JavaScript de Andy Harris

TEMA 5 – Nuevos elementos de formulario

En el tema anterior, los programas recibían las entradas del usuario en cuadros de texto y áreas de texto. Aunque los elementos básicos de texto son muy flexibles, HTML y JavaScript nos permiten usar otros elementos de formulario. En este tema vamos a aprender a trabajar con otros elementos de formulario de HTML y cómo JavaScript interactúa con ellos. Concretamente vamos a:

• Usar otros elementos de texto, cómo contraseña y cuadros de entrada ocultos (*hidden input boxes*).

• Crear en interactuar con formularios que contengan cuadros de validación (*check boxes*) .

• Utilizar botones de radio (*radio buttons*) y entender las diferencias con los cuadros de validación.

• Organizar los elementos de pantalla en vectores (*arrays*).

• Construir y utilizar listas de selección (*selection lists*) para la entrada del usuario.

Proyecto: “La Historia disparatada 2.0”

En este tema, vamos a construir una versión avanzada del programa del tema 4 “La Historia disparatada”. La idea será básicamente la misma, pero esta tendrá nuevos elementos de formulario (ver Figura 5.1). Esta versión será más fácil para el usuario, porque no tendrán que teclear casi nada. Estos elementos aportan también ventajas para el programador, ya que limitan los valores que puede introducir el usuario y además sólo se permitirán valores válidos.

Trabajando con objetos de tipo texto

Hemos visto cómo manejar cuadros de texto y áreas de texto en el tema anterior. HTML tiene una serie de elementos muy parecidos desde el punto de vista del programador. Estos elementos se crean (como la mayoría de los elementos de formulario) con variantes de la etiqueta `<input>`.

El programa “Palabra de paso”

El programa “Palabra de paso” muestra algunos de los elementos de entrada de tipo texto (ver Figura 5.2). Aunque sólo vemos cuatro elementos en la página, realmente contiene cinco elementos.

Figura 5.1
Figura 5.2

La distribución de la pantalla es relativamente sencilla. Este es el código HTML que la genera:

```
<body>
<center><h1>contraseña<hr /></h1>
<form name="miForm" id="miForm" action="">
<table border="1">
<tr>
<td>Por favor, teclee su contraseña</td>
<td><input type="password" name="pwdIntento" id="pwdIntento"/>
</tr>
<tr> <td colspan="2"><center>
<input type="button" value="Click para validar"
     onclick="compruebaPassword()">
</center> </td>
</tr>
<tr> <td colspan="2">
<center> <textArea name="txtSalida" id="txtSalida" rows="1" cols="35">
</textarea> </center> </td>
</tr>
</table>
<input type="hidden" name="hdnSecreto" value="JavaScript">
</form>
<hr />
</center>
</body>
```

El formulario se distribuye con la ayuda de una tabla, y la tabla contiene cuatro elementos. El primero es un mensaje de texto, a continuación un cuadro de contraseña, luego un botón y por último un área de texto. Fuera de la tabla tenemos un quinto elemento, un campo oculto (*hidden field*). Este elemento contiene el valor correcto de la palabra de paso. Cuando el usuario hace click en el botón, la función compruebaPassword() se ejecuta. Este es el código de la función:

```
<head>
<title>Contraseña</title>
<script type="text/javascript">
function compruebaPassword(){
// Contraseña
// muestra el uso de varios elementos de tipo texto
var intento = document.miForm.pwdIntento.value;
var secreto = document.miForm.hdnSecreto.value;
if (intento == secreto){
document.miForm.txtSalida.value = "Puede seguir.";
} else {
document.miForm.txtSalida.value = "contraseña incorrecta.";
} // end if
} // end function compruebaPassword
</script>
</head>
```

Esta función obtiene los valores de los diferentes elementos de texto, compara los valores y devuelve el mensaje apropiado.

Cuadros de contraseña

Los cuadros de contraseña con muy parecidos a los cuadros de texto para lo que a y JavaScript se refiere. La única diferencia real es que `<input type="password">` los valores que le usuario teclea se sustituyen por asteriscos (*). Los caracteres que el usuario teclea se guardan con su valor real, pero en la pantalla sólo se ven asteriscos. Esto evita que cualquiera que esté mirando pueda ver la contraseña que teclea. Los cuadros de constraseña tienen una propiedad value, y podemos usar JavaScript para copiar valores desde y hacia un cuadro de contraseña al igual que con un cuadro de texto.

CUIDADO – Los cuadros de contraseña son potencialmente inseguros, especialmente cuando trabajamos con JavaScript. El usuario puede averiguar fácilmente la contraseña mirando el código fuente de la paǵina. JavaScript no es suficientemente seguro para transacciones que impliquen seguridad, pero los cuadros de contraseña pueden ser útiles para programas que no necesiten un alto nivel de seguridad.

Campos ocultos

El otro elemento interesante de este ejemplo es el campo oculto. Si nos fijamos en el código HTML, veremeos este elemento:

```
<input type="hidden"
      name="hdnSecreto"
      value="JavaScript">
```

El código es similar a los cuadros de texto y cuadros de constraseña, pero el comportamiento de un campo oculto es diferente. Almacena el valor en el formulario, pero el valor no se muestra al usuario. En este caso, hemos usado el campo oculto para almacenar el valor de la contraseña. No obstante, es se puede hacer en JavaScript con variables y los campos ocultos no suelen ser necesarios en los programas JavaScript.

CONSEJO – Porqué existen los campos ocultos? HTML soporta campos ocultos porqué los programadores pueden trabajar con otros lenguajes. A veces, el formulario está conectado a un programa en el servidor. En este tipo de programas los campos ocultos pueden ser útiles para que el programador guarde valores que el programa puede necesitar más adelante.

Usando los cuadros de validación (*CheckBoxes*)

A veces, el usuario tiene que elegir un opción de un grupo de elementos. En otros casos, el usuario puede elegir cualquier número de las opciones disponibles. El programa “Selector de música” presenta las opciones al usuario con cuadros de validación (*checkboxes*) que le permiten hacer ambas cosas.

Figura 5.3

El programa “Selector de música”

Cómo vemos en la Figura 5.3, el programa “Selector de música” pide al usuario que seleccione su género musical favorito mediante un grupo de cuadros de validación. El usuario puede elegir cualquiera de una serie de cuadros de validación. Es importante destacar que los cuadros de

validación no son mutuamente excluyentes. El estado de cualquier cuadro de validación no tiene efecto sobre los demás. La mayoría de usuarios ya conoce este tipo de elementos y saben el tipo de comportamiento que tienen.

Código para crear cuadros de validación en HTML

Para comprender la forma de escribir código para un grupo de cuadros de validación, ayuda ver el código HTML. Los cuadros de validación (*Checkboxes*) sobre otra variante del elemento `input`. El código HTML que genera la pantalla del programa “Selector de música”es:

```
<body>
<center>
<h1>Selector de música<hr /></h1>
<form name="miForm" id="miForm" action="">
<h3>Por favor, qué música te gusta:</h3>
<table border="0">
<tr><td>
<input type="checkbox" name="chkCountry" value="country" />country
</td></tr>
<tr><td>
<input type="checkbox" name="chkRock" value="rock" />rock</td>
</tr>
<tr><td>
<input type="checkbox" name="chkRap" value="rap" />rap </td>
</tr>
<tr><td><input type="checkbox" name="chkClasica" value="clasica"
/>clásica </td>
</tr>
<tr><td><input type="checkbox" name="chkBlues" value="blues" />blues
</td>
</tr>
<tr><td><input type="button" value="Haga Click"
onclick="procesaMusica()" />
</td></tr>
</table>
<textarea name="txtSalida" rows="10" cols="35"> </textarea>
</form>
</center>
<hr />
</body>
```

Lo anterior es una página HTML relativamente sencilla. Incluye un formulario y una serie de cuadros de validación. Cada cuadro de validación tiene un nombre y un valor. Fíjese que el valor del cuadro de validación no es visible para el usuario. Si quiere que aparezca una etiqueta en el cuadro de validación, hay que escribir cómo texto HTML normal. Es muy habitual, colocar los cuadros de validación en una lista o en una tabla. El código también contiene un botón para ejecutar la función `procesaMusica()` y un aŕea de texto para mostrar el resultado generado por la función.

Comportamiento de los cuadros de validación (*CheckBoxes*)

El código para manipular los cuadros de validación es razonablemente sencillo. El objeto cuadro de valicación (*checkbox*) tiene una propiedad `value,` que se corresponde con el valor

almacenado cuando se crea el objeto en HTML. Los cuadros de validación también tienen otra propiedad fundamental `checked`. Esta propiedad contiene los valores `true` o `false`. Si el usuario selecciona el cuadro de validación (y este estaba previamente sin validar), la propiedad `checked` vale `true`. Normalmente, vamos a querer que nuestro programa realice algo sólo si el usuario ha validado el cuadro de validación. En nuestro programa, vamos a realizar una acción en función de si los cuadros de validación están validados o no. Este es el código para la función `procesaMusica()`:

```
function procesaMusica(){
// Selector de música
// muestra el uso de los checkboxes
document.miForm.txtSalida.value = "";
if (document.miForm.chkCountry.checked == true){
document.miForm.txtSalida.value += "Te gusta la música ";
document.miForm.txtSalida.value += document.miForm.chkCountry.value;
document.miForm.txtSalida.value += "\n";
} // end if
if (document.miForm.chkRock.checked == true){
document.miForm.txtSalida.value += "Te gusta la música ";
document.miForm.txtSalida.value += document.miForm.chkRock.value;
document.miForm.txtSalida.value += "\n";
} // end if
if (document.miForm.chkRap.checked == true){
document.miForm.txtSalida.value += "Te gusta la música ";
document.miForm.txtSalida.value += document.miForm.chkRap.value;
document.miForm.txtSalida.value += "\n";
} // end if
if (document.miForm.chkClasica.checked == true){
document.miForm.txtSalida.value += "Te gusta la música ";
document.miForm.txtSalida.value += document.miForm.chkClasica.value;
document.miForm.txtSalida.value += "\n";
} // end if
if (document.miForm.chkBlues.checked == true){
document.miForm.txtSalida.value += "Te gusta la música ";
document.miForm.txtSalida.value += document.miForm.chkBlues.value;
document.miForm.txtSalida.value += "\n";
} // end if
} // end function procesaMusica
```

Aunque parece que hay mucho código, la función es relativamente sencilla. Si miramos el código, vemos que es bastante repetitivo. Básicamente, se trata de una serie de sentencias if. Cada una comprueba un cuadro de validación para ver si está validado, o no. Si lo está, la función escribe el valor en el área de texto.

CONSEJO – Los cuadros de validación se utilizan para permitir al usuario activar o desactivar opciones. Por ejemplo, en una pantalla de preferencias, para permitir al usuario que active o desactive las preferencias que prefiera.

Es imporante recordar que cada cuadro de validación es básicamente una unidad independiente. Los valores de los cuadros de validación no están relacionados de ninguna manera, y cada cuadro de validación debe evaluarse de forma independiente.

Usando los botones de Radio

Los cuadros de validación tienen un elemento primo cercano los botones de radio. Los botones de radio son similares a los cuadros de validación (*checkboxes*), pero se comportan de forma diferente cuando se colocan dentro de un grupo. Se utilizan cuando queremos que el usuario elija un único elemento de un grupo. Los botones de radio deben su nombre a los antiguos autoradios que tenían pequeños botones para seleccionar las emisoras presintonizadas. Cuando se pulsaba un botón los demás saltaban automáticamente, de forma que sólo se sintonizaba una emisora cada vez.

Creando el programa “Selector de tamaño”

El programa “Selector de tamaño” ofrece un ejemplo de botones de radio HTML-JavaScript en acción. Las Figuras 5.4 y 5.5 muestran el programa. Cuando el usuario hace click en el botón OK, el programa responde cambiando el tamaño según la selección.

Figura 5.4
Figura 5.5

Generando los botones de radio en HTML

El código HTML para crear los botones de radio es muy similar al de otros componentes de pantalla que hemos visto. Pero vamos a ver las diferencias:

```
<body>
<center>
<h1>Selector de tamaño<hr></h1>
<form name="miForm" id="miForm" action="">
<h3>Qué tamaño prefiere?</h3>
<table border="0">
<tr>
<td><input type="radio" name="radTam" value="small" />Pequeño</td>
<td><input type="radio" name="radTam" value="medium" />Medio </td>
</tr>
<tr>
<td><input type="radio" name="radTam" value="large" />Grande </td>
<td> <input type="radio" name="radTam" value="jumbo" />Jumbo </td>
</tr>
<tr> <td colspan="2"> <center>
<input type="button" value="OK" onclick="procesarTam()" /> </center>
</td>
</tr>
</table>
<textarea name="txtSalida" id="txtSalida" rows="5" cols="40">
</textarea>
</form>
</center>
<hr />
</body>
```

Observad que el nombre de los botones de radio es el mismo para todos! En la próxima sección explicaremos el porqué. Por ahora, sólo añadir que el código HTML es bastante sencillo.

Tenemos un botón con una función asociada al evento `onclick` y un área de texto para mostrar la salida.

Colocando los Botones en Vectores (*Arrays*)

Lo que hace a los botones de radio especiales, es la forma en que actúan cuando se colocan en grupos. Los cuadros de validación (*Checkboxes*), cómo recordará, son bastante independientes. No echan cuenta de las selecciones de los cuadros de validación vecinos. Los botones de radio, sin embargo, trabajan en equipo. Cada botón de radio forma parte de un grupo. Cuando uno de los botones del grupo se selecciona, los demás se deseleccionan. De alguna manera, es necesario especificar a qué grupo pertenece cada botón. En HTML, especificamos que pertenecen al mismo grupo dándoles el mismo nombre. Aunque parezca un poco extraño, no es raro ver varios elementos con el mismo nombre. Si juega al golf, puede tener una tarjeta para llevar la puntuación similar a la Figura 5.6. Un campo de golf pequeño puede tener nueve hoyos. Los hoyos están numerados. Cada jugador lleva la puntuación de los golpes que necesita para introducir la bola en el hoyo. El jugador hace referencia al primer resultado cómo golpes del hoyo 1, a continuación, golpes del hoy 2, y así sucesivamente.

Figura 5.6

Los lenguajes de programación llaman a esta estructura un vector (*array*). Un array es un grupo de elementos similares, todos con el mismo nombre, pero con diferente índice numérico. JavaScript hace referencia a un array usando la notación de corchetes ([]). Al generar un grupo de botones de radio, todos con el mismo nombre, hemos creado un array. Los lenguajes de programación empiezan a contar los elementos de un array a partir del 0, así el botón pequeño será `radTam[0]`, el mediano `radTam[1]`, y así sucesivamente.

Usando variables para simplificar el código

El código para un grupo de botones der radio, puede parecer complicado cuando lo vemos por primera vez, pero no es para tanto. Antes de ver el código en si, veamos la estrategia. Vamos a utilizar un bucle for para generar los números del 0 al 3. Dentro del bucle, vamos a mirar cada botón y comprobar si ha sido seleccionado. Si lo ha sido, vamos a guardar el valor que especifica en una variable. Una vez termina el bulce, la variable tendrá almacenado el valor del botón seleccionado. Este es el código que lo realiza:

```
<script type="text/javascript">
function procesarTam(){
// Seleccionar tamaño
// Demuestra el uso de los botones de radio
var tam;
var i;
var opcionSeleccionada;
for (i = 0; i <= 3; i++){
opcionSeleccionada = document.miForm.radTam[i];
if (opcionSeleccionada.checked == true){
tam = opcionSeleccionada.value;
} // end if
} // end for loop
document.miForm.txtSalida.value = "OK, te conseguiré el tamaño ";
document.miForm.txtSalida.value += tam + ".";
```

```
} // end function
</script>
```

La función empieza de forma relativamente sencilla. La variable tam almacenará el tamaño seleccionado por el usuario. La variable i se usa para controlar el bucle. (Este es uno de los caos en que el valor de la variable de control del bucle no tiene otra utilidad, y por tanto el nombre i es un nombre adecuado). La última variable `opcionSeleccionada` es un poco más complicado. JavaScript permite almacenar todo tipo de datos en variables. En este caso, vamos a almacenar un objeto en la variable. Esto nos va a permitir trabajar de forma más cómoda, cómo vamos a ver. La línea siguiente empieza un bucle for que se va a repetir 3 veces:

```
for (i = 0; i <= 3; i++){
```

El valor de i será 0, 1, 2, y 3. Recordemos que JavaScript empieza a contar en 0. Cómo tenemos cuatro botones de radio, los numeraremos del 0 al 3.

La línea siguientes obtiene el botón de orden i:

```
opcionSeleccionada = document.miForm.radTam[i];
```

Cuando `i` vale 0, `document.miForm.radTam[0]` (el botón de radio con el valor small asociado) se copiará en la variable `opcionSeleccionada`. Cada vez que se ejecute el bucle, otro botón de radio se copiará en la variable `opcionSeleccionada`. La línea siguiente comprueba si el usuario ha seleccionado el botón de radio:

```
if (opcionSeleccionada.checked == true){
```

Si el valor es `true`, la línea siguiente copia el valor de la propiedad value del objeto en la variable

```
tam = opcionSeleccionada.value;
```

Si el botón de radio actual no está seleccionado, no sucede nada. El bucle sigue ejecutándose hasta que todos los botones se comprueban. Las dos últimas líneas generan la salida para el área de texto:

```
document.miForm.txtSalida.value = "OK, te conseguiré el tamaño ";
document. miForm.txtSalida.value += tam + ".";
```

Podemos repetir esta estrategia cada vez que queramos recuperar el valor del botón de radio seleccionado:

1. Crear las variables para controlar el bucle, para el resultado y para almacenar una referencia al botón de radio.
2. Crear un blucle for que vaya desde 0 hasta el número de botones menos 1.
3. Asignar el botón de radio actual a la variable.
4. Si el botón está seleccionado, copiar el valor en la variable resultado.
5. Repetir el bucle hasta que se han comprobado todos los botones.

Por supuesto, para cada grupo de botones de radio que usemos en la página, necesitamos repetir este proceso.

Usando el objeto select

El último elemento que vamos a ver en este Tema es la lista desplegable (*select*). También se le conoce cómo lista desplegable en muchos lenguajes de programación. El programa “Selector de Color” es un ejemplo que utiliza el objeto *select*.

Creando el programa “Selector de Color”

El programa “Selector de Color” muestra una lista de colores (ver Figura 5.7). Cuando el usuario hace click sobre la lista, esta se despliega y muestra los valores diponibles. Una vez se selecciona un color, este aparece cómo seleccionado en la lista. Cuando el usuario hace click en el botón OK, el color de fondo del documento toma el color seleccionado (ver Figura 5.8). Este tipo de selección es eficiente, porque ocupa menos espacio en pantalla que los botones de radio.

Figura 5.7
Figura 5.8

Creando la lista desplegable en HTML

Este es el código HTML que genera la lista desplegable:

```
<body>
<center><h1>Selector de Color<hr /></h1>
<h3>Por favor, elige un color</h3>
<form name="miForm" id="miForm" action="">
<select name="selColor" id="selColor">
<option value="red">Rojo</option>
<option value="orange">Naranja</option>
<option value="yellow">Amarillo</option>
<option value="green">Verde</option>
<option value="blue">Azul</option>
<option value="indigo">Añil</option>
<option value="violet">Violeta</option>
</select>
<br />
<input type="button" value="Cambiar color" onclick="cambiaColor()" />
</form>
</center>
<hr />
</body>
```

Recordará que el objeto select en HTML es básicamente un contenedor para una serie de opciones. El objeto select debe tener un nombre, y cada opción debe tener un valor. No es necesario dar nombre a cada una de las opciones. Cómo el objeto select contiene una serie de opciones, se almacenan cómo un vector (*array*).

Obteniendo la elección

El procedimiento para obtener el valor de la opción seleccionada, es más sencillo que con los botones de radio, porque el objeto select nos da información valiosa. Cuando queríamos evaluar un conjunto de botones de radio, teníamos que mirar en cada botón para ver si era el seleccionado. El objeto select tiene una propiedad predefinida que nos dice qué opción es la que se ha seleccionado. He aquí el código:

```
<script type="text/javascript">
function cambiaColor(){
// Selector de Color
// demuestra una lista desplegable, u objeto select
var laLista = document.miForm.selColor;
var laOpcion = laLista[laLista.selectedIndex];
var elColor = laOpcion.value;
document.bgColor = elColor;
} // end function cambiaColor
</script>
```

La brevedad del código puede sorprendernos. Básicamente, el código sencillamente hace una copia del objeto select. Obtiene el valor seleccionado y lo copia en la variable `elColor`. Esta línea hace una copia del objeto select:

```
var laLista = document.miForm.selColor;
```

Esta línea no es absolutamente necesaria, pero hace que el código que sigue sea más fácil de leer, porque nos evitamos repetir `document.miForm.selColor` cada vez. La línea siguiente es similar:

```
var laOpcion = laLista[laLista.selectedIndex];
```

Esta línea genera una variable para guardar el elemento seleccionado. El objeto `select` tiene una propiedad `selectedIndex`, que devuelve el índice del elemento seleccionado. La variable `laLista` tiene un vector de opciones, accedemos a la opción seleccionada mediante el valor de `laLista.selectedIndex`, esta línea copia a la variable `laOpcion` el objeto opción que ha seleccionado el usuario. A continuación es trivial recuperar el valor de la opción:

```
var elColor = laOpcion.value;
```

Una vez hemos copiado el valor del color en una variable, basta con modificar la propiedad `bgColor` del documento para cambiar el color de fondo:

```
document.bgColor = elColor;
```

Usando objetos select de selección múltiple

Además del comportamiento que acabamos de ver, un objeto select puede configurarse para acpetar múltiples selecciones por parte del usuario. El programa “Selector de Color II” que vamos a ver es una variación del “Selector de Color” que aprovecha estas capacidades.

Creando el programa “Selector de Color II”

Para este programa, vamos a cambiar el objeto select para que admita selecciones múltiples. Además, el programa añade un área de texto para la salida. El usuario puede seleccionar una serie de elementos usando combinaciones Shift+click o Ctrl+click. Cuando el usuario hace click en el botón, el elemento seleccionado más próximo a la parte superior de la lista se utilizará cómo nuevo color de fondo, y los demás valores seleccionados se copiarán al área de texto.

Modificando el HTML para soportar selecciones múltiples

Sólo son necesarias un par de modificaciones al objeto select, en HTML, para conseguir el efector deseado:

```
<select name="selColor" multiple="multiple" size="7">
```

El atributo `multiple` le dice al navegador que se permiten las selecciones múltiples. No obstante, una selección múltiple no tiene sentido en una lista desplegable. Para tener un número de opciones visibles a la vez se usa el atributo `size`.

Figura 5.9

En esta página, la lista tiene siete elementos, y hemos puesto el valor de `size` en 7 para verlos todos. El otro cambio que hemos hecho al código HTML es la creación de un área de texto para la salida.

Escribiendo el código para gestionar las selecciones múltiples

El código JavaScript para la selección empieza igual que en el programa “Selector de Color”. Se usa exactamente el mismo código para obtener el valor del color del fondo. Cuando tenemos un objeto select con múltiples selecciones, la propiedad selectedIndex devuelve el valor del índice del primer elemento seleccionado. Este es el código:

```
function cambiaColor(){
// Selector de Color
// demuestra una lista desplegable, u objeto select
var laLista = document.miForm.selColor;
var laOpcion = laLista[laLista.selectedIndex];
var elColor = laOpcion.value;
document.bgColor = elColor;
//listar todos los colores seleccionados
var listaColores = "";
var i = 0;
for (i = 0; i < laLista.length; i++){
laOpcion = laLista.options[i];
if (laOpcion.selected == true){
listaColores += laOpcion.value + "\n";
} // end if
} // end for loop
document.miForm.txtSalida.value = listaColores;
} // end function cambiaColor
```

El nuevo código aparece después del comentario `//listar todos los colores seleccionados`. El código es muy similar al que hemos utilizado para comprobar un conjunto de botones de radio. Primero, inicializamos una serie de variables para controlar el bucle for y

una cadena de caracteres para almacenar el mensaje que vamos a escribir en el área de texto. A continuación, empezamos el bucle for:

```
for (i = 0; i < laLista.length; i++){
```

El bucle se ejecuta tantas veces cómo elementos opción hay en el objeto `select`. La propiedad `length` nos dice el número de elementos en el objeto `select`. La variable i tiene los valores desde 0 hasta el número de opciones menos 1. La siguiente línea coge la opción actual y la guarda en la variable `laOpcion`:

```
laOpcion = laLista.options[i];
```

A continuación usamos una sentencia if para comprobar si la opción actual está seleccionada:

```
if (laOpcion.selected == true){
```

Si el usuario ha seleccioando esa opción, la siguiente línea copia el valor en la variable , que el programa copiará después en el área de texto:

```
listaColores += laOpcion.value + "\n";
```

Volviendo a “La Historia disparatada 2.0”

El programa “La Historia disparatada 2.0” se entiende fácilmente una vez hemos aprendido a manejar todos estos elementos, ya que no es más que una combinación de los mismos.

El código HTML para “La Historia disparatada 2.0”

El código HTML crea el escenario. No es demasiado compleo, pero es necesario planificarlo bien, o se convertirá en complejo cuando se escriba el código JavaScript. Esta es la parte de HTML : (Veremos el JavaScript a continuación)

```
<body>
<center> <h1>Historia disparatada 2.0<hr /></h1>
<form name="miForm" id="miForm" action="">
<h3>Please enter the following information</h3>
<table border = 1>
<tr><td>Un nombre de persona</td>
<td><input type="text" name="txtPersona" id="txtPersona" /></td>
</tr>
<tr> <td>descripción</td>
<td> <input type="checkbox" name="chkDiablo" value="diablo" />diablo
<input type="checkbox" name="chkGoofy" value="goofy" />goofy
<input type="checkbox" name="chkDisfunc" value="disfuncional" />
disfuncional
<input type="checkbox" name="chkChiflado" value="chiflado" />
chiflado</td>
</tr>
<tr> <td>Un ruido absurdo</td>
<td> <input type="radio" name="optSonido" value="phht!" /> phht!
<input type="radio" name="optSonido" value="boing!" /> boing!
<input type="radio" name="optSonido" value="whoosh!" /> whoosh!
```

```
<input type="radio" name="optSonido" value="splat!" /> splat! </td>
</tr>
<tr>
<td>Una parte del cuerpo</td>
<td> <select name="selCuerpo">
<option value="dientes">dientes</option>
<option value="riñones">riñones</option>
<option value="mejilla">mejilla</option>
<option value="codo">codo</option>
<option value="cerebro">cerebro</option>
</select> </td>
</tr>
<tr> <td>Un vehículo</td>
<td><input type="text" name="txtVehiculo" id="txtVehiculo" /></td>
</tr>
<tr> <td>Un animal</td>
<td><input type="text" name="txtAnimal" id="txtAnimal" /></td>
</tr>
<tr> <td colspan="2"><center>
<input type="button" value="Click para el cuento"
onclick="creaHD()" /> </td>
</tr>
</table>
<textarea name="txtSalida" rows="10" cols="40" wrap="hard">
</textarea>
</form>
</center>
<hr />
</body>
```

Todo el formulario está encajado en una tabla. Es muy difícil organizar un formulario sin la ayuda de una tabla a menos que utilicemos hojas de estilo adecuadamente. Si no se siente cómodo con las tablas, es un buen momento para repasarlas. También es importante repasar los atributos colspan y rowspan (que permiten que las celdas se expandan y ocupen más espacio del que en principio les correspondería). Estos atributos nos van a permitir flexibilizar el diseño de nuestras páginas.

El formulario contiene un cierto número de elementos de entrada de texto. Estos se han copiado directamente del programa del tema anterior. Los elementos de descripción son nuevos. Se han introducido para mostrar el uso de los cuadros de validación (*checkboxes*). El usuario puede elegir cualquier combinación de cuadros de validación. Tiene sentido agruparlos en el formulario, pero esto no es en absoluto necesario. Notad cómo cada cuadro de validación tiene un nombre diferente. El programa obtiene el valor del sonido mediante un conjunto de botones de radio. Aunque la historia puede hacer uso de más de un elemento descriptivo, sólo utiliza un sonido. Cómo pretendemos que el usuario seleccione un único sonido, este va a ser un buen candidato para un conjunto de botones de radio. Colocar todos los botones de radio en un grupo tiene sentido. Notad cómo todos los botones de radio tienen el mismo nombre, de forma que se pueda acceder a ellos mediante un vector (*array*). El usuario seleccionará la parte del cuerpo mediante una lista deplegable. Esta elección también tiene sentido, ya que sólo necesitamos un elemento. Los demás elementos, tales cómo el botón y el área de texto son iguales al programa anterior.

EN EL MUNDO REAL – La elección de los elementos de la interfaz de usuario, pueden ser algo más que por motivos estéticos. Usar el tipo correcto de elemento de entrada, simplificará la tarea del usuario que sólo tendrá que elegir las opciones con la ayuda del ratón, pero además simplificará la del programador que no tendrá que comprobar que las opciones que ha introducido el usuario son válidas.

El código JavaScript es muy similiar al del tema anterior adaptándolo a los nuevos elementos de interfaz incorporados:

```
function creaHD(){
// Historia disparatada 2.0
// crea una historia disparatada a partir de un conjunto de entradas
// crear las variables
var sonido;
var persona;
var parte;
var descripcion;
var vehiculo;
var animal;
var historia = "";
//obtner las variables cuadro de texto
persona = window.document.miForm.txtPersona.value;
vehiculo = document.miForm.txtVehiculo.value;
animal = document.miForm.txtAnimal.value;
//obtener descripción
descripcion = "";
if (document.miForm.chkDiablo.checked==true){
descripcion += document.miForm.chkDiablo.value;
descripcion += ", ";
} // end if
if (document.miForm.chkGoofy.checked==true){
descripcion += document.miForm.chkGoofy.value;
descripcion += ", ";
} // end if
if (document.miForm.chkDisfunc.checked==true){
descripcion += document.miForm.chkDisfunc.value;
descripcion += ", ";
} // end if
if (document.miForm.chkChiflado.checked==true){
descripcion += document.miForm.chkChiflado.value;
descripcion += ", ";
} // end if
//historia += "descripcion: \t" + descripcion + "\n";
//obtener sonido
for (i = 0; i <= 3; i++){
if (document.miForm.optSonido[i].checked == true){
sonido = document.miForm.optSonido[i].value;
} // end if
} // end for loop
// historia += "sonido: \t" + sonido + "\n";
// obtener parte del cuerpo
var laLista = document.miForm.selCuerpo;
var laOpcion = laLista[laLista.selectedIndex];
parte = laOpcion.value;
// historia += "parte: \t" + parte + "\n";
```

```
historia = "Un día, una persona que se llamaba " ;
historia += persona;
historia += " estaba paseando por la calle. Cuando, de pronto, ";
historia += persona;
historia += " oyó un horrible sonido: ";
historia += sonido;
historia += ". ";
historia += persona;
historia += " miró a su alrededor y vió que el ";
historia += sonido;
historia += " venía de un ";
historia += vehiculo;
historia += " que bajaba la calle a toda velocidad. ";
historia += persona;
historia += " se asustó mucho. ";
historia += persona;
historia += " vió que el ";
historia += vehiculo;
historia += " era conducido por el diablo Super-";
historia += animal;
historia += ". En su día, un ";
historia += animal;
historia += " normal, había sufrido una extraña transformación después
";
historia += "de caer en un depósito de basura nuclear. ";
historia += "Super-";
historia += animal;
historia += " siguió asustando a ";
historia += persona;
historia += " con el horrible ";
historia += sonido;
historia += " , pero,  ";
historia += persona;
historia += " no se inmutó. \"No puedes asustarme, Super-";
historia += animal;
historia += "! Yo sé cómo girar la ";
historia += parte;
historia += "!\" \nFin.";
document.miForm.txtSalida.value = historia;
} // end function creaHD
```

Este programa no tiene demasiadas novedades. Conforme un programa se hace más largo es más importante documentarlo adecuadamente. Hemos incluido comentarios que van explicando lo que hace el programa. También es buena idea usar separaciones verticales (líneas en blanco) para separar distintas partes del código.

Tabla 5.1 Objetos y propiedades usados para entradas (*input*)

Es muy fácil obtener los valores de los cuadros de texto. Para obtener el valor de la descripción necesitamos comprobar todos los cuadros de validación (*checkboxes*). Comprobamos, cada uno de los cuadros para comprobar si está validado (`checked==true`), si es así se copia el valor en la variable `descripcion`.

CONSEJO – Echemos un vistazo a esta línea:
//historia += "descripción: \t" + descripcion + "\n";
La línea está comentada, pero tiene un papel importante. Cuando estamos comprobando el programa, queremos asegurarnos que los distintos pasos se están ejecutando correctamente. No queremos concatenar todas las variables hasta asegurarnos que sus valores son los correctos. Cuando hemos acabado con la depuración, se comenta la línea, pero no se elimina por si más adelante la necesitamos si es que algo no funciona correctamente.

Hemos construido la historia de la misma forma que lo hicimos en el programa del tema anterior, con una serie de concatenaciones de cadenas de caracteres.

Resumen

En este tema, hemos aprendido a utilizar más elementos de entrada de formularios HTML y tratarlos con JavaScript. Hemos visto el comportamiento de los cuadros de validación (*checkboxes*), botones de radio (*radio buttons*) y listas desplegables. También hemos visto estrategias generales para obtener valores de estos elementos desde el código JavaScript. En el próximo tema vamos a ver nuevas maneras de generar salida al usuario.

EJERCICIOS

5.1. Escribir una versión del programa del ejercicio 3.3 en que el usuario contesta al ordenador: alto, bajo o correcto utilizando botones de radio.
5.2. Escribir un prototipo de un formulario de pedido que utilice los elementos de entrada que hemos visto. El programa mostrará las opciones seleccionadas en un área de texto.
5.3. Escribir un programa “Piedra, Papel, Tijeras”, en que el usuario seleccione su opción con botones de radio, al pulsar un botón, el ordenador seleccionará su jugada de forma aleatoria, mostrará su jugada y un mensaje diciendo quién ha ganado. (Las tijeras cortan el papel, el papel envuelve la piedra, la piedra rompe las tijeras).
5.4. Una vez vistos los vectores (*arrays*), escribir una versión del programa de la lotería primitiva que compruebe que no hay números repetidos. Una vez se tenga una combinación válida, mostrarla al usuario ordenada de menor a mayor.
5.5 Escribir una función aleatorio(min,max), que recibe un argumento min (mínimo) y un argumento max (máximo) y que devuelve un número aleatorio entre el mínimo y el máximo.
5.6 Escribir una función letraDNI(num), que recibe un argumento num (número del DNI) y que devuelve la letra correspondiente.