JavaScript de Andy Harris

TEMA 4 – Programación basada en objetos

Los principios de programación que hemos visto en los temas anteriores no han cambiado a penas desde los primeros lenguajes de programación. JavaScript también soporta alguno de los principios más modernos de la programación. En concreto, la programación orientada a objetos. Estos son algunos de los conceptos que vamos a ver:

• Las caracaterísticas básicas de los objetos.

• Cómo funcionan las propiedades, los métodos y los eventos.

• Cómo usar algunas propiedades del DOM (*Document Object Model*).

• Cómo usar algunos métodos del DOM.

• Cómo usar los eventos.

• Cómo escribir funciones para encapsular el código.

• Cómo gestionar las entradas y salidas de forma más moderna.

El proyecto: “La Historia disparatada”

El proyecto principal de este Tema es bastante divertido. Cuando empieza el programa “La Historia disparatada”, la página web contiene un formulario con un conjunto de elementos de pantalla (ver Figura 4.1). Se trata de un formulario con una serie de cuadros de texto, un botón y un amplia área de texto. Este programa no utiliza los cuadros de diálogo. El programa realiza todas las entradas y salidas directamente sobre la página web (ver Figuras 4.2 y 4.3).

Figure 4.1
Figure 4.2
Figure 4.3

Los resultados de este programa pueden ser bastante divertidos. Por supuesto, el programa se hace todavía más divertido cuando creamos nuestra propia historia y el usuario introduce más palabras. Desde el punto de vista de la programación, este juego ofrece un montón de elementos nuevos. Hasta ahora, nuestros programas han tenido poco que ver con el HTML. Este programa ya integra el HTML y JavaScript.

Objetos y HTML

Para conseguir que los programas en JavaScript interactúen con las páginas web, necesitamos saber cómo ve JavaScript las páginas web. Este concepto, puede parecer un poco extraño al

principio, pero es una idea muy importante que nos va a brindar un montón de posibiliades una vez asimilada.

El programa “Cambia el Color”

Las Figuras 4.4 y 4.5 muestran el programa “Cambia el Color”, que muestra cómo JavaScript puede interactuar con la página web que lo aloja. El programa va pasando por una serie de colores y cambia dinámicamente el color de fondo de la página. Este es el código del programa:

```
<html>
<head>
<title>Cambia el Color</title>
<script>
// Cambia el Color
// Demuestra el uso del método bgColor

document.bgColor = "red";
alert("Listo para otro color?");
document.bgColor = "orange";
alert("Listo para otro color?");
document.bgColor = "yellow";
alert("Listo para otro color?");
document.bgColor = "green";
alert("Listo para otro color?");
document.bgColor = "blue";
alert("Listo para otro color?");
document.bgColor = "indigo";
alert("Listo para otro color?");
document.bgColor = "violet";
alert("Listo para otro color");
document.bgColor = "black";
alert("Listo para otro color?");
document.bgColor = "white";
</script>
</head>
<body>
<h1>Cambia el Color<br /></h1>
<hr />
</body>
</html>
```

El programa “Cambia el Color” es bastante repetitivo. El único concepto nuevo que aparece es:

```
document.bgColor = "violet";
```

Cómo vemos, el programa copia el valor “violet” a algo llamado `document.bgColor`, que es una entidad especial utilizada para hacer referencia al color de fondo del documento.

Figura 4.4
Figura 4.5

Características de los objetos

JavaScript usa la programación basada en objetos, un paradigma de programación, es un esquema de programación en el que ciertos elementos (cómo las páginas web, por ejemplo) están definidos como objetos. En el mundo real, usamos objetos continuamente. Cualquier tipo de objeto — ya sea una página web, una grapadora o un elefante — tiene ciertos elementos que lo caracterizan. Los programadores les llaman funcionalidades del objeto: propiedades, métodos y eventos. Para explicar estos conceptos, empezaremos hablando de objetos del mundo real. A continuación explicaremos cómo utiliza los objetos JavaScript para modificar una página web.

Propiedades

Una propiedad es una característica particular de un objeto. Un objeto vaca puede tener propiedades cómo nombre, edad, peso, especie y propietario. Un objeto grapadora puede tener cómo propiedades color, fabricante, numeroGrapa, tamaño. Podemos hacer un simil entre propiedades y adjetivos en un lenguaje natural. En el fondo, contienen datos es decir se corportan como las variables. Algunas propiedades contienen valores de texto, otras números, otras valores booleanos (verdader/falso). Las propiedades le permiten al objeto almacenar información, técnicamente al conjunto de valores de las propiedades se le conoce cómo el **estado** (*state*) del objeto.

Métodos

Si una propiedad es un adjetivo, un método es cómo un verbo. Un método es una acción que puede hacer un objeto. Hemos visto algunos ejemplos de métodos, `toUpperCase()` de los objetos de tipo `String`, o el método `random()` del objeto `Math`. Los métodos del objeto vaca podrían ser comer, mugir, darLeche. Los métodos del objeto grapadora podrían ser grapar, abrirGrapadora, recargarGrapas. Los métodos permiten a los objetos realizar acciones. Técnicamente se conoce como el comportamiento (*behaviour*) del objeto.

Eventos

Los más parecido a los eventos en un lenguaje natural son mensajes del tipo “Houston, tenemos un problema”. Un evento es un mensaje que un objeto puede enviar a otros objetos. Normalmente para avisar que ha sucedido algo. El objeto vaca puede tener eventos del tipo hambre, defecar o nacimiento. La grapadora puede tener un evento atasco, que ocurrirá cuando se atasque una grapa y la grapadora deje de funcionar. Los eventos facultan a los objetos para comunicar su estado a otros objetos.

TRUCO – Si le cuesta ver la diferencia entre métodos y eventos: puede aplicar el siguiente criterio eventos son cosas que puede hacer el objeto y métodos son cosas que le suceden al objeto.

Leyendo las propiedades de un objeto

Hemos usado algunas propiedades en el programa “Cambia el Color”. Hemos cambiado de forma reiterada la propiedad `.bgcolor` del objeto `document`. El programa “Información del Documento” profundiza un poco más en el objeto `document` y nos muestra algunas de sus propiedades.

EN EL MUNDO REAL – La programación con objetos se ha convertido en uno de los paradigmas más usados actualmente. JavaScript es un ejemplo de lenguaje de programación basado en objetos. Esto significa que JavaScript tiene soporte para objetos, pero la implementación de estos objetos dista mucho de seguir el paradigma de la orientación a objetos. Lenguajes cómo C++ y Java si siguen las reglas de los lenguajes orientados a objetos. No obstante, la habilidad para describir entidades cómo objetos ha revolucionado la programación. El uso de objeto facilita la programación en equipo y facilita la depuración de los programas.

El programa “Información del documento”

El programa “Información del documento” muestra una serie de cuadros de diálogo con información sobre la página web que contiene el código JavaScript (ver Figura 4.6). JavaScript tiene un objeto llamado document. Este objeto hace referencia a la página HTML dónde reside el código JavaScript. El objeto document, cómo la mayoría de objetos, tiene propiedades, eventos y métodos.

Figura 4.6

Este es el código que muestra algunas propiedades del objeto `document`:

```
<html>
<head>
<title>Información del Documento</title>
<script>
// Información del Documento
// Muestra algunas propiedades del objeto document
alert("Color de fondo: " + document.bgColor);
alert("Dominio: " + document.domain);
alert("Último cambio: " + document.lastModified);
alert("URL: " + document.location);
alert("Última página: " + document.referrer);
alert("Título: " + document.title);
</script>
</head>
<body>
<center> <h1>Información del Documento</h1> </center>
<hr />
</body>
</html>
```

El código consiste en una serie de sentecias alert que incluyen referencias a propiedades del objeto `document`.

Propiedades del Documento

El objeto `document` es muy importante en programación JavaScript. Representa la página web en la que está alojado el código JavaScript. Las propiedades del objeto nos dan mucha información sobre la página web. La Tabla 4.1 muestra algunas de las propiedades del objeto `document`. Además de estas propiedades, si una página web contiene formularios y elementos de formularios, también estarán disponibles cómo propiedades. Una propiedad del objeto document es muy parecido a una variable, excepto que no tiene que definirla, ya lo está en el

objeto. Alguna propiedades, cómo `bgColor`, pueden leerse o escribirse. Podemos mostrar el valor `document.bgColor`, o puede asignarle un valor para cambiar el color de fondo de la página.

Propiedades del objeto document	
alinkColor	Color de los enlaces activos
bgColor	El color de fondo del documento.
classes	Las clases definidas en la declaración de estilos CSS.
domain	Nombre del dominio del servidor de la página.
fgColor	El color del texto. Para ver los cambios hay que reescribir la página.
forms array	Un array con todos los formularios de la página.
ids	Para acceder a estilos CSS.
images array	Cada una de las imágenes de la página introducidas en un array.
lastModified	La fecha de última modificación del documento.
linkColor	El color de los enlaces.
links array	Un array con cada uno de los enlaces de la página.
location	La URL del documento que se está visualizando. Es de solo lectura.
referrer	La página de la que viene el usuario.
title	El titulo de la página.
URL	Lo mismo que location, pero es aconsejable utilizar location
vlinkColor	El color de los enlaces visitados.

Cuestiones de las plataformas

Aunque JavaScript es un entorno razonablemente estándar, los fabricantes de navegadores han sido horriblemente inconsistentes en la forma de definir el objeto `document`. Netscape y Microsoft ambos tienen objetos `document`, pero son diferentes. Cada fabricante ha implementado diferentes propiedades, eventos y métodos para su objeto `document`. Incluso cuando la propiedades se llamen igual su comportamiento puede ser diferente. Esto es algo muy frustrante para un programador de JavaScript. Afortunadamente, hay un conjunto de propiedades que actúan de forma similar en ambos navegadores. En general, hablaremos de las propiedades que son comunes a ambos.

TRUCO – Compruebe siempre sus programas en los navegadores más difundidos. Algo que funciona perfectamente en uno, puede no funcionar en otro.

Métodos

El objeto document tiene también algunos métodos interesantes. Estas son las cosas que puede hacer el objeto `document`. El programa “Métodos del Documento” muestra alguno de los métodos más importantes del objeto `document`.

El programa “Métodos del Documento”

El programa “Métodos del Documento” utilizar la sentencia `prompt` para obtener la entrada del usuario, pero luego el programa incorpora el valor directamente en la página web (ver Figuras 4.7 y 4.8).

Figure 4.7
Figure 4.8

Una vez obtenida la entrada del usuario, el programa la utiliza para modificar la página web cómo si estuviese escribiendo directamente en el navegador. El programa llama al método `write()` del objeto `document`. Echemos un vistazo al código:

```
<html>
<head><title>Métodos del Documento</title>
<script>
// Métodos del Documento
// Muestra la forma de escribir en el documento
var nombreUsuario = prompt("Hola! Cómo te llamas?");
</script>
</head>
<body>
<h1>Métodos del Documento</h1><br />
<h3>
<script>
document.write("Bienvenido, ");
document.write(nombreUsuario);
document.write("!!");
document.close();
</script>
</h3>
<hr />
</body>
</html>
```

Esta página tiene dos áreas de código. La primera crea la variable `nombreUsuario` y obtiene un valor mediante la sentencia `prompt`. Dentro del body hay otro trozo de código. Esta segunda parte, llama al método `write()` del objeto `document` varias veces, por último llama al método `close()` de `document`.

CUIDADO – El código anterior crea el contenido de la página. Si mira el código fuente de la página una vez que se ha ejecutado el programa, verá que no es exactamente lo mismo. Veremos esto en un momento.

Usando document.write()

El método `document.write()` nos permite escribir texto en la página actual. Esto es una facilidad importante, porque se puede modificar el código HTML que ve el usuario. Además de texto plano, se pueden escribir etiquetas HTML que le navegador interpretará cómo si fueran de la página web original.

Usando document.close()

Los documento HTML no viajan por Internet de una pieza. Si no que lo hacen en forma de paqutes de texto, que el navegador luego interpreta como información para presentar. De hecho la página le manda al navegador una señal para indicarle que ha terminado de enviar la información a representar. Cuando se generan trozos de páginas web mediante JavaScript, se está cambiando la página conforme se carga. El navegador no va a escribir la información sólo porque se ha ejecutado el método `document.write()`. Sólo lo hace cuando recibe la señal que se ha terminado de escribir mediante el método `close()` del objeto `document`.

Eventos

Recuerde que los objetos tienen propiedades, métodos y eventos. Para mostrar cómo funcionan los eventos, vamos a ver un ejemplo clásico de evento, hacer click sobre un botón.

El programa “No haga Click”

El programa “No haga Click” le plantea un desafío importante al usuario. Presenta un formulario HTML con un botón, cómo se muestra en las Figuras 4.9 y 4.10.

EN EL MUNDO REAL – Aunque el método document. write() parece una buena forma de interactuar con el usuario, tiene serias limitaciones. El método no es realmente interactivo, ya que el documento debe cerrarse sólo una vez. Los programadores sólo usan el método para tareas muy sencillas cómo actualizar la hora actual o personalizar la página. El Tema 6, Salida dinámica, veremos cómo trabajar con marcos y ventanas. En este caso el método document. write() es más útil.

Figure 4.9
Figure 4.10

Cuando el usuario hace click sobre el botón, el programa muestra un mensaje de queja. A continuación espera a que el usuario haga click de nuevo. Este es el código:

```
<html>
<head>
<title>No haga Click</title>
</head>
<body>
<h1>No haga Click<br></h1>
<hr>
<form name="miForm" id="miForm" action="">
<input type="button"
   value="No haga Click aquí!!"
   onclick= 'alert("Pero no te he dicho que no hagas Click!!")' />
</form>
</body>
</html>
```

Este programa es un poco especial ya que no contiene etiquetas `<script>`. Pero incorpora algún código JavaScript. El otro elemento interesante de este programa es el uso de formularios HTML.

El objeto form

Muchos programas JavaScript trabajan con formularios HTML, los formularios son contenedores de controles muy interesantes cómo cajas de texto, áreas de texto y botones. Para crear un formulario, para crear un formulario HTML se usa la pareja de etiquetas `<form>` `</form>`. Dentro del formulario, se pueden añadir elementos `<input>` cómo cajas de texto, botones y botones de radio. Es una buena idea ponerle nombres a los formularios y a los elementos que contienen, de manera que se pueda hacer referencia a ellos en el código. Hemos dado nombre al objeto form cuando lo hemos definido:

```
<form name="miForm" id="miForm" action="">
```

El atributo name se utiliza para dar nombre a varios elementos HTML. En XHTML además del atributo name se ha estandarizado el atributo id para identificar los elementos.

CUIDADO – No olvide cerrar sus formularios. Si no incluye la etiqueta </form>, algunos navegadores muestran resultados extraños.

Dentro del formulario, hemos colocado un botón. Este es el código que hace el trabajo:

```
<input type="button"
   value="No haga Click aquí!!"
   onclick= 'alert("Pero no te he dicho que no hagas Click!!")' />
```

El objeto `input` es un objeto HTML muy versátil. Se usa para generar distintos elementos; en este caso hemos creado un botón. El atributo type determina qué tipo de elemento hay que generar. El elemento value contiene el literal que aparecerá sobre el botón. El atributo `onclick` es la parte que llama al código.

El evento onclick

Los botones en HTML tienen un atributo `onclick`. El valor de este atributo es muy especial. Puede contener una llamada a código JavaScript. Cuando el usuario pulsa el botón, el evento onclick se dispara y el navegador ejecuta la línea de código JavaScript. En este caso, el atributo onclick contiene código JavaScript para generar la sentencia `alert`.

TRUCO – Tome nota del uso de las comillas simples y dobles. JavaScript y HTML permiten el uso alternativo de las comillas simples y dobles para evitar confusiones.

Aunque el evento `onclick` es muy útil, puede resultar muy limitado si se restringe al uso de una línea de código JavaScript. Afortunadamente, JavaScript, cómo la mayoría de los lenguajes, tiene funciones, que nos permiten programar práctiacmente cualquier código, de forma que una única sentencia en el abributo onclick puede llamar a un número ilimitado de comandos. Vamos a verlo con el programa “No haga Click, con funciones”.

El programa “No haga Click, con funciones”

Las Figuras 4.11 y 4.12 muestran el programa “No haga click, con funciones”. Aunque se parece mucho al anterior la pequeña diferencia es fundamental.

Figura 4.11
Figura 4.12

Ahora, cuando el usuario hace click sobre el botón, el programa hace dos cosas: Muestra un cuadro de diálogo y cambia el color del fondo. Evidentemente la limitación de una línea de código impide hacer esto si no usamos algún truco. Este programa incorpora nuevos elementos estructurales:

```
<html>
<head>
<title>No hacer click, con funciones</title>
<script>
function protestar(){
// Función no hacer click
// demuestra el uso de las funciones
document.bgColor = "red";
alert (""Pero no te he dicho que no hagas Click!!"");
document.bgColor = "white";
} // end function
</script>
</head>
<body>
<h1>No hacer click, con funciones<br /></h1>
<hr>
<form name="miForm" id="miForm" action="">
<input type="button"
   value="No haga Click aquí!!"
   onclick="protestar()" />
</form>
</body>
</html>
```

La cabecera un trozo de código que el evento onclick del botón parece invocar. El botón sigue teniendo una línea de código JavaScript en el atributo onclick, pero esta línea no se parece a ningún código JavaScript que hayamos visto.

El propósito de las funciones

Si alguién os pregunta qué habéis hecho esta mañana, probablemente contestaréis algo parecido a: “Me he levantado, me he dado una ducha, me he vestido, he desayunado y he venido al Instituto.” Probablemente no va a describir todos los detalles del desayuno (“He ido a la cocina, he echado un vaso de leche, le he echado una cucharada de café, otra de azúcar, ...”) porque la frase “he desayunado” engloba todos estos detalles. Las **funciones** trabajan exactamente de la misma manera. **Juntamos una serie de comandos y le damos un nombre**. Por tanto, cada vez que queramos que el ordenador repita esa serie de pasos de nuevo, usaremos el nombre del nuevo comando que acabamos de definir. Esto es exactamente lo que hace la función definida en el programa “No hacer click, con funciones”.

Creando una función

Echemos un vistazo de nuevo al trozo de código de la cabecera del HTML:

```
function protestar(){
// Función no hacer click
// demuestra el uso de las funciones
document.bgColor = "red";
alert (""Pero no te he dicho que no hagas Click!!"");
document.bgColor = "white";
} // end function
```

Este código es una **definición de una función**. El término protestar() es el nombre que le hemos dado a la función. Los **paréntesis** son parte de la sintaxis de definición de la función. En este caso, están vacíos lo que significa que no se le envían valores especiales (llamados **parámetros**) a la función. (Veremos funciones con parámetros en otro tema). El código que define las tareas que realiza la función va entre las llaves de apertura y de cierre. La mayoría de programas JavaScript que haremos consistirán de una o varias funciones definidas en la cabecera del HTML que serán invocadas desde los eventos de los elementos de formulario dentro del body del documento HTML. Definir la función en el área de cabecera nos garantiza que estará cargada en el navegador cuando algún evento la necesite.

Llamando a una función desde un evento

Una vez definida la función, se convierte en un nuevo comando del lenguaje JavaScript que podremos usar cuando lo necesitemos. He aquí una forma de utilizarla:

```
<input type="button"
   value="No haga Click aquí!!"
        onclick="protestar()" />
```

Cuando el usuario hace click en el botón, el programa llama automáticamente a la función `protestar()`. El control del programa pasa a la función y se ejecutan las instrucciones que están definidas en la función.

Entradas y salidas controladas por eventos

El uso de los formularios de HTML y las capacidades de gestión de eventos que ofrecen pueden combinarse para conseguir entradas y salidas más elegantes que los cuadros de diálogo. Los cuadros de diálogo nos han servido bien, y nos seguirán sirviendo para depurar programas, pero interrumpen el normal desarrollo de un programa. Necesitamos integrar las entradas y salidas en la página web de forma más natural.

Creando el programa “Capturando el nombre”

Las figuras 4.13 y 4.14 muestran el programa “Capturando el nombre”, que usa cuadros de texto para gestionar las entradas y salidas. Este programa no utiliza los cuadros de diálogo. El programa “Capturando el nombre” realiza las entradas y salidas manipulando elementos del formulario. Este programa introduce un nuevo elemento los cuadros de texto HTML. Estos elementos van a sustituir los cuadros de diálogo

Figura 4.13
Figura 4.14

He aquí el código completo:

```
<html>
<head><title>Capturando el nombre</title>
<script>
function copiarNombre(){
// Capturando el nombre
// demuestra I/O basadas en formularios
var nombreUsuario = document.miForm.txtNombre.value;
var saludo = "Hola, qué tal, ";
saludo += nombreUsuario;
saludo += "!!";
document.miForm.txtSaludo.value = saludo;
} // end function copiarNombre
</script>
</head>
<body>
<h1>Capturando el nombre<br /></h1>
<form name="miForm" id="miForm" action="">
<table border="1">
<tr>
<td>Por favor introduzca su nombre:</td>
<td><input type="text" name="txtNombre" id="txtNombre" /></td>
</tr>
<tr> <td colspan="2"><center>
<input type="button"
   value="Haga Click aqui" onclick="copiarNombre()" />
</center></td>
</tr>
<tr> <td colspan = 2><center>
<input type="text" name="txtSaludo" id="txtSaludo" /> </center></td>
</tr>
</table>
</form>
</center>
</body>
</html>
```

El código consiste en una funcion JavaScript escrita en la cabecera, y una serie de elementos de formulario de HTML que juegan un papel importante en el proceso. La forma más fácil de comprender lo que está pasando es mirar el código en dos partes: la parte HTML y la función `copiarNombre()`.

Escribiendo el código HTML

En primer lugar, echemos un vistazo al código HTML. Cuando estemos estudiando código JavaScript escrito por otros programadores, es una buena idea empezar mirando el código HTML, porque el HTML construye el escenario. Nuestro código HTML consiste en un formulario con tres elementos. El formulario se llama miForm y contiene dos cuadros de texto y un botón.

TRUCO – Las tablas y los formularios suelen ser una buena combinación. A menudo, escondo el marco (<table border="0">), pero me permiten organizar la pantalla cómo quiero. Conforme vamos ganando experiencia, tenemos que

sustituir el uso de las tablas por las hojas de estilo para posicionar los elementos de un formulario y desarrollar un estilo personal para nuestras páginas.

Echemos un vistazo al código que genera el primer cuadro de texto:

```
<input type="text" name="txtNombre" id="txtNombre" />
```

Esta es otra version de la etiqueta `input`, con el atributo `type` inicializado con `"text"`, con lo que obtenemos un cuadro de texto. El navegador coloca un cuadro de texto en el formulario. No se especifican más atributos, luego aparece vacío para que el usuario pueda teclear texto en él. El atributo `name` se establece en `"txtNombre"`. Veremos en un instante que es fundamental darle un nombre al cuadro de texto.

TRUCO – Muchos programadores usan prefijos cómo txt para decir que se trata de cuadros de texto, cmd para decir que se trata de botones de comando. Esta convención hace más fácil gestionar su código cuando tenemos un gran número de objetos definidos. Cuando vemos txtNombre ya podemos saber que nos estamos refiriendo a un cuadro de texto (Sobre convenciones y notaciones).

El código que genera el otro cuadro de texto es muy similar, a este le hemos llamado txtSaludo. No tiene atributo value especificado, y por tanto se muestra en blanco. Este es código para el botón:

```
<input type="button"
   value="Haga Click aqui" onclick="copiarNombre()" />
```

Este código genera un botón con la etiqueta “Haga Click aqui”. Cuando el usuario hace click, la función copiarNombre() se ejecuta. La siguiente sección explica lo que hace la función copiarNombre(). Es un código un poco diferente al que hemos visto hasta ahora:

```
function copiarNombre(){
// Capturando el nombre
// demuestra I/O basadas en formularios
var nombreUsuario = document.miForm.txtNombre.value;
var saludo = "Hola, qué tal, ";
saludo += nombreUsuario;
saludo += "!!";
document.miForm.txtSaludo.value = saludo;
} // end function copiarNombre
```

El código empieza creando las variables saludo y nombreUsuario. La línea que obtiene el dato del nombre del usuario asusta un poco pero realmente no es tan compleja. `document.miForm.txtNombre.value` no es más que la forma de decir el contenido del cuadro de texto llamado txtNombre, del formulario miForm, que está en el objeto document. Este cuadro de texto tiene una propiedad llamada value que representa el texto que se ha tecleado en el cuadro de texto (ver Figura 4.15). Cuando construimos un formulario HTML, estamos creando una estructura compleja. El formulario es un nodo, y todos los objetos que forman parte del formulario son nodos a su vez. Cada uno de estos objetos tiene sus propias propiedades, cómo por ejemplo value. Observemos la línea siguiente:

```
var nombreUsuario = document.miForm.txtNombre.value;
```

El resultado de esta línea es copiar el contenido de la propiedad value del cuadro de texto `txtNombre` en la variable `nombreUsuario`. `document.miForm.txtNombre.value` se convierte asi en una especie de variable a la que le podemos asignar datos y leer su contenido, cómo a cualquier otra variable. Una vez hemos entendido esto, la línea siguiente tiene pleno sentido:

```
document.miForm.txtSaludo.value = saludo;
```

Esta línea copia el contenido de la variable `saludo` en la propiedad `value` del cuadro de texto `txtSaludo` haciéndolo visible al usuario.

Figura 4.15

Volviendo al programa “La Historia disparatada”

El programa “La Historia disparatada” se hace mucho más fácil, una vez que sabemos hacer entradas y salidas mediante elementos de formulario. Cómo el código se hace bastante largo vamos a verlo por secciones.

Planificando el juego

El primer paso es construir la historia. Hay muchas formas de hacerlo. Si tiene facilidad para escribir historias, escriba una y guarde las palabras que quiere utilizar para rellenar huecos. Escriba la historia y la lista de palabras para rellenar antes de empezar a programar. Esto nos permitirá saber lo que tenemos que hacer en cada parte del programa. Haga un boceto del formulario en papel. Escriba los nombres de los objetos que va a utilizar (cuadros de texto, áreas de texto, botones). Este trabajo previo facilitará mucho la escritura del código.

Construyendo el HTML

Cree primero el HTML. Siga el diseño del boceto para colocar los distintos elementos en su sitio. No olvide dar nombres al formulario, a los cuadros de texto, al área de texto. El botón no necesita nombre, porqué no vamos a acceder a sus propiedades, pero se le puede dar uno cómo buena costumbre. Este es el código HTML: (Todavía no contiene código JavaScript. Lo veremos en la siguiente sección).

```
<body>
<h1>Historia disparatada<hr /></h1>
<form name="miForm" if="miForm" action="">
<h3>Por favor, introduzca la información siguiente</h3>
<table border = 1>
<tr>
<td>Un nombre de persona</td>
<td><input type="text" name="txtPersona" id="txtPersona" /></td>
</tr>
<tr> <td>Un sonido estúpido</td>
<td><input type="text" name="txtSonido" id="txtSonido" /></td>
</tr>
<tr> <td>Una parte del cuerpo</td>
<td><input type="text" name="txtParte" id="txtParte" /></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="Crear el cuento" onclick ="crearHD()" />
</td>
</tr>
</table>
<textarea name="txtHistoria" id="txtHistoria" rows="10" cols="40"
wrap="true"> </textarea>
</form>
</center>
<hr />
</body>
</html>
```

Fijarse especialmente en el uso de un área de texto. La salida será una historia bastante larga, y necesitamos un lugar para colocarla. Un área de texto es perfecta para esto. Activando el atributo `wrap` nos aseguramos que toda la historia se va a ver, sin necesidad de incorporar un montón de retornos de carro. Notar también que hemos incluido la llamada a la función crearLD() en el código del botón. Es bueno probar el código HTML antes de empezar a trabajar con el código JavaScript.

Obteniendo las entradas

Una vez probado el HTML, podemos centrarnos en la función. Este es el inicio de la definición:

```
function crearHD(){
// Historia disparatada
// crea una historia sin sentido a partir de trozos de entradas
// obtiene los valores del formulario
var persona = window.document.miForm.txtPersona.value;
var sonido = document.miForm.txtSonido.value;
var parte = document.miForm.txtParte.value;
var vehiculo = document.miForm.txtVehiculo.value;
var animal = document.miForm.txtAnimal.value;
var historia = "";
```

Esta parte del programa solamente crea las variables y obtiene los valores de los elementos del formulario. No es coincidencia que persona y txtPersona se parezcan tanto. En un programa repetitivo cómo este, es bueno generar algún tipo de convención a lo hora de dar nombres para que sea fácil hacer un seguimiento de los elementos del formulario y de las variables. La variable historia almacenará el contenido completo de la historia antes de mostrarla en el área de texto.

Construyendo un String muy largo

Construir la historia se convierte en una de las tareas más sencillas:

```
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 ";
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.txtHistoria.value = historia;
} // end function crearHD
```

Hemos alternado texto y variables. Hubiese sido muy tentador crear concatenaciones muy largas del tipo:

```
historia += "Super-" + animal + " siguió asustando a ";
```

El problema de estas líneas es que son muy difíciles de depurar.

TRAMPA – La razón por la que estas líneas son tan difíciles de depurar es que no se sabe muy bien a qué altura de la línea se produce el error así como distinguir lo que son literales de lo que son variables.

Otra cosa a destacar es el uso de `"\"` en la última línea de la historia. La barra invertida indica que las comillas que vienen a continuación no deben interpretarse cómo el final de la cadena de caracteres sino cómo el carácter comillas.
Una vez construida la historia, es sencillo copiar el valor de la variable al área de texto `txtHistoria` para mostrar la salida.

El objeto Date

Sobre el objeto `Date` recae todo el trabajo con fechas en Javascript, como obtener una fecha, el día y la hora actuales y otras cosas. Para trabajar con fechas necesitamos instanciar un objeto de la clase `Date` y con él ya podemos realizar las operaciones que necesitemos.
Un objeto de la clase `Date` se puede crear de dos maneras distintas. Por un lado podemos crear el objeto con el día y hora actuales y por otro podemos crearlo con un día y hora concretos. Esto depende de los parámetros que pasemos al construir los objetos.
Para crear un objeto fecha con el día y hora actuales colocamos los paréntesis vacíos al llamar al constructor de la clase `Date`.

```
miFecha = new Date();
```

Para crear un objeto fecha con un día y hora distintos de los actuales tenemos que indicar entre paréntesis los valores con que inicializar el objeto. Hay varias maneras de expresar un día y hora válidas, por eso podemos construir una fecha guiándonos por varios esquemas. Estos son dos de ellos, suficientes para crear todo tipo de fechas y horas.

```
miFecha = new Date(año,mes,dia,hora,minutos,segundos);
miFecha = new Date(año,mes,dia);
```

Los valores que debe recibir el constructor son siempre numéricos. Un detalle, el mes comienza por 0, es decir, enero es el mes 0. Si no indicamos la hora, el objeto fecha se crea con hora 00:00:00.

Métodos	
getDate()	Devuelve el día del mes.
getDay()	Devuelve el día de la semana (atención empieza en 0 para Domingo).
getHours()	Devuelve la hora.
getMinutes()	Devuelve los minutos.
getMonth()	Devuelve el mes (atención al mes que empieza por 0).
getSeconds()	Devuelve los segundos.
getTime()	Devuelve los milisegundos transcurridos entre el día 1 de enero de 1970 y la fecha correspondiente al objeto al que se le pasa el mensaje.
getFullYear()	Retorna el año con todos los dígitos. Usar este método para estar seguros de que funcionará todo bien en fechas posteriores al año 2000.
setDate()	Actualiza el día del mes.
setHours()	Actualiza la hora.

setMinutes()	Cambia los minutos.
setMonth()	Cambia el mes (atención al mes que empieza por 0).
setSeconds()	Cambia los segundos.
setFullYear()	Cambia el año de la fecha al número que recibe por parámetro.

Resumen

Este tema trata de conceptos mayores. Hemos visto los objetos y sus características principales. Hemos empezado a explorar las propiedades, los métodos y los eventos. Hemos echado un vistazo al DOM (*Document Object Model*). Hemos visto cómo la programación guiada por eventos liga el código JavaScript a la página web. Hemos visto cómo copiar valores a y desde los cuadros de texto y las áreas de texto. En el próximo tema, veremos otros elementos de formulario cómo botones de radio, cajas de validación (*checkboxes*) y listas desplegables (*dropdown lists*).

EJERCICIOS – Tema 4

4.1. Escribir un programa en el que el usuario escriba un color en un cuadro de texto, haga click en un botón y cambie en color del fondo.

4.2. Escribir un programa que obtiene el nombre en un cuadro de texto, el apellido en otro y muestra “apellido, nombre” en un tercero.

4.3. Escribir un programa que muestre la fecha actual al principio de la página, con el formato de fecha español (p.e. lunes 8 de octubre de 2012).

SUPER-EJERCICIO – Tema 4 (0.5 puntos)

Escribir un programa con dos cuadros de texto (mes y año) en los que el usuario escribirá los valores del mes y año que quiere representar.

1. Comprobar que la entrada del cuadro de texto mes es correcta (El usuario ha escrito un valor de mes válido). Si no, mostrar un mensaje de que no es correcta y volver a preguntar.
2. Comprobar que la entrada del año es correcta (año > 1900 y año < 2100). Si no, mostrar un mensaje con el error y volver a preguntar.
3. Una vez hemos obtenido valores correctos el programa deberá mostrar en la página web el calendario de ese mes con un formato similar al siguiente:

OCTUBRE 2012

Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				