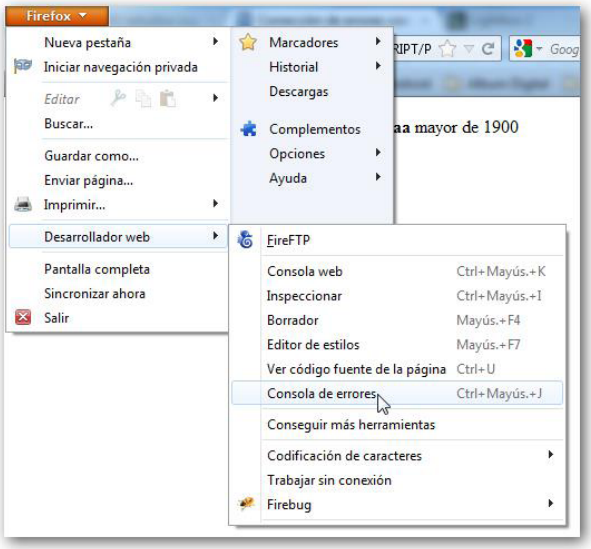

Figura 3.87. Forma de acceder a la “Consola de errores”.

Esta consola nos diferencia:

- Los mensajes de información.
- Los mensajes de aviso.
- Los mensajes de error.

Además, nos va a permitir visualizar todos los errores de la página simultáneamente. En cada error detectado se indicará la posible solución mediante un mensaje en inglés y además muestra un trozo de código del script donde se ha producido el error. Aparte de eso, pulsando sobre el enlace incluido se accede a la línea exacta del archivo concreto en el que se ha producido el error.

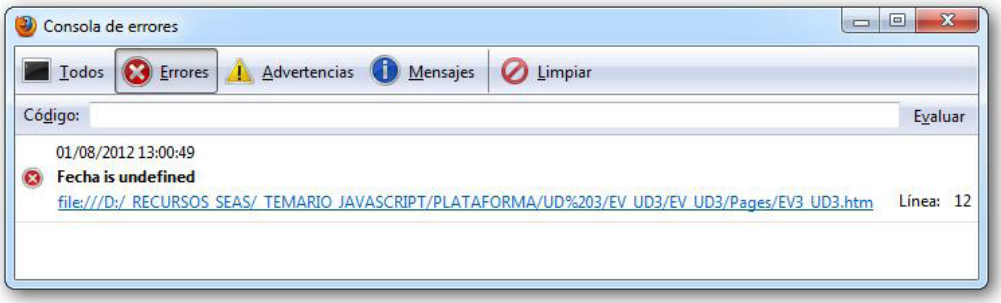

Figura 3.88. Ventana del error con el enlace que nos abre el archivo exacto.

Además, en este navegador se pueden instalar complementos o extensiones, que son pequeñas mejoras y ampliaciones. De todas estas extensiones existentes, Firebug es una de las más interesantes para los desarrolladores de aplicaciones web. La podemos descargar gratuitamente desde http://getfirebug.com/.

Una vez instalada en el navegador, accederemos mediante la tecla de función F12. Nos aparecerá una ventana similar a la que trae IE por defecto, y que nos mostrará infinidad de información acerca de todo el contenido del error y de la página HTML:

Figura 3.89. Información ofrecida por la Extensión Firebug.

Si pinchamos sobre el mensaje de error, nos redirigirá a la línea exacta del script donde se está produciendo el error:

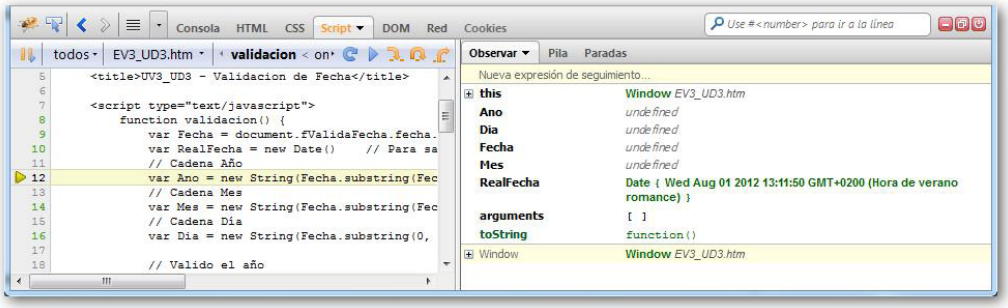

Figura 3.90. Línea exacta del error del Script.

3.5.3. Google Chrome

Este navegador también dispone por defecto de un depurador de páginas web. Se accede de la misma manera que en Firebug de Firefox o que el depurador de IE, mediante la tecla de función F12.

También dispone de innumerables herramientas para poder depurar y mostrarnos la información relativa al error:

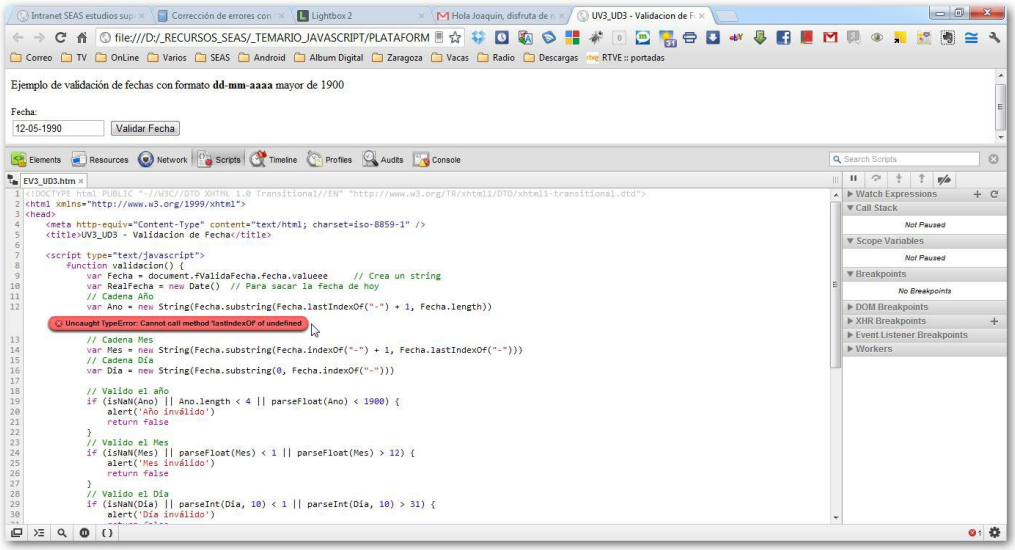

Figura 3.91. Google Chrome.

RESUMEN

- En esta unidad hemos profundizado en los conceptos de JavaScript y realizado scripts más complejos y con funcionalidades más aplicables al entorno de desarrollo web.
- Nos hemos centrado en analizar y en profundizar en los formularios, conociendo las propiedades y funciones que nos ofrece JavaScript para operar con ellos.
- Hemos aprendido a:
  - Obtener el valor de los campos de un formulario, de los cuadros de texto y textareas, de los RadioButton, de los CheckBox y de las listas desplegables.
  - Establecer el foco en un elemento determinado dentro de un formulario.
  - Evitar el envío duplicado de un formulario a un servidor deshabilitando el botón de envío.
  - Limitar los caracteres en un textarea, definiendo el máximo permitido a ingresar.
  - Acotar un carácter determinado en un cuadro de texto, permitiendo la introducción de unos determinados
- Hemos visto que la validación es la principal de las utilidades para las que vamos a usar el lenguaje JavaScript. También se ha mostrado como:
  - Obligar a rellenar los campos que sean necesarios en un formulario.
  - Comprobar cadenas de texto, para ver si se han introducido únicamente números.
  - Obligar a seleccionar un valor de una lista desplegable.
  - Comprobar:
    - Una dirección de email, el formato de una fecha un DNI y NIE y un número de teléfono.
    - Si se ha seleccionado un checkbox o un radiobutton.
- Hemos aprendido a realizar e introducir en nuestras páginas web relojes, calendarios, tooltips, menús desplegables y galerías de imágenes.
- Hemos visto las herramientas de depuración que tienen los navegadores para detectar errores en nuestros scripts.