Desarrollo Web en entorno Cliente: JAVASCRIPT

UNIDAD 5. OBJETOS DEL FORMULARIO

5.1 JERARQUÍA DE OBJETOS

5.2 OBJETO form

TIPOS DE BOTONES

5.3 OBJETO submit

5.4 OBJETO reset

5.5 OBJETO button

CAMPOS Y ÁREAS DE TEXTO

5.6 OBJETO text

5.7 OBJETO textarea

5.8 OBJETO password

5.9 OBJETO fileUpload

5.10 OBJETO hidden

5.11 OBJETO radio ***(botones de radio)***

5.12 OBJETO checkbox ***(casillas de verificación)***

LISTAS DESPLEGABLES

5.13 OBJETO select

5.14 OBJETO option

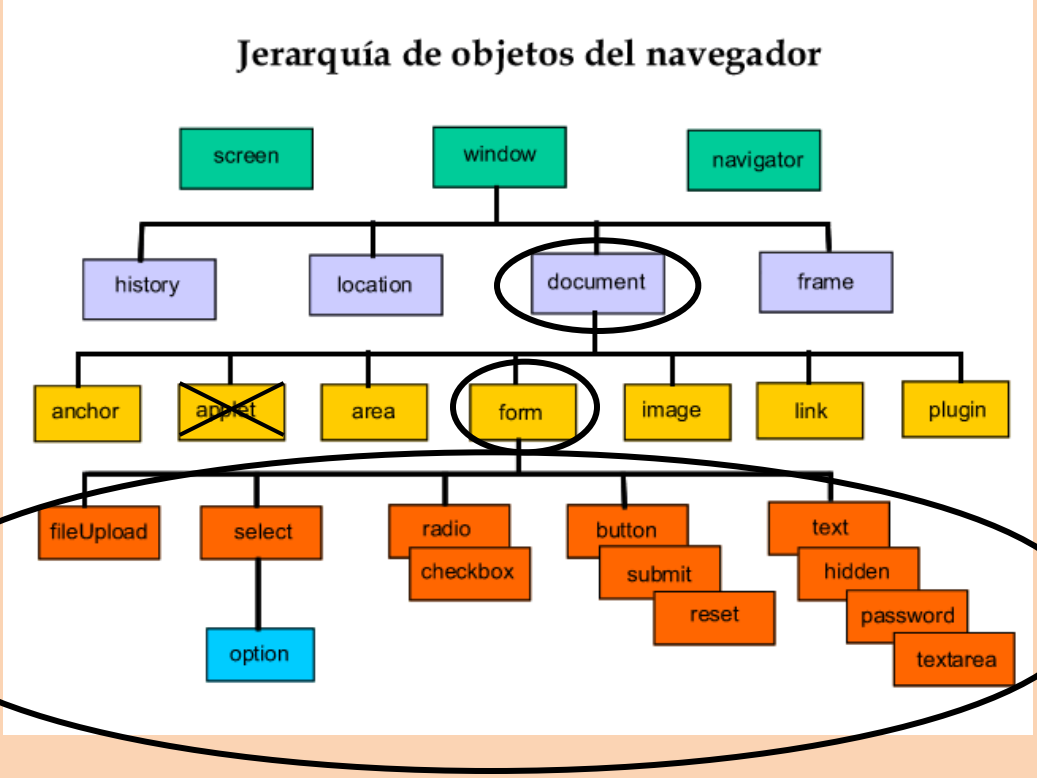
5.15 VALIDAR FORMULARIOS

- **Validación de un carácter**
- **Validaciones alfabéticas**
- **Validaciones numéricas**
- **Validaciones de fechas**
- **Validación según un patrón**
- **Validación de números de tarjetas de crédito**

UNIDAD 5. OBJETOS DEL FORMULARIO

5.1 JERARQUÍA DE OBJETOS DEL NAVEGADOR

Ahora vamos a estudiar la jerarquía que presentan los objetos del navegador, en la siguiente figura podemos observar gráficamente dicha jerarquía, atendiendo a la relación contenedor - contenido



Applet eliminado en HTML 5.

El frame es sustituido por , <iframe> y <object> en HTML 5.

La siguiente tabla muestra los objetos del navegador del modelo Javascript y sus correspondientes etiquetas HTML.

Objeto de Javascript	Etiqueta HTML
window	
~~Frame~~ iframe	~~<frame>~~ <iframe>
document	<body>
form	<form>
button	<input type="button">
checkbox	<input type="checkbox">
hidden	<input type="hidden">

fileUpload	<input type="file">
password	<input type="password">
radio	<input type="radio">
reset	<input type="reset">
select	<select>
submit	<input type="submit">
text	<input type="text">
textarea	<textarea>
link	<a href="">
anchor	<a name="">
~~applet~~	~~<applet>~~
image	<img>
plugin	<embed>
area	<map>
history	
location	
navigator	

5.2 OBJETO form

Los formularios son la herramienta que permite transferir hacia el servidor los datos introducidos por el cliente para que estos sean procesados. Javascript define un conjunto de objetos relacionados directamente con los elementos que se pueden incluir en un formulario HTML.

Propiedades del objeto form

Propiedad	Significado
action	Acción que queremos realizar cuando se submite un formulario (dirección de correo o url del programa que procesa los datos del formulario). Corresponde con el atributo **action** del formulario.
elements	Array que contiene cada uno de los elementos definidos en el formulario, en el mismo orden en que aparecen en el código HTML.
encoding	Tipo de codificación del formulario. Corresponde al atributo **enctype**
length	El número de elementos que contiene el formulario
method	Método a utilizar para enviar la información al servidor. Corresponde al atributo **method**.
name	El nombre del formulario, que corresponde con el atributo **name** del formulario.
~~target~~	~~La ventana o frame donde aparecerá la respuesta que genere el servidor después de enviar el formulario. Cuando se submita se actualizará la ventana o frame indicado. Corresponde con el atributo **target** del formulario.~~

Por ejemplo, podríamos cambiar la URL que recibiría la información del formulario con la instrucción.

```
document.miFormulario.action = "miPágina.asp"
```

Métodos del objeto form

Método	Significado
reset()	Simula el efecto que tiene pulsar el botón de tipo **reset** (borra los datos que haya introducido el usuario en el formulario).
submit()	Envía el formulario al servidor. Tiene el mismo efecto que pulsar el botón de tipo **submit** del formulario.

Para lograr un control programático sobre un formulario, lo primero que debe saber es que el **objeto document** dispone de una **propiedad forms**, que devuelve un array con todos los formularios definidos en dicho documento. Por añadidura cada uno de estos elementos del array es un **objeto form** con el que se puede trabajar directamente.

Para referirnos a un objeto form, debemos especificar su índice correspondiente dentro del **array forms**, o bien, **el nombre del formulario**:

window.document.forms[indice]
window.document.forms[nombre]
window.document.forms.nombre
window.document.nombre

- **window** es el **objeto window** que contiene el documento con el formulario con el que deseamos trabajar. Si dicho formulario se encuentra en el mismo documento que el **script**, puede omitir **window**.
- **indice** es la posición, dentro del **array forms**, que especifica el vínculo con el que vamos a trabajar.
- **nombre** es el nombre del formulario ofrecido por el atributo **name** de la etiqueta **form**.

Por ejemplo, consideremos la siguiente etiqueta <form>:

```
<form action="/cgi-win/formtest.exe" method="post" name="formulario_prueba">
```

Asumiendo que éste es el primer formulario definido en el documento actual, las siguientes expresiones son equivalentes:

document.forms[0]
document.forms["formulario_prueba"]
document.forms.formulario_prueba
document.formulario_prueba

De todas las **propiedades** del objeto form, la más utilizada es, sin duda, **elements**, que devuelve un array que contiene todos los campos del formulario. Para referirnos a uno de ellos, puede usar el **array elements**, o bien, ***el propio objeto a través de su nombre como mostramos a continuación***:

form.elements[indice]

form.elements[nombre]
form.nombre

- **form** es el **objeto form** que contiene el campo con el que vamos a trabajar.
- **indice** es la posición, dentro del array **elements**, que especifica el campo deseado.
- **nombre** es el nombre del campo utilizado en el atributo **name** su etiqueta.

Por ejemplo, consideremos la siguiente etiqueta <input>:

```
<input type="text" name="campo_texto">
```

Asumiendo que este campo del objeto form se encuentra en un formulario llamado mi_formulario y que es el primero definido en dicho formulario, las siguientes expresiones hacen referencia al mismo campo:

mi_formulario.elements[0]
mi_formulario.elements["campo_texto"]
mi_formulario.campo_texto

Con este sencillo ejemplo vemos cómo acceder a los formularios y elementos de los mismos sin necesidad de conocer sus nombres:

```
<html>
<head>
</head>
<body>
<form action="mailto:nobody@punto.es" method="post" name="ElPrimero">
Hola, soy el primer formulario del documento. Tengo algunos elementos: <p>
Caja 1: <input type="text" size="5" maxlength="10" name="Linea1"><br>
Caja 2: <input type="text" size="5" maxlength="10" name="Linea2"><br>
Caja 3: <input type="text" size="5" maxlength="10" name="Linea3">
</form> <p>
<form action="mailto:nobody@punto.es" method="get" name="ElSegundo">
Hola, soy el segundo formulario del documento. Tengo algunos elementos: <p>
<input type="radio" name="Radio1" value="a" checked> Radio 1<br>
<input type="radio" name="Radio1" value="b"> Radio 2<br>
<input type="radio" name="Radio1" value="c"> Radio 3
</form> <p>

  <script language="javascript">
    <!--
    var i, j;
    document.write('Tenemos ' + document.forms.length
       + ' formularios en el documento.');
    for (i = 0; i < document.forms.length; i++)
    with(document) {
      write('<p>document.forms[' + i + '].name = ' + forms[i].name +
           '<br>');
      write('document.forms[' + i + '].action = ' + forms[i].action +
           '<br>');
      write('document.forms[' + i + '].method = ' + forms[i].method +
           '<p>');
      write('Tengo ' + forms[i].elements.length + ' elementos:<p>');
      for (j = 0; j < document.forms[i].elements.length; j++)
        write('document.forms[' + i + '].elements[' + j + '].name = '
```

```
            + forms[i].elements[j].name + '<br>');
      }
      -->
    </script>
  </body>
  </html>
```

Ejemplo anterior (Acceso a los formularios y a sus elementos**)**: **ContenidoFormularios.html**

Otro Ejemplo (Calculadora sencilla**):** **Calculadora_sencilla.html**

5.3 OBJETO submit (es un botón)

Propiedades del objeto submit

Propiedad	Significado
form	Referencia al formulario que contiene al objeto.
name	Mantiene el valor del atributo **name** (nombre asignado al botón)
type	Mantiene el valor del atributo **type**
value	Mantiene el valor del atributo **value** (carateres que aparecen etiquetando al botón)

Métodos del objeto submit

Método	Significado
blur()	Elimina el foco del botón
click()	Simula el efecto que tiene pulsar el botón
focus()	Asigna el foco al botón

5.4 OBJETO reset (es un botón)

Propiedades del objeto reset

Propiedad	Significado
form	Referencia al formulario que contiene al objeto.
name	Mantiene el valor del atributo **name** (nombre asignado al botón)
type	Mantiene el valor del atributo **type**
value	Mantiene el valor del atributo **value** (carateres que aparecen etiquetando al botón)

Métodos del objeto reset

Método	Significado
blur()	Elimina el foco del botón
click()	Simula el efecto que tiene pulsar el botón
focus()	Asigna el foco al botón

Ejemplo (uso del método reset() para reiniciar todos los campos de un formulario**):**
Ejemplo_método_reset.html

5.5 OBJETO button

Propiedades del objeto button

Propiedad	Significado
form	Referencia al formulario que contiene al objeto.
name	Mantiene el valor del atributo **name** (nombre asignado al botón)
type	Mantiene el valor del atributo **type**
value	Mantiene el valor del atributo **value** (carateres que aparecen etiquetando al botón)

Métodos del objeto button

Método	Significado
blur()	Elimina el foco del botón
click()	Simula el efecto que tiene pulsar el botón
focus()	Asigna el foco al botón

Tenemos tres tipos de botones: un botón genérico, button, que no tiene acción asignada, y dos botones específicos, submit y reset. Estos dos últimos sí que tienen una acción asignada al ser pulsados: el primero envía el formulario y el segundo limpia los valores del formulario.

Como los tres tipos de botones comparten propiedades y métodos, veremos todos los ejemplos al mismo tiempo:

Aceptar, o rechazar, una reinicialización de campos.

```
<html>
<head>
  <script language="javascript">
    <!--
```

```
    function funcion_reset() {
        if (confirm("¿Está seguro de que quiere reinicializar todos los" +
                       " campos del formulario?")) {
            return true
        }
        else {
            return false
         }
     }
    -->
  </script>
</head>
<body>
<form action="mailto:nobody@punto.es"
      method="post"
      onReset="return funcion_reset()">
<input type="text" name="text_field" value="Texto predeterminado">
<br>
<input type="submit">
<input type="reset">
</form>
</body>
</html>
```

Ejemplo anterior (Comprobar una reinicialización de campos**):**
ComprobarResetCampos.html

Como forzar un envío o una reinicialización. Los métodos submit() y reset().

```
<html>
<head>
  <script language="javascript">
    <!--
    function funcion_button() {
         var submit_ok = confirm("Haga clic en Aceptar para enviar el " +
                     " formulario o en Cancelar para reinicializarlo:")
        if (submit_ok) {
            document.forms[0].submit()
        }
        else {
            document.forms[0].reset()
        }
    }
    -->
  </script>
</head>
<body>
<form action="mailto:nobody@punto.es"
      method="post">
<input type="text" name="text_field" value="Texto predeterminado">
<br>
<input type="button"
       value="Enviar / Reinicializar"
       onClick="funcion_button()">
```

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

Ejemplo anterior (Comprobar el envío o la reinicialización de un formulario**):**
ComprobarEnvioReset.html

5.6 OBJETO text

Propiedades del objeto text

Propiedad	Significado
defaultValue	Mantiene el valor del atributo **value** (valor por defecto asignado al campo o al área de texto).
form	Referencia al formulario que contiene al objeto.
name	Mantiene el valor del atributo **name** (nombre asignado al campo o área de texto)
type	Mantiene el valor del atributo **type**
value	Mantiene el valor del campo o del área de texto

Métodos del objeto text

Método	Significado
blur()	Elimina el foco del campo o del área de texto
focus()	Asigna el foco al campo o al área de texto
select()	Selecciona el texto que tiene el campo o el área de texto

5.7 OBJETO textarea

Propiedades del objeto textarea

Propiedad	Significado
defaultValue	Mantiene el valor del atributo **value** (valor por defecto asignado al campo o al área de texto).
form	Referencia al formulario que contiene al objeto.
name	Mantiene el valor del atributo **name** (nombre asignado al campo o área de texto)
type	Mantiene el valor del atributo **type**
value	Mantiene el valor del campo o del área de texto

Métodos del objeto textarea

Método	Significado
blur()	Elimina el foco del campo o del área de texto
focus()	Asigna el foco al campo o al área de texto
select()	Selecciona el texto que tiene el campo o el área de texto

5.8 OBJETO password

Propiedades del objeto password

Propiedad	Significado
defaultValue	Mantiene el valor del atributo **value** (valor por defecto asignado al campo password).
form	Referencia al formulario que contiene al objeto.
name	Mantiene el valor del atributo **name** (nombre asignado al campo password)
type	Mantiene el valor del atributo **type**
value	Mantiene el valor actual del campo password

Métodos del objeto password

Método	Significado
blur()	Elimina el foco del campo password
focus()	Asigna el foco al campo password
select()	Selecciona el texto que contiene el campo password

Los objetos text, textarea y password representan los campos de texto y las áreas de texto dentro de un formulario. Además, el objeto password es exactamente igual que el text salvo en que no muestra los caracteres introducidos por el usuario, poniendo asteriscos (*) en su lugar. Los tres tienen las mismas propiedades y métodos, por ello estudiamos los ejemplos juntos:

Creación de un campo de texto de solo lectura

```
<html>
<head>
  <script language="javascript">
    <!--
    function quitar_foco(campo_actual) {
    var formulario_actual = campo_actual.form
    var total_campos = formulario_actual.elements.length
    // ¿Hay otro campo?
    if (total_campos > 1) {
      // Si es así, obtener el índice del campo actual
```

```
      for (contador = 0; contador < total_campos; contador++) {
        if (campo_actual.name ==
                    formulario_actual.elements[contador].name) {
          var indice_actual = contador
          break
        }
      }
      // ¿Es éste el último campo?
      if (indice_actual == formulario_actual.elements.length-1) {
        // Si es así, configurar el foco en el primer campo
        formulario_actual.elements[0].focus()
      }
      else {
        // En caso contrario, configurar el foco en el campo siguente
        formulario_actual.elements[indice_actual + 1].focus()
      } // fin else
    } // fin if total_campos
    } // fin function
    -->
  </script>
</head>
<body>
<table border="1" cellspacing="5">
<tr>
<td valign="top">
<h3>Formulario 1: dos cuadros de texto</h3>
<form>
<b>Intente editar el siguiente texto:</b><br>
<input type="text" value="Texto de sólo lectura" name="campo1"
    onFocus="quitar_foco(this)">
<br>
<b>Este cuadro de texto es normal:</b><br>
<input type="text" value="Texto editable" name="campo2">
</form>
</td>
<td valign="top">
<h3>Formulario 2: tres cuadros de texto</h3>
<form>
<b>Cuadro de texto normal:</b><br>
<input type="text" value="Texto editable" name="campo1">
<br>
<b>Intente editar el siguiente texto:</b><br>
<input type="text" value="Texto de sólo lectura"  name="campo2"
    onFocus="quitar_foco(this)">
<br>
<b>Este cuadro de texto es normal:</b><br>
<input type="text" value="Texto editable" name="campo3">
</form>
</td>
</tr>
</table>
</body>
</html>
```

Ejemplo anterior (Creación de un campo de texto de solo lectura**):**
CuadroTextoSoloLectura.html

Comprobar login alfanumérico y password no vacía

```
<html>
<head>
  <script language="javascript">
    <!--
    function verificar_login(valor_login)
      {
      var mayusculas="ABCDEFGHIJKLMNÑOPQRSTUVWXYZ";
      var minusculas="abcdefghijklmnñopqrstuvwxyz";
      var numeros="0123456789";
      var caracteres_especiales="#@&!·$%&/()=?¿+*{}Ç-_.:,;[]";
      var valido=true;
      for (n=0;n<valor_login.length;n++)
       {
       if ((numeros.indexOf(valor_login.charAt(n))==-1) &&
           (mayusculas.indexOf(valor_login.charAt(n))==-1) &&
           (minusculas.indexOf(valor_login.charAt(n))==-1))
         {
         valido=false;
         break;
         }
       }
       return (valido);
     }

   function verificar_y_enviar(formulario)
     {
     var campo_login=formulario.login.value;
     if (campo_login=="")
       {
       alert("El campo login no puede quedar vacío");
       }
     else if (formulario.clave.value=="")
       {
       alert("El campo password no puede quedar vacío");
       }
     else if (!verificar_login(campo_login))
       {
       alert("El login debe ser alfanumérico");
       }
     else
       {
       formulario.submit();
       }
     }
    -->
  </script>
</head>
<body>
<h4 align="center">Introduce login y password</h4>
<hr>
<form method="post" action="mailto:nobody@punto.es">
<table>
<tr><td><strong>Login (alfanumérico):</strong></td>
<td><input type="text" name="login" size="10"></td></tr>
<tr><td><strong>Password:</strong></td>
```

```
<td><input type="password" name="clave" size="10"><br></td></tr>
<tr><td align="right"><input type="button" value="Enviar"
    onClick="verificar_y_enviar(document.forms[0])"></td>
<td align="left"><input type="reset" value="Borrar"></td></tr>
</table>
</form>
</body>
</html>
```

Ejemplo anterior (Comprobación login alfanumérico y password no vacía**)**:
LoginAlfanumerico.html

Ejercicio alumnado: Dados los ejemplos anteriores, realizar alguna mejora o cambio, en ellos.

5.9 OBJETO fileUpload

Propiedades del objeto fileUpload

Propiedad	Significado
form	Referencia al formulario que contiene al objeto.
name	Mantiene el valor del atributo **name** (nombre asignado al campo file)
type	Mantiene el valor del atributo **type**
value	Mantiene el valor actual del campo file (nombre del fichero a enviar)

Métodos del objeto fileUpload

Método	Significado
blur()	Elimina el foco del campo file
focus()	Asigna el foco al campo file
select()	Selecciona el texto que contiene el campo file

El objeto fileUpload se refiere a una etiqueta <input> que tiene su atributo type configurado a file.

Esto visualiza un cuadro de texto y un botón Examinar. Al hacer click en este botón se abre el típico cuadro de diálogo de apertura de archivos del sistema operativo, de modo que el usuario puede seleccionar un archivo. Cuando el usuario sale del cuadro de diálogo, la ruta de acceso y el nombre del archivo seleccionado se introducen en el cuadro de texto. (El usuario también puede escribir estos datos directamente). Cuando se envía el formulario, un programa CGI o un script de servidor procesa el archivo para cargarlo en un directorio del servidor.

Desde el punto de vista de la programación, no se puede hacer mucho con el objeto fileUpload, ya que no es posible cambiar su propiedad value desde un script. Lo único que se puede hacer es comprobar el valor y rechazar el envío si el archivo seleccionado es erróneo (por ejemplo).

Esta limitación se debe a motivos de seguridad para evitar que un sitio descargue archivos desde el ordenador de los usuarios sin que ellos lo sepan. Por ejemplo, un sitio podría descargar un archivo de claves de acceso windows y después utilizar métodos conocidos para romper la encriptación.

Formulario con etiqueta <input type="file">

```
<html>
<head>
  <script language="javascript">
    <!--
    function comprobar_vacio() {
      if (document.forms[0].archivo.value == "") {
        alert("Especifique el fichero para enviar al servidor.")
        return false
      }
      else {
        return true
      }
    }
```

```
    -->
  </script>
</head>
<body>
<h1>Envianos tus archivos!!</h1>
<form name="formulario">
<input type="file" name="archivo"><br>
<input type="submit" value="Enviar" onClick="return comprobar_vacio()">
</form>
</body>
</html>
```

Ejemplo anterior (Formulario con etiqueta <input type=”file”>**):** **fileUpload.html**

5.10 OBJETO hidden

Propiedades del objeto hidden

Propiedad	Significado
form	Referencia al formulario que contiene al objeto.
name	Mantiene el valor del atributo **name** (nombre asignado al campo oculto)
type	Mantiene el valor del atributo **type**
value	Mantiene el valor actual del campo oculto

Métodos del objeto hidden

Netscape no define ningún método para el objeto hidden, por lo que no existen métodos comunes a los dos navegadores.

Gracias a este objeto podemos almacenar información extra en el formulario de forma completamente invisible al usuario, pues no se verá en ningún momento que tenemos estos campos en el documento. Es parecido a un campo de texto (objeto text) salvo que no tiene valor por defecto (no tiene sentido pues el usuario no va a modificarlo) y que no se puede editar.

Enviar la fecha del cliente en el campo hidden de un formulario

```
<html>
<head>
  <script language="javascript">
    <!--
    function funcion_button()
      {
```

```
      var dFechaHoy=new Date();
      frm.hddOculto.value=dFechaHoy;
      alert(frm.hddOculto.value);
      frm.submit();
      }

    function VisFecha2()
      {
      var Mes=['enero','febrero','marzo','abril','mayo','junio','julio',
               'agosto','septiembre','octubre','noviembre','diciembre'];
      var Dia=['domingo','lunes','martes','miércoles','jueves','viernes',
               'sábado'];

      return Dia[this.getDay()]+" "+this.getDate()+" de "+
                Mes[this.getMonth()]+" de "+this.getFullYear();
      }

    Date.prototype.toString=VisFecha2;
    -->
  </script>
</head>
<body>
<form name="frm" action="mailto:nobody@punto.es" method="post">
<input type="hidden" name="hddOculto">
<br>
<input type="button"  value="Pasar valor a hidden y enviar"
   onClick="funcion_button()">
</form>
</body>
</html>
```

Ejemplo anterior (Enviar la fecha del cliente en el campo hidden de un formulario**):**
CampoHidden.html

5.11 OBJETO radio

Propiedades del objeto radio

Propiedad	Significado
checked	Mantiene el valor del atributo **checked** (indica si el elemento radio está seleccionado o no).
defaultChecked	Indica si el elemento radio debe estar seleccionado por defecto.
length	Número de elementos que hay en un grupo de botones de radio.
form	Referencia al formulario que contiene al objeto.
name	Mantiene el valor del atributo **name** (nombre asignado al elemento radio)
type	Mantiene el valor del atributo **type**
value	Mantiene el valor del atributo **value** (valor que se envía asociado al elemento radio cuando se manda el formulario al servidor).

Métodos del objeto radio

Método	Significado
blur()	Elimina el foco del objeto radio
click()	Simula el efecto que tiene pulsar el elemento radio
focus()	Asigna el foco al objeto radio

Los objetos radio o botones de radio sólo nos permiten elegir una de entre todas las posibilidades que hay. Están pensados para posibilidades mutuamente excluyentes (no se puede ser a la vez mayor de 18 años y menor de 18 años, no se puede estar a la vez soltero y casado, etc.).

Búsqueda del botón de radio activado de un grupo

```
<html>
<head>
  <script language="javascript">
    <!--
    function vis_radio(formulario_actual) {

      // Obtener el índice del botón activado
      var indice_radio = radio_activo(formulario_actual.grupo_buscadores)

      // Si un botón está activado, mostrar su valor en el cuadro de texto
      if (indice_radio >= 0) {
          formulario_actual.radio_seleccionado.value =
              formulario_actual.grupo_buscadores[indice_radio].value
      }
    }

    function radio_activo(grupo_radio) {

      // Repasar el grupo
      for (contador = 0; contador < grupo_radio.length; contador++) {

          // Al encontrar el botón activado, devolver el índice
          if (grupo radio[contador].checked) {
              return contador
          }
      }
      // Si no hay ningún botón activado, devolver -1
      return -1
    }
    -->
  </script>
</head>
<body>
<form>
<input type="radio"  name="grupo_buscadores" value="AV" checked>Altavista
<br>
<input type="radio"  name="grupo_buscadores" value="Google">Google
<br>
<input type="radio"  name="grupo_buscadores" value="Lycos">Lycos
<br>
<input type="radio"  name="grupo_buscadores" value="Yahoo">Yahoo
```

```
<br>
<input type="text" name="radio_seleccionado">
<br>
<input type="button"
    value="Mostrar valor del botón de radio activado"
    onClick="vis_radio(this.form)">
</form>
</body>
</html>
```

Ejemplo anterior (Búsqueda del botón de radio activado de un grupo**):**
BotonRadioSeleccionado.html

NOTA: Recordar que podemos saber si un objeto radio está activo o nó, o sea, checkeado o no, si usamos id´s diferentes en cada uno de ellos y podremos preguntar si la propiedad checked es true o false:

```
if (document.getElementById('radio1').checked)
   {
     alert('no tienes estudios');
   }
```

No olvidar definir el id de cada objeto radio:

```
<form>
<input type="radio" id="radio1" name="estudios">Sin estudios
<br>
<input type="radio" id="radio2" name="estudios">Primarios
<br>
<input type="radio" id="radio3" name="estudios">Secundarios
<br>
<input type="radio" id="radio4" name="estudios">Universitarios
<br>
<input type="button" value="Mostrar" onClick="mostrarSeleccionado()">
</form>
```

5.12 OBJETO checkbox

Propiedades del objeto checkbox

Propiedad	Significado
checked	Mantiene el valor del atributo **checked** (indica si el checkbox está seleccionado o no).
defaultChecked	Indica si el checkbox debe estar seleccionado por defecto.
form	Referencia al formulario que contiene al objeto.
name	Mantiene el valor del atributo **name** (nombre asignado al elemento checkbox)
type	Mantiene el valor del atributo **type**
value	Mantiene el valor del atributo **value** (valor que se envía asociado al elemento checkbox cuando se manda el formulario al servidor).

Métodos del objeto checkbox

Método	Significado
blur()	Elimina el foco del objeto checkbox
click()	Simula el efecto que tiene pulsar el elemento checkbox
focus()	Asigna el foco al objeto checkbox

Las checkboxes o casillas de verificación nos permiten seleccionar varias opciones marcando el cuadrito que aparece a su izquierda. El cuadrito pulsado equivale a un sí y sin pulsar a un no o, lo que es lo mismo, a true o false.

Desactivación de campos de formulario

```
<html>
<head>
  <script language="javascript">
    <!--
    function marcar_y_activar(formulario_actual) {

    // Obtener el estado actual de la casilla de verificación
    var controles_activados = formulario_actual.control_activado.checked

      // Si está desactivada mover el foco a la casilla de verificación
      if (!controles_activados) {
         formulario_actual.control_activado.focus()
      }
    }
    -->
  </script>
</head>
<body>
<form>
<input type="checkbox" name="control_activado">
Active esta casilla de verificación para activar los siguientes
cuadros de texto
<br>
<input type="text" onFocus="marcar_y_activar(this.form)">
<br>
<input type="text" onFocus="marcar_y_activar(this.form)">
</form>
</body>
</html>
```

Ejemplo anterior (Desactivación de campos de formulario**):**

CasillaVerificacionActivarCuadroTexto.html

Controlar el número de casillas de verificación activadas

```
<html>
<head>
  <script language="javascript">
    <!--
    // Esta variable configura el número máximo
    // de casillas de verificación que se pueden activar
    var maximo_activadas = 3

    // Esta variable configura el número mínimo
    // de casillas de verificación que se deben activar
    var minimo_activadas = 1

    // Esta variable codifica el número
    // total de casilla de verificación activadas.
    var total_activadas = minimo_activadas

    function contar_activadas(casilla_actual) {

      // Comprobar el estado de la casilla de verificación actual
      if (casilla_actual.checked) {

        // Si está activada, incrementar total_activadas
        total_activadas++
      }
      else {

        // Si está desactivada, disminuir total_activadas
        total_activadas--
      }

      // ¿El usuario ha sobrepasado el máximo?
      if (total_activadas > maximo_activadas) {

        // Si es así, desactivar la casilla de verificación actual
        casilla_actual.checked = false
        total_activadas = maximo_activadas
        // Mostrar un mensaje
        alert ("No puede activar más de " +
             maximo_activadas +
             (maximo_activadas == 1 ? " casilla de verificación." :
             " casillas de verificación."))
      }

      // ¿Se ha quedado el usuario por debajo del mínimo?
      if (total_activadas < minimo_activadas) {

        // Si es así, activar la casilla de verificación actual
        casilla_actual.checked = true
        total_activadas = minimo_activadas

        // Mostrar un mensaje
        alert ("Debe activar al menos " +
             minimo_activadas +
             (minimo_activadas == 1 ? " casilla de verificación." :
             " casillas de verificación."))
      }
    }
```

```
    function reinicializar_cuenta() {
      total_activadas = 0
    }
    -->
  </script>
</head>
<body>
<form onReset="reinicializar_cuenta()">
<input type="checkbox" name="Social" checked
      onClick="contar_activadas(this)">Agorafobia
              (Miedo a los espacios abiertos)
<br>
...
...
<input type="checkbox" name="Extranjeros"
      onClick="contar_activadas(this)">Xenofobia
              (Miedo a los extranjeros)
<br>
<p>
<input type="reset">
</form>
</body>
</html>
```

Ejemplo anterior (Controlar el número de casillas de verificación activadas**)**:

ControlarCasillaVerificacionActivadas.html

5.13 OBJETO select

Propiedades del objeto select

Propiedad	Significado
form	Referencia al formulario que contiene al objeto.
length	Mantiene el número de opciones definidas en el elemento select
name	Mantiene el valor del atributo **name** (nombre asignado al elemento select)
options	Array que mantiene una entrada por cada opción del elemento select, en el mismo orden en el que aparecen en el código HTML. Este atributo tiene además las propiedades descritas en el objeto **option**.
selectedIndex	Mantiene el valor del índice de la opción actualmente seleccionada (o el valor del índice de la primera opción marcada si se han seleccionado varias opciones).
type	Mantiene el valor del atributo **type**.

Métodos del objeto select

Método	Significado
blur()	Elimina el foco del objeto select.
focus()	Asigna el foco al objeto select.

Este objeto representa una lista de opciones dentro de un formulario. Puede tratarse de una lista desplegable de la que podremos escoger alguna (o algunas) de sus opciones (objeto option).

5.14 OBJETO option

Propiedades del objeto option

Propiedad	Significado
defaultSelected	Indica si la opción debe estar seleccionada por defecto.
selected	Indica si la opción está seleccionada o no.
text	Mantiene el texto de la opción.
value	Mantiene el valor del atributo **value** de la opción (valor que se envía asociado al elemento option cuando se manda el formulario al servidor).

NOTA: Recordar que con el **atributo size** podemos indicar el número de elementos del select que se verán, y el **atributo multiple** nos permite seleccionar más de un elemento usando las teclas <ctrl> o <Mayús>:

```
<SELECT NAME="lista1" MULTIPLE SIZE=3>
  <OPTION>Valor 1
  <OPTION SELECTED>Valor 2
  <OPTION>Valor 3
  <OPTION>Valor 4
  <OPTION>Valor 5
  <OPTION>Valor 6
  <OPTION>Valor 7
</SELECT>
```

Obtención de múltiples opciones seleccionadas

```
<html>
<head>
  <script language="javascript">
    <!--
    function obtener_seleccionados(lista_actual) {

      var array_seleccionados = new Array()
      var indice_actual = 0
      for (var contador = 0; contador < lista_actual.options.length;
                              contador++) {
        if (lista_actual.options[contador].selected) {
          array_seleccionados[indice_actual] =
                             lista_actual.options[contador].index
         indice_actual++
        }
      }
    return array_seleccionados
    }

    function vis_seleccionados(lista_actual) {

       var elegidos = new Array()
       elegidos = obtener_seleccionados(lista_actual)
       var mensaje = "Ha seleccionado los siguientes elementos:\n\n"
       for (contador = 0; contador < elegidos.length; contador++) {
         mensaje += lista_actual.options[elegidos[contador]].text + "\n"
       }
       alert(mensaje)
    }
    -->
  </script>
</head>
<body>
<form>
<select name="seleccion_multiple" multiple>
<option>Listado 28.1</option>
...
...
<option>Listado 28.8</option>
</select>
<br>
<input type="button"
    value="Mostrar selecciones"
    onClick="vis_seleccionados(this.form.seleccion_multiple)">
</form>
</body>
</html>
```

Ejemplo anterior (Obtención de múltiples opciones seleccionadas**):**
ListaSeleccionMultiple.html

Crear una lista que permita la navegación por diferentes páginas

```
<html>
<head>
  <script language="javascript">
    <!--
    function ir(lista_navegacion) {

      // Obtener la opción actualmente seleccionada
      var pagina nueva =
       lista_navegacion.options[lista_navegacion.selectedIndex].value

      // ¿Es una opción de página?
      if (pagina_nueva != "") {
        self.location = pagina_nueva
      }
    }
    -->
  </script>
</head>
<body>
<form>
<select name="sitios_navegar">
<option value="ContenidoFormularios.htm">
          Contenido de formularios</option>
<option value="ComprobarEnvioReset.htm">
          Métodos submit() y reset()</option>
...
...
<option value="CasillaVerificacionActivarCuadroTexto.htm">
          Controles desactivados</option>
</select>
<input type="button"
    value="¡Ir allí!"
    onClick="ir(this.form.sitios_navegar)">
</form>
</body>
</html>
```

Ejemplo anterior (Crear una lista que permita la navegación por diferentes páginas**):**
ListaNavegacion.html

Adición o borrado de opciones de una lista

```
<html>
<head>
  <script language="javascript">
    <!--
    // Crear el nuevo array de dos dimensiones
    var array_estados = new Array(63)

    // Rellenar el array con dos valores en cada elemento
    array_estados_populares()

    // Esta función asigna valores a las
```

```
// propiedades del objeto de dos dimensiones
function Estado(nombre, codigo) {
  this.nombre = nombre
  this.codigo = codigo
}

// Esta función inserta dos valores en cada elemento del array
function array_estados_populares() {
  array_estados[0] = new Estado("Alabama", "AL")
  array_estados[1] = new Estado("Alaska", "AK")
  ....
  ....
  array_estados[61] = new Estado("Quebec", "QC")
  array_estados[62] = new Estado("Saskatchewan", "SK")
}

function lista_populares(lista_actual) {

  // Reiniciar la lista
  lista_actual.length = 0

  // Revisar el array de estados
  for (contador = 0; contador < array_estados.length; contador++) {

    // Añadir el código de estado como valor
    // y el nombre del estado como texto
    lista_actual.options[contador] =
       new Option(array_estados[contador].nombre,
       array_estados[contador].codigo)
  }

  // Netscape 4 necesita actualizar la página
  /*if (its_ns4) {
      history.go(0)
    } */
  // Seleccionar el primer elemento
  lista_actual.options[0].selected = true
}

function vis_codigo(lista_actual) {

  // Asegúrese de que la lista está completa
  if (lista_actual.length > 0) {

    // Borrar la opción seleccionada
    alert(lista_actual.options[lista_actual.selectedIndex].value)
  }
}

function borrar_seleccionado(lista_actual) {

  // Asegúrese de que la lista está completa
  if (lista_actual.length > 0) {

   // Borrar la opción seleccionada
   lista_actual.options[lista_actual.selectedIndex] = null
  }
}
-->
```

```
  </script>
</head>
<body>
<form>
<select name="lista_estados">
</select>
<p>
<input type="button"
    value="Poblar la lista de estados"
    onClick="lista_populares(this.form.lista_estados)">
<p>
<input type="button"
    value="Mostrar código del estado"
    onClick="vis_codigo(this.form.lista_estados)">
<p>
<input type="button"
    value="Borrar opción seleccionada"
    onClick="borrar_seleccionado(this.form.lista_estados)">
</form>
</body>
</html>
```

Ejemplo anterior (Adición o borrado de opciones de una lista**)**: **ListaDinamica.html**

5.15 VALIDACIÓN DE FORMULARIOS

Una de las deficiencias mayores que presenta el lenguaje HTML reside en la imposibilidad de realizar controles sobre la información que el usuario introduce a través de los formularios. Esto provoca que los controles pertinentes tengan que realizarse en la parte del servidor, ralentizando el proceso de introducción de información, ya que en caso de producirse errores en la misma, se multiplican los accesos al servidor.

El lenguaje JavaScript proporciona los mecanismos necesarios para realizar la validación de la información dentro de la propia máquina cliente mejorando, de esta manera, el rendimiento global de los sistemas Web.

En este orden de cosas vamos a estudiar los siguientes apartados:

- **Validación de un carácter**
- **Validaciones alfabéticas**
- **Validaciones numéricas**
- **Validaciones de fechas**
- **Validación según un patrón**
- **Validación de números de tarjetas de crédito**

Validación de un caracter

En este apartado vamos a desarrollar una serie de funciones que permitirán validar los tipos de datos que aparecen con más frecuencia dentro del elemento input de tipo text de un formulario. Las ideas que subyacen dentro de dichas funciones pueden ser aplicadas para realizar cualquier otro tipo de validación que surja dentro de un problema concreto.

Las operaciones que se realizan para determinar la validez de la información son de dos categorías.

Por un lado, se hacen comprobaciones sintácticas encaminadas a determinar si los caracteres que forman las cadenas de información pertenecen a una determinada categoría y si su combinación sigue un determinado patrón.

Por otro lado, las comprobaciones semánticas controlan que la combinación de dichos caracteres conforme una información coherente.

Por ejemplo, si queremos validar un dato del tipo fecha primero determinaremos su corrección sintáctica, comprobando que está formado por uno o dos números, seguidos del carácter "/", seguido de uno o dos números, seguidos del carácter "/" y seguido de cuatro números. En segundo lugar, realizaremos las comprobaciones semánticas determinando la correcta relación entre los números de la fecha que representan el día y el mes (enero tiene 31 días, febrero 28...).

Antes de desarrollar las funciones especificas para la validación de los distintos tipos de datos, vamos a ver algunas funciones básicas de apoyo para la realización de las comprobaciones sintácticas. La tabla siguiente nos muestra el nombre de estas funciones junto con su descripción:

Función	Descripción
es_minuscula(car)	Determina si el carácter que recibe como parámetro se trata de una letra minúscula incluyendo la letra **ñ**, las **vocales acentuadas** y la letra **ü**.

es_mayuscula(car)	Determina si el carácter que recibe como parámetro se trata de una letra mayúscula incluyendo la letra **Ñ**, las **vocales acentuadas** y la letra **Ü**.
es_letra(car)	Determina si el carácter que recibe como parámetro se trata de una letra minúscula o mayúscula incluyendo la letra **ñ**, las **vocales acentuadas** y la letra **ü**.
es_numero(car)	Determina si el carácter que recibe como parámetro se trata de un **número**.
es_signo(car)	Determina si el carácter que recibe como parámetro se trata de un **+** o un **-**.

La implementación de estas funciones sigue la misma pauta. Se declara una cadena con todos los caracteres que pertenecen a la clase que se está tratando, y a continuación se pregunta por medio del método indexOf si el carácter recibido como parámetro pertenece a dicha cadena. Veamos la codificación:

```
function es_minuscula(car)
  {
  var MINUSCULAS="abcdefghijklmnnopgrstuvwxyzáéíóúü";
  return MINUSCULAS.indexOf(car) >= 0;
  }
function es_mayuscula(car)
  {
  var MAYUSCULAS="ABCDEFGHIJKLMNNOPQRSTUVWXYZÁÉÍÓÚÜ";
  return MAYUSCULAS.indexOf(car)>=0;
  }
function es_letra(car)
  {
  return es_minuscula(car) || es_mayuscula(car);
  }
function es_numero(car)
  {
  var NUMEROS="0123456789";
  return NUMEROS.indexOf(car) >=0;
  }
function es_signo(car)
  {
  var SIGNOS="+-";
  return SIGNOS.indexOf(car) >=0;
  }
```

Para el control de errores vamos a definirnos un objeto que va a contener dos atributos: uno con el texto explicativo del error que se ha producido y otro con la posición dentro de la cadena inspeccionada donde se ha detectado dicho error. Este objeto va a ser empleado por todas las funciones de validación de datos que vamos a desarrollar más adelante.

En el código JavaScript que aparece a continuación se detalla la función constructora del objeto llamada crear_error() así como el método rellenar() que permite actualizar los atributos de dicho objeto y devuelve siempre el valor false.

```
function crear_error()
  {
  this.mensaje="";
  this.posicion=0;
  this.rellenar=rellenar;
  }
```

```
function rellenar(mensaje,posicion)
  {
  this.mensaje=mensaje;
  this.posicion=posicion;
  return false;
  }

var error=new crear_error();
```

Ejercicio alumnado: Dado el ejemplo anterior crear una página web en la que uses las funciones de validación de un carácter, solicitando un valor al usuario y utilizando botones para probar si es minúsculas, mayúsculas, etc. (**FuncionesValidacion.html**)

Seguidamente vamos a presentar las validaciones típicas que se realizan dentro de un formulario. Para ello desarrollaremos un conjunto de funciones que devolverán un valor booleano resultado de determinar si una cadena es del tipo que comprueba la función.

Validaciones alfabéticas

El primer tipo de información que vamos a validar es la formada por caracteres del alfabeto castellano. Es bastante habitual que cuando rellenemos un formulario perteneciente a una página HTML, tengamos que introducir nuestro nombre, la calle en la que vivimos y, en general, una serie de información escrita en nuestro idioma. Seguidamente presentamos tres funciones que nos permiten determinar si una cadena está formada por caracteres pertenecientes al alfabeto castellano en minúsculas, en mayúsculas o sin hacer distinción entre unas y otras:

```
function comprobar_palabra_en_minusculas(contenido,error)
  {
  if(contenido.length ==0)
   return error.rellenar("Campo vacio no contiene ningun valor",1);
      for(var i=0;i < contenido.length; i++)
    if(!es_minuscula(contenido.charAt(i)))
      return error.rellenar("Falta caracter alfabético" +
                            " en minúsculas",i+1);
  return true
  }

function comprobar_palabra_en_mayusculas(contenido,error)
  {
  if (contenido.length ==0)
   return error.rellenar("Campo vacio no contiene ningun valor",1);
      for(var i=0;i < contenido.length; i++)
    if (!es_mayuscula(contenido.charAt(i)))
      return error.rellenar("Falta caracter alfabético" +
                            " en mayúsculas",i+1);
  return true
  }

function comprobar_palabra(contenido,error)
```

```
{
if(contenido.length==0)
 return error.rellenar("Campo vacio no contiene ningun valor",1);
    for(var i=0;i < contenido.length; i++)
      if(!es_letra(contenido.charAt(i)))
     return error.rellenar("Falta caracter alfabético",i+1);
return true
}
```

Las tres funciones tienen una estructura similar que consiste en recorrer cada uno de los caracteres de la cadena inspeccionada comprobando que pertenecen a la categoría deseada, apoyándose para ello respectivamente, en las funciones básicas es_minuscula(), es_mayuscula() y es_letra() desarrolladas con anterioridad. Adicionalmente se comprueba que la cadena inspeccionada no está vacía.

Ejercicio alumnado: Dado el ejemplo anterior probarlo (**ValidacionAlfabetica2.html**)

Validaciones numéricas

En la mayoría de aplicaciones Web surge con bastante frecuencia la necesidad de introducir información de carácter numérico. Aunque el propio lenguaje JavaScript proporciona medios para convertir cadenas a números a través de las funciones parseInt() y parseFloat() es necesario desarrollar un conjunto de funciones específicas que determinen si un dato es de tipo numérico.

Los datos de tipo numérico a su vez se dividen en una serie de subtipos de los que nosotros vamos a distinguir tres: **naturales, enteros y reales**.

Los **números naturales** se caracterizan por estar formados nada más que por caracteres numéricos. En definitiva son números sin signo que sirven fundamentalmente para contar cosas. La función que determine si una cadena es un numero natural válido, básicamente realiza una inspección de dicha cadena comprobando por medio de la función **es_numero()** que todos los caracteres que la forman son números.

```
function comprobar_natural(contenido,sin_ceros_izquierda,error)
  {
  if(contenido.length==0)
    return error.rellenar("Campo vacio no contiene ningun valor",1);
  for(var i=0;i < contenido.length; i++)
    if(!es_numero(contenido.charAt(i)))
      return error.rellenar("Carácter ilegal en un número",i+1);
  if(sin_ceros_izquierda &&
     (contenido.charAt(0)=='0' && contenido.length>1))
    return error.rellenar("Número con cero ó ceros a la izquierda",2);
  return true;
  }
```

Como se puede apreciar en el código anterior, si la variable **sin_ceros_izquierda** contiene el valor **true**, se realiza una comprobación semántica consistente en descartar como números naturales válidos aquellos que tenga un cero a la izquierda, como ocurre en las cadenas "00", "01" o "001".

Los **números enteros** son una ampliación de los naturales ya que opcionalmente pueden estar precedidos de un signo positivo o negativo. El código siguiente es de una función que valida números enteros.

```
function comprobar_entero(contenido,sin_ceros_izquierda,error)
  {
  if(contenido.length==0)
    return error.rellenar("Campo vacio no contiene ningun valor",1);
  for(var i=0;i < contenido.length; i++)
    if(!es_numero(contenido.charAt(i)))
      if (!(i==0 && es_signo(contenido.charAt(0))))
        return error.rellenar("Carácter ilegal en un número",i+1);
  if (es_signo(contenido.charAt(0)) && contenido.length==1)
    return error.rellenar("Solo tiene un signo, inserte un número",2);
  if (sin_ceros_izquierda &&
       (contenido.charAt(0)=='0' && contenido.length>1) ||
       (es_signo(contenido.charAt(0)) && contenido.charAt(1) == '0' &&
       contenido.length > 2))
    return error.rellenar("Número con cero ó ceros a la izquierda",2);
  return true;
  }
```

La base de la codificación anterior es una inspección de la cadena contenido con el objeto de determinar que todos los caracteres que la forman son números, excepto el primero que también puede ser un signo (caracteres + y -). Además se realizan las comprobaciones sintácticas que determinan que la cadena no está vacía y que no está compuesta por un único carácter de tipo signo. Finalmente, si la variable sin_ceros_izquierda tiene el valor true, se realiza una comprobación semántica consistente en desestimar como cadenas de tipo entero aquellas que tengan más de un cero a la izquierda.

La validación de **números reales** vamos a realizarla apoyándonos en las funciones desarrolladas anteriormente. Al fin y al cabo, un número real se puede definir como la concatenación de un número entero, el carácter punto decimal, y un número natural con ceros a la izquierda.

```
function comprobar_real(contenido,sin_ceros_izquierda,error)
  {
     var punto_decimal=",";
  if(contenido.length==0)
   return error.rellenar("Campo vacio no contiene ningun valor",1);
  posicion_punto=contenido.indexOf(punto_decimal);
  if (posicion_punto < 0)
    return error.rellenar("Número real sin punto decimal",
                          contenido.length+1);
  var parte_entera=contenido.substring(0,posicion_punto);
  var parte decimal=contenido.substring(posicion punto+1,
                                        contenido.length);
  if (parte_entera.length < 1)
    return error.rellenar("Falta la parte entera del número",
                          posicion_punto);
  if (parte decimal.length < 1)
    return error.rellenar("Falta la parte decimal del número",
                          posicion_punto+1);
  if (!comprobar_entero(parte_entera,true,error))
    return false;
  if (!comprobar_natural(parte_decimal,false,error))
    return false;
  return true;
  }
```

Los detalles más importantes del código anterior son los siguientes:

- Se busca el carácter **punto_decimal** dentro de la cadena **contenido** utilizando el método **indexOf()**. Si no existe se descarta como válida la cadena.
- Se divide la cadena en dos partes utilizando el método **substring()**: la parte entera que va desde el principio de la cadena hasta el carácter que se encuentra antes del **punto_decimal** y la parte decimal que va desde el caracter después del **punto_decimal** hasta el final de la cadena.
- Se comprueba que tanto la parte entera como el decimal tienen al menos 1 carácter.
- Se comprueba que la cadena correspondiente a la parte entera es un número entero invocando a la función desarrollada **comprobar_entero()** y por medio de la función **comprobar_natural()** que la parte decimal se trata de un número natural que puede llevar ceros a la izquierda.

Nótese que solo se considera número real aquel que tiene un punto decimal. En el caso de que deseemos considerar como real también cualquier numero entero, podríamos utilizar la función comprobar_real2(), que comprueba dicha circunstancia:

```
function comprobar_real2(contenido,sin_ceros_izquierda,error)
  {
  if (!comprobar_entero(contenido,sin_ceros_izquierda,error)  &&
      !comprobar_real(contenido,sin_ceros_izquierda,error))
    return false;
  return true;
  }
```

Ejemplo anterior (Comprobar los algoritmos de validación numérica**):**
ValidacionNumerica.html

Validaciones de fechas

Otro tipo de datos utilizado con frecuencia es el tipo de datos fecha. Antes de desarrollar la función correspondiente, tenemos que definir cuál va a ser el formato de fecha que queremos validar ya que existen distintos tipos. En concreto, vamos a tratar con fechas que cumplan las siguientes reglas sintácticas:

- Las fechas estarán formadas por 3 números separados por un carácter especial denominado separador.
- El primer número denota el día del mes y estará formado por 1 o 2 dígitos. Para valores de número de día menores de 10 se podrá optar por incluir un cero a la izquierda o no hacerlo. De esta manera el primer día del mes se expresara como **1** o **01**.
- El segundo número representa el número de mes y también estará formado por 1 o 2 dígitos. Por ejemplo, febrero se corresponde con **2** o **02**
- El tercer número representa el numero de año de la fecha y obligatoriamente tendra 4 dígitos. Así el número **0799** es la representación del año **799**.

La función que nos valida fechas va a tener dos partes bien diferenciadas encargadas, respectivamente, de realizar las comprobaciones sintácticas y semánticas.

La sintaxis correcta de una fecha se determina inspeccionando los caracteres que la forman comprobando que coinciden, o bien con un carácter numérico, o bien con un carácter separador de fecha. Además, deben existir dos caracteres del tipo separador. Esto precisamente es lo que hace el siguiente código:

```
// Comprobación de la sintáxis de una fecha
for (var i=0; i < contenido.length; i++)
  {
  var caracter=contenido.charAt(i);
  if (!es_numero(caracter) && caracter !=      separador_fecha)
    return error.rellenar("Carácter ilegal en una fecha",i+1);
  if (caracter == separador_fecha)
    numero_separadores++;
  }
if (numero_separadores !=       2)
  return error.rellenar("Faltan separadores en una fecha",i+1);
```

Para afirmar que una fecha es correcta desde el punto de vista semántico tenemos que comprobar que los valores de los elementos que la forman (día, mes y año)) son coherentes con la realidad que representan. Esto quiere decir que, por ejemplo, el valor que toma el mes debe estar entre 1 y 12, o que si el mes es el numero 4 (abril), el valor del día puede oscilar entre 1 y 30. Por lo tanto, para poder realizar comprobaciones tendremos que obtener por separado los valores del día, mes y año que es precisamente lo que realiza el siguiente código:

```
// Comprobación de la semántica de una fecha
var posicion_separador_1=contenido.indexOf(separador_fecha);
var dia=contenido.substring(0,posicion_separador_1);
var posicion_separador_2=
          contenido.indexOf(separador_fecha, posicion_separador_1+1);
var mes=contenido.substring(posicion_separador_1+1,posicion_separador_2);
var anio=contenido.substring(posicion_separador_2+1,10);
```

Lo que hemos hecho es localizar en qué posición se encuentran los dos separadores de fecha utilizando el método indexOf() y a continuación, tomando como referencia esos valores, obtener las subcadenas correspondientes al dia, mes y año por medio del método substring(): la subcadena que contiene el día está comprendida entre la posición que ocupa el primer carácter de la cadena y la del primer carácter separador de fecha; el mes se obtendrá de la subcadena comprendida entre la posición siguiente a la del primer carácter separador de fecha y la del segundo carácter separador de fecha y finalmente, la subcadena que representa al año está delimitada por las posiciones ocupadas por el carácter siguiente al segundo carácter separador de fecha y el último carácter de la cadena que se está validando.

Una vez obtenidos los elementos que componen una fecha pasamos a realizar las comprobaciones necesarias. Un año será correcto siempre y cuando esté formado por 4 cifras. En cuanto al valor numérico de un mes se estimará como válido cuando se encuentre en el rango que va de 1 a 12. Lo más complicado es validar el día de la fecha ya que dependiendo del mes podrá tomar unos valores u otros. Es más, incluso es necesario tener en cuenta el valor del año, ya que febrero que es el mes numero 2 puede tener 28 o 29 días dependiendo de si se trata de un año bisiesto.

```
if (anio.length != 4)
  return error.rellenar("Año incorrecto en una fecha",6);
if (mes < 1 || mes > 12)
  return error.rellenar("Mes incorrecto en una fecha",4);
```

```
if ((dia < 1 || dia > 31) || (mes == 4 && dia > 30) ||
    (mes ==      6 && dia > 30) || (mes == 9 && dia > 30) ||
    (mes == 11 && dia > 30) || (mes == 2 && es_bisiesto(anio) &&
     dia > 29) ||
    (mes ==      2 && !es_bisiesto(anio) && dia > 28))
  return error.rellenar("Día incorrecto en una fecha",2);
return true
```

La función es_bisiesto() determina si un año cumple esta propiedad aplicando la regla que dice que un año es bisiesto cuando, o bien es divisible por 4 pero no por 100, o bien es divisible por 400. La codificación es la siguiente:

```
function es_bisiesto(anio)
  {
  if (((anio % 4 == 0) && anio % 100 != 0) || anio % 400 == 0)
    return true;
  return false;
  }
```

Juntando las porciones de código que hemos explicado por separado en este apartado, obtenemos la función comprobar_fecha() para validar cadenas de caracteres cuyo contenido queremos que sea una fecha y que podemos estudiar en el siguiente ejemplo:

Ejercicio alumnado: Dados los ejemplos anterior probarlo validando una fecha solicitada al usuario (**ValidacionFecha.html**)

Validación según un patrón

En este apartado vamos a desarrollar una función que validará una cadena teniendo en cuenta una serie de reglas sintácticas expresadas por medio de un formato o patrón.

Consideramos que un patrón es una cadena de caracteres cada uno de los cuales puede tener un significado especial. Realmente en un patrón distinguiremos entre caracteres normales que se representan a sí mismos y caracteres especiales o metacaracteres cuyo significado depende de unas reglas preestablecidas.

Cuando definimos un patrón nos creamos un modelo que representa a un conjunto de cadenas que concuerdan o cumplen con dicho modelo. La función que vamos a desarrollar determinará si una cadena se ajusta a un patrón, es decir, si la cadena pertenece al conjunto de cadenas definidas por el patrón.

Para construir un patrón concatenamos caracteres y metacaracteres en función del conjunto de cadenas que queramos representar. En nuestro caso vamos a considerar los metacaracteres que se describen en la siguiente tabla:

Metacaracteres	Nombre	Significado
Aa	ALFABETICO	Representa cualquier carácter alfabético perteneciente al alfabeto castellano.
Nn	NUMERICO	Representa cualquier carácter numérico
Xx	ALFANUMERICO	Representa cualquier carácter alfanumérico perteneciente al alfabeto castellano.

<	MAYUSCULAS	Activa o desactiva el control de mayúsculas en los caracteres alfabéticos y alfanuméricos.
>	MINUSCULAS	Activa o desactiva el control de minúsculas en los caracteres alfabéticos y alfanuméricos.
[	APERTURACONJUNTO	Carácter delimitador inicial de un conjunto de caracteres.
]	CIERRECONJUNTO	Carácter delimitador final de un conjunto de caracteres.
%	ESCAPE	Provoca que el carácter que le sigue no se interprete como metacarácter.

Veamos algunos **ejemplos de patrones** que aclaren el significado de los metacaracteres que aparecen en la tabla anterior.

Con el patrón "SoLnN" se ajustarán todas aquellas cadenas que empiecen por el carácter "S" seguido del carácter "o", seguido del carácter "L", seguido de un carácter numérico y seguido de otro carácter numérico. De esta manera serán válidas las cadenas "SOL00","SoL18" y "SoL24" y no lo serán "SOLOO", "SoL222" y "SoLnN".

Con el patrón "aAnnX" se ajustarán todas aquellas cadenas que empiecen por una letra del alfabeto castellano, seguida de otra letra del alfabeto castellano, seguida de un carácter numérico, seguido de otro carácter numérico y seguido de un carácter que puede ser un número o una letra. Por lo tanto, serán válidas las cadenas "aallx","bc22d" y "dd222", pero no lo serán "22222", "2aallx" y "bc22+".

Con el patrón "aX>X" concordarán todas aquellas cadenas que empiecen por una letra del alfabeto castellano (se activa el control de letras minúsculas), seguida de una letra en minúsculas del alfabeto castellano, seguida de otra letra en minúsculas del alfabeto castellano (se activa el control de letras mayúsculas), seguida de un carácter que puede ser un número o una letra en mayúsculas del alfabeto castellano (se desactiva el control de letras mayúsculas) y seguido de otro carácter que puede ser un número o una letra del alfabeto castellano. De esta manera serán válidas las cadenas "AaaAa", "xcdMA" y "abcD7" y no lo serán "AAAAA", "aaaaa" y "AaaAa5".

Con el patrón "%[[abc],[123]%]" concordarán todas aquellas cadenas que comiencen por el carácter "[" (este carácter pierde su condición de metacaracter por ir precedido del carácter "%"), seguido de uno de los caracteres pertenecientes al conjunto formado por "a", "b" y "c", seguido del carácter ",", seguido por un número entre 1 y 3, y finalmente seguido del carácter "]" (este carácter está escapado). Las cadenas "[a,1]","[b,3]" y "[c,2]" concuerdan con el patrón, mientras que no lo hacen "[a,4]","a,l" y "[[a 1]]".

La función que vamos a desarrollar recibe dos parámetros que son la cadena que queremos validar y el patrón que debe cumplir dicha cadena. El eje central de la función es un bucle que va obteniendo cada uno de los caracteres o metacaracteres del patrón y comprobando que tienen su correspondencia con los caracteres de la cadena tal y corno muestra el siguiente esquema:

```
for (var i=0 ;i<patron.length; i++)
  {
  if (ALFABETICO (patron.charAt(i))
    // Comprobar que el siguiente carácter de la cadena es alfabético. Si
    // está activada la comprobación de mayúsculas el carácter de la cadena
    // debe estar en mayúsculas. Si está activada la comprobación de
```

```
  // minúsculas el carácter de la cadena debe estar en minúsculas.
else
if (NUMERICO (patron.charAt(i))
  // Comprobar que e1 siguiente carácter de la cadena es numérico.
else
if (ALFANUMERICO (patron.charAt(i))
  // Comprobar que el siguiente carácter de la cadena es alfanumérico. Si
  // está activada la comprobación de mayúsculas e1 carácter de la cadena
  // debe estar en mayúsculas. Si está activada la comprobación de
  // minúsculas el carácter de la cadena debe estar en minúsculas.
else
if (APERTURACONJUNTO (patron.charAt(i))
  // Obtener los siguientes caracteres del patrón has de encontrar el
  // carácter CIERRECONJUNTO. Comprobar que e1 siguiente carácter de
  // la cadena es uno de los caracteres obtenidos que forman parte del
  // conjunto.
else
if (MAYUSCULAS (patron.charAt(i))
  // Cambiar el estado de la comprobación de mayúsculas.
else
if (MINUSCULAS (patron.charAt(i))
  // Cambiar el estado de la comprobación de minúsculas.
else
if (ESCAPE (patron.charAt(i))
  // Comprobar que el siguiente carácter de la cadena es el siguiente
  // carácter del patrón.
else // CARÁCTER NORMAL
  // Comprobar que el siguiente carácter de la cadena es el carácter
  // actual del patrón.
}
```

Vamos a profundizar en el desarrollo del esquema anterior. En primer lugar, necesitamos dos variables booleanas que nos indiquen el estado de la comprobación de letras mayúsculas y minúsculas. Estas variables se van a llamar respectivamente en_mayusculas y en_minusculas. También necesitamos dos variables enteras llamadas i y j que nos indiquen la posición del carácter que se está procesando en cada momento correspondiente al patrón y a la cadena examinada.

El tratamiento de los metacaracteres MAYUSCULAS y MINUSCULAS va a consistir en cambiar de estado la variable correspondiente y desactivar la complementaria. Por ejemplo el código asociado con MAYUSCULAS es:

```
if (MAYUSCULAS.indexOf(patron.charAt(i)) >= 0)
  {
  en_mayusculas=!en_mayusculas;
  en_minusculas=false;
  }
```

Los metacaracteres ALFABETICO, NUMERICO y ALFANUMERICO se tratan de una manera parecida que básicamente consiste en comprobar que el carácter situado en la posición j-ésima de la cadena pertenece al conjunto de datos que cada metacarácter representa. Dicha comprobación se realiza por medio de las funciones básicas de comprobación de sintaxis definidas al principio de este apartado dedicado a validaciones. Valga como ejemplo el código asociado al metacarácter ALFABETICO teniendo en cuenta los distintos estados del control de mayúsculas y minúsculas.

```
  // comprobar carácter ALFABÉTICO
  if (ALFABETICO.indexOf(patron.charAt(i)) >=   0 &&
```

```
   !en_mayusculas && !en_minusculas)
  if (es_letra(contenido.charAt(j)))
    j++;
  else
    return error.rellenar("Falta carácter alfabético",j+1);
else
//; comprobar caracter ALFABÉTICO en minúsculas
if (ALFABETICO.indexOf(patron.charAt(i)) >= 0 &&
    !en_mayusculas && en_minusculas)
  if (es_minuscula(contenido.charAt(j)))
    j++;
  else
    return error.rellenar("Falta carácter alfabético en" +
                          " minúsculas",j+1);
else
// comprobar caracter ALFABÉTICO an mayúsculas
if (ALFABETICO.indexOf(patron.charAt(i)) >= 0 &&
    en_mayusculas && !en_minusculas)
  if (es_mayuscula(contenido.charAt(j)))
    j++;
  else
    return error.rellenar("Falta carácter alfabético en" +
                          " mayúsculas",j+1);
```

Cuando el elemento a tratar dentro del patrón sea un carácter específico, el trabajo de nuestra función se limitará a comprobar que el carácter respectivo dentro de la cadena coincide con el del patrón. Si dentro del patrón nos encontramos el metacarácter ESCAPE, avanzaremos hasta el siguiente carácter y lo trataremos como un carácter específico.

```
if (ESCAPE.indexOf(patron.charAt(i)) >= 0)
  {
  i++;
  if (i >= patron.length)
    return error.rellenar("Formato del patron incorrecto",j+1);
  }
if (patron.charAt(i) == contenido.charAt(j))
  j++;
else
    return error.rellenar("Falta carácter específico",j+1);
```

Finalmente, en el tratamiento del metacarácter APERTURACONJUNTO hay que realizar dos acciones: construir un conjunto de caracteres válidos y comprobar que el carácter actual de la cadena pertenece al conjunto construido. Para realizar la primera acción, obtendremos los siguientes caracteres del patrón por medio de un bucle y los iremos concatenando en una cadena llamada CONJUNTO. Dicho bucle terminará cuando encontremos el metacarácter CIERRECONJUNTO. Lógicamente, si se quiere incluir como carácter del conjunto el metacarácter anterior será necesario escaparlo. La segunda acción se realiza comprobando que el carácter actual de la cadena examinada coincide con alguno de los contenidos en la cadena CONJUNTO utilizando el método indexOf(). El código para esta parte de la función es el siguiente:

```
if (APERTURACONJUNTO.indexOf(patron.charAt(i)) >= 0)
  {
  var CONJUNTO="";
  for (i=i+1;  i < patron.length; i++)
    if (ESCAPE.indexOf(patron.charAt(i)) >= 0)
```

```
    {
    i++;
    if (i >= patron.length)
      return error.rellenar("Formato del patrón incorrecto",j+1);
    CONJUNTO+=patron.charAt(i);
    }
  else
    if (CIERRECONJUNTO.indexOf(patron.charAt(i)) >= 0 )
      break;
    else
      CONJUNTO+=patron.charAt(i);
if (i >= patron.length)
  return error.rellenar("Formato del patrón incorrecto",j+1);
if (CONJUNTO.indexOf(contenido.charAt(j))>=0)
  j++;
else
  return error.rellenar("Falta carácter de un conjunto",j+1);
}
```

La función que nos valida una cadena que coincide con un patrón se completa con algunas comprobaciones adicionales que permiten determinar que no existen más caracteres en el patrón que en la cadena o viceversa y que el patrón está bien formado (siempre existe un carácter después del metacarácter ESCAPE y los conjuntos de datos se abren y cierran correctamente). El siguiente ejemplo permite comprobar todo el código anterior.

Ejemplo anterior (Comprobar la validación según un patrón**):** **ValidacionPatron.html**

Validación de números de tarjetas de crédito

En la mayoría de aplicaciones de comercio electrónico, el pago de las mercancías adquiridas se realiza por medio de una tarjeta de crédito. Resulta necesario, pues, tener una función escrita en JavaScript que determine si un número de tarjeta de crédito es correcto. De esta manera, validaremos esta información sin necesidad de realizar ningún proceso en el servidor. En cualquier caso, **la función que vamos a desarrollar solo nos proporciona la seguridad de que el número de tarjeta de crédito cumple las reglas establecidas para su formación, pero no garantiza que la tarjeta tenga crédito o esté activada.** Para comprobar estos últimos datos necesitaremos comunicarnos con la entidad emisora de la tarjeta por medio de algún mecanismo y que ésta nos acepte el cargo correspondiente al importe de los objetos adquiridos.

Dependiendo de la entidad emisora nos encontramos con unos valores característicos determinados en relación al prefijo por el que comienzan y el número de dígitos que forman el número de tarjeta. Estos valores característicos aparecen resumidos en la siguiente tabla:

Tipo de tarjeta	Prefijo	Número de dígitos
American Express	34, 37	15
Diner´s Club	300-305, 36, 38	14
Master Card	51-55	16
Visa	4	13,16

Para determinar que **un número de tarjeta de crédito es correcto** se aplica el algoritmo que se describe a continuación:

Se numeran los dígitos que forman el número de la tarjeta de derecha a izquierda comenzando por el número 1. Esto quiere decir que el dígito más a la derecha es el número 1, el segundo dígito más a la derecha el número 2, y así sucesivamente.

Todos los dígitos pares según la numeración establecida se multiplican por 2. Si el resultado es un número con 2 dígitos se reduce al valor obtenido de sumar los dos dígitos. Veamos algunos ejemplos: 5 x 2 = 10 se reduce a 1 + 0 = 1; 6 x 2 = 12 se reduce a1 1 + 2 = 3; 9 x 2 = 18 se reduce a1 1 + 8 = 9.

Se suman los dígitos impares con los productos de los dígitos pares que se calcularon en el paso anterior.

Se divide la suma anterior por 10. Si el resto de la división es cero el número de tarjeta de crédito es correcto, en caso contrario, se considerara incorrecto.

Para aclarar el algoritmo anterior vamos a desarrollar un ejemplo. Supongamos que queremos validar la tarjeta de crédito número 345833102446893 de la entidad American Express. Como primer paso, numeramos sus dígitos:

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
3	4	5	8	3	3	1	0	2	4	4	6	8	9	3

Ahora multiplicamos por 2 los dígitos situados en posiciones pares:

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
3	8	5	16	3	6	1	0	2	8	4	12	8	18	3

Reducimos los números de dos dígitos a un solo dígito:

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
3	8	5	7	3	6	1	0	2	8	4	3	8	9	3

Sumamos los valores anteriores y obtenemos el valor 70 que es un número divisible por 10 por lo que se trata de un número correcto de tarjeta de crédito.

La función que nos va a validar un número de tarjeta de crédito, comenzará comprobando que tanto el prefijo como la longitud de dicho número son correctos, teniendo en cuenta qué entidad es la emisora de la tarjeta. Por ejemplo, si la tarjeta pertenece a la entidad Visa, comprobaremos que su numeración comienza por el número 4 (obteniendo el primer carácter de la cadena) y que dicha numeración está formada por 13 o 16 dígitos (obteniendo el tamaño de la cadena):

```
case "VISA":
          if(!(contenido.length==13 || contenido.length==16))
      return error.rellenar("Número incorrecto de dígitos la tarjeta" +
                            " VISA tiene 13 ó 16",contenido.length);
       if (contenido.substring(0,1)!="4")
      return error.rellenar("Las tarjetas VISA comienzan" +
                                 " por el dígito 4",1);
          break;
```

La implementación del algoritmo que valida la corrección de un número de tarjeta de crédito es la siguiente:

```
//comprobación del número de tarjeta
var sum1=0;
var sum2=0;
var impar=true;
for( var i=(contenido.length-1);i>=0;i--)
  {
  if (!es_numero(contenido.charAt(i)))
    return error.rellenar("Las tarjetas de crédito" +
                    " solo contienen dígitos",contenido.length);
  else if(impar)
    {
    sum1+=contenido.charAt(i)*1;
    }
  else
    {
    var aux=contenido.charAt(i)*2;
    if (aux>=10)
      aux-=9;
    sum2+=aux;
    }
  impar=(impar==true?false:true);
  }
if ((sum1+sum2)%10!=0)
  return error.rellenar("No concuerda" +
              " el dígito de control",contenido.length);
return true;
```

Como se puede apreciar se utilizan dos variables llamadas suml y sum2. En la primera se acumula la suma de los dígitos situados en posiciones impares y en la segunda, la suma del doble reducido a un dígito de los dígitos situados en posiciones pares. La variable booleana impar se utiliza para controlar si se trata de un dígito situado en una posición de este tipo o no.

Por medio de un bucle que comienza en el último carácter de la cadena y acaba en el primero, vamos obteniendo cada uno de los caracteres y comprobando que son números (dígitos) con la función es_numero(). Además dentro de dicho bucle, en función del valor de la variable impar se acumulará el dígito correspondiente en las variables suml o sum2. En el caso de tratarse de un dígito situado en posición par se calcula su producto por 2 y si el valor resultante sobrepasa 10 se le reduce a un solo dígito restándole 9 unidades. Este último valor obtenido es el que realmente se acumula en sum2. La variable impar tomará alternativamente los valores true y false en cada iteración del bucle.

Finalmente, una vez procesados todos los caracteres de la cadena se suman los dos acumuladores sum1 y sum2 y sobre este resultado se calcula el resto de dividir por 10. Si este resto es cero se trata de un número de tarjeta correcto, en caso contrario, definitivamente es un número de tarjeta inválido.

El siguiente ejemplo permite comprobar todo el código anterior.

Ejemplo anterior (Comprobar la validación de una tarjeta de crédito**):**
ValidacionTarjetaCredito.html