CURSO DE JAVASCRIPT

POR
IVÁN RODRÍGUEZ RUIZ

ÍNDICE

1 INTRODUCCIÓN

1.1 ¿Qué es JavaScript?

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas.

Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

A pesar de su nombre, JavaScript no guarda ninguna relación directa con el lenguaje de programación Java. Legalmente, JavaScript es una marca registrada de la empresa Sun Microsystems, como se puede ver en http://www.sun.com/suntrademarks/.
Se define:

- **Como orientado a objetos:** paradigma de programación que usa los objetos en sus interacciones, para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

- **Basado en prototipos:** es un estilo de programación orientada a objetos en el cual, las "clases" no están presentes, y la re-utilización de procesos (conocida como herencia en lenguajes basados en clases) se obtiene a través de la clonación de objetos ya existentes, que sirven de prototipos, extendiendo sus funcionalidades. Este modelo es conocido como orientado a prototipos, o programación basada en instancias

- **Imperativo**: En la programación imperativa se describe paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado del programa y hallar la solución, es decir, un algoritmo en el que se describen los pasos necesarios para solucionar el problema.

- **Débilmente tipado**: Los lenguajes de programación no tipados o débilmente tipados no controlan los tipos de las variables que declaran, de este modo, es posible usar variables de cualquier tipo en un mismo escenario. Por ejemplo, una función puede recibir como parámetro un valor entero, cadena de caracteres, flotante...

- **Dinámico**: conjunto de técnicas que permiten crear sitios web interactivos utilizando una combinación de lenguaje HTML estático, un lenguaje interpretado en el lado del cliente (como JavaScript), el lenguaje de hojas de estilo en cascada (CSS) y la jerarquía de objetos de un Document Object Model (DOM).

Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, en bases de datos locales al navegador...aunque existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

JavaScript se diseñó con una sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo Java y JavaScript no están relacionados y tienen semánticas y propósitos diferentes.
Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. JavaScript se interpreta en el agente de usuario, al mismo tiempo que las sentencias van descargándose junto con el código HTML.

Mas información: http://es.wikipedia.org/wiki/JavaScript
http://es.wikipedia.org/wiki/HTML_din%C3%A1mico
http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos
http://es.wikipedia.org/wiki/Tipado_fuerte
http://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa
http://es.wikipedia.org/wiki/Programaci%C3%B3n_basada_en_prototipos

1.2 Breve historia

A principios de los años 90, la mayoría de usuarios que se conectaban a Internet lo hacían con módems a una velocidad máxima de 28.8 kbps. En esa época, empezaban a desarrollarse las primeras aplicaciones web y las páginas web comenzaban a incluir formularios complejos.

Con unas aplicaciones web cada vez más complejas y una velocidad de navegación tan lenta, surgió la necesidad de un lenguaje de programación que se ejecutara en el navegador del usuario. De esta forma, si el usuario no rellenaba correctamente un formulario, no se le hacía esperar mucho tiempo hasta que el servidor volviera a mostrar el formulario indicando los errores existentes.

Brendan Eich, un programador que trabajaba en Netscape, pensó que podría solucionar este problema adaptando otras tecnologías existentes (como ScriptEase) al navegador Netscape Navigator 2.0, que iba a lanzarse en 1995. Inicialmente, Eich denominó a su lenguaje LiveScript.

Posteriormente, Netscape firmó una alianza con Sun Microsystems para el desarrollo del nuevo lenguaje de programación. Además, justo antes del lanzamiento Netscape decidió cambiar el nombre por el de JavaScript. La razón del cambio de nombre fue exclusivamente por marketing, ya que Java era la palabra de moda en el mundo informático y de Internet de la época.

La primera versión de JavaScript fue un completo éxito y Netscape Navigator 3.0 ya incorporaba la siguiente versión del lenguaje, la versión 1.1. Al mismo tiempo, Microsoft lanzó JScript con su navegador Internet Explorer 3. JScript era una copia de JavaScript al que le cambiaron el nombre para evitar problemas legales.

Para evitar una guerra de tecnologías, Netscape decidió que lo mejor sería estandarizar el lenguaje JavaScript. De esta forma, en 1997 se envió la especificación JavaScript 1.1 al organismo ECMA European Computer Manufacturers Association).

ECMA creó el comité TC39 con el objetivo de "estandarizar de un lenguaje de script multiplataforma e independiente de cualquier empresa". El primer estándar que creó el comité TC39 se denominó ECMA-262, en el que se definió por primera vez el lenguaje ECMAScript.

Por este motivo, algunos programadores prefieren la denominación ECMAScript para referirse al lenguaje JavaScript. De hecho, JavaScript no es más que la implementación que realizó la empresa Netscape del estándar ECMAScript. La organización internacional para la estandarización (ISO) adoptó el estándar ECMA-262 a través de su comisión IEC, dando lugar al estándar ISO/IEC-16262.

1.3 Especificaciones oficiales

ECMA ha publicado varios estándares relacionados con ECMAScript. En Junio de 1997 se publicó la primera edición del estándar ECMA-262. Un año después, en Junio de 1998 se realizaron pequeñas modificaciones para adaptarlo al estandar ISO/IEC-16262 y se creó la segunda edición.

La tercera edición del estándar ECMA-262 (publicada en Diciembre de 1999) es la versión que utilizan los navegadores actuales y se puede consultar gratuitamente en http://www.ecma-international.org/publications/standards/Ecma-262.htm

Actualmente se encuentra en desarrollo la cuarta versión de ECMA-262, que podría incluir novedades como paquetes, namespaces, definición explícita de clases, etc.

ECMA también ha definido varios estándares relacionados con ECMAScript, como el estándar ECMA-357, que define una extensión conocida como E4X y que permite la integración de JavaScript y XML.

1.4 Cómo incluir JavaScript en documentos XHTML

La integración de JavaScript y XHTML es muy flexible, ya que existen al menos tres formas para incluir código JavaScript en las páginas web.

1.4.1 Incluir JavaScript en el mismo documento XHTML

El código JavaScript se encierra entre etiquetas <script> y se incluye en cualquier parte del documento. Aunque es correcto incluir cualquier bloque de código en cualquier zona de la página, se recomienda definir el código JavaScript dentro de la cabecera del documento (dentro de la etiqueta <head>).
Veamos un ejemplo completo: **Ej01.04a-JavaScriptHead.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Ejemplo de código JavaScript en el propio documento</title>
   <script type="text/javascript">
      window.alert("Un mensaje de prueba");
   </script>
</head>

<body>
   <p>Un párrafo de texto.</p>
</body>
</html>
```

Para que la página XHTML resultante sea válida, es necesario añadir el atributo type a la etiqueta <script>. Los valores que se incluyen en el atributo type están estandarizados y para el caso de JavaScript, el valor correcto es text/javascript.

Este método se emplea cuando se define un bloque pequeño de código o cuando se quieren incluir instrucciones específicas en un determinado documento HTML que completen las instrucciones y funciones que se incluyen por defecto en todos los documentos del sitio web.

El principal inconveniente es que si se quiere hacer una modificación en el bloque de código, es necesario modificar todas las páginas que incluyen ese mismo bloque de código JavaScript.

1.4.2 Definir JavaScript en un archivo externo

Esta es **la opción mas recomendable** de las 3 que vamos a ver.
Las instrucciones JavaScript se pueden incluir en un archivo externo de tipo JavaScript que los documentos XHTML enlazan mediante la etiqueta <script>. Se pueden crear todos los archivos JavaScript que sean necesarios y cada documento XHTML puede enlazar tantos archivos JavaScript como necesite.
Un ejemplo:

JavaScriptExterno.js

```
window.alert("Un mensaje de prueba");
```

Ej01.04b-JavaScriptExterno.html

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Ejemplo de código JavaScript Externo</title>
   <script type="text/javascript" src="/js/JavaScriptExterno.js"></script>
</head>

<body>
   <p>Un párrafo de texto.</p>
</body>
</html>
```

Además del atributo type, este método requiere definir el atributo src, que es el que indica la URL correspondiente al archivo JavaScript que se quiere enlazar. Cada etiqueta <script> solamente puede enlazar un único archivo, pero en una misma página se pueden incluir tantas etiquetas <script> como sean necesarias.

Los archivos de tipo JavaScript son documentos normales de texto con la extensión .js, que se pueden crear con cualquier editor de texto como Notepad, Wordpad, EmEditor, UltraEdit, Vi, etc.

La principal ventaja de enlazar un archivo JavaScript externo es que se simplifica el código XHTML de la página, que se puede reutilizar el mismo código JavaScript en todas las páginas del sitio web y que cualquier modificación realizada en el archivo JavaScript se ve reflejada inmediatamente en todas las páginas XHTML que lo enlazan.

1.4.3 Incluir JavaScript en los elementos XHTML

Este último método es el menos utilizado, ya que consiste en incluir trozos de JavaScript dentro del código XHTML de la página:

Ej01.04c-JavaScriptInterno.html

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Ejemplo de código JavaScript Body</title>
</head>

<body>
   <p onclick="alert('Un mensaje de prueba');">Un párrafo de texto.</p>
</body>
</html>
```

El mayor inconveniente de este método es que ensucia innecesariamente el código XHTML de la página y complica el mantenimiento del código JavaScript. En general, este método sólo se utiliza para definir algunos eventos y en algunos otros casos especiales, como se verá más adelante.

1.5 Etiqueta noscript

Algunos navegadores no disponen de soporte completo de JavaScript, otros navegadores permiten bloquearlo parcialmente e incluso algunos usuarios bloquean completamente el uso de JavaScript porque creen que así navegan de forma más segura.

En estos casos, es habitual que si la página web requiere JavaScript para su correcto funcionamiento, se incluya un mensaje de aviso al usuario indicándole que debería activar JavaScript para disfrutar completamente de la página.

El lenguaje HTML define la etiqueta <noscript> para mostrar un mensaje al usuario cuando su navegador no puede ejecutar JavaScript.

El siguiente código muestra un ejemplo del uso de la etiqueta <noscript>:

Ej01.05a-NoScript.html

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Ejemplo de código NoScript</title>
</head>

<body>
      <noscript>
         <h1> ATENCIÓN!!</h1>
         <p> Su navegador tiene JavaScript DESHABILITADO!!!</p>
      </noscript>
      <p> Esto es el resto de la página...</p>
</body>
</html>
```

La etiqueta <noscript> se debe incluir en el interior de la etiqueta <body> (normalmente se incluye al principio de <body>). El mensaje que muestra <noscript> puede incluir cualquier elemento o etiqueta XHTML.

1.6 Glosario básico

Script: cada uno de los programas, aplicaciones o trozos de código creados con el lenguaje de programación JavaScript. Unas pocas líneas de código forman un script y un archivo de miles de líneas de JavaScript también se considera un script. A veces se traduce al español directamente como "guión", aunque script es una palabra más adecuada y comúnmente aceptada.

Sentencia: cada una de las instrucciones que forman un script.

Palabras reservadas: son las palabras (en inglés) que se utilizan para construir las sentencias de JavaScript y que por tanto no pueden ser utilizadas libremente.
Las palabras actualmente reservadas por JavaScript son:
break, case, catch, continue, default, delete, do, else, finally, for, function, if, in, instanceof, new, return, switch, this, throw, try, typeof, var, void, while, with.

1.7 Sintaxis

La sintaxis de un lenguaje de programación se define como el conjunto de reglas que deben seguirse al escribir el código fuente de los programas para considerarse como correctos para ese lenguaje de programación.

La sintaxis de JavaScript es muy similar a la de otros lenguajes de programación como Java y C. Las normas básicas que definen la sintaxis de JavaScript son las siguientes:

No se tienen en cuenta los espacios en blanco y las nuevas líneas: como sucede con XHTML, el intérprete de JavaScript ignora cualquier espacio en blanco sobrante, por lo que el código se puede ordenar de forma adecuada para entenderlo mejor (tabulando las líneas, añadiendo espacios, creando nuevas líneas, etc.)

Se distinguen las mayúsculas y minúsculas: al igual que sucede con la sintaxis de las etiquetas y elementos XHTML. Sin embargo, si en una página XHTML se utilizan indistintamente mayúsculas y minúsculas, la página se visualiza correctamente, siendo el único problema la no validación de la página. En cambio, si en JavaScript se intercambian mayúsculas y minúsculas el script no funciona.

No se define el tipo de las variables: al crear una variable, no es necesario indicar el tipo de dato que almacenará. De esta forma, una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script.

No es necesario terminar cada sentencia con el carácter de punto y coma (;): en la mayoría de lenguajes de programación, es obligatorio terminar cada sentencia con el carácter ;. Aunque JavaScript no obliga a hacerlo, es conveniente seguir la tradición de terminar cada sentencia con el carácter del punto y coma (;).

Se pueden incluir comentarios: los comentarios se utilizan para añadir información en el código fuente del programa. Aunque el contenido de los comentarios no se visualiza por pantalla, si que se envía al navegador del usuario junto con el resto del script, por lo que es necesario extremar las precauciones sobre la información incluida en los comentarios.

JavaScript define dos tipos de comentarios: los de una sola línea y los que ocupan varias líneas.

Ejemplo de comentario de una sola línea:

```
// a continuación se muestra un mensaje
alert("mensaje de prueba");
```

Los comentarios de una sola línea se definen añadiendo dos barras oblicuas (//) al principio de la línea.

Ejemplo de comentario de varias líneas:

```
/* Los comentarios de varias líneas son muy útiles
 * cuando se necesita incluir bastante información
 * en los comentarios */
      alert("mensaje de prueba");
```

Los comentarios multilínea se definen encerrando el texto del comentario entre los símbolos /* y */.

1.8 Posibilidades y limitaciones

Desde su aparición, JavaScript siempre fue utilizado de forma masiva por la mayoría de sitios de Internet. La aparición de Flash disminuyó su popularidad, ya que Flash permitía realizar algunas acciones imposibles de llevar a cabo mediante JavaScript.

Sin embargo, la aparición de las aplicaciones AJAX programadas con JavaScript le ha devuelto una popularidad sin igual dentro de los lenguajes de programación web.

En cuanto a las limitaciones, JavaScript fue diseñado de forma que se ejecutara en un entorno muy limitado que permitiera a los usuarios confiar en la ejecución de los scripts.

De esta forma, los scripts de JavaScript no pueden comunicarse con recursos que no pertenezcan al mismo dominio desde el que se descargó el script. Los scripts tampoco pueden cerrar ventanas que no hayan abierto esos mismos scripts. Las ventanas que se crean no pueden ser demasiado pequeñas ni demasiado grandes ni colocarse fuera de la vista del usuario (aunque los detalles concretos dependen de cada navegador).

Además, los scripts no pueden acceder a los archivos del ordenador del usuario (ni en modo lectura ni en modo escritura) y tampoco pueden leer o modificar las preferencias del navegador.

Por último, si la ejecución de un script dura demasiado tiempo (por ejemplo por un error de programación) el navegador informa al usuario de que un script está consumiendo demasiados recursos y le da la posibilidad de detener su ejecución.

A pesar de todo, existen alternativas para poder saltarse algunas de las limitaciones anteriores. La alternativa más utilizada y conocida consiste en firmar digitalmente el script y solicitar al usuario el permiso para realizar esas acciones.

1.9 JavaScript y navegadores

Los navegadores más modernos disponibles actualmente incluyen soporte de JavaScript hasta la versión correspondiente a la tercera edición del estándar ECMA-262.

La mayor diferencia reside en el dialecto utilizado, ya que mientras Internet Explorer utiliza JScript, el resto de navegadores (Firefox, Chrome, Opera, Safari, Konqueror) utilizan JavaScript.

1.10 JavaScript en otros entornos

La inigualable popularidad de JavaScript como lenguaje de programación de aplicaciones web se ha extendido a otras aplicaciones y otros entornos no relacionados con la web.

Herramientas como Adobe Acrobat permiten incluir código JavaScript en archivos PDF. Otras herramientas de Adobe como Flash y Flex utilizan ActionScript, un dialecto del mismo estándar de JavaScript.

Photoshop permite realizar pequeños scripts mediante JavaScript y la versión 6 de Java incluye un nuevo paquete (denominado javax.script) que permite integrar ambos lenguajes.

Por último, aplicaciones como Yahoo Widgets (cuyo proyecto está parado) y el Dashboard de Apple utilizan JavaScript para programar sus widgets.

2 EL PRIMER SCRIPT

A continuación, se muestra un script sencillo pero completo:

Ej02.01a-Confirm.html

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Ejemplo de código JavaScript en el propio documento</title>
   <script type="text/javascript">
      window.confirm("¿Te ha gustado este script?");
   </script>
</head>

<body>
   <p>Un párrafo de texto.</p>
</body>
</html>
```

En este ejemplo, el script se incluye como un bloque de código dentro de una página XHTML. Por tanto, en primer lugar se debe crear una página XHTML correcta que incluya la declaración del DOCTYPE, el atributo xmlns, las secciones <head> y <body>, la etiqueta <title>, etc.

Aunque el código del script se puede incluir en cualquier parte de la página, se recomienda incluirlo en la cabecera del documento, es decir, dentro de la etiqueta <head>.

A continuación, el código JavaScript se debe incluir entre las etiquetas <script>...</script>. Además, para que la página sea válida, es necesario definir el atributo type de la etiqueta <script>. Técnicamente, el atributo type se corresponde con "el tipo MIME", que es un estándar para identificar los diferentes tipos de contenidos. El "tipo MIME" correcto para JavaScript es text/javascript.

Una vez definida la zona en la que se incluirá el script, se escriben todas las sentencias que forman la aplicación. Este primer ejemplo es tan sencillo que solamente incluye una sentencia: window.confirm("Hola Mundo!");.

La instrucción confirm() es una de las utilidades que incluye JavaScript y permite mostrar un mensaje en la pantalla del usuario junto con los botones aceptar y cancelar. Si se visualiza la página web de este primer script en cualquier navegador, automáticamente se mostrará una ventana con el mensaje "Hola Mundo!".

Ejercicio 01

Modificar el primer script para que:

1. Todo el código JavaScript se encuentre en un archivo externo llamado codigo.js y el script siga funcionando de la misma manera.
2. Después del primer mensaje, se debe mostrar otro mensaje que diga "Soy el primer script"
3. Añadir algunos comentarios que expliquen el funcionamiento del código
4. Añadir en la página XHTML un mensaje de aviso para los navegadores que no tengan activado el soporte de JavaScript

3 PROGRAMACIÓN BÁSICA

Antes de comenzar a desarrollar programas y utilidades con JavaScript, es necesario conocer los elementos básicos con los que se construyen las aplicaciones. Si ya sabes programar en algún lenguaje de programación, este capítulo te servirá para conocer la sintaxis específica de JavaScript.

Si nunca has programado, este capítulo explica en detalle y comenzando desde cero los conocimientos básicos necesarios para poder entender posteriormente la programación avanzada, que es la que se utiliza para crear las aplicaciones reales.

3.1 Variables

Las variables en los lenguajes de programación siguen una lógica similar a las variables utilizadas en otros ámbitos como las matemáticas. Una variable es un elemento que se emplea para almacenar y hacer referencia a otro valor. Gracias a las variables es posible crear "programas genéricos", es decir, programas que funcionan siempre igual independientemente de los valores concretos utilizados.

De la misma forma que si en Matemáticas no existieran las variables no se podrían definir las ecuaciones y fórmulas, en programación no se podrían hacer programas realmente útiles sin las variables.

Si no existieran variables, un programa que suma dos números podría escribirse como:

```
resultado = 3 + 1
```

El programa anterior es tan poco útil que sólo sirve para el caso en el que el primer número de la suma sea el 3 y el segundo número sea el 1. En cualquier otro caso, el programa obtiene un resultado incorrecto.

Sin embargo, el programa se puede rehacer de la siguiente manera utilizando variables para almacenar y referirse a cada número:

```
numero_1 = 3
numero_2 = 1
resultado = numero_1 + numero_2
```

Los elementos numero_1 y numero_2 son variables que almacenan los valores que utiliza el programa. El resultado se calcula siempre en función del valor almacenado por las variables, por lo que este programa funciona correctamente para cualquier par de números indicado. Si se modifica el valor de las variables numero_1 y numero_2, el programa sigue funcionando correctamente.

Las variables en JavaScript se crean mediante la palabra reservada var. De esta forma, el ejemplo anterior se puede realizar en JavaScript de la siguiente manera:

```
var numero_1;
numero_1 = 3;
var numero_2 = 1;
var resultado = numero_1 + numero_2;
```

La palabra reservada var solamente se debe indicar al definir por primera vez la variable, lo que se denomina declarar una variable. Cuando se utilizan las variables en el resto de instrucciones del script, solamente es necesario indicar su nombre.

En otras palabras, en el ejemplo anterior sería un error indicar lo siguiente:

```
var numero_1 = 3;
var numero_2 = 1;
var resultado = var numero_1 + var numero_2;
```

Si cuando se declara una variable se le asigna también un valor, se dice que la variable ha sido inicializada. En JavaScript no es obligatorio inicializar las variables, ya que se pueden declarar por una parte y asignarles un valor posteriormente. Por tanto, el ejemplo anterior se puede rehacer de la siguiente manera:

```
var numero_1;
var numero_2;

numero_1 = 3;
numero_2 = 1;

var resultado = numero_1 + numero_2;
```

Una de las características más sorprendentes de JavaSript para los programadores habituados a otros lenguajes de programación es que tampoco es necesario declarar las variables. En otras palabras, se pueden utilizar variables que no se han definido anteriormente mediante la palabra reservada var. El ejemplo anterior también es correcto en JavaScript de la siguiente forma:

```
var numero_1 = 3;
var numero_2 = 1;
resultado = numero_1 + numero_2;
```

La variable resultado no está declarada, por lo que JavaScript crea una variable global (más adelante se verán las diferencias entre variables locales y globales) y le asigna el valor correspondiente. De la misma forma, también sería correcto el siguiente código:

```
numero_1 = 3;
numero_2 = 1;
resultado = numero_1 + numero_2;
```

En cualquier caso, se recomienda declarar todas las variables que se vayan a utilizar.

El nombre de una variable también se conoce como identificador y debe cumplir las siguientes normas:

Sólo puede estar formado por letras, números y los símbolos $ (dólar) y _ (guión bajo).
El primer carácter no puede ser un número.

Por tanto, las siguientes variables tienen nombres correctos:

```
var $numero1;
var _$letra;
var $$$otroNumero;
var $_a__$4;
```

Sin embargo, las siguientes variables tienen identificadores incorrectos:

```
var 1numero;       // Empieza por un número
var numero;1_123;  // Contiene un carácter ";"
```

3.2 Tipos de variables

Aunque todas las variables de JavaScript se crean de la misma forma (mediante la palabra reservada var), la forma en la que se les asigna un valor depende del tipo de valor que se quiere almacenar (números, textos, etc.)

3.2.1 Numéricas

Se utilizan para almacenar valores numéricos enteros (llamados integer en inglés) o decimales (llamados float en inglés). En este caso, el valor se asigna indicando directamente el número entero o decimal. Los números decimales utilizan el carácter . (punto) en vez de , (coma) para separar la parte entera y la parte decimal:

```
var iva = 16;         // variable tipo entero
var total = 234.65;   // variable tipo decimal
```

Veamos un ejemplo usando un botón de formulario:

Ej03.02a-VariableNum.html

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Variable Numérica</title>
<script language="javascript">
var x;
x = 5
x = x+1

   function mensaje ()
   {  // Asi usamos la variable: "mensaje" + variable
      alert ("Resultado: "+x);
   }
</script>
</head>

<body>
   <form name="formulario1">
      <input type="button" name="boton1" value="clic" onClick=mensaje()>
   </form>
</body>
</html>
```

En el ejemplo anterior usamos el evento onClick en el formulario que llama a mensaje( ), una función (que ya veremos mas detenidamente mas adelante) de tipo void (sin argumentos).

Es importante reseñar que los pasos empleados a la hora realizar el cálculo han sido:
var x; → Definimos la variable, no obligatorio pero si recomendable
x = 5 → Asignamos un valor a la variable.

MUY IMPORTANTE: ¡No igualamos; asignamos! No es lo mismo
x = x+1 → Usamos el valor introducido en la variable, sumamos1 y lo ASIGNAMOS a la misma variable. Es fundamental comprender esto.

Una variable es una porción de la memoria de la máquina reservada a la que aplicamos un nombre. En ella introducimos un valor (NO IGUALAMOS) que luego podemos emplear a nuestro antojo. La suma la realiza el coprocesador matemático:
http://es.wikipedia.org/wiki/Coprocesador
El resultado es enviado a donde queramos, en este caso a la misma variable.

3.2.2 Cadenas de texto

Se utilizan para almacenar caracteres, palabras y/o frases de texto. Para asignar el valor a la variable, se encierra el valor entre comillas dobles o simples, para delimitar su comienzo y su final:

```
var mensaje = "Bienvenido a nuestro sitio web";
var nombreProducto = 'Producto ABC';
var letraSeleccionada = 'c';
```

En ocasiones, el texto que se almacena en las variables no es tan sencillo. Si por ejemplo el propio texto contiene comillas simples o dobles, la estrategia que se sigue es la de encerrar el texto con las comillas (simples o dobles) que no utilice el texto:

```
/* El contenido de texto1 tiene comillas simples, por lo que
    se encierra con comillas dobles */
      var texto1 = "Una frase con 'comillas simples' dentro";

/* El contenido de texto2 tiene comillas dobles, por lo que
    se encierra con comillas simples */
      var texto2 = 'Una frase con "comillas dobles" dentro';
```

No obstante, a veces las cadenas de texto contienen tanto comillas simples como dobles. Además, existen otros caracteres que son difíciles de incluir en una variable de texto (tabulador, ENTER, etc.) Para resolver estos problemas, JavaScript define un mecanismo para incluir de forma sencilla caracteres especiales y problemáticos dentro de una cadena de texto.

El mecanismo consiste en sustituir el carácter problemático por una combinación simple de caracteres. A continuación se muestra la tabla de conversión que se debe utilizar:

Si se quiere incluir...	**Se debe incluir...**
Una nueva línea	\n
Un tabulador	\t
Una comilla simple	\'
Una comilla doble	\"
Una barra inclinada	\\

De esta forma, el ejemplo anterior que contenía comillas simples y dobles dentro del texto se puede rehacer de la siguiente forma:

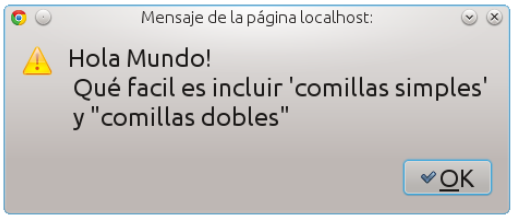
```
var texto1 = 'Una frase con \'comillas simples\' dentro';
var texto2 = "Una frase con \"comillas dobles\" dentro";
```

Este mecanismo de JavaScript se denomina "mecanismo de escape" de los caracteres problemáticos, y es habitual referirse a que los caracteres han sido "escapados".
Veamos un ejemplo completo: **Ej03.02b-VariableCadena.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <script language="javascript">
    var cadena = " Curso de FPE \n Programador de Aplicaciones.";
      function mensaje () // Ojo a los espacios en la cadena...
      {
        alert (""+cadena);      }
  </script>
</head>
<body>
  <form name="formulario1">
      <input type="button" name="boton1" value="clic" onClick=mensaje()>
  </form>
</body>
</html>
```

Ejercicio 02

Modificar el primer script del capítulo anterior para que:



1. El mensaje que se muestra se almacene en una variable llamada mimensaje y el funcionamiento del script sea el mismo.
2. El mensaje mostrado sea el de la siguiente imagen:

Y todo en un archivo externo llamado **Ejercicio_02.js.**

3.2.3 Arrays

En ocasiones, a los arrays se les llama vectores, matrices e incluso arreglos. No obstante, el término array es el más utilizado y es una palabra comúnmente aceptada en el entorno de la programación.

Un array es una colección de variables, que pueden ser todas del mismo tipo o cada una de un tipo diferente. Su utilidad se comprende mejor con un ejemplo sencillo: si una aplicación necesita manejar los días de la semana, se podrían crear siete variables de tipo texto:

```
var dia1 = "Lunes";
var dia2 = "Martes";
...
var dia7 = "Domingo";
```

Aunque el código anterior no es incorrecto, sí que es poco eficiente y complica en exceso la programación. Si en vez de los días de la semana se tuviera que guardar el nombre de los meses del año, el nombre de todos los países del mundo o las mediciones diarias de temperatura de los últimos 100 años, se tendrían que crear decenas o cientos de variables.

En este tipo de casos, se pueden agrupar todas las variables relacionadas en una colección de variables o array. El ejemplo anterior se puede rehacer de la siguiente forma:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado",
"Domingo"];
```

Ahora, una única variable llamada dias almacena todos los valores relacionados entre sí, en este caso los días de la semana. Para definir un array, se utilizan los caracteres [y] para delimitar su comienzo y su final y se utiliza el carácter , (coma) para separar sus elementos:

```
var nombre_array = [valor1, valor2, ..., valorN];
```

Una vez definido un array, es muy sencillo acceder a cada uno de sus elementos. Cada elemento se accede indicando su posición dentro del array. La única complicación, que es responsable de muchos errores cuando se empieza a programar, es que las posiciones de los elementos empiezan a contarse en el 0 y no en el 1:

```
var diaSeleccionado = dias[0];    // diaSeleccionado = "Lunes"
var otroDia = dias[5];            // otroDia = "Sábado"
```

En el ejemplo anterior, la primera instrucción quiere obtener el primer elemento del array. Para ello, se indica el nombre del array y entre corchetes la posición del elemento dentro del array. Como se ha comentado, las posiciones se empiezan a contar en el 0, por lo que el primer elemento ocupa la posición 0 y se accede a el mediante dias[0].

El valor dias[5] hace referencia al elemento que ocupa la sexta posición dentro del array dias. Como las posiciones empiezan a contarse en 0, la posición 5 hace referencia al sexto elemento, en este caso, el valor Sábado.

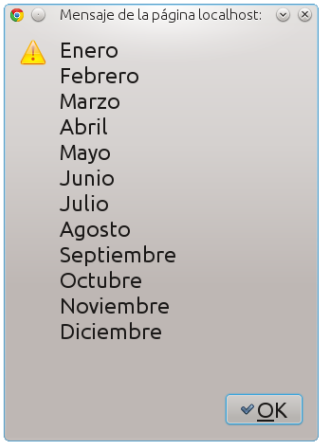
Veamos un ejemplo completo: **Ej03.02c-VariableArray.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>Variables Array</title>
  <script type="text/javascript">
    var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];
    var diahoy = dias [0];
      function mensaje ()
      {
        alert ("Hoy es: "+diahoy);
      }
 </script>
</head>
<body>
    <form name="formulario1">
      <input type="button" name="boton1" value="clic" onClick=mensaje()>
  </form>
</body>
</html>
```

Es esencial que nos demos cuenta que no podemos usar el array sin haberla, previamente, definido. Así, si intercambiamos:

```
   var diahoy = dias [0];
   var dias = ["Lunes", "Martes",
"Miércoles", "Jueves",
"Viernes", "Sábado", "Domingo"];
```

Nos aparecerá el mensaje: Hoy es: **undefined.**



Ejercicio 03

Crear un array llamado meses y que almacene el nombre de los doce meses del año. Mostrar por pantalla los doce nombres utilizando la función alert().
El resultado tiene que ser el de la siguiente imagen:

3.2.4 Booleanos

Las variables de tipo boolean o booleano también se conocen con el nombre de variables de tipo lógico. Aunque para entender realmente su utilidad se debe estudiar la programación avanzada con JavaScript del siguiente capítulo, su funcionamiento básico es muy sencillo.

Una variable de tipo boolean almacena un tipo especial de valor que solamente puede tomar dos valores: **true (verdadero) o false (falso)**. No se puede utilizar para almacenar números y tampoco permite guardar cadenas de texto.

Los únicos valores que pueden almacenar estas variables son true y false, por lo que no pueden utilizarse los valores verdadero y falso. A continuación se muestra un par de variables de tipo booleano:

```
var clienteRegistrado = false;
var ivaIncluido = true;
```

3.3 Operadores

Las variables por sí solas son de poca utilidad. Hasta ahora, sólo se ha visto cómo crear variables de diferentes tipos y cómo mostrar su valor mediante la función alert(). Para hacer programas realmente útiles, son necesarias otro tipo de herramientas.

Los operadores permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar diferentes variables. De esta forma, los operadores permiten a los programas realizar cálculos complejos y tomar decisiones lógicas en función de comparaciones y otros tipos de condiciones.

3.3.1 Asignación

El operador de asignación es el más utilizado y el más sencillo. Este operador se utiliza para guardar un valor específico en una variable. El símbolo utilizado es = (no confundir con el operador == que se verá más adelante):
var numero1 = 3;

NOTA IMPORTANTE: Fijaros que en el párrafo anterios se dice "para guardar" y no "para igualar". En una variable introducimos datos, no lo igualamos a nada, es una porción de memoría RAM a la que ponemos un nombre para poder usarla a nuestro antojo.

A la izquierda del operador, siempre debe indicarse el nombre de una variable. A la derecha del operador, se pueden indicar variables, valores, condiciones lógicas, etc:

```
var numero1 = 3;
var numero2 = 4;

/* Error, la asignación siempre se realiza a una variable,
    por lo que en la izquierda no se puede indicar un número */
5 = numero1;

// Ahora, la variable numero1 vale 5
numero1 = 5;

// Ahora, la variable numero1 vale 4
numero1 = numero2;
```

3.3.2 Incremento y decremento

Estos dos operadores solamente son válidos para las variables numéricas y se utilizan para incrementar o decrementar en una unidad el valor de una variable.

Veamos un ejemplo sencillo: **Ej03.03a-OperadorIncremento.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>Operadores</title>
  <script type="text/javascript">
   var numero = 5;
   ++numero;
   alert(numero);  // numero = 6
 </script>
</head>
<body>
</body>
</html>
```

El operador de incremento se indica mediante el prefijo ++ en el nombre de la variable. El resultado es que el valor de esa variable se incrementa en una unidad. Por tanto, el anterior ejemplo es equivalente a:

```
var numero = 5;
numero = numero + 1;
alert(numero);  // numero = 6
```

De forma equivalente, el operador decremento (indicado como un prefijo -- en el nombre de la variable) se utiliza para decrementar el valor de la variable:

```
var numero = 5;
--numero;
alert(numero);  // numero = 4
```

El anterior ejemplo es equivalente a:

```
var numero = 5;
numero = numero - 1;
alert(numero);  // numero = 4
```

Los operadores de incremento y decremento no solamente se pueden indicar como prefijo del nombre de la variable, sino que también es posible utilizarlos como sufijo. En este caso, su comportamiento es similar pero muy diferente. En el siguiente ejemplo:

```
var numero = 5;
numero++;
alert(numero);  // numero = 6
```

El resultado de ejecutar el script anterior es el mismo que cuando se utiliza el operador ++numero, por lo que puede parecer que es equivalente indicar el operador ++ delante o detrás del identificador de la variable. Sin embargo, el siguiente ejemplo muestra sus diferencias:

Ej03.03b-IncrementoSufijo.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>Incremento</title>
  <script type="text/javascript">
      function mensaje ()
      {
        var numero1 = 5;
        var numero2 = 2;
        numero3 = numero1++ + numero2;
        // numero3 = 7, numero1 = 6

        /* Probamos el cambio:
         * var numero1 = 5;
         * var numero2 = 2;
         * numero3 = ++numero1 + numero2;
         * // numero3 = 8, numero1 = 6
         * */
      }
  </script>
</head>
<body>
    <form name=”formulario1”>
      <input type="button" name="boton1" value="clic" onclick="mensaje();" />
  </form>
</body>
</html>
```

Si el operador ++ se indica como prefijo del identificador de la variable, su valor se incrementa antes de realizar cualquier otra operación. Si el operador ++ se indica como sufijo del identificador de la variable, su valor se incrementa después de ejecutar la sentencia en la que aparece.

Por tanto, en la instrucción numero3 = numero1++ + numero2;, el valor de numero1 se incrementa después de realizar la operación (primero se suma y numero3 vale 7, después se incrementa el valor de numero1 y vale 6). Sin embargo, en la instrucción numero3 = ++numero1 + numero2;, en primer lugar se incrementa el valor de numero1 y después se realiza la suma (primero se incrementa numero1 y vale 6, después se realiza la suma y numero3 vale 8).

3.3.3 Lógicos

Los operadores lógicos son imprescindibles para realizar aplicaciones complejas, ya que se utilizan para tomar decisiones sobre las instrucciones que debería ejecutar el programa en función de ciertas condiciones. El resultado de cualquier operación que utilice operadores lógicos siempre es un valor lógico o booleano.

3.3.3.1 Negación

Uno de los operadores lógicos más utilizados es el de la negación. Se utiliza para obtener el valor contrario al valor de la variable. Veamos un ejemplo nuevo modificando el anterior:

Ej03.03c-Negación.html

```
<!DOCTYPE html>
...
  <title>Variables Array</title>
  <script type="text/javascript">
      function mensaje ()
      {
        var visible = true;
        alert(!visible);  // Muestra "false" y no "true"
      }
  </script>
...
```

La negación lógica se obtiene prefijando el símbolo ! al identificador de la variable. El funcionamiento de este operador se resume en la siguiente tabla:

variable	!variable
true	false
false	true

Si la variable original es de tipo booleano, es muy sencillo obtener su negación. Sin embargo, ¿qué sucede cuando la variable es un número o una cadena de texto? Para obtener la negación en este tipo de variables, se realiza en primer lugar su conversión a un valor booleano:

- Si la variable contiene un número, se transforma en false si vale 0 y en true para cualquier otro número (positivo o negativo, decimal o entero).
- Si la variable contiene una cadena de texto, se transforma en false si la cadena es vacía ("") y en true en cualquier otro caso.

```
var cantidad = 0;
vacio = !cantidad;  // vacio = true

cantidad = 2;
vacio = !cantidad;  // vacio = false

var mensaje = "";
mensajeVacio = !mensaje;  // mensajeVacio = true

mensaje = "Bienvenido";
mensajeVacio = !mensaje;  // mensajeVacio = false
```

3.3.3.2 AND

La operación lógica AND obtiene su resultado combinando dos valores booleanos. El operador se indica mediante el símbolo && (doble ampersand) y su resultado solamente es true si los dos operandos son true:

variable1	variable2	variable1 && variable2
true	true	true
true	false	false
false	true	false
false	false	false

```
var valor1 = true;
var valor2 = false;
resultado = valor1 && valor2; // resultado = false
valor1 = true;
valor2 = true;
resultado = valor1 && valor2; // resultado = true
```

3.3.3.3 OR

La operación lógica OR también combina dos valores booleanos. El operador se indica mediante el símbolo || y su resultado es true si alguno de los dos operandos es true:

variable1	variable2	variable1 || variable2
true	true	true
true	false	true
false	true	true
false	false	false

```
var valor1 = true;
var valor2 = false;
resultado = valor1 || valor2; // resultado = true

valor1 = false;
valor2 = false;
resultado = valor1 || valor2; // resultado = false
```

3.3.4 Matemáticos

JavaScript permite realizar manipulaciones matemáticas sobre el valor de las variables numéricas. Los operadores definidos son: suma (+), resta (-), multiplicación (*) y división (/).
NOTA: En este ejemplo usaremos window.prompt que permite solicitar valores al usuario.
Ejemplo: **Ej03.03d-Producto.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>Producto</title>
  <script type="text/javascript">
    var operador1 = window.prompt ("Escriba 1er Operador: ");
    var operador2 = window.prompt ("Escriba 2° Operador: ");
    var resultado = operador1 * operador2;
    window.alert ("El resultado es: "+resultado);
  </script>
</head>
<body></body>
</html>
```

Otros ejemplos:

```
var numero1 = 10;
var numero2 = 5;

resultado = numero1 / numero2;  // resultado = 2
resultado = 3 + numero1;        // resultado = 13
resultado = numero2 - 4;        // resultado = 1
resultado = numero1 * numero 2; // resultado = 50
```

Además de los cuatro operadores básicos, JavaScript define otro operador matemático que no es sencillo de entender cuando se estudia por primera vez, pero que es muy útil en algunas ocasiones.

Se trata del operador "módulo", que calcula el resto de la división entera de dos números. Si se divide por ejemplo 10 y 5, la división es exacta y da un resultado de 2. El resto de esa división es 0, por lo que módulo de 10 y 5 es igual a 0.

Sin embargo, si se divide 9 y 5, la división no es exacta, el resultado es 1 y el resto 4, por lo que módulo de 9 y 5 es igual a 4.

El operador módulo en JavaScript se indica mediante el símbolo %, que no debe confundirse con el cálculo del porcentaje. Veamos un ejemplo: **Ej03.03e-RestoModulo.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>Producto</title>
  <script type="text/javascript">
    var operador1 = window.prompt ("Escriba 1er Divisor: ");
    var operador2 = window.prompt ("Escriba 2° Divisor: ");
    var resultado = operador1 % operador2;
    window.alert ("El Resto o Módulo es: "+resultado);
  </script>
</head>
<body></body>
</html>
```

Mas ejemplos:

```
var numero1 = 10;
var numero2 = 5;
resultado = numero1 % numero2; // resultado = 0

numero1 = 9;
numero2 = 5;
resultado = numero1 % numero2; // resultado = 4
```

Los operadores matemáticos también se pueden combinar con el operador de asignación para abreviar su notación:

```
var numero1 = 5;
numero1 += 3;  // numero1 = numero1 + 3 = 8
numero1 -= 1;  // numero1 = numero1 - 1 = 4
numero1 *= 2;   // numero1 = numero1 * 2 = 10
numero1 /= 5;   // numero1 = numero1 / 5 = 1
numero1 %= 4;   // numero1 = numero1 % 4 = 1
```

3.3.5 Relacionales

Los operadores relacionales definidos por JavaScript son idénticos a los que definen las matemáticas: mayor que (>), menor que (<), mayor o igual (>=), menor o igual (<=), igual que (==) y distinto de (!=).

Los operadores que relacionan variables son imprescindibles para realizar cualquier aplicación compleja, como se verá en el siguiente capítulo de programación avanzada. El resultado de todos estos operadores siempre es un valor booleano:

Veamos un ejemplo: **Ej03.03f-Relacionales.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>Producto</title>
  <script type="text/javascript">
      var dato1, dato2;
      var operador1, operador2;
      var resultado;

      function ejecutar (){
         dato1 = window.prompt ("Escribe un n°: ");
         dato2 = window.prompt ("Escribe otro n°: ");

         operador1 = parseInt (dato1);
         operador2 = parseInt (dato2);
         resultado = operador1 >= operador2;                    // TRUE
         window.alert (" Si "+operador1+ " >=" + operador2
                   +" el resultado será TRUE; " + "\n"
                   +" En caso contrario será FALSE: " + "\n"
                   +" Resultado Final: "+resultado);
      }
  </script>
</head>
<body></body>
</html>
```

Mas ejemplos:

```
var numero1 = 3;
var numero2 = 5;
resultado = numero1 > numero2; // resultado = false
resultado = numero1 < numero2; // resultado = true

numero1 = 5;
numero2 = 5;
resultado = numero1 >= numero2; // resultado = true
resultado = numero1 <= numero2; // resultado = true
resultado = numero1 == numero2; // resultado = true
resultado = numero1 != numero2; // resultado = false
```

Se debe tener especial cuidado con el operador de igualdad (==), ya que es el origen de la mayoría de errores de programación, incluso para los usuarios que ya tienen cierta experiencia desarrollando scripts. El operador == se utiliza para comparar el valor de dos variables, por lo que es muy diferente del operador =, que se utiliza para asignar un valor a una variable:

```
// El operador "=" asigna valores
var numero1 = 5;
resultado = numero1 = 3;  // numero1 = 3 y resultado = 3
 // El operador "==" compara variables
var numero1 = 5;
resultado = numero1 == 3; // numero1 = 5 y resultado = false
```

Los operadores relacionales también se pueden utilizar con variables de tipo cadena de texto:

Veamos un ejemplo: **Ej03.03g-Comparaciones.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>Producto</title>
  <script type="text/javascript">
    var texto1 = "hola";
    var texto2 = "hola";
    var texto3 = "adios";

    resultado = texto1 == texto3; // resultado = false
    alert(""+resultado);
  </script>
</head>
<body></body>
</html>
```

Cuando se utilizan cadenas de texto, los operadores "mayor que" (>) y "menor que" (<) siguen un razonamiento no intuitivo: se compara letra a letra comenzando desde la izquierda hasta que se encuentre una diferencia entre las dos cadenas de texto. Para determinar si una letra es mayor o menor que otra, las mayúsculas se consideran menores que las minúsculas y las primeras letras del alfabeto son menores que las últimas (a es menor que b, b es menor que c, A es menor que a, etc.)

Ejercicio 04
A partir del siguiente array que se proporciona: var valores = [true, 5, false, "hola", "adios", 2];

1. Determinar cual de los dos elementos de texto es mayor
2. Utilizando exclusivamente los dos valores booleanos del array, determinar los operadores necesarios para obtener un resultado true y otro resultado false
   (habrá que emplear AND y OR).
3. Determinar el resultado de las cinco operaciones matemáticas realizadas con los dos elementos numéricos.

3.4 Estructuras de control de flujo

Los programas que se pueden realizar utilizando solamente variables y operadores son una simple sucesión lineal de instrucciones básicas.

Sin embargo, no se pueden realizar programas que muestren un mensaje si el valor de una variable es igual a un valor determinado y no muestren el mensaje en el resto de casos. Tampoco se puede repetir de forma eficiente una misma instrucción, como por ejemplo sumar un determinado valor a todos los elementos de un array.

Para realizar este tipo de programas son necesarias las estructuras de control de flujo, que son instrucciones del tipo "si se cumple esta condición, hazlo; si no se cumple, haz esto otro". También existen instrucciones del tipo "repite esto mientras se cumpla esta condición".

Si se utilizan estructuras de control de flujo, los programas dejan de ser una sucesión lineal de instrucciones para convertirse en programas inteligentes que pueden tomar decisiones en función del valor de las variables.

3.4.1 Estructura if.

La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación es la estructura if. Se emplea para tomar decisiones en función de una condición. Su definición formal es:

```
if(condicion) {
  ...
}
```

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro de {...}. Si la condición no se cumple (es decir, si su valor es false) no se ejecuta ninguna instrucción contenida en {...} y el programa continúa ejecutando el resto de instrucciones del script.

Ejemplo:

```
var mostrarMensaje = true;

if(mostrarMensaje) {
  alert("Hola Mundo");
}
```

En el ejemplo anterior, el mensaje sí que se muestra al usuario ya que la variable mostrarMensaje tiene un valor de true y por tanto, el programa entra dentro del bloque de instrucciones del if.

El ejemplo se podría reescribir también como:

```
var mostrarMensaje = true;

if(mostrarMensaje == true) {
  alert("Hola Mundo");
}
```

En este caso, la condición es una comparación entre el valor de la variable mostrarMensaje y el valor true. Como los dos valores coinciden, la igualdad se cumple y por tanto la condición es cierta, su valor es true y se ejecutan las instrucciones contenidas en ese bloque del if.

La comparación del ejemplo anterior suele ser el origen de muchos errores de programación, al confundir los operadores == y =.

Las comparaciones siempre se realizan con el operador ==, ya que el operador = solamente asigna valores:

```
var mostrarMensaje = true;

// Se comparan los dos valores
if(mostrarMensaje == false) {
  ...
}

// Error - Se asigna el valor "false" a la variable
if(mostrarMensaje = false) {
  ...
}
```

La condición que controla el if() puede combinar los diferentes operadores lógicos y relacionales mostrados anteriormente:

```
var mostrado = false;

if(!mostrado) {
  alert("Es la primera vez que se muestra el mensaje");
}
```

Los operadores AND y OR permiten encadenar varias condiciones simples para construir condiciones complejas:

```
var mostrado = false;
var usuarioPermiteMensajes = true;

if(!mostrado && usuarioPermiteMensajes) {
  alert("Es la primera vez que se muestra el mensaje");
}
```

La condición anterior está formada por una operación AND sobre dos variables. A su vez, a la primera variable se le aplica el operador de negación antes de realizar la operación AND. De esta forma, como el valor de mostrado es false, el valor !mostrado sería true. Como la variable usuarioPermiteMensajes vale true, el resultado de !mostrado && usuarioPermiteMensajes sería igual a true && true, por lo que el resultado final de la condición del if() sería true y por tanto, se ejecutan las instrucciones que se encuentran dentro del bloque del if().

Ejercicio 05

1. Crear un script que pregunte la edad del usuario y lo introduzca en una variable.
2. Si dicha edad es igual o superior a 18, mostrar el mensaje "Eres mayor de Edad".
3. Si tiene JUSTO 18, mostrar además el mensaje "Enhorabuena por tu mayoría de Edad!"
4. Si la edad es inferior a 18, mostrar el mensaje "Eres menor de edad".

NOTA: Se presupone, por ahora, que el usuario sólo va a escribir un valor numérico.

Ejercicio 05 Especial

Realizar un script que haga lo siguiente:

1. Crear un botón que pedirá USUARIO y CONTRASEÑA.
2. Pedir el nombre de usuario que se grabará en una variable global
3. Pedir una contraseña que se tendrá que repetir.
4. Si AMBAS contraseñas coinciden, se grabará en una variable global.
5. Crear OTRO botón que pedirá el acceso a la web: usuario y contraseña.
6. Mostrar SI el usuario es correcto y SI la contraseña es correcta.
7. Crear un contador (variable global) que tras 3 intentos fallidos de acceso redirija a: http://abduzeedo.com/4023

3.4.2 If...else.

En ocasiones, las decisiones que se deben realizar no son del tipo "si se cumple la condición, hazlo; si no se cumple, no hagas nada". Normalmente las condiciones suelen ser del tipo "si se cumple esta condición, hazlo; si no se cumple, haz esto otro".

Para este segundo tipo de decisiones, existe una variante de la estructura if llamada if...else. Su definición formal es la siguiente:

```
if(condicion) {
  ...
}
else {
  ...
}
```

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro del if(). Si la condición no se cumple (es decir, si su valor es false) se ejecutan todas las instrucciones contenidas en else { }. Ejemplo:

```
var edad = 18;

if(edad >= 18) {
  alert("Eres mayor de edad");
}
else {
  alert("Todavía eres menor de edad");
}
```

Si el valor de la variable edad es mayor o igual que el valor numérico 18, la condición del if() se cumple y por tanto, se ejecutan sus instrucciones y se muestra el mensaje "Eres mayor de edad". Sin embargo, cuando el valor de la variable edad no es igual o mayor que 18, la condición del if() no se cumple, por lo que automáticamente se ejecutan todas las instrucciones del bloque else { }. En este caso, se mostraría el mensaje "Todavía eres menor de edad".

El siguiente ejemplo compara variables de tipo cadena de texto:

```
var nombre = "";

if(nombre == "") {
  alert("Aún no nos has dicho tu nombre");
}
else {
  alert("Hemos guardado tu nombre");
}
```

La condición del if() anterior se construye mediante el operador ==, que es el que se emplea para comparar dos valores (no confundir con el operador = que se utiliza para asignar valores). En el ejemplo anterior, si la cadena de texto almacenada en la variable nombre es vacía (es decir, es igual a "") se muestra el mensaje definido en el if(). En otro caso, se muestra el mensaje definido en el bloque else { }.

La estructura if...else se puede encadenar para realizar varias comprobaciones seguidas:

```
if(edad < 12) {
  alert("Todavía eres muy pequeño");
}
else if(edad < 19) {
  alert("Eres un adolescente");
}
else if(edad < 35) {
  alert("Aun sigues siendo joven");
}
else {
  alert("Piensa en cuidarte un poco más");
}
```

No es obligatorio que la combinación de estructuras if...else acabe con la instrucción else, ya que puede terminar con una instrucción de tipo else if().

Ejercicio 06

El cálculo de la letra del Documento Nacional de Identidad (DNI) es un proceso matemático sencillo que se basa en obtener el resto de la división entera del número de DNI y el número 23. A partir del resto de la división, se obtiene la letra seleccionándola dentro de un array de letras.

El array de letras es:

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

Por tanto si el resto de la división es 0, la letra del DNI es la T y si el resto es 3 la letra es la A. Con estos datos, elaborar un pequeño script que:

1. Almacene en una variable el número de DNI indicado por el usuario y en otra variable la letra del DNI que se ha indicado. (Pista: si se quiere pedir directamente al usuario que indique su número y su letra, se puede utilizar la función prompt())
2. En primer lugar (y en una sola instrucción) se debe comprobar si el número es menor que 0 o mayor que 99999999. Si ese es el caso, se muestra un mensaje al usuario indicando que el número proporcionado no es válido y el programa no muestra más mensajes.
3. Si el número es válido, se calcula la letra que le corresponde según el método explicado anteriormente.
4. Una vez calculada la letra, se debe comparar con la letra indicada por el usuario. Si no coinciden, se muestra un mensaje al usuario diciéndole que la letra que ha indicado no es correcta. En otro caso, se muestra un mensaje indicando que el número y la letra de DNI son correctos.

3.4.3 Estructura For

Las estructuras if y if...else no son muy eficientes cuando se desea ejecutar de forma repetitiva una instrucción. Por ejemplo, si se quiere mostrar un mensaje cinco veces, se podría pensar en utilizar el siguiente if:

```
var veces = 0;

if(veces < 4) {
  alert("Mensaje");
  veces++;
}
```

Se comprueba si la variable veces es menor que 4. Si se cumple, se entra dentro del if(), se muestra el mensaje y se incrementa el valor de la variable veces. Así se debería seguir ejecutando hasta mostrar el mensaje las cinco veces deseadas.

Sin embargo, el funcionamiento real del script anterior es muy diferente al deseado, ya que solamente se muestra una vez el mensaje por pantalla. La razón es que la ejecución de la estructura if() no se repite y la comprobación de la condición sólo se realiza una vez, independientemente de que dentro del if() se modifique el valor de la variable utilizada en la condición.

La estructura for permite realizar este tipo de repeticiones (también llamadas bucles) de una forma muy sencilla. No obstante, su definición formal no es tan sencilla como la de if():

```
for(inicializacion; condicion; actualizacion) {
  ...
}
```

La idea del funcionamiento de un bucle for es la siguiente: "mientras la condición indicada se siga cumpliendo, repite la ejecución de las instrucciones definidas dentro del for. Además, después de cada repetición, actualiza el valor de las variables que se utilizan en la condición".

- La "inicialización" es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.
- La "condición" es el único elemento que decide si continua o se detiene la repetición.
- La "actualización" es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

Veamos un ejemplo: **Ej03.04a-BucleFor.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>BucleFor</title>
  <script type="text/javascript">
      var mensaje = "Comienza el bucle";
      for(var i = 0; i < 5; i++) {
         alert("Iteración "+i);
      }
  </script>
</head>
<body></body>
</html>
```

La parte de la inicialización del bucle consiste en:

```
var i = 0;
```

Por tanto, en primer lugar se crea la variable i y se le asigna el valor de 0. Esta zona de inicialización solamente se tiene en consideración justo antes de comenzar a ejecutar el bucle. Las siguientes repeticiones no tienen en cuenta esta parte de inicialización.

La zona de condición del bucle es:

```
i < 5
```

Los bucles se siguen ejecutando mientras se cumplan las condiciones y se dejan de ejecutar justo después de comprobar que la condición no se cumple. En este caso, mientras la variable i valga menos de 5 el bucle se ejecuta indefinidamente.

Como la variable i se ha inicializado a un valor de 0 y la condición para salir del bucle es que i sea menor que 5, si no se modifica el valor de i de alguna forma, el bucle se repetiría indefinidamente.

Por ese motivo, es imprescindible indicar la zona de actualización, en la que se modifica el valor de las variables que controlan el bucle:

```
i++
```

En este caso, el valor de la variable i se incrementa en una unidad después de cada repetición. La zona de actualización se ejecuta después de la ejecución de las instrucciones que incluye el for.
Si queremos que el incremento sea diferente (por ejemplo de 2 en 2) pondremos:

```
i+=2
```
IMPORTANTE: Si se pone `i+2` SE BLOQUEA el navegador.

Así, durante la ejecución de la quinta repetición el valor de i será 4. Después de la quinta ejecución, se actualiza el valor de i, que ahora valdrá 5. Como la condición es que i sea menor que 5, la condición ya no se cumple y las instrucciones del for no se ejecutan una sexta vez.

Normalmente, la variable que controla los bucles for se llama i, ya que recuerda a la palabra índice y su nombre tan corto ahorra mucho tiempo y espacio.

El ejemplo anterior que mostraba los días de la semana contenidos en un array se puede rehacer de forma más sencilla utilizando la estructura for:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado",
"Domingo"];

for(var i=0; i<7; i++) {
  alert(dias[i]);
}
```

Ejercicio 7

1. Solicitar al usuario un número.
2. Presentar por pantalla (en ventanas emergentes) la tabla de multiplicar (del 1 al 9) de dicho número.
3. (para NOTA!!) Presentar la tabla de multiplicar de los primeros 9 números SIN PEDIR ningún dato al usuario. (pista: hay que usar FOR ANIDADOS)

Ejercicio 7 Especial

El factorial de un número entero n es una operación matemática que consiste en multiplicar todos los factores n x (n-1) x (n-2) x ... x 1. Así, el factorial de 5 (escrito como 5!) es igual a: 5! = 5 x 4 x 3 x 2 x 1 = 120

Utilizando la estructura for, crear un script que calcule el factorial de un número entero.

3.4.4 Estructura for...in

Una estructura de control derivada de for es la estructura for...in. Su definición exacta implica el uso de objetos, que es un elemento de programación avanzada que no se va a estudiar. Por tanto, solamente se va a presentar la estructura for...in adaptada a su uso en arrays. Su definición formal adaptada a los arrays es:

```
for(indice in array) {
  ...
}
```

Si se quieren recorrer todos los elementos que forman un array, la estructura for...in es la forma más eficiente de hacerlo, como se muestra en el siguiente ejemplo:

Veamos un ejemplo: **Ej03.04b-BucleForIn.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>BucleForIn</title>
  <script type="text/javascript">
   var dias =
      ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];
   for(i in dias) {
      alert(dias[i]);
   }
  </script>
</head>
<body></body>
</html>
```

La variable que se indica como indice es la que se puede utilizar dentro del bucle for...in para acceder a los elementos del array. De esta forma, en la primera repetición del bucle la variable i vale 0 y en la última vale 6.

Esta estructura de control es la más adecuada para recorrer arrays (y objetos), ya que evita tener que indicar la inicialización y las condiciones del bucle for simple y funciona correctamente cualquiera que sea la longitud del array. De hecho, sigue funcionando igual aunque varíe el número de elementos del array.

Ejercicio 7b Especial

1. Crear las siguientes listas del Supermercado
   - Frutas (5 elementos)
   - Pescados (5 elementos)
   - Carnes (5 elementos)
   - Droguería (5 elementos)
2. Crear un menú web donde se preguntará al usuario que lista visualizar.
3. Recoger la opción elegida y en función de la misma, mostrar la lista con los elementos en vertical usando for...in.

(TRAS VER EL MÉTODO SPLIT) **Ejercicio 7c**
Modificar el ejercicio anterior para que las listas sean pedidas al usuario .
Es decir, tendremos 1menú principal con 2 opciones:
a) Introducir listas del supermercado.
b) Visualizar las listas guardadas
A su vez cada opción da lugar a otro menú para Introducir/ver cada una de las listas.
En la introducción, preguntar previamente al usuario por el carácter separador (para el split).
Por último, controlar que la lista visualizada o introducida está en blanco.

3.5 Funciones y propiedades básicas de JavaScript

JavaScript incorpora una serie de herramientas y utilidades (llamadas funciones y propiedades, como se verá más adelante) para el manejo de las variables. De esta forma, muchas de las operaciones básicas con las variables, se pueden realizar directamente con las utilidades que ofrece JavaScript.

3.5.1 Funciones útiles para cadenas de texto.

A continuación se muestran algunas de las funciones más útiles para el manejo de cadenas :
Para ello usaremos el objeto String cuyo constructor (que ya veremos que es) es:
new String ( );

length, calcula la longitud de una cadena de texto (el número de caracteres que la forman)

```
var mensaje = "Hola Mundo";
var numeroLetras = mensaje.length; // numeroLetras = 10
```

Veamos un ejemplo usando DOM: **Ej03.05a-Length.html**

```
<!DOCTYPE html>
<html lang="es">        <!--Ej03.05a-Length.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      // Para crearnos una instancia del Objeto String (cadena)...
      var cadena = new String ();

      var numcaracteres;

      function mensaje (){
         cadena = window.prompt (" Escriba una cadena: ");
         numcaracteres =cadena.length;
         window.alert (" Total Caracteres: "+numcaracteres);
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Ver Longitud" onclick="mensaje ();" />
      </form>
  </body>
</html>
```

+, se emplea para concatenar varias cadenas de texto

```
var mensaje1 = "Hola";
var mensaje2 = " Mundo";
var mensaje = mensaje1 + mensaje2; // mensaje = "Hola Mundo"
```

Además del operador +, también se puede utilizar la función **concat()**

```
var cadena1 = "Primera Cadena";
var cadena2 = " - Segunda Cadena";
var cadenafinal = cadena1.concat (cadena2);
window.alert (cadenafinal);
```

Las cadenas de texto también se pueden unir con variables numéricas:

```
var variable1 = "Hola ";
var variable2 = 3;
var mensaje = variable1 + variable2; // mensaje = "Hola 3"
```

Cuando se unen varias cadenas de texto es habitual olvidar añadir un espacio de separación entre las palabras:

```
var mensaje1 = "Hola";
var mensaje2 = "Mundo";
var mensaje = mensaje1 + mensaje2; // mensaje = "HolaMundo"
```

Los espacios en blanco se pueden añadir al final o al principio de las cadenas y también se pueden indicar forma explícita:

```
var mensaje1 = "Hola";
var mensaje2 = "Mundo";
var mensaje = mensaje1 + " " + mensaje2; // mensaje = "Hola Mundo"
```

toUpperCase(), transforma todos los caracteres de la cadena a sus correspondientes caracteres en mayúsculas:

```
var mensaje1 = "Hola";
var mensaje2 = mensaje1.toUpperCase(); // mensaje2 = "HOLA"
```

toLowerCase(), transforma todos los caracteres de la cadena a sus correspondientes caracteres en minúsculas:

```
var mensaje1 = "HolA";
var mensaje2 = mensaje1.toLowerCase(); // mensaje2 = "hola"
```

Veamos un ejemplo: **Ej03.05b-toLowerUpperCase.html**

```
<!DOCTYPE html>
<html lang="es">        <!--Ej03.05b-toLowerCase.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      var cadena = new String ();
      var resultado;

      function mayusculas() {
         cadena = window.prompt (" Escriba la cadena: ");
         resultado = cadena.toUpperCase();
         window.alert (resultado);          }

      function minusculas() {
         cadena = window.prompt (" Escriba la cadena: ");
         resultado = cadena.toLowerCase();
         window.alert (resultado);          }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Poner en Mayúsculas" onclick="mayusculas ();" />
         <br /><br />
         <input type="button" value="Poner en Minúsculas" onclick="minusculas ();" />
      </form>
  </body>
</html>
```

charAt(posicion), obtiene el carácter que se encuentra en la posición indicada:

```
var mensaje = "Hola";
var letra = mensaje.charAt(0); // letra = H
letra = mensaje.charAt(2);     // letra = l
```

CharCodeAt, devuelve el código UNICODE del carácter que se encuentra en la posición pasada por parámetro. Este ir desde 0 hasta el total de caracteres -1.

```
var texto = "Hola Mundo";
alert ("Pos3 en Hola Mundo = "+texto.charCodeAt (3));
```

fromCharCode: el método devuelve un String a partir de códigos UNICODE.
Para saber los caracteres estandar: http://www.unicode.org/charts/
Por ejemplo, entre el Cuneiform, elegimos el 12011, poniendo el siguiente código:
var texto = new String ();
texto = String.fromCharCode (12011,12012);
alert ("Carácter UNICODE Cuneiforme 12011 y 12012: "+texto);

Veamos un ejemplo: **Ej03.05c-funcionesCadena.html**

```
<!DOCTYPE html>
<html lang="es">        <!--Ej03.05c-funcionesCadenas.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">

      var cadena = new String ();
      var posicion;

      function verLetra() {
         cadena = window.prompt (" Escriba la Cadena: ");
         posicion = window.prompt (" Escriba la Posición de la Letra: ");
         window.alert (" La letra solicitada es: " + cadena.charAt(posicion));
      }

      function verLetraUnicode() {
         cadena = window.prompt (" Escriba la Cadena: ");
         posicion = window.prompt (" Escriba la Posición de la Letra: ");
         window.alert (" La letra solicitada es: " + cadena.charCodeAt(posicion));
      }

      function unicodeLetra() {
         posicion = window.prompt (" Escriba valor Unicode: ");
         cadena = String.fromCharCode (posicion);
         window.alert ("El Cáracter: "+cadena);
      }

   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Ver letra" onclick="verLetra ();" />
         <br /><br />
         <input type="button" value="Ver letra Unicode" onclick="verLetraUnicode ();" />
         <br /><br />
         <input type="button" value="Unicode a Letra" onclick="unicodeLetra ();" />
      </form>
  </body>
</html>
```

indexOf(caracter), calcula la posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si el carácter se incluye varias veces dentro de la cadena de texto, se devuelve su primera posición empezando a buscar desde la izquierda. Si la cadena no contiene el carácter, la función devuelve el valor -1:

```
var mensaje = "Hola";
var posicion = mensaje.indexOf('a'); // posicion = 3
posicion = mensaje.indexOf('b');     // posicion = -1
```

Su función análoga es lastIndexOf():

lastIndexOf(caracter), calcula la última posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si la cadena no contiene el carácter, la función devuelve el valor -1:

```
var mensaje = "Hola";
var posicion = mensaje.lastIndexOf('a'); // posicion = 3
posicion = mensaje.lastIndexOf('b');     // posicion = -1
```

La función lastIndexOf() comienza su búsqueda desde el final de la cadena hacia el principio, aunque la posición devuelta es la correcta empezando a contar desde el principio de la palabra.

Veamos un ejemplo: **Ej03.05d-lastIndexOf.html**

```
<!DOCTYPE html>
<html lang="es">        <!--Ej03.05d-lastIndexOf.html -->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      var cadena = new String ();
      var letra = new String ();
      var posicion;

      function buscarLetra() {
         cadena = window.prompt (" Escriba la Cadena: ");
         letra = window.prompt (" Escriba la Letra: ");
         posicion = cadena.indexOf (letra);
         window.alert (" La posición es: "+posicion);
      }

      function buscarUltimaLetra() {
         cadena = window.prompt (" Escriba la Cadena: ");
         letra = window.prompt (" Escriba la Letra: ");
         posicion = cadena.lastIndexOf(letra);
         window.alert (" La posición es: "+posicion);
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Buscar Letra" onclick="buscarLetra ();" />
         <br /><br />
         <input type="button" value="Buscar Ultima Letra"
               onclick="buscarUltimaLetra ();" />
      </form>
  </body>
</html>
```

substring(inicio, final), extrae una porción de una cadena de texto. El segundo parámetro es opcional. Si sólo se indica el parámetro inicio, la función devuelve la parte de la cadena original correspondiente desde esa posición hasta el final:

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(2); // porcion = "la Mundo"
porcion = mensaje.substring(5);     // porcion = "Mundo"
porcion = mensaje.substring(7);     // porcion = "ndo"
```

Si se indica un inicio negativo, se devuelve la misma cadena original:

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(-2); // porcion = "Hola Mundo"
```

Cuando se indica el inicio y el final, se devuelve la parte de la cadena original comprendida entre la posición inicial y la inmediatamente anterior a la posición final (es decir, la posición inicio está incluida y la posición final no):

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(1, 8); // porcion = "ola Mun"
porcion = mensaje.substring(5, 8);     // porcion = "Mun"
```

Si se indica un final más pequeño que el inicio, JavaScript los considera de forma inversa, ya que automáticamente asigna el valor más pequeño al inicio y el más grande al final:

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(5, 0); // porcion = "Hola "
porcion = mensaje.substring(0, 5);     // porcion = "Hola "
```

Veamos un ejemplo: **Ej03.05e-substring.html**

```
<html lang="es">       <!--Ej03.05e-substring.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      var cadena = new String ();
      var inicio, fin;
      var cadenacortada;

      function subcadena (){
         cadena = window.prompt (" Escriba cadena a cortar: ");
         inicio = window.prompt (" Escriba Valor Caracter Inicial: ");
         fin = window.prompt (" Escriba Valor Caracter Final: ");

         // Ej: Cadena: "Hola Hijos del Rock & Roll"
         cadenacortada = cadena.substring (inicio,fin);
         window.alert (" La Cadena Cortada será: "+cadenacortada);
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Cortar Cadena" onclick="subcadena ();" />
      </form>
  </body>
</html>
```

Slice, Devuelve una parte del objeto.
Formato: objetoString.slice (inicio [,fin]).Si se omite fin busca hasta el final del objeto.

Veamos un ejemplo: **Ej03.05f-slice.html**

```
<!DOCTYPE html>
<html lang="es">       <!--Ej03.05f-slice.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      var cadena = new String ();
      var inicio, fin;
      var cadenacortada;

      function subcadena (){
         cadena = window.prompt (" Escriba cadena a cortar: ");
         inicio = window.prompt (" Escriba Valor Caracter Inicial: ");

         // Ej: Cadena: "Hola Hijos del Rock & Roll"
         cadenacortada = cadena.slice (inicio);
         window.alert (" La Cadena Cortada será: "+cadenacortada);
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Cortar Cadena" onclick="subcadena ();" />
      </form>
  </body>
</html>
```

split(separador), convierte una cadena de texto en un array de cadenas de texto. La función parte la cadena de texto determinando sus trozos a partir del carácter separador indicado:

```
var mensaje = "Hola Mundo, soy una cadena de texto!";
var palabras = mensaje.split(" ");
// palabras = ["Hola", "Mundo,", "soy", "una", "cadena", "de", "texto!"];
```

Con esta función se pueden extraer fácilmente las letras que forman una palabra:

```
var palabra = "Hola";
var letras = palabra.split(""); // letras = ["H", "o", "l", "a"]
```

Veamos un ejemplo: **Ej03.05g-split.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <script type="text/javascript">
   var texto = new String ("Hola Peña del Curso");
   var array1 = new Array ();
   var indice;
   function mensaje ()
   {
      alert ("Cadena: Hola Peña del Curso");
      array1 = texto.split (" ");
      for (indice in array1)
         alert (""+array1[indice]);
}
  </script>
</head>
<body>
   <form name="form1">
      <input type="button" value="Mostrar Mensaje" onclick="mensaje ();" />
   </form>
</body> </html>
```

replace(cadenaBuscada,cadenaReemplazo), realiza una búsqueda en la cadena original de la cadenaBuscada. En caso de encontrarla, realiza el reemplazo en la primera ocasión (es decir, no sirve para reemplazar varias cadenas).

Veamos un ejemplo: **Ej03.05g-replace.html**

```
<!DOCTYPE html>
<html lang="es">        <!--Ej03.05g-replace.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      function ejecutar () {
         var cadenaEscogida = new String ();
         cadenaEscogida = window.prompt (" Escriba la cadena: ");
         var cadenaBuscar = window.prompt (" Escriba la cadena a buscar: ");
         var cadenaReemplazar = window.prompt (" Escriba el Reemplazo: ");

         var nuevaCadena = cadenaEscogida.replace (cadenaBuscar,cadenaReemplazar);
         window.alert (" La cadena con el Reemplazo es: \n"+nuevaCadena);
      }

   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Iniciar Programa" onclick="ejecutar ();" />
      </form>
  </body>
</html>
```

OTROS METODOS

localeCompare(expresiónCadena), devuelve un valor booleano que indica si dos cadenas son equivalentes en la configuración regional actual.

```
var compara = true;
var mensaje = "Niño";
compara = mensaje.localeCompare("nino");
// Devuelve false que se asigna a compara.
```

localeCompare(expresiónCadena), devuelve un valor booleano que indica si dos cadenas son equivalentes en la configuración regional actual.

```
var compara = true;
var mensaje = "Niño";
compara = mensaje.localeCompare("nino");
// Devuelve false que se asigna a compara.
```

3.5.2 Funciones útiles para arrays

A continuación se muestran algunas de las funciones más útiles para el manejo de arrays:

length, calcula el número de elementos de un array

```
var vocales = ["a", "e", "i", "o", "u"];
var numeroVocales = vocales.length; // numeroVocales = 5
```

concat(), se emplea para concatenar los elementos de varios arrays

```
var array1 = new Array ();
array1 = [1, 2, 3];
array2 = array1.concat(4, 5, 6);        // array2 = [1, 2, 3, 4, 5, 6]
array3 = array1.concat(7, 8, 9);        // array3 = [1, 2, 3, 7, 8, 9]
```

join(unificador), es la función contraria a split(). Une todos los elementos de un array para formar una cadena de texto. Para unir los elementos se utiliza el carácter separador indicado

```
var array = ["hola", "mundo"];
var mensaje = array.join(""); // mensaje = "holamundo"
mensaje = array.join(" ");    // mensaje = "hola mundo"
```

pop(), elimina el último elemento del array y lo devuelve. El array original se modifica y su longitud disminuye en 1 elemento.

```
var array = [1, 2, 3];
var ultimo = array.pop();
// ahora array = [1, 2], ultimo = 3
```

Veamos un ejemplo **Ej03.05h-pop.html**:

```
<!DOCTYPE html>
<html lang="es">        <!--Ej03.05h-pop.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      var array1 = new Array();
      array1 = ["Enero", "Febrero", "Marzo"];

      function ejecutar (){
         var cadena = "";
         var cadena2 = "";
         cadena = array1.pop();
         window.alert (cadena);
         cadena2 = array1.join ("\n");
         window.alert (cadena2);
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Ejecutar Programa" onclick="ejecutar ();" />
      </form>
</body> </html>
```

push(), añade un elemento al final del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez)

```
var array = [1, 2, 3];
array.push(4);
// ahora array = [1, 2, 3, 4]
```

Seguimos el ejemplo anterior: **Ej03.05i-push.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <script type="text/javascript">
   var dias = ["Lunes", "Martes", "Miércoles"];
   dias.push("Jueves");
   function mensaje ()
   {
      for (i in dias)
         alert (""+dias[i]);
   }
  </script>
</head>
<body>
   <form name="form1">
      <input type="button" value="Mostrar Mensaje" onclick="mensaje ();" />
   </form>
</body> </html>
```

shift(), elimina el primer elemento del array y lo devuelve. El array original se ve modificado y su longitud disminuida en 1 elemento.

```
var array = [1, 2, 3];
var primero = array.shift();
// ahora array = [2, 3], primero = 1
```

unshift(), añade un elemento al principio del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez)

```
var array = [1, 2, 3];
array.unshift(0);
// ahora array = [0, 1, 2, 3]
```

reverse(), modifica un array colocando sus elementos en el orden inverso a su posición original:

```
var array = [1, 2, 3];
array.reverse();
// ahora array = [3, 2, 1]
```

Ejercicio3 Especial

Recuperar el ejercicio3 con los meses del año y mostrar los meses al revés, usando las funciones reverse() y empleando for...in.

3.5.3 Funciones útiles para números

A continuación se muestran algunas de las funciones y propiedades más útiles para el manejo de números.

NaN, (del inglés, "Not a Number") JavaScript emplea el valor NaN para indicar un valor numérico no definido (por ejemplo, la división 0/0).

```
var numero1 = 0;
var numero2 = 0;
alert(numero1/numero2); // se muestra el valor NaN
```

isNaN(), permite proteger a la aplicación de posibles valores numéricos no definidos

```
var numero1 = 0;
var numero2 = 0;
if(isNaN(numero1/numero2)) {
  alert("La división no está definida para los números indicados");
}
else {
  alert("La división es igual a => " + numero1/numero2);
}
```

Infinity, hace referencia a un valor numérico infinito y positivo (también existe el valor –Infinity para los infinitos negativos)

```
var numero1 = 10;
var numero2 = 0;
alert(numero1/numero2); // se muestra el valor Infinity
```

toFixed(digitos), devuelve el número original con tantos decimales como los indicados por el parámetro digitos y realiza los redondeos necesarios. Se trata de una función muy útil por ejemplo para mostrar precios.

```
var numero1 = 4564.34567;
numero1.toFixed(2); // 4564.35
numero1.toFixed(6); // 4564.345670
numero1.toFixed(); // 4564
```

4 PROGRAMACIÓN AVANZADA

Las estructuras de control, los operadores y todas las utilidades propias de JavaScript que se han visto en los capítulos anteriores, permiten crear scripts sencillos y de mediana complejidad.

Sin embargo, para las aplicaciones más complejas son necesarios otros elementos como las funciones y otras estructuras de control más avanzadas, que se describen en este capítulo.

4.1 Funciones

Cuando se desarrolla una aplicación compleja, es muy habitual utilizar una y otra vez las mismas instrucciones. Un script para una tienda de comercio electrónico por ejemplo, tiene que calcular el precio total de los productos varias veces, para añadir los impuestos y los gastos de envío.

Cuando una serie de instrucciones se repiten una y otra vez, se complica demasiado el código fuente de la aplicación, ya que:

- El código de la aplicación es mucho más largo porque muchas instrucciones están repetidas.
- Si se quiere modificar alguna de las instrucciones repetidas, se deben hacer tantas modificaciones como veces se haya escrito esa instrucción, lo que se convierte en un trabajo muy pesado y muy propenso a cometer errores.

Las funciones son la solución a todos estos problemas, tanto en JavaScript como en el resto de lenguajes de programación. Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente.

En el siguiente ejemplo, las instrucciones que suman los dos números y muestran un mensaje con el resultado se repiten una y otra vez:

```
var resultado;

var numero1 = 3;
var numero2 = 5;

// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);

numero1 = 10;
numero2 = 7;

// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);

numero1 = 5;
numero2 = 8;

// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);
...
```

Aunque es un ejemplo muy sencillo, parece evidente que repetir las mismas instrucciones a lo largo de todo el código no es algo recomendable. La solución que proponen las funciones consiste en extraer las instrucciones que se repiten y sustituirlas por una instrucción del tipo "en este punto, se ejecutan las instrucciones que se han extraído":

```
var resultado;

var numero1 = 3;
var numero2 = 5;

/* En este punto, se llama a la función que suma
    2 números y muestra el resultado */

numero1 = 10;
numero2 = 7;

/* En este punto, se llama a la función que suma
    2 números y muestra el resultado */

numero1 = 5;
numero2 = 8;

/* En este punto, se llama a la función que suma
    2 números y muestra el resultado */
...
```

Para que la solución del ejemplo anterior sea válida, las instrucciones comunes se tienen que agrupar en una función a la que se le puedan indicar los números que debe sumar antes de mostrar el mensaje.

Por lo tanto, en primer lugar se debe crear la función básica con las instrucciones comunes. Las funciones en JavaScript se definen mediante la palabra reservada function, seguida del nombre de la función. Su definición formal es la siguiente:

```
function nombre_funcion() {
  ...
}
```

El nombre de la función se utiliza para llamar a esa función cuando sea necesario. El concepto es el mismo que con las variables, a las que se les asigna un nombre único para poder utilizarlas dentro del código. Después del nombre de la función, se incluyen dos paréntesis cuyo significado se detalla más adelante. Por último, los símbolos { y } se utilizan para encerrar todas las instrucciones que pertenecen a la función (de forma similar a como se encierran las instrucciones en las estructuras if o for).

Volviendo al ejemplo anterior, se crea una función llamada suma_y_muestra de la siguiente forma:

```
function suma_y_muestra() {
  resultado = numero1 + numero2;
  alert("El resultado es " + resultado);
}
```

Aunque la función anterior está correctamente creada, no funciona como debería ya que le faltan los "argumentos", que se explican en la siguiente sección. Una vez creada la función, desde cualquier punto del código se puede llamar a la función para que se ejecuten sus instrucciones (además de "llamar a la función", también se suele utilizar la expresión "invocar a la función").

La llamada a la función se realiza simplemente indicando su nombre, incluyendo los paréntesis del final y el carácter ";" para terminar la instrucción:

```
var resultado;

var numero1 = 3;
var numero2 = 5;

suma_y_muestra();

numero1 = 10;
numero2 = 7;

suma_y_muestra();

numero1 = 5;
numero2 = 8;

suma_y_muestra();
...
```

El código del ejemplo anterior es mucho más eficiente que el primer código que se mostró, ya que no existen instrucciones repetidas. Las instrucciones que suman y muestran mensajes se han agrupado bajo una función, lo que permite ejecutarlas en cualquier punto del programa simplemente indicando el nombre de la función.

Lo único que le falta al ejemplo anterior para funcionar correctamente es poder indicar a la función los números que debe sumar. Cuando se necesitan pasar datos a una función, se utilizan los "argumentos", como se explica en la siguiente sección.

Veamos un ejemplo: **Ej04.01a-Funciones.html**

```
<!DOCTYPE html>
<html lang="es">        <!--Ej04.01a-Funciones.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      var sumando1;
      var sumando2;
      var resultado;

      function ejecutar (){
         sumando1 = parseInt(window.prompt (" Escriba Primer sumando: "));
         sumando2 = parseInt(window.prompt (" Escriba Segundo sumando: "));
         suma();
      }

      function suma () {
         resultado = sumando1 + sumando2;
         window.alert (" La suma es: " +resultado);
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Sumar y Mostrar" onclick="ejecutar ();" />
      </form>
  </body>
</html>
```

4.1.1.1 Argumentos y valores de retorno

Las funciones más sencillas no necesitan ninguna información para producir sus resultados. Sin embargo, la mayoría de funciones de las aplicaciones reales deben acceder al valor de algunas variables para producir sus resultados.

Las variables que necesitan las funciones se llaman argumentos. Antes de que pueda utilizarlos, la función debe indicar cuántos argumentos necesita y cuál es el nombre de cada argumento. Además, al invocar la función, se deben incluir los valores que se le van a pasar a la función. Los argumentos se indican dentro de los paréntesis que van detrás del nombre de la función y se separan con una coma (,).

Siguiendo el ejemplo anterior, la función debe indicar que necesita dos argumentos, correspondientes a los dos números que tiene que sumar:

```
function suma_y_muestra(primerNumero, segundoNumero) { ... }
```

A continuación, para utilizar el valor de los argumentos dentro de la función, se debe emplear el mismo nombre con el que se definieron los argumentos:

```
function suma_y_muestra(primerNumero, segundoNumero) { ... }
  var resultado = primerNumero + segundoNumero;
  alert("El resultado es " + resultado);            }
```

Dentro de la función, el valor de la variable primerNumero será igual al primer valor que se le pase a la función y el valor de la variable segundoNumero será igual al segundo valor que se le pasa. Para pasar valores a la función, se incluyen dentro de los paréntesis utilizados al llamar a la función:

```
// Definición de la función
function suma_y_muestra(primerNumero, segundoNumero) { ... }
  var resultado = primerNumero + segundoNumero;
  alert("El resultado es " + resultado);            }

// Declaración de las variables
var numero1 = 3;
var numero2 = 5;

// Llamada a la función
suma_y_muestra(numero1, numero2);
```

En el código anterior, se debe tener en cuenta que:

- Aunque casi siempre se utilizan variables para pasar los datos a la función, se podría haber utilizado directamente el valor de esas variables: suma_y_muestra(3, 5);
- El número de argumentos que se pasa a una función debería ser el mismo que el número de argumentos que ha indicado la función. No obstante, JavaScript no muestra ningún error si se pasan más o menos argumentos de los necesarios (cosa poco recomendable).
- El orden de los argumentos es fundamental, ya que el primer dato que se indica en la llamada, será el primer valor que espera la función; el segundo valor indicado en la llamada, es el segundo valor que espera la función y así sucesivamente.
- Se puede utilizar un número ilimitado de argumentos, aunque si su número es muy grande, se complica en exceso la llamada a la función.
- No es obligatorio que coincida el nombre de los argumentos que utiliza la función y el nombre de los argumentos que se le pasan. En el ejemplo anterior, los argumentos que se pasan son numero1 y numero2 y los argumentos que utiliza la función son primerNumero y segundoNumero.

A continuación se muestra otro ejemplo de una función que calcula el precio total de un producto a partir de su precio básico:

```
// Definición de la función
function calculaPrecioTotal(precio) {
  var impuestos = 1.16;
  var gastosEnvio = 10;
  var precioTotal = ( precio * impuestos ) + gastosEnvio;
}

// Llamada a la función
calculaPrecioTotal(23.34);
```

La función anterior toma como argumento una variable llamada precio y le suma los impuestos y los gastos de envío para obtener el precio total. Al llamar a la función, se pasa directamente el valor del precio básico mediante el número 23.34.

No obstante, el código anterior no es demasiado útil, ya que lo ideal sería que la función pudiera devolver el resultado obtenido para guardarlo en otra variable y poder seguir trabajando con este precio total:

```
function calculaPrecioTotal(precio) {
  var impuestos = 1.16;
  var gastosEnvio = 10;
  var precioTotal = ( precio * impuestos ) + gastosEnvio;
}

// El valor devuelto por la función, se guarda en una variable
var precioTotal = calculaPrecioTotal(23.34);

// Seguir trabajando con la variable "precioTotal"
```

Afortunadamente, las funciones no solamente puede recibir variables y datos, sino que también pueden devolver los valores que han calculado. Para devolver valores dentro de una función, se utiliza la palabra reservada return. Aunque las funciones pueden devolver valores de cualquier tipo, solamente pueden devolver un valor cada vez que se ejecutan.

```
function calculaPrecioTotal(precio) {
  var impuestos = 1.16;
  var gastosEnvio = 10;
  var precioTotal = ( precio * impuestos ) + gastosEnvio;
  return precioTotal;
}

var precioTotal = calculaPrecioTotal(23.34);

// Seguir trabajando con la variable "precioTotal"
```

Para que la función devuelva un valor, solamente es necesario escribir la palabra reservada return junto con el nombre de la variable que se quiere devolver. En el ejemplo anterior, la ejecución de la función llega a la instrucción return precioTotal; y en ese momento, devuelve el valor que contenga la variable precioTotal.

Como la función devuelve un valor, en el punto en el que se realiza la llamada, debe indicarse el nombre de una variable en el que se guarda el valor devuelto:

```
var precioTotal = calculaPrecioTotal(23.34);
```

Si no se indica el nombre de ninguna variable, JavaScript no muestra ningún error y el valor devuelto por la función simplemente se pierde y por tanto, no se utilizará en el resto del programa. En este caso, tampoco es obligatorio que el nombre de la variable devuelta por la función coincida con el nombre de la variable en la que se va a almacenar ese valor.

Si la función llega a una instrucción de tipo return, se devuelve el valor indicado y finaliza la ejecución de la función. Por tanto, todas las instrucciones que se incluyen después de un return se ignoran y por ese motivo la instrucción return suele ser la última de la mayoría de funciones.

Para que el ejemplo anterior sea más completo, se puede añadir otro argumento a la función que indique el porcentaje de impuestos que se debe añadir al precio del producto. Evidentemente, el nuevo argumento se debe añadir tanto a la definición de la función como a su llamada:

```
function calculaPrecioTotal(precio, porcentajeImpuestos) {
  var gastosEnvio = 10;
  var precioConImpuestos = (1 + porcentajeImpuestos/100) * precio;
  var precioTotal = precioConImpuestos + gastosEnvio;
  return precioTotal;
}

var precioTotal = calculaPrecioTotal(23.34, 16);
var otroPrecioTotal = calculaPrecioTotal(15.20, 4);
```

Para terminar de completar el ejercicio anterior, se puede rendondear a dos decimales el precio total devuelto por la función:

```
function calculaPrecioTotal(precio, porcentajeImpuestos) {
  var gastosEnvio = 10;
  var precioConImpuestos = (1 + porcentajeImpuestos/100) * precio;
  var precioTotal = precioConImpuestos + gastosEnvio;
  return precioTotal.toFixed(2);
}

var precioTotal = calculaPrecioTotal(23.34, 16);
```

Veamos un ejemplo a partir de los anteriores: **Ej04.01b-Parametros_opcionA.html**

```
<!DOCTYPE html>
<html lang="es">        <!--Ej04.01b-Parametros_opcionA.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">

      function ejecutar (){
         var base = parseInt (window.prompt (" Escriba la Base del Rectángulo"));
         var altura = parseInt (window.prompt (" Escriba la altura del Rectángulo"));
         areaRectangulo (base, altura);
      }

      function areaRectangulo(baseRectangulo, alturaRectangulo) {
         var area;
         area = baseRectangulo * alturaRectangulo;
         window.alert (" El Área es: " + area);
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Sumar y Mostrar" onclick="ejecutar ();" />
      </form>
  </body>
</html>
```

Ahora el mismo ejemplo pero algo mas complicado usando return (omito código):
Ej04.01b-Parametros_opcionA.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej04.01b-Parametros_opcionB.html-->
...
      function ejecutar (){
         var resultado;
         var base = parseInt (window.prompt (" Escriba la Base del Rectángulo"));
         var altura = parseInt (window.prompt (" Escriba la altura del Rectángulo"));
         resultado = areaRectangulo (base, altura);
         window.alert (" El Área es: " + resultado);
      }
      function areaRectangulo(baseRectangulo, alturaRectangulo) {
         var area;
         area = baseRectangulo * alturaRectangulo;
         return area;
      }
...
</html>
```

Por último os paso la forma de sacar la raiz cuadrada, **Ej04.01b-RaizCuadrada.html**:
NOTA: Para sacar el ² hay que pulsar [MAYUS] + [^] y se activar el modo superindice:

```
<!DOCTYPE html>
...
function ejecutar (){
         var valor = window.prompt (" Escriba valor: ");
         var raiz = Math.sqrt(valor);
         window.alert (" La raiz ² es: " + raiz);
      }
```

Ejercicio 08

Escribir el código de una función a la que se pasa como parámetro un número entero y devuelve como resultado una cadena de texto que indica si el número es par o impar. Mostrar por pantalla el resultado devuelto por la función.

Ejercicio 09

Definir una función que muestre información sobre una cadena de texto que se le pasa como argumento. A partir de la cadena que se le pasa, la función determina si esa cadena está formada sólo por mayúsculas, sólo por minúsculas o por una mezcla de ambas.

Ejercicio 10

Definir una función que determine si la cadena de texto que se le pasa como parámetro es un palíndromo, es decir, si se lee de la misma forma desde la izquierda y desde la derecha. Ejemplo de palíndromo complejo: "La ruta nos aporto otro paso natural".

Ejercicio 10 Especial

1. Crear una función que muestre el resultado de una Ecuación de segundo grado.
   Mas información: http://es.wikipedia.org/wiki/Ecuaci%C3%B3n_de_segundo_grado
   Usar la función matemática: Math.sqrt (num).
2. Crear una aplicación que solicite la elección de una opción de un menú:
   - Mutiplicar
   - Dividir
   - Resto
   - Potencia
   - Raiz Cuadrada
3. Desarrollar las funciones especificas de cada opción.

4.2 Ámbito de las variables

El ámbito de una variable (llamado "scope" en inglés) es la zona del programa en la que se define la variable. JavaScript define dos ámbitos para las variables: global y local.

El siguiente ejemplo ilustra el comportamiento de los ámbitos:

```
function creaMensaje() {
  var mensaje = "Mensaje de prueba";
}
creaMensaje();
alert(mensaje);
```

El ejemplo anterior define en primer lugar una función llamada creaMensaje que crea una variable llamada mensaje. A continuación, se ejecuta la función mediante la llamada creaMensaje(); y seguidamente, se muestra mediante la función alert() el valor de una variable llamada mensaje.

Sin embargo, al ejecutar el código anterior no se muestra ningún mensaje por pantalla. La razón es que la variable mensaje se ha definido dentro de la función creaMensaje() y por tanto, es una variable local que solamente está definida dentro de la función.

Cualquier instrucción que se encuentre dentro de la función puede hacer uso de esa variable, pero todas las instrucciones que se encuentren en otras funciones o fuera de cualquier función no tendrán definida la variable mensaje.

De esta forma, para mostrar el mensaje en el código anterior, la función alert() debe llamarse desde dentro de la función creaMensaje():

```
function creaMensaje() {
  var mensaje = "Mensaje de prueba";
  alert(mensaje);
}
creaMensaje();
```

Además de variables locales, también existe el concepto de variable global, que está definida en cualquier punto del programa (incluso dentro de cualquier función).

```
var mensaje = "Mensaje de prueba";

function muestraMensaje() {
  alert(mensaje);
}
```

El código anterior es el ejemplo inverso al mostrado anteriormente. Dentro de la función muestraMensaje() se quiere hacer uso de una variable llamada mensaje y que no ha sido definida dentro de la propia función. Sin embargo, si se ejecuta el código anterior, sí que se muestra el mensaje definido por la variable mensaje.

El motivo es que en el código JavaScript anterior, la variable mensaje se ha definido fuera de cualquier función. Este tipo de variables automáticamente se transforman en variables globales y están disponibles en cualquier punto del programa (incluso dentro de cualquier función).

De esta forma, aunque en el interior de la función no se ha definido ninguna variable llamada mensaje, la variable global creada anteriormente permite que la instrucción alert() dentro de la función muestre el mensaje correctamente.

Si una variable se declara fuera de cualquier función, automáticamente se transforma en variable global independientemente de si se define utilizando la palabra reservada var o no. Sin embargo, las variables definidas dentro de una función pueden ser globales o locales.

Si en el interior de una función, las variables se declaran mediante var se consideran locales y las variables que no se han declarado mediante var, se transforman automáticamente en variables globales.

Por lo tanto, se puede rehacer el código del primer ejemplo para que muestre el mensaje correctamente. Para ello, simplemente se debe definir la variable dentro de la función sin la palabra reservada var, para que se transforme en una variable global:

```
function creaMensaje() {
  mensaje = "Mensaje de prueba";
}

creaMensaje();
alert(mensaje);
```

¿Qué sucede si una función define una variable local con el mismo nombre que una variable global que ya existe? En este caso, las variables locales prevalecen sobre las globales, pero sólo dentro de la función:

```
var mensaje = "gana la de fuera";

function muestraMensaje() {
  var mensaje = "gana la de dentro";
  alert(mensaje);
}

alert(mensaje);
muestraMensaje();
alert(mensaje);
```

El código anterior muestra por pantalla los siguientes mensajes:
gana la de fuera
gana la de dentro
gana la de fuera

Dentro de la función, la variable local llamada mensaje tiene más prioridad que la variable global del mismo nombre, pero solamente dentro de la función.

¿Qué sucede si dentro de una función se define una variable global con el mismo nombre que otra variable global que ya existe? En este otro caso, la variable global definida dentro de la función simplemente modifica el valor de la variable global definida anteriormente:

```
var mensaje = "gana la de fuera";
function muestraMensaje() {
  mensaje = "gana la de dentro";
  alert(mensaje);
}

alert(mensaje);
muestraMensaje();
alert(mensaje);
```

En este caso, los mensajes mostrados son:

gana la de fuera
gana la de dentro
gana la de dentro

La recomendación general es definir como variables locales todas las variables que sean de uso exclusivo para realizar las tareas encargadas a cada función. Las variables globales se utilizan para compartir variables entre funciones de forma sencilla.

4.3 Sentencias break y continue

La estructura de control for es muy sencilla de utilizar, pero tiene el inconveniente de que el número de repeticiones que se realizan sólo se pueden controlar mediante las variables definidas en la zona de actualización del bucle.

Las sentencias break y continue permiten manipular el comportamiento normal de los bucles for para detener el bucle o para saltarse algunas repeticiones. Concretamente, la sentencia break permite terminar de forma abrupta un bucle y la sentencia continue permite saltarse algunas repeticiones del bucle.

El siguiente ejemplo muestra el uso de la sentencia break:

```
var cadena = "En un lugar de la Mancha de cuyo nombre no quiero acordarme...";
var letras = cadena.split("");
var resultado = "";

for(i in letras) {
  if(letras[i] == 'a') {
    break;
  }
  else {
    resultado += letras[i];
  }
}
alert(resultado);
// muestra "En un lug"
```

Veamos un ejemplo: **Ej04.03a-Break.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <script type="text/javascript">
   var veces;
    var opcion;

    function ejecutar () {
      for (veces=1; veces<=5; veces++){
          opcion = window.prompt (" Menú de Opciones:"+ "\n"+
                                  " 0 -> Salir       "+ "\n"+
                                  " 1 -> Opción 1    "+ "\n"+
                                  " 2 -> Opción 2    "+ "\n"+
                                  " Quedan "+(5-veces)+" intentos"+ "\n"+
                                  " Elije opción: "   +"\n");

          window.alert ("Has elegido la opción: "+opcion);
          if (opcion==0)
            break;
      }
    }
  </script>
</head>
<body>
   <form name="form1">
      <input type="button" value="Ver Menú" onclick="ejecutar() ();" />
   </form>
</body>
</html>
```

Si el programa llega a una instrucción de tipo break;, sale inmediatamente del bucle y continúa ejecutando el resto de instrucciones que se encuentran fuera del bucle for. En el ejemplo anterior, se recorren todas las letras de una cadena de texto y cuando se encuentra con la primera letra "a", se detiene la ejecución del bucle for.

La utilidad de break es terminar la ejecución del bucle cuando una variable toma un determinado valor o cuando se cumple alguna condición.

En ocasiones, lo que se desea es saltarse alguna repetición del bucle cuando se dan algunas condiciones. Siguiendo con el ejemplo anterior, ahora se desea que el texto de salida elimine todas las letras "a" de la cadena de texto original:

```
var cadena = "En un lugar de la Mancha de cuyo nombre no quiero acordarme...";
var letras = cadena.split("");
var resultado = "";

for(i in letras) {
  if(letras[i] == 'a') {
    continue;
  }
  else {
    resultado += letras[i];
  }
}
alert(resultado);
// muestra "En un lugr de l Mnch de cuyo nombre no quiero cordrme..."
```

En este caso, cuando se encuentra una letra "a" no se termina el bucle, sino que no se ejecutan las instrucciones de esa repetición y se pasa directamente a la siguiente repetición del bucle for.

La utilidad de continue es que permite utilizar el bucle for para filtrar los resultados en función de algunas condiciones o cuando el valor de alguna variable coincide con un valor determinado.

Veamos un ejemplo: **Ej04.03b-Continue.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <script type="text/javascript">
   var indice;
    var opcion;
    var omitir;
    var resultado;

    function ejecutar () {
      opcion = window.prompt (" Elija Tabla multiplicar: ")
      omitir = window.prompt (" Elija valor Tabla multiplicar a omitir: ")

      for (indice=1; indice<=10; indice++)
      {
        resultado = opcion * indice;

         if (indice == omitir)
            continue;

         window.alert (opcion + " * " + indice + "= " +resultado);
      }
    }
  </script>
</head>
<body>
   <form name="form1">
      <input type="button" value="Tabla Multiplicar" onclick="ejecutar() ();" />
   </form>
</body>
</html>
```

4.4 Otras estructuras de control

Las estructuras de control de flujo que se han visto (if, else, for) y las sentencias que modifican su comportamiento (break, continue) no son suficientes para realizar algunas tareas complejas y otro tipo de repeticiones. Por ese motivo, JavaScript proporciona otras estructuras de control de flujo diferentes y en algunos casos más eficientes.

4.4.1 Estructura while

La estructura while permite crear bucles que se ejecutan ninguna o más veces, dependiendo de la condición indicada. Su definición formal es:

```
while(condicion) {
  ...
}
```

El funcionamiento del bucle while se resume en: "mientras se cumpla la condición indicada, repite indefinidamente las instrucciones incluidas dentro del bucle".

Si la condición no se cumple ni siquiera la primera vez, el bucle no se ejecuta. Si la condición se cumple, se ejecutan las instrucciones una vez y se vuelve a comprobar la condición. Si se sigue cumpliendo la condición, se vuelve a ejecutar el bucle y así se continúa hasta que la condición no se cumpla.

Evidentemente, las variables que controlan la condición deben modificarse dentro del propio bucle, ya que de otra forma, la condición se cumpliría siempre y el bucle while se repetiría indefinidamente.

El siguiente ejemplo utiliza el bucle while para sumar todos los números menores o iguales que otro número:

```
var resultado = 0;
var numero = 100;
var i = 0;

while(i <= numero) {
  resultado += i;
  i++;
}

alert(resultado);
```

El programa debe sumar todos los números menores o igual que otro dado. Por ejemplo si el número es 5, se debe calcular: 1 + 2 + 3 + 4 + 5 = 15

Este tipo de condiciones "suma números mientras sean menores o iguales que otro número dado") se resuelven muy fácilmente con los bucles tipo while, aunque también se podían resolver con bucles de tipo for.

En el ejemplo anterior, mientras se cumpla la condición, es decir, mientras que la variable i sea menor o igual que la variable numero, se ejecutan las instrucciones del bucle.

Dentro del bucle se suma el valor de la variable i al resultado total (variable resultado) y se actualiza el valor de la variable i, que es la que controla la condición del bucle. Si no se actualiza el valor de la variable i, la ejecución del bucle continua infinitamente o hasta que el navegador permita al usuario detener el script.

Veamos un ejemplo (omito algunas partes): **Ej04.04a-While.html**

```
<!DOCTYPE html>
...
  <script type="text/javascript">
   var veces=0;
   var opcion;
   function ejecutar () {
      while (opcion!=0){
          veces++;
          opcion = window.prompt (" Menú de Opciones:"+ "\n"+
                                  " PANTALLA " +veces + "\n"+
                                  " 0 -> Salir       "+ "\n"+
                                  " 1 -> Opción 1    "+ "\n"+
                                  " 2 -> Opción 2    "+ "\n"+
                                  " Elije opción: "   + "\n");

          window.alert (" Has elegido la opción: "+opcion);
      }
      window.alert (" Gracias por usar este programa! ");
    }
  </script>
</head>
<body>
   <form name="form1">
      <input type="button" value="Ver Menú" onclick="ejecutar() ();" />
   </form>
</body>
</html>
```

4.4.2 Estructura do...while

El bucle de tipo do...while es muy similar al bucle while, salvo que en este caso siempre se ejecutan las instrucciones del bucle al menos la primera vez. Su definición formal es:

```
do {
  ...
} while(condicion);
```

De esta forma, como la condición se comprueba después de cada repetición, la primera vez siempre se ejecutan las instrucciones del bucle. Es importante no olvidar que después del while() se debe añadir el carácter ; (al contrario de lo que sucede con el bucle while simple).

Utilizando este bucle se puede calcular fácilmente el factorial de un número:
(os lo dejo hecho ;D): **Ej04.04b-DoWhile.html**

```
<!DOCTYPE html>
...
  <script type="text/javascript">
    var resultado = 1;
    var numero;

    function ejecutar () {
      var numeroelegido = window.prompt (" Elija el número para el Factorial: ");
      numero = numeroelegido;
      do {
        resultado *= numero;  // resultado = resultado * numero
        numero--;
      } while(numero > 0);
      window.alert(" El Factorial de "+ numeroelegido + " es : "+resultado);
   }
  </script>
</head>
<body>
   <form name="form1">
      <input type="button" value="Ver Factorial" onclick="ejecutar() ();" />
   </form>
</body>
</html>
```

4.4.3 Estructura switch

La estructura if...else se puede utilizar para realizar comprobaciones múltiples y tomar decisiones complejas. Sin embargo, si todas las condiciones dependen siempre de la misma variable, el código JavaScript resultante es demasiado redundante:

```
if(numero == 5) {
  ...
}
else if(numero == 8) {
  ...
}
else if(numero == 20) {
  ...
}
else {
  ...
}
```

En estos casos, la estructura switch es la más eficiente, ya que está especialmente diseñada para manejar de forma sencilla múltiples condiciones sobre la misma variable. Su definición formal puede parecer compleja, aunque su uso es muy sencillo:

```
switch(variable) {
  case valor_1:
    ...
    break;
  case valor_2:
    ...
    break;
  ...
  case valor_n:
    ...
    break;
  default:
    ...
    break;
}
```

El anterior ejemplo realizado con if...else se puede rehacer mediante switch:

```
switch(numero) {
  case 5:
    ...
    break;
  case 8:
    ...
    break;
  case 20:
    ...
    break;
  default:
    ...
    break;
}
```

La estructura switch se define mediante la palabra reservada switch seguida, entre paréntesis, del nombre de la variable que se va a utilizar en las comparaciones. Como es habitual, las instrucciones que forman parte del switch se encierran entre las llaves { y }.

Dentro del switch se definen todas las comparaciones que se quieren realizar sobre el valor de la variable. Cada comparación se indica mediante la palabra reservada case seguida del valor con el que se realiza la comparación. Si el valor de la variable utilizada por switch coincide con el valor indicado por case, se ejecutan las instrucciones definidas dentro de ese case.

Normalmente, después de las instrucciones de cada case se incluye la sentencia break para terminar la ejecución del switch, aunque no es obligatorio. Las comparaciones se realizan por orden, desde el primer case hasta el último, por lo que es muy importante el orden en el que se definen los case.

¿Qué sucede si ningún valor de la variable del switch coincide con los valores definidos en los case? En este caso, se utiliza el valor default para indicar las instrucciones que se ejecutan en el caso en el que ningún case se cumpla para la variable indicada.

Aunque default es opcional, las estructuras switch suelen incluirlo para definir al menos un valor por defecto para alguna variable o para mostrar algún mensaje por pantalla.

Un ejemplo: **Ej04.04c-Switch.html**
NOTA: vamos a usar el objeto Date y, dentro de este, la función getDay.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <script type="text/javascript">

    function ejecutar () {
      var dia = new Date();
      var diahoy = dia.getDay ();
      var diasemana = "";

      switch (diahoy) {
        case 1: diasemana = "lunes";      break;
        case 2: diasemana = "martes";     break;
        case 3: diasemana = "miercoles";  break;
        case 4: diasemana = "jueves";     break;
        case 5: diasemana = "viernes";    break;
        case 6: diasemana = "sabado";     break;
        case 7: diasemana = "domingo";    break;
      }

      window.alert ("hoy es: "+diasemana);
    }
  </script>
</head>
<body>
   <form name="form1">
      <input type="button" value="Dia Semana" onclick="ejecutar() ();" />
   </form>
</body>
</html>
```

5 DOM

La creación del Document Object Model o DOM es una de las innovaciones que más ha influido en el desarrollo de las páginas web dinámicas y de las aplicaciones web más complejas.

DOM permite a los programadores web acceder y manipular las páginas XHTML como si fueran documentos XML. De hecho, DOM se diseñó originalmente para manipular de forma sencilla los documentos XML.

A pesar de sus orígenes, DOM se ha convertido en una utilidad disponible para la mayoría de lenguajes de programación (Java, PHP, JavaScript) y cuyas únicas diferencias se encuentran en la forma de implementarlo.

5.1 Árbol de nodos

Una de las tareas habituales en la programación de aplicaciones web con JavaScript consiste en la manipulación de las páginas web. De esta forma, es habitual obtener el valor almacenado por algunos elementos (por ejemplo los elementos de un formulario), crear un elemento (párrafos, , etc.) de forma dinámica y añadirlo a la página, aplicar una animación a un elemento (que aparezca/desaparezca, que se desplace, etc.).

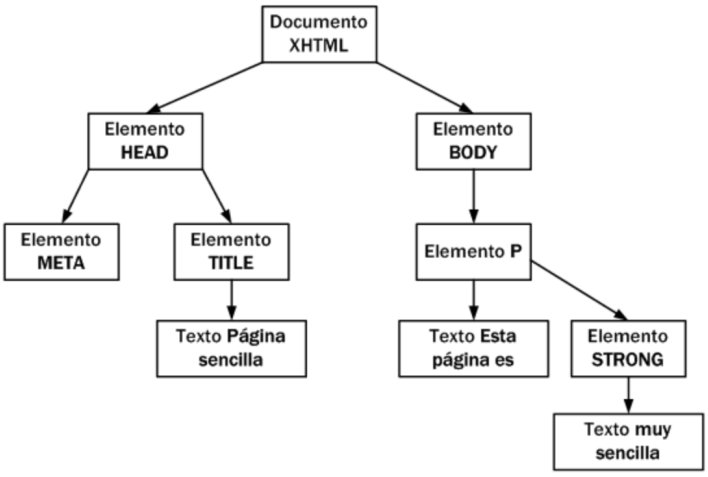
Todas estas tareas habituales son muy sencillas de realizar gracias a DOM. Sin embargo, para poder utilizar las utilidades de DOM, es necesario "transformar" la página original. Una página HTML normal no es más que una sucesión de caracteres, por lo que es un formato muy difícil de manipular. Por ello, los navegadores web transforman automáticamente todas las páginas web en una estructura más eficiente de manipular.

Esta transformación la realizan todos los navegadores de forma automática y nos permite utilizar las herramientas de DOM de forma muy sencilla. El motivo por el que se muestra el funcionamiento de esta transformación interna es que condiciona el comportamiento de DOM y por tanto, la forma en la que se manipulan las páginas.

DOM transforma todos los documentos XHTML en un conjunto de elementos llamados nodos, que están interconectados y que representan los contenidos de las páginas web y las relaciones entre ellos. Por su aspecto, la unión de todos los nodos se llama "árbol de nodos".

La siguiente página XHTML sencilla:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8" />
<title>Página Sencilla</title>
</head>
<body>
<p>Esta página es
<strong>muy sencilla</strong>
</p>
</body>
</html>
```



Se transforma en el siguiente árbol de nodos:

En el esquema anterior, cada rectángulo representa un nodo DOM y las flechas indican las relaciones entre nodos. Dentro de cada nodo, se ha incluido su tipo (que se verá más adelante) y su contenido. La raíz del árbol de nodos de cualquier página XHTML siempre es la misma: un nodo de tipo especial denominado "Documento".

A partir de ese nodo raíz, cada etiqueta XHTML se transforma en un nodo de tipo "Elemento". La conversión de etiquetas en nodos se realiza de forma jerárquica. De esta forma, del nodo raíz solamente pueden derivar los nodos HEAD y BODY. A partir de esta derivación inicial, cada etiqueta XHTML se transforma en un nodo que deriva del nodo correspondiente a su "etiqueta padre".

La transformación de las etiquetas XHTML habituales genera dos nodos: el primero es el nodo de tipo "Elemento" (correspondiente a la propia etiqueta XHTML) y el segundo es un nodo de tipo "Texto" que contiene el texto encerrado por esa etiqueta XHTML.

Así, la siguiente etiqueta XHTML:

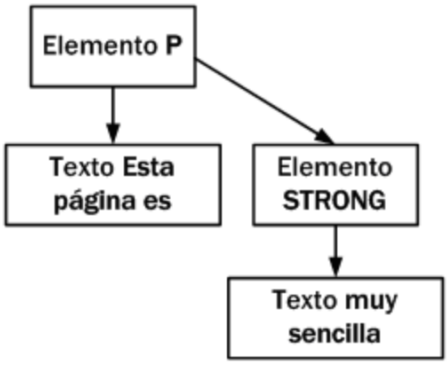
```
<title>Página sencilla</title>
```

Genera los siguientes dos nodos:

De la misma forma, la siguiente etiqueta XHTML:

```
<p>Esta página es <strong>muy sencilla</strong></p>
```

Genera los siguientes nodos:



- Nodo de tipo "Elemento" correspondiente a la etiqueta <p>.
- Nodo de tipo "Texto" con el contenido textual de la etiqueta <p>.
- Como el contenido de <p> incluye en su interior otra etiqueta XHTML, la etiqueta interior se transforma en un nodo de tipo "Elemento" que representa la etiqueta y que deriva del nodo anterior.
- El contenido de la etiqueta genera a su vez otro nodo de tipo "Texto" que deriva del nodo generado por .

La transformación automática de la página en un árbol de nodos siempre sigue las mismas reglas:

- Las etiquetas XHTML se transforman en dos nodos: el primero es la propia etiqueta y el segundo nodo es hijo del primero y consiste en el contenido textual de la etiqueta.
- Si una etiqueta XHTML se encuentra dentro de otra, se sigue el mismo procedimiento anterior, pero los nodos generados serán nodos hijo de su etiqueta padre.

Como se puede suponer, las páginas XHTML habituales producen árboles con miles de nodos. Aun así, el proceso de transformación es rápido y automático, siendo las funciones proporcionadas por DOM (que se verán más adelante) las únicas que permiten acceder a cualquier nodo de la página de forma sencilla e inmediata.

5.2 Tipos de nodos

La especificación completa de DOM define 12 tipos de nodos, aunque las páginas XHTML habituales se pueden manipular manejando solamente cuatro o cinco tipos de nodos:

- Document, nodo raíz del que derivan todos los demás nodos del árbol.
- Element, representa cada una de las etiquetas XHTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
- Attr, se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas XHTML, es decir, uno por cada par atributo=valor.
- Text, nodo que contiene el texto encerrado por una etiqueta XHTML.
- Comment, representa los comentarios incluidos en la página XHTML.

Los otros tipos de nodos existentes que no se van a considerar son DocumentType, CDataSection, DocumentFragment, Entity, EntityReference, ProcessingInstruction y Notation.

5.3 Acceso directo a los nodos

Una vez construido automáticamente el árbol completo de nodos DOM, ya es posible utilizar las funciones DOM para acceder de forma directa a cualquier nodo del árbol. Como acceder a un nodo del árbol es equivalente a acceder a "un trozo" de la página, una vez construido el árbol, ya es posible manipular de forma sencilla la página: acceder al valor de un elemento, establecer el valor de un elemento, mover un elemento de la página, crear y añadir nuevos elementos, etc.

DOM proporciona dos métodos alternativos para acceder a un nodo específico: acceso a través de sus nodos padre y acceso directo.

Las funciones que proporciona DOM para acceder a un nodo a través de sus nodos padre consisten en acceder al nodo raíz de la página y después a sus nodos hijos y a los nodos hijos de esos hijos y así sucesivamente hasta el último nodo de la rama terminada por el nodo buscado. Sin embargo, cuando se quiere acceder a un nodo específico, es mucho más rápido acceder directamente a ese nodo y no llegar hasta él descendiendo a través de todos sus nodos padre.

Por ese motivo, no se van a presentar las funciones necesarias para el acceso jerárquico de nodos y se muestran solamente las que permiten acceder de forma directa a los nodos.

Por último, es importante recordar que el acceso a los nodos, su modificación y su eliminación solamente es posible cuando el árbol DOM ha sido construido completamente, es decir, después de que la página XHTML se cargue por completo. Más adelante se verá cómo asegurar que un código JavaScript solamente se ejecute cuando el navegador ha cargado entera la página XHTML.

5.3.1 GetElementsByTagName()

Como sucede con todas las funciones que proporciona DOM, la función getElementsByTagName() tiene un nombre muy largo, pero que lo hace autoexplicativo.

La función getElementsByTagName(nombreEtiqueta) obtiene todos los elementos de la página XHTML cuya etiqueta sea igual que el parámetro que se le pasa a la función.

El siguiente ejemplo muestra cómo obtener todos los párrafos de una página XHTML:

```
var parrafos = document.getElementsByTagName("p");
```

El valor que se indica delante del nombre de la función (en este caso, document) es el nodo a partir del cual se realiza la búsqueda de los elementos. En este caso, como se quieren obtener todos los párrafos de la página, se utiliza el valor document como punto de partida de la búsqueda.

El valor que devuelve la función es un array con todos los nodos que cumplen la condición de que su etiqueta coincide con el parámetro proporcionado. El valor devuelto es un array de nodos DOM, no un array de cadenas de texto o un array de objetos normales. Por lo tanto, se debe procesar cada valor del array de la forma que se muestra en las siguientes secciones.

De este modo, se puede obtener el primer párrafo de la página de la siguiente manera:

```
var primerParrafo = parrafos[0].innerHTML;
```

NOTA: como se puede comprobar hemos usado el método innerHTML para escribir código HTML (en este caso el contenido del párrafo).

De la misma forma, se podrían recorrer todos los párrafos de la página con el siguiente código:

```
for(var i=0; i<parrafos.length; i++) {
  var parrafo = parrafos[i].innerHTML;
}
```

Veamos un ejemplo completo: **Ej05.03a-GetElementsByTagName.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <script type="text/javascript">
    var parrafos;

    function ejecutar () {
      parrafos = document.getElementsByTagName("p");

      for(var i=0; i<parrafos.length; i++) {
        var parrafo = parrafos[i].innerHTML;
        window.alert (parrafo);
      }
    }

  </script>
</head>
<body>
   <p>Hola k ase</p>
    <p>Todos los redes sociales</p>

    <form name="form1">
      <input type="button" value="Ver Mensajes" onclick="ejecutar() ();" />
   </form>
</body>
</html>
```

La función getElementsByTagName() se puede aplicar de forma recursiva sobre cada uno de los nodos devueltos por la función. En el siguiente ejemplo, se obtienen todos los enlaces del primer párrafo de la página:

```
var parrafos = document.getElementsByTagName("p");
var primerParrafo = parrafos[0];
var enlaces = primerParrafo.getElementsByTagName("a");
```

5.3.2 getElementsByName()

La función getElementsByName() es similar a la anterior, pero en este caso se buscan los elementos cuyo atributo name sea igual al parámetro proporcionado. En el siguiente ejemplo, se obtiene directamente el único párrafo con el nombre indicado:

```
var parrafoEspecial = document.getElementsByName("especial");
<p name="prueba">...</p>
<p name="especial">...</p>
<p>...</p>
```

Normalmente el atributo name es único para los elementos HTML que lo definen, por lo que es un método muy práctico para acceder directamente al nodo deseado. En el caso de los elementos HTML radiobutton, el atributo name es común a todos los radiobutton que están relacionados, por lo que la función devuelve una colección de elementos.

Veamos un ejemplo similar al anterior: **Ej05.03b-getElementsByName.html**

```
<!DOCTYPE html> <html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <script type="text/javascript">
    var parrafos;

    function ejecutar () {
      parrafos = document.getElementsByName("especial");

      for(var i=0; i<parrafos.length; i++) {
        var parrafo = parrafos[i].innerHTML;
        window.alert (parrafo);
      }
    }
 </script>
</head>
<body>
   <p>Un párrafo normal</p>
   <p name="especial">Hola k ase</p>
   <p name="especial">Todos los redes sociales</p>
   <button onclick="ejecutar();">
      Mostrar Párrafos Especiales</button>
</body>
</html>
```

5.3.3 getElementById()

La función getElementById() es la más utilizada cuando se desarrollan aplicaciones web dinámicas. Se trata de la función preferida para acceder directamente a un nodo y poder leer o modificar sus propiedades.

La función getElementById() devuelve el elemento XHTML cuyo atributo id coincide con el parámetro indicado en la función. Como el atributo id debe ser único para cada elemento de una misma página, la función devuelve únicamente el nodo deseado.

Ej05.03c-getElementById.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej05.03b-getElementById.html -->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      function ejecutar() {
         var parrafoElegido = window.prompt (" Seleccione Párrafo (1,2,3): ");
         var parrafo = document.getElementById (parrafoElegido);
         document.write (parrafo.innerHTML);        // Hay que colocar el innerHTML
      }
   </script>
</head>
  <body>
   <p id="1"> Primer Párrafo</p>
   <p id="2"> Segundo Párrafo</p>
   <p id="3"> Tercer Párrafo</p>
   <input type="button" value="Ver párrafo" onclick="ejecutar();" />
  </body>
</html>
```

La función getElementById() es tan importante y tan utilizada en todas las aplicaciones web, que casi todos los ejemplos y ejercicios que siguen la utilizan constantemente.

5.4 Creación y eliminación de nodos

Acceder a los nodos y a sus propiedades (que se verá más adelante) es sólo una parte de las manipulaciones habituales en las páginas. Las otras operaciones habituales son las de crear y eliminar nodos del árbol DOM, es decir, crear y eliminar "trozos" de la página web.

5.4.1 Creación de elementos XHTML simples

Como se ha visto, un elemento XHTML sencillo, como por ejemplo un párrafo, genera dos nodos: el primer nodo es de tipo Element y representa la etiqueta <p> y el segundo nodo es de tipo Text y representa el contenido textual de la etiqueta <p>.

Por este motivo, crear y añadir a la página un nuevo elemento XHTML sencillo consta de cuatro pasos diferentes:

- Creación de un nodo de tipo Element que represente al elemento.
- Creación de un nodo de tipo Text que represente el contenido del elemento.
- Añadir el nodo Text como nodo hijo del nodo Element.
- Añadir el nodo Element a la página, en forma de nodo hijo del nodo correspondiente al lugar en el que se quiere insertar el elemento.

De este modo, si se quiere añadir un párrafo simple al final de una página XHTML, es necesario incluir el siguiente código JavaScript:

Ej05.04a-InsertarNodo.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej05.04a-InsertarNodo.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      function ejecutar() {
         var texto = window.prompt (" Escriba el texto del párrafo: ");

         var etiqueta = document.createElement("p");
         var contenido = document.createTextNode (texto);
         etiqueta.appendChild (contenido);
         document.body.appendChild(etiqueta);
      }
   </script>
</head>
  <body>
   <p> Primer Párrafo</p>
   <p> Segundo Párrafo</p>
   <input type="button" value="Insertar párrafo" onclick="ejecutar();" />
  </body>
</html>
```

El proceso de creación de nuevos nodos puede llegar a ser tedioso, ya que implica la utilización de tres funciones DOM:

- **createElement(etiqueta):** crea un nodo de tipo Element que representa al elemento XHTML cuya etiqueta se pasa como parámetro.
- **createTextNode(contenido):** crea un nodo de tipo Text que almacena el contenido textual de los elementos XHTML.
- **nodoPadre.appendChild(nodoHijo):** añade un nodo como hijo de otro nodo. Se debe utilizar al menos dos veces con los nodos habituales: en primer lugar se añade el nodo Text como hijo del nodo Element y a continuación se añade el nodo Element como hijo de algún nodo de la página.

Veamos otro ejemplo pero metiendo una tabla de multiplicar: **Ej05.04b-NodoMultiplicar.html**

```
<!DOCTYPE html>
<html lang="es">        <!--Ej05.01d-NodoMultiplicar.html -->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      function ejecutar() {
         var num = prompt (" Elija la tabla de Multiplicar: ");

         // Defino la etiqueta que se va a presentar
         var etiqueta = document.createElement ("p");
         // Defino el contenido de la etiqueta (aquí, la tabla de Multiplicar)
         var textoNodo = document.createTextNode (multiplicar(num));

         // Completo el nodo agregando el contenido a la etiqueta
         etiqueta.appendChild(textoNodo);    // El nodo completo
         // Añado el nodo (etiqueta+contenido) al parrafo
         parrafo.appendChild(etiqueta);
      }

      function multiplicar(tabla) {
         var cadena = "";
         var i;
         for (i=1; i<=10; i++) {
            cadena = cadena + i + " x " + tabla + " = " + (tabla * i) + "; ";
         }
         return cadena;
      }

      // Esta parte del ejemplo vale para el siguiene apartado...
      function borrar() {
         parrafo.removeChild (parrafo.lastChild);
      }
   </script>
</head>
  <body>
      <p id="parrafo">
         <!--Aquí dentro irá cada tabla como un párrafo diferente...-->
      </p>
      <form name="form1">
         <input type="button" value="Multiplicar" onclick="ejecutar();" />
         <input type="button" value="Borrar" onclick="borrar();" />
      </form>
  </body>
</html>
```

5.4.2 Eliminación de nodos

Afortunadamente, eliminar un nodo del árbol DOM de la página es mucho más sencillo que añadirlo. En este caso, solamente es necesario utilizar la función removeChild():

```
var parrafo = document.getElementById("provisional");
parrafo.parentNode.removeChild(parrafo);
<p id="provisional">...</p>
```

La función removeChild() requiere como parámetro el nodo que se va a eliminar. Además, esta función debe ser invocada desde el elemento padre de ese nodo que se quiere eliminar. La forma más segura y rápida de acceder al nodo padre de un elemento es mediante la propiedad nodoHijo.parentNode.

Así, para eliminar un nodo de una página XHTML se invoca a la función removeChild() desde el valor parentNode del nodo que se quiere eliminar. Cuando se elimina un nodo, también se eliminan automáticamente todos los nodos hijos que tenga, por lo que no es necesario borrar manualmente cada nodo hijo.

Veamos un ejemplo: **Ej05.04c-BorrarNodo.html**

```
<!DOCTYPE html>
<html lang="es">        <!--Ej05.04c-BorrarNodo.html -->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      function ejecutar() {
         // Obtenemos el elemento
         var lista = document.getElementById("li");

         // Obtenemos el padre de dicho elemento
         var padre = lista.parentNode;

         // Eliminamos el hijo (lista) del elemento padre
         padre.removeChild(lista);
      }
   </script>
</head>
  <body>
      <ul>
         <li id="li">Elemento1</li>
         <li id="li">Elemento2</li>
         <li id="li">Elemento3</li>
      </ul>
      <form name="form1">
         <input type="button" value="Borrar Párrafo" onclick="ejecutar ();" />
      </form>
  </body>
</html>
```

Podemos incluir en el mismo código la opción de añadir elementos y quitarlos. Eso si, en este último caso, quitando desde el final:

Ej05.04d-InsertarBorrarNodo.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej05.04d-InsertarBorrarNodo.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      function insertar() {
         var textoNodo = window.prompt (" Escriba el texto del nuevo elemento: ");

         var etiqueta = document.createElement("li");
         var contenido = document.createTextNode (textoNodo);
         etiqueta.appendChild (contenido);

         miLista.appendChild(etiqueta);
         //document.getElementById("miLista").appendChild(etiqueta);
      }
      function borrar() {
         miLista.removeChild (miLista.lastChild);
      }
   </script>
</head>
  <body>
   <ul id="miLista">
      <li> Primer elemento de la lista</li>
   </ul>
   <input type="button" value="Añadir Elemento" onclick="insertar();" />
   <input type="button" value="Borrar Elemento" onclick="borrar();" />
  </body>
</html>
```

En este caso hemos usado lastChild para quitar el último elemento de la lista (y no el primero que es lo que haríamos si usásemos sólo removeChild o dentro del argumento, miLista.firstChild).

5.5 Acceso directo a los atributos XHTML

Una vez que se ha accedido a un nodo, el siguiente paso natural consiste en acceder y/o modificar sus atributos y propiedades. Mediante DOM, es posible acceder de forma sencilla a todos los atributos XHTML y todas las propiedades CSS de cualquier elemento de la página.

Los atributos XHTML de los elementos de la página se transforman automáticamente en propiedades de los nodos. Para acceder a su valor, simplemente se indica el nombre del atributo XHTML detrás del nombre del nodo.
El siguiente ejemplo obtiene de forma directa todos los enlaces de la página:

Ej05.05a-AccesoAtributos.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej05.05a-AccesoAtributos.html -->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <script language="JavaScript">
    function ejecutar() {
      var enlaces  = document.getElementsByTagName("a");
      var cadena = "Los enlaces de la página son: \n";
      var i;
      for (i=0; i<enlaces.length; i++)
        cadena += enlaces[i].href + "\n";
      alert (cadena);
    }
  </script>
</head>
<body>
  <p><a href="http://www.google.es">Google</a></p>
  <p><a href="http://www.yahoo.es">Yahoo</a></p>
  <input type="button" value="Ver enlaces" onClick="ejecutar();">
</body>
</html>
```

Las propiedades CSS no son tan fáciles de obtener como los atributos XHTML. Para obtener el valor de cualquier propiedad CSS del nodo, se debe utilizar el atributo style. El siguiente ejemplo obtiene el valor de la propiedad margin de la imagen:

```
var imagen = document.getElementById("imagen");
alert(imagen.style.margin);
<img id="imagen" style="margin:0; border:0;" src="logo.png" />
```

Aunque el funcionamiento es homogéneo entre distintos navegadores, los resultados no son exactamente iguales.

Si el nombre de una propiedad CSS es compuesto, se accede a su valor modificando ligeramente su nombre:

```
var parrafo = document.getElementById("parrafo");
alert(parrafo.style.fontWeight);  // muestra "bold"
<p id="parrafo" style="font-weight: bold;">...</p>
```

La transformación del nombre de las propiedades CSS compuestas consiste en eliminar todos los guiones medios (-) y escribir en mayúscula la letra siguiente a cada guión medio.
A continuación se muestran algunos ejemplos:

- font-weight se transforma en fontWeight
- line-height se transforma en lineHeight
- border-top-style se transforma en borderTopStyle
- list-style-image se transforma en listStyleImage

El único atributo XHTML que no tiene el mismo nombre en XHTML y en las propiedades DOM es el atributo class. Como la palabra class está reservada por JavaScript, no es posible utilizarla para acceder al atributo class del elemento XHTML.

En su lugar, DOM utiliza el nombre className para acceder al atributo class de XHTML:

```
var parrafo = document.getElementById("parrafo");
alert(parrafo.class); // muestra "undefined"
alert(parrafo.className); // muestra "normal"
<p id="parrafo" class="normal">...</p>
```

Veamos un ejemplo **Ej05.05b-AccesoEstilos.html**

```
<!DOCTYPE html>
<html lang="es">        <!--Ej05.05b-AccesoEstilos.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      function ejecutar() {
         var campos = new Array ();
         var i;
         var cadena="";
         campos = document.getElementsByClassName ("campos");
         for (i=0; i<campos.length; i++) {
            cadena +=   " Input "+campos[i].name +
                      " tiene de color " +campos[i].style.background + " \n ";
         }
         alert (cadena);
      }
   </script>
</head>
  <body>
   Usuario: <input type="text" name="usuario" class="campos"
            style="width: 200px; background: pink;" /> <br />
   Contraseña: <input type="password" name="clave" class="campos"
            style="width: 100px; background: aquamarine;" /> <br />
   <input type="button" value="Ver Estilos" onclick="ejecutar();" />
  </body>
</html>
```

Además podemos modificar los estilos en tiempo real (basado en el ejemplo anterior):
Ej05.05c-EstilosDinamicos.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej05.05c-EstilosDinamicos.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      function ejecutar() {
         var campos = new Array ();
         var i;
         var fondo="";
         fondo = window.prompt (" Seleccione el fondo de los Input: ");
         var bordes = window.prompt (" Seleccione el borde de los Input: ");
         var esquinas = window.prompt (" Seleccione las esquinas de los Input: ");
         campos = document.getElementsByClassName ("campos");
         for (i=0; i<campos.length; i++) {
            campos[i].style.backgroundColor = fondo;
            campos[i].style.border = bordes;
            campos[i].style.borderRadius = esquinas;
         }
      }
   </script>
</head>
  <body>
   Usuario: <input type="text" name="usuario" class="campos"
            style="width: 200px;" /> <br />
   Contraseña: <input type="password" name="clave" class="campos"
            style="width: 100px;" /> <br />
   <input type="button" value="cambiar color" onclick="ejecutar();" />
  </body>
</html>
```

5.6 Ejercicios sobre DOM

Ejercicio 11

A partir de la página web proporcionada y utilizando las funciones DOM, mostrar por pantalla la siguiente información:

- Número de enlaces de la página
- Dirección a la que enlaza el penúltimo enlace
- Numero de enlaces que enlazan a http://prueba
- Número de enlaces del tercer párrafo

Ejercicio11.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>Ejercicio 11 - DOM básico</title>
<script type="text/javascript">
   window.onload = function() {
   // Numero de enlaces de la pagina
   // Dirección del penúltimo enlace
   // Numero de enlaces que apuntan a http://prueba
   // Numero de enlaces del tercer párrafo  }
</script>
</head>
<body>
<p>Lorem ipsum dolor sit amet, <a href="http://prueba">consectetuer adipiscing elit</a>. Sed
mattis enim vitae orci. Phasellus libero. Maecenas nisl arcu, consequat congue, commodo nec,
commodo ultricies, turpis. Quisque sapien nunc, posuere vitae, rutrum et, luctus at, pede.
Pellentesque massa ante, ornare id, aliquam vitae, ultrices porttitor, pede. Nullam sit amet
nisl elementum elit convallis malesuada. Phasellus magna sem, semper quis, faucibus ut,
rhoncus non, mi. <a href="http://prueba2">Fusce porta</a>. Duis pellentesque, felis eu
adipiscing ullamcorper, odio urna consequat arcu, at posuere ante quam non dolor. Lorem ipsum
dolor sit amet, consectetuer adipiscing elit. Duis scelerisque. Donec lacus neque, vehicula
in, eleifend vitae, venenatis ac, felis. Donec arcu. Nam sed tortor nec ipsum aliquam
ullamcorper. Duis accumsan metus eu urna. Aenean vitae enim. Integer lacus. Vestibulum
venenatis erat eu odio. Praesent id metus.</p>

<p>Aenean at nisl. Maecenas egestas dapibus odio. Vestibulum ante ipsum primis in faucibus
orci luctus et ultrices posuere cubilia Curae; Proin consequat auctor diam. <a
href="http://prueba">Ut bibendum blandit est</a>. Curabitur vestibulum. Nunc malesuada
porttitor sapien. Aenean a lacus et metus venenatis porta. Suspendisse cursus, sem non dapibus
tincidunt, lorem magna porttitor felis, id sodales dolor dolor sed urna. Sed rutrum nulla
vitae tellus. Sed quis eros nec lectus tempor lacinia. Aliquam nec lectus nec neque aliquet
dictum. Etiam <a href="http://prueba3">consequat sem quis massa</a>. Donec aliquam euismod
diam. In magna massa, mattis id, pellentesque sit amet, porta sit amet, lectus. Curabitur
posuere. Aliquam in elit. Fusce condimentum, arcu in scelerisque lobortis, ante arcu
scelerisque mi, at cursus mi risus sed tellus.</p>

<p>Donec sagittis, nibh nec ullamcorper tristique, pede velit feugiat massa, at sollicitudin
justo tellus vitae justo. Vestibulum aliquet, nulla sit amet imperdiet suscipit, nunc erat
laoreet est, a <a href="http://prueba">aliquam leo odio sed sem</a>. Quisque eget eros
vehicula diam euismod tristique. Ut dui. Donec in metus sed risus laoreet sollicitudin. Proin
et nisi non arcu sodales hendrerit. In sem. Cras id augue eu lorem dictum interdum. Donec
pretium. Proin <a href="http://prueba4">egestas</a> adipiscing ligula. Duis iaculis laoreet
turpis. Mauris mollis est sit amet diam. Curabitur hendrerit, eros quis malesuada tristique,
ipsum odio euismod tortor, a vestibulum nisl mi at odio. <a href="http://prueba5">Sed non
lectus non est pellentesque</a> auctor.</p>
</body>
</html>
```

Ejercicio 12

Completar la función muestra de JavaScript para que realice lo siguiente:
a) Ocultar el enlace Seguir leyendo.
b) Mostrar el texto incluido dentro del
Una vez acabada esta parte, realizar lo siguiente:
a) Crear un nuevo evento en el enlace2 que llamará a la función oculta().
b) Dicha función ocultará el enlace2, el texto el span y volverá a mostrar el enlace Seguir leyendo.

Ejercicio12.html

```
<!DOCTYPE html> <html lang="es">
<head>
  <meta charset="UTF-8" />
<title>Ejercicio 12 - DOM básico y atributos XHTML</title>
<style type="text/css">
   .oculto { display: none; }
   .visible { display: inline; }
</style>
<script type="text/javascript">
   function muestra() { }
</script>
</head>
<body>
<p id="texto">Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed mattis enim vitae
orci. Phasellus libero. Maecenas nisl arcu, consequat congue, commodo nec, commodo ultricies,
turpis. Quisque sapien nunc, posuere vitae, rutrum et, luctus at, pede. Pellentesque massa
ante, ornare id, aliquam vitae, ultrices porttitor, pede. <span id="adicional"
class="oculto">Nullam sit amet nisl elementum elit convallis malesuada. Phasellus magna sem,
semper quis, faucibus ut, rhoncus non, mi. Duis pellentesque, felis eu adipiscing ullamcorper,
odio urna consequat arcu, at posuere ante quam non dolor. Lorem ipsum dolor sit amet,
consectetuer adipiscing elit. Duis scelerisque. Donec lacus neque, vehicula in, eleifend
vitae, venenatis ac, felis. Donec arcu. Nam sed tortor nec ipsum aliquam ullamcorper. Duis
accumsan metus eu urna. Aenean vitae enim. Integer lacus. Vestibulum venenatis erat eu odio.
Praesent id metus.</span></p>
<a id="enlace1" href="#" onclick="muestra();">Seguir leyendo</a> <br />
<a id="enlace2" href="#" class="oculto">Atrás</a> <br />
</body>
</html>
```

Ejercicio 13

Completar el código JavaScript proporcionado para que se añadan nuevos elementos a la lista cada vez que se pulsa sobre el botón. Utilizar las funciones DOM para crear nuevos nodos y añadirlos a la lista existente. Realizar la misma operación pero borrando los distintos elementos empezando por el final

Ejercicio13.html

```
<!DOCTYPE html> <html lang="es">
<head>
  <meta charset="UTF-8" />
<title>Ejercicio 13 - DOM básico y atributos XHTML</title>
<script type="text/javascript">
function insertar() {
}
</script>
</head>
<body>
<ul id="lista">
   <li>Elemento Lista 1</li>
   <li>Elemento Lista 2</li>
   <li>Elemento Lista 3</li>
   <li>Elemento Lista 4</li>
   <li>Elemento Lista 5</li>
</ul>
<input type="button" value="Añadir elemento" onclick="insertar();">
<input type="button" value="Borrar elemento" onclick="borrar();">
</body></html>
```

6 EVENTOS

Hasta ahora, todas las aplicaciones y scripts que se han creado tienen algo en común: se ejecutan desde la primera instrucción hasta la última de forma secuencial. Gracias a las estructuras de control de flujo (if, for, while) es posible modificar ligeramente este comportamiento y repetir algunos trozos del script y saltarse otros trozos en función de algunas condiciones.

Este tipo de aplicaciones son poco útiles, ya que no interactúan con los usuarios y no pueden responder a los diferentes eventos que se producen durante la ejecución de una aplicación. Afortunadamente, las aplicaciones web creadas con el lenguaje JavaScript pueden utilizar el modelo de programación basada en eventos. En este tipo de programación, los scripts se dedican a esperar a que el usuario "haga algo" (que pulse una tecla, que mueva el ratón, que cierre la ventana del navegador). A continuación, el script responde a la acción del usuario normalmente procesando esa información y generando un resultado.

Los eventos hacen posible que los usuarios transmitan información a los programas. JavaScript define numerosos eventos que permiten una interacción completa entre el usuario y las páginas/aplicaciones web. La pulsación de una tecla constituye un evento, así como pinchar o mover el ratón, seleccionar un elemento de un formulario, redimensionar la ventana del navegador, etc.

JavaScript permite asignar una función a cada uno de los eventos. De esta forma, cuando se produce cualquier evento, JavaScript ejecuta su función asociada. Este tipo de funciones se denominan "event handlers" en inglés y suelen traducirse por "manejadores de eventos".

6.1 Modelos de eventos

Crear páginas y aplicaciones web siempre ha sido mucho más complejo de lo que debería serlo debido a las incompatibilidades entre navegadores. A pesar de que existen decenas de estándares para las tecnologías empleadas, los navegadores no los soportan completamente o incluso los ignoran.

Las principales incompatibilidades se producen en el lenguaje XHTML, en el soporte de hojas de estilos CSS y sobre todo, en la implementación de JavaScript. De todas ellas, la incompatibilidad más importante se da precisamente en el modelo de eventos del navegador. Así, existen hasta tres modelos diferentes para manejar los eventos dependiendo del navegador en el que se ejecute la aplicación.

6.1.1 Modelo básico de eventos

Este modelo simple de eventos se introdujo para la versión 4 del estándar HTML y se considera parte del nivel más básico de DOM. Aunque sus características son limitadas, es el único modelo que es compatible en todos los navegadores y por tanto, el único que permite crear aplicaciones que funcionan de la misma manera en todos los navegadores.

6.1.2 Modelo de eventos estándar

Las versiones más avanzadas del estándar DOM (DOM nivel 2) definen un modelo de eventos completamente nuevo y mucho más poderoso que el original. Todos los navegadores modernos lo incluyen, salvo Internet Explorer.

6.1.3 Modelo de eventos de Internet Explorer

Internet Explorer utiliza su propio modelo de eventos, que es similar pero incompatible con el modelo estándar. Se utilizó por primera vez en Internet Explorer 4 y Microsoft decidió seguir utilizándolo en el resto de versiones.

6.2 Modelo básico de eventos

6.2.1 Tipos de eventos

En este modelo, cada elemento o etiqueta XHTML define su propia lista de posibles eventos que se le pueden asignar. Un mismo tipo de evento (por ejemplo, pinchar el botón izquierdo del ratón) puede estar definido para varios elementos XHTML diferentes y un mismo elemento XHTML puede tener asociados varios eventos diferentes.

El nombre de cada evento se construye mediante el prefijo on, seguido del nombre en inglés de la acción asociada al evento. Así, el evento de pinchar un elemento con el ratón se denomina onclick y el evento asociado a la acción de mover el ratón se denomina onmousemove.

La siguiente tabla resume los eventos más importantes definidos por JavaScript:

Evento	Descripción	Elementos para los que está definido
onblur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
onclick	Pinchar y soltar el ratón	Todos los elementos
ondblclick	Pinchar dos veces seguidas con el ratón	Todos los elementos
onfocus	Seleccionar un elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onkeydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>
onkeypress	Pulsar una tecla	Elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onload	La página se ha cargado completamente	<body>
onmousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos
onmouseout	El ratón *"sale"* del elemento (pasa por encima de otro elemento)	Todos los elementos
onmouseover	El ratón *"entra"* en el elemento (pasa por encima del elemento)	Todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
onreset	Inicializar el formulario (borrar todos sus datos)	<form>
onresize	Se ha modificado el tamaño de la ventana del navegador	<body>
onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar el formulario	<form>
onunload	Se abandona la página (por ejemplo al cerrar el navegador)	<body>

Los eventos más utilizados en las aplicaciones web tradicionales son onload para esperar a que se cargue la página por completo, los eventos onclick, onmouseover, onmouseout para controlar el ratón y onsubmit para controlar el envío de los formularios.

Algunos eventos de la tabla anterior (onclick, onkeydown, onkeypress, onreset, onsubmit) permiten evitar la "acción por defecto" de ese evento. Más adelante se muestra en detalle este comportamiento, que puede resultar muy útil en algunas técnicas de programación.

Las acciones típicas que realiza un usuario en una página web pueden dar lugar a una sucesión de eventos. Al pulsar por ejemplo sobre un botón de tipo <input type="submit"> se desencadenan los eventos onmousedown, onclick, onmouseup y onsubmit de forma consecutiva.

6.2.2 Manejadores de eventos

Un evento de JavaScript por sí mismo carece de utilidad. Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado, por lo que la aplicación puede responder ante cualquier evento que se produzca durante su ejecución.

Las funciones o código JavaScript que se definen para cada evento se denominan "manejador de eventos" y como JavaScript es un lenguaje muy flexible, existen varias formas diferentes de indicar los manejadores:

- Manejadores como atributos de los elementos XHTML.
- Manejadores como funciones JavaScript externas.
- Manejadores "semánticos".

6.2.2.1 Manejadores de eventos como atributos XHTML

Se trata del método más sencillo y a la vez menos profesional de indicar el código JavaScript que se debe ejecutar cuando se produzca un evento. En este caso, el código se incluye en un atributo del propio elemento XHTML. En el siguiente ejemplo, se quiere mostrar un mensaje cuando el usuario pinche con el ratón sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias por
pinchar');" />
```

En este método, se definen atributos XHTML con el mismo nombre que los eventos que se quieren manejar. El ejemplo anterior sólo quiere controlar el evento de pinchar con el ratón, cuyo nombre es onclick. Así, el elemento XHTML para el que se quiere definir este evento, debe incluir un atributo llamado onclick.

El contenido del atributo es una cadena de texto que contiene todas las instrucciones JavaScript que se ejecutan cuando se produce el evento. En este caso, el código JavaScript es muy sencillo (alert('Gracias por pinchar');), ya que solamente se trata de mostrar un mensaje.

En este otro ejemplo, cuando el usuario pincha sobre el elemento se muestra un mensaje y cuando el usuario pasa el ratón por encima del elemento, se muestra otro mensaje:
NOTA: Como puede comprobarse, tiene preferencia este último evento, onmouseover...

Ej06.02a-EventosHTML.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej06.02a-EventosHTML.html -->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
    <style type="text/css">
      div {
        width: 300px;
        height: 300px;
        background: aquamarine;             }
 </style>
  <script language="JavaScript">
    function pulsado() {
      alert ("Has pulsado en la Caja!");     }
    function encima() {
      alert (" Las características de la caja son: \n" +
             " 300px de ancho y 300px de alto \n" +
             " Color: Aguamarina");          }
  </script>
</head>
<body>
    <div onclick="pulsado();" onmouseover="encima();"></div>
    <input type="button" value="Probar" onClick="ejecutar();" />
</body> </html>
```

Este otro ejemplo incluye una de las instrucciones más utilizadas en las aplicaciones JavaScript más antiguas:

```
<body onload="alert('La página se ha cargado completamente');">
   ...
</body>
```

El mensaje anterior se muestra después de que la página se haya cargado completamente, es decir, después de que se haya descargado su código HTML, sus imágenes y cualquier otro objeto incluido en la página.

El evento onload es uno de los más utilizados ya que, como se vio en el capítulo de DOM, las funciones que permiten acceder y manipular los nodos del árbol DOM solamente están disponibles cuando la página se ha cargado completamente.

6.2.2.2 Manejadores de eventos y variable this

JavaScript define una variable especial llamada this que se crea automáticamente y que se emplea en algunas técnicas avanzadas de programación. En los eventos, se puede utilizar la variable this para referirse al elemento XHTML que ha provocado el evento. Esta variable es muy útil para ejemplos como el siguiente:

Cuando el usuario pasa el ratón por encima del , el color del borde se muestra de color negro. Cuando el ratón sale del , se vuelve a mostrar el borde con el color gris claro original.
Elemento original:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver">
  Sección de contenidos...
</div>
```

Si no se utiliza la variable this, el código necesario para modificar el color de los bordes, sería el siguiente:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver"
onmouseover="document.getElementById('contenidos').style.borderColor='black';"
onmouseout="document.getElementById('contenidos').style.borderColor='silver';">
  Sección de contenidos...
</div>
```

El código anterior es demasiado largo y demasiado propenso a cometer errores. Dentro del código de un evento, JavaScript crea automáticamente la variable this, que hace referencia al elemento XHTML que ha provocado el evento. Así, el ejemplo anterior se puede reescribir de la siguiente manera:
NOTA: No funciona si se intenta sacar el código en una función externa: **Ej06.02b-This.html**

```
<!DOCTYPE html>
<html lang="es">          <!--Ej06.02b-This.html -->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
    <style type="text/css">
      div {  width: 300px;             height: 150px;
             border: 5px solid blue;    }
  </style>
</head>
<body>
    <div onmouseover="this.style.borderColor='red';"
onmouseout="this.style.borderColor='blue';"></div>
    <input type="button" value="Cambiar Bordes" onClick="ejecutar();" />
</body>
</html>
```

El código anterior es mucho más compacto, más fácil de leer y de escribir y sigue funcionando correctamente aunque se modifique el valor del atributo id del .

6.2.2.3 Manejadores como funciones JavaScript externas

La definición de los manejadores de eventos en los atributos XHTML es el método más sencillo pero menos aconsejable de tratar con los eventos en JavaScript. El principal inconveniente es que se complica en exceso en cuanto se añaden algunas pocas instrucciones, por lo que solamente es recomendable para los casos más sencillos.

Si se realizan aplicaciones complejas, como por ejemplo la validación de un formulario, es aconsejable agrupar todo el código JavaScript en una función externa y llamar a esta función desde el elemento XHTML.

Siguiendo con el ejemplo anterior que muestra un mensaje al pinchar sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias por
pinchar');" />
```

Utilizando funciones externas se puede transformar en:

```
function muestraMensaje() {
  alert('Gracias por pinchar');
}
<input type="button" value="Pinchame y verás" onclick="muestraMensaje()" />
```

Esta técnica consiste en extraer todas las instrucciones de JavaScript y agruparlas en una función externa. Una vez definida la función, en el atributo del elemento XHTML se incluye el nombre de la función, para indicar que es la función que se ejecuta cuando se produce el evento. La llamada a la función se realiza de la forma habitual, indicando su nombre seguido de los paréntesis y de forma opcional, incluyendo todos los argumentos y parámetros que se necesiten.

El principal inconveniente de este método es que en las funciones externas no se puede seguir utilizando la variable this y por tanto, es necesario pasar esta variable como parámetro a la función:

Ej06.02c-ThisExterno.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej06.02b-This.html-->
<head>
   <meta charset="UTF-8" />
   <title>JavaScript</title>
   <style type="text/css">
      div {    width: 300px;  height: 150px;
            background: aquamarine;
            border: 5px solid blue;
      }
      img {    width: 300px;  height: 150px;
            display: inline;
      }
   </style>
   <script type="text/javascript">
      function dentro(objeto) {
         objeto.style.background = "yellow";              }

      function fuera(objeto) {
         objeto.style.background = "red";          }

      function oculta(objeto) {
         objeto.style.display = "none";            }
   </script>
</head>
<body>
   <div onmouseover="dentro(this);"
         onmouseout="fuera(this);">  ...
   </div>
   <br />
   <img src="sevilla.jpg" onmouseout="oculta(this);" />
</body> </html>
```

Veamos otro ejemplo adaptado a distintos navegadores (usando switch...case):

```
function resalta(elemento) {
  switch(elemento.style.borderColor) {
    case 'silver':
    case 'silver silver silver silver':
    case '#c0c0c0':
      elemento.style.borderColor = 'black';
      break;
    case 'black':
    case 'black black black black':
    case '#000000':
      elemento.style.borderColor = 'silver';
      break;
  }
}

<div style="width:150px; height:60px; border:thin solid silver"
onmouseover="resalta(this)" onmouseout="resalta(this)">
  Sección de contenidos...
</div>
```

En el ejemplo anterior, la función externa es llamada con el parámetro this, que dentro de la función se denomina elemento. La complejidad del ejemplo se produce sobre todo por la forma en la que los distintos navegadores almacenan el valor de la propiedad borderColor. Mientras que Firefox almacena (en caso de que los cuatro bordes coincidan en color) el valor black, Internet Explorer lo almacena como black black black black y Opera almacena su representación hexadecimal #000000.

6.2.2.4 Manejadores "semánticos"

Los métodos que se han visto para añadir manejadores de eventos (como atributos XHTML y como funciones externas) tienen un grave inconveniente: "ensucian" el código XHTML de la página.

Como es conocido, una de las buenas prácticas básicas en el diseño de páginas y aplicaciones web es la separación de los contenidos (XHTML) y su aspecto o presentación (CSS). Siempre que sea posible, también se recomienda separar los contenidos (XHTML) y su comportamiento o programación (JavaScript).

Mezclar el código JavaScript con los elementos XHTML solamente contribuye a complicar el código fuente de la página, a dificultar la modificación y mantenimiento de la página y a reducir la semántica del documento final producido.

Afortunadamente, existe un método alternativo para definir los manejadores de eventos de JavaScript. Esta técnica es una evolución del método de las funciones externas, ya que se basa en utilizar las propiedades DOM de los elementos XHTML para asignar todas las funciones externas que actúan de manejadores de eventos. Así, el siguiente ejemplo:

```
<input id="pinchable" type="button" value="Pinchame y verás"
onclick="alert('Gracias por pinchar');" />
```

Se puede transformar en:

Ej06.02d-ManejadorSemantico.html

```
<!DOCTYPE html>
<html lang="es">          <!--Ej06.02d-ManejadorSemantico.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <script language="JavaScript">
    //Creamos una funcion externa
    function muestraMensaje() {
      alert('Gracias por pinchar');
    }

    // Asigno la función externa al elemento
    window.onload = function() {
      document.getElementById("pinchable").onclick = muestraMensaje;
    }
  </script>
</head>
<body>
  <input id="pinchable" type="button" value="Púlsame" />
</body>
</html>
```

La técnica de los manejadores semánticos consiste en:

- Asignar un identificador único al elemento XHTML mediante el atributo id.
- Crear una función de JavaScript encargada de manejar el evento.
- Asignar la función externa al evento correspondiente en el elemento deseado.

El último paso es la clave de esta técnica. En primer lugar, se obtiene el elemento al que se desea asociar la función externa:

```
document.getElementById("pinchable");
```

A continuación, se utiliza una propiedad del elemento con el mismo nombre que el evento que se quiere manejar. En este caso, la propiedad es onclick:

```
document.getElementById("pinchable").onclick = ...
```

Por último, se asigna la función externa mediante su nombre sin paréntesis. Lo más importante (y la causa más común de errores) es indicar solamente el nombre de la función, es decir, prescindir de los paréntesis al asignar la función:

```
document.getElementById("pinchable").onclick = muestraMensaje;
```

Si se añaden los paréntesis después del nombre de la función, en realidad se está ejecutando la función y guardando el valor devuelto por la función en la propiedad onclick de elemento.

```
// Asignar una función externa a un evento de un elemento
document.getElementById("pinchable").onclick = muestraMensaje;

// Ejecutar una función y guardar su resultado en una propiedad de un elemento
document.getElementById("pinchable").onclick = muestraMensaje();
```

La gran ventaja de este método es que el código XHTML resultante es muy "limpio", ya que no se mezcla con el código JavaScript. Además, dentro de las funciones externas asignadas sí que se puede utilizar la variable this para referirse al elemento que provoca el evento.

El único inconveniente de este método es que la página se debe cargar completamente antes de que se puedan utilizar las funciones DOM que asignan los manejadores a los elementos XHTML. Una de las formas más sencillas de asegurar que cierto código se va a ejecutar después de que la página se cargue por completo es utilizar el evento onload:

```
window.onload = function() {
  document.getElementById("pinchable").onclick = muestraMensaje;
}
```

La técnica anterior utiliza el concepto de funciones anónimas, que no se va a estudiar, pero que permite crear un código compacto y muy sencillo. Para asegurarse que un código JavaScript va a ejecutarse después de que la página se haya cargado completamente, sólo es necesario incluir esas instrucciones entre los símbolos { y }:

```
window.onload = function() {
  ...
}
```

En el siguiente ejemplo, se añaden eventos a los elementos de tipo input=text de un formulario complejo (es decir, sólo a los input de texto se les aplica algún cambio):

Ej06.02e-ManejadorSemanticoEstilo.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej06.02e-ManejadorSemanticoEstilo.html -->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      window.onload = function () {
         var i;
         var input = new Array ();
         inputs = document.getElementsByTagName ("input");

         for (i=0; i<inputs.length; i++) {
            if (inputs[i].type == "text" ||
               inputs[i].type == "password") {

               inputs[i].onfocus = resalta;
               inputs[i].onblur = noresalta;
            }
         }
      }

      function resalta() {
         this.style.backgroundColor = "khaki";
         this.style.color = "red";
      }

      function noresalta() {
         this.style.backgroundColor = "white";
         this.style.color = "black";
      }
   </script>
</head>
  <body>
   <form action="#" method="post" id="formulario">
      Usuario <input type="text" id="usuario" /> <br />
      Contraseña <input type="password" id="clave" /> <br />
      <input type="button" value="Enviar" />
   </form>
  </body>
</html>
```

Ejercicio 14

A partir de la página web proporcionada, completar el código JavaScript para que:

1. Cuando se pinche sobre el primer enlace, se oculte su sección relacionada
2. Cuando se vuelva a pinchar sobre el mismo enlace, se muestre otra vez esa sección de contenidos
3. Completar el resto de enlaces de la página para que su comportamiento sea idéntico al del primer enlace
4. Cuando una sección se oculte, debe cambiar el mensaje del enlace asociado (pista: propiedad innerHTML)

Ejercicio14.html

```
<!DOCTYPE html> <html lang="es">
<head>
  <meta charset="UTF-8" />
<title>Ejercicio 14 - DOM básico y atributos XHTML</title>

<script type="text/javascript">
function muestraOculta() {
}
</script>
</head>

<body>

<p id="contenidos_1">Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed mattis enim
vitae orci. Phasellus libero. Maecenas nisl arcu, consequat congue, commodo nec, commodo
ultricies, turpis. Quisque sapien nunc, posuere vitae, rutrum et, luctus at, pede.
Pellentesque massa ante, ornare id, aliquam vitae, ultrices porttitor, pede. Nullam sit amet
nisl elementum elit convallis malesuada. Phasellus magna sem, semper quis, faucibus ut,
rhoncus non, mi. Duis pellentesque, felis eu adipiscing ullamcorper, odio urna consequat arcu,
at posuere ante quam non dolor. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis
scelerisque.</p>
<a id="enlace_1" href="#">Ocultar contenidos</a>

<br/>

<p id="contenidos_2">Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed mattis enim
vitae orci. Phasellus libero. Maecenas nisl arcu, consequat congue, commodo nec, commodo
ultricies, turpis. Quisque sapien nunc, posuere vitae, rutrum et, luctus at, pede.
Pellentesque massa ante, ornare id, aliquam vitae, ultrices porttitor, pede. Nullam sit amet
nisl elementum elit convallis malesuada. Phasellus magna sem, semper quis, faucibus ut,
rhoncus non, mi. Duis pellentesque, felis eu adipiscing ullamcorper, odio urna consequat arcu,
at posuere ante quam non dolor. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis
scelerisque.</p>
<a id="enlace_2" href="#">Ocultar contenidos</a>

<br/>

<p id="contenidos_3">Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed mattis enim
vitae orci. Phasellus libero. Maecenas nisl arcu, consequat congue, commodo nec, commodo
ultricies, turpis. Quisque sapien nunc, posuere vitae, rutrum et, luctus at, pede.
Pellentesque massa ante, ornare id, aliquam vitae, ultrices porttitor, pede. Nullam sit amet
nisl elementum elit convallis malesuada. Phasellus magna sem, semper quis, faucibus ut,
rhoncus non, mi. Duis pellentesque, felis eu adipiscing ullamcorper, odio urna consequat arcu,
at posuere ante quam non dolor. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis
scelerisque.</p>
<a id="enlace_3" href="#">Ocultar contenidos</a>

</body>
</html>
```

6.3 Obteniendo información del evento (objeto event)

Normalmente, los manejadores de eventos requieren información adicional para procesar sus tareas. Si una función por ejemplo se encarga de procesar el evento onclick, quizás necesite saber en que posición estaba el ratón en el momento de pinchar el botón.

No obstante, el caso más habitual en el que es necesario conocer información adicional sobre el evento es el de los eventos asociados al teclado. Normalmente, es muy importante conocer la tecla que se ha pulsado, por ejemplo para diferenciar las teclas normales de las teclas especiales (ENTER, tabulador, Alt, Ctrl., etc.).

JavaScript permite obtener información sobre el ratón y el teclado mediante un objeto especial llamado event. Desafortunadamente, los diferentes navegadores presentan diferencias muy notables en el tratamiento de la información sobre los eventos.

La principal diferencia reside en la forma en la que se obtiene el objeto event. Internet Explorer considera que este objeto forma parte del objeto window y el resto de navegadores lo consideran como el único argumento que tienen las funciones manejadoras de eventos.

Aunque es un comportamiento que resulta muy extraño al principio, todos los navegadores modernos excepto Internet Explorer crean mágicamente y de forma automática un argumento que se pasa a la función manejadora, por lo que no es necesario incluirlo en la llamada a la función manejadora. De esta forma, para utilizar este "argumento mágico", sólo es necesario asignarle un nombre, ya que los navegadores lo crean automáticamente.

En resumen, en los navegadores tipo Internet Explorer, el objeto event se obtiene directamente mediante:

```
var evento = window.event;
```

Por otra parte, en el resto de navegadores, el objeto event se obtiene mágicamente a partir del argumento que el navegador crea automáticamente:

```
function manejadorEventos(elEvento) {
  var evento = elEvento;
}
```

Si se quiere programar una aplicación que funcione correctamente en todos los navegadores, es necesario obtener el objeto event de forma correcta según cada navegador. El siguiente código muestra la forma correcta de obtener el objeto event en cualquier navegador:

```
function manejadorEventos(elEvento) {
  var evento = elEvento || window.event;
}
```

Una vez obtenido el objeto event, ya se puede acceder a toda la información relacionada con el evento, que depende del tipo de evento producido.

6.3.1 Información sobre el evento.

La propiedad type indica el tipo de evento producido, lo que es útil cuando una misma función se utiliza para manejar varios eventos:

```
var tipo = evento.type;
```

La propiedad type devuelve el tipo de evento producido, que es igual al nombre del evento pero sin el prefijo on.

Mediante esta propiedad, se puede rehacer de forma más sencilla el ejemplo anterior en el que se resaltaba una sección de contenidos al pasar el ratón por encima:

Ej06.03a-EventoType.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej06.03a-EventoType.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <script language="JavaScript">
    function resalta(elEvento) {
      var evento = elEvento || window.event; // Compatibilidad IE

      // En el case hay que poner el evento SIN el on por delante
      switch(evento.type) {
        case 'mouseover':
          this.style.borderColor = 'red';
          break;
        case 'mouseout':
          this.style.borderColor = 'blue';
          break;
      }
    }

    window.onload = function() {
      document.getElementById("caja").onmouseover = resalta;
      document.getElementById("caja").onmouseout = resalta;
    }
  </script>
</head>
<body>
  <div id="caja" style="width:150px; height:60px; border:5px solid silver">
    Sección de contenidos...
  </div>
</body>
</html>
```

6.3.2 Información sobre los eventos de teclado

De todos los eventos disponibles en JavaScript, los eventos relacionados con el teclado son los más incompatibles entre diferentes navegadores y por tanto, los más difíciles de manejar. En primer lugar, existen muchas diferencias entre los navegadores, los teclados y los sistemas operativos de los usuarios, principalmente debido a las diferencias entre idiomas.

Además, existen tres eventos diferentes para las pulsaciones de las teclas (onkeyup, onkeypress y onkeydown). Por último, existen dos tipos de teclas: las teclas normales (como letras, números y símbolos normales) y las teclas especiales (como ENTER, Alt, Shift, etc.)

Cuando un usuario pulsa una tecla normal, se producen tres eventos seguidos y en este orden: onkeydown, onkeypress y onkeyup. El evento onkeydown se corresponde con el hecho de pulsar una tecla y no soltarla; el evento onkeypress es la propia pulsación de la tecla y el evento onkeyup hace referencia al hecho de soltar una tecla que estaba pulsada.

La forma más sencilla de obtener la información sobre la tecla que se ha pulsado es mediante el evento onkeypress. La información que proporcionan los eventos onkeydown y onkeyup se puede considerar como más técnica, ya que devuelven el código interno de cada tecla y no el carácter que se ha pulsado.

A continuación se incluye una lista con todas las propiedades diferentes de todos los eventos de teclado tanto en Internet Explorer como en el resto de navegadores:

- Evento **keydown**:
  - Mismo comportamiento en todos los navegadores:
    - Propiedad **keyCode**: código interno de la tecla
    - Propiedad **charCode**: no definido
- Evento **keypress**:
  - Internet Explorer:
    - Propiedad **keyCode**: el código del carácter de la tecla que se ha pulsado
    - Propiedad **charCode**: no definido
  - Resto de navegadores:
    - Propiedad **keyCode**: para las teclas normales, no definido. Para las teclas especiales, el código interno de la tecla.
    - Propiedad **charCode**: para las teclas normales, el código del carácter de la tecla que se ha pulsado. Para las teclas especiales, 0.
- Evento **keyup**:
  - Mismo comportamiento en todos los navegadores:
    - Propiedad **keyCode**: código interno de la tecla
    - Propiedad **charCode**: no definido

Para convertir el código de un carácter (no confundir con el código interno) al carácter que representa la tecla que se ha pulsado, se utiliza la función String.fromCharCode().

A continuación se incluye un script que muestra toda la información sobre los tres eventos de teclado:

Ej06.03b-EventoTeclado.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej06.03b-EventoTeclado.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <script language="JavaScript">
    window.onload = function() {
      document.onkeyup = muestraInformacion;
      document.onkeypress = muestraInformacion;
      document.onkeydown = muestraInformacion;
    }

    function muestraInformacion(elEvento) {
      var evento = window.event || elEvento;

      var mensaje = "Tipo de evento: " + evento.type + "<br>" +
                    "Propiedad keyCode: " + evento.keyCode + "<br>" +
                    "Propiedad charCode: " + evento.charCode + "<br>" +
                    "Carácter pulsado: " + String.fromCharCode(evento.charCode);

      // Con el <id>.innerHTML escribimos en la caja
      info.innerHTML += "<br />--------------------------------------<br />" + mensaje;
}
  </script>
</head>
<body>
  <div id="info"></div>
</body>
</html>
```

En los eventos keydown y keyup, la propiedad keyCode sigue valiendo lo mismo en los dos casos. El motivo es que keyCode almacena el código interno de la tecla, por lo que si se pulsa la misma tecla, se obtiene el mismo código, independientemente de que una misma tecla puede producir caracteres diferentes (por ejemplo mayúsculas y minúsculas).

En el evento keypress, el valor de la propiedad charCode varía, ya que el carácter a, no es el mismo que el carácter A. En este caso, el valor de charCode coincide con el código ASCII del carácter pulsado.

Las teclas especiales no disponen de la propiedad charCode, ya que sólamente se guarda el código interno de la tecla pulsada en la propiedad keyCode, en este caso el código 9. Si se pulsa la tecla Enter, se obtiene el código 13, la tecla de la flecha superior produce el código 38, etc. No obstante, dependiendo del teclado utilizado para pulsar las teclas y dependiendo de la disposición de las teclas en función del idioma del teclado, estos códigos podrían variar.

La propiedad keyCode en el evento keypress contiene el código ASCII del carácter de la tecla, por lo que se puede obtener directamente el carácter mediante String.fromCharCode(keyCode).

Si se pulsa la tecla A, la información mostrada es idéntica al caso de pulsar a, salvo que el código que muestra el evento keypress cambia por 65, que es el código ASCII de la tecla A.

Los códigos mostrados para las teclas especiales coinciden con los de Firefox y el resto de navegadores, pero recuerda que pueden variar en función del teclado que se utiliza y en función de la disposición de las teclas para cada idioma.

Por último, las propiedades altKey, ctrlKey y shiftKey almacenan un valor booleano que indica si alguna de esas teclas estaba pulsada al producirse el evento del teclado. Sorprendentemente, estas tres propiedades funcionan de la misma forma en todos los navegadores:

```
if (evento.altKey) {
  alert('Estaba pulsada la tecla ALT');
}
```

El evento keypress es el único que permite obtener el carácter realmente pulsado, ya que al pulsar sobre la tecla 2 habiendo pulsado la tecla Shift previamente, se obtiene el carácter ", que es precisamente el que muestra el evento keypress.

El siguiente código de JavaScript permite obtener de forma correcta en cualquier navegador el carácter correspondiente a la tecla pulsada:

Ej06.03c-EventoTeclaPulsada.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />                  <!--Ej06.03c-EventoTeclaPulsada.html-->
<title>JavaScript</title>
<script type="text/javascript">
   window.onload = function () {
      document.onkeypress = info;
   }
   function info(evento) {
      var caracter = evento.charCode || evento.keyCode;
      var tecla = String.fromCharCode (caracter);
      alert ("La tecla pulsada es: " + tecla);
   }
</script>
</head>
<body>
   Pulse una tecla para verla en pantalla...
</body> </html>
```

6.3.3 Información sobre los eventos de ratón

La información más relevante sobre los eventos relacionados con el ratón es la de las coordenadas de la posición del puntero del ratón. Aunque el origen de las coordenadas siempre se encuentra en la esquina superior izquierda, el punto que se toma como referencia de las coordenadas puede variar.

De esta forma, es posible obtener la posición del ratón respecto de la pantalla del ordenador, respecto de la ventana del navegador y respecto de la propia página HTML (que se utiliza cuando el usuario ha hecho scroll sobre la página). Las coordenadas más sencillas son las que se refieren a la posición del puntero respecto de la ventana del navegador, que se obtienen mediante las propiedades clientX y clientY.

Las coordenadas de la posición del puntero del ratón respecto de la pantalla completa del ordenador del usuario se obtienen de la misma forma, mediante las propiedades screenX y screenY:

Ej06.03d-EventoRaton.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />                  <!--Ej06.03d-EventoRaton.html-->
<title>JavaScript</title>
<script type="text/javascript">
   window.onload = function () {
      document.onmousemove= info;
   }

   function info(evento) {
      var coordX = evento.clientX;   // Coord. X respecto al Navegador
      var coordY = evento.clientY;   // Coord. Y respecto al Navegador
      var scrX = evento.screenX;     // Coord. X respecto a la Pantalla
      var scrY = evento.screenY;     // Coord. Y respecto a la Pantalla

      var cadena =   " PosX -> " + coordX + "<br />" +
                  " PosY -> " + coordY + "<br />" +
                  " PantX -> " + scrX + "<br />" +
                  " PantY -> " + scrY + "<br />";

      var caja = document.getElementById ("caja");
      caja.innerHTML = cadena;
   }
</script>
</head>
<body>
   <div id="caja">   ... </div>
</body>
</html>
```

En muchas ocasiones, es necesario obtener otro par de coordenadas diferentes: las que corresponden a la posición del ratón respecto del origen de la página. Estas coordenadas no siempre coinciden con las coordenadas respecto del origen de la ventana del navegador, ya que el usuario puede hacer scroll sobre la página web. Internet Explorer no proporciona estas coordenadas de forma directa, mientras que el resto de navegadores sí que lo hacen.

De esta forma, es necesario detectar si el navegador es de tipo Internet Explorer y en caso afirmativo realizar un cálculo sencillo:

Ej06.03e-EventoRatonScroll.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej06.03e-EventoRatonScroll.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <script language="JavaScript">
    function muestraInformacion(elEvento) {
      var evento = elEvento || window.event;
      // Detectar si el navegador es Internet Explorer
      var ie = navigator.userAgent.toLowerCase().indexOf('msie')!=-1;

      if(ie) {
        coordenadaX = evento.clientX + document.body.scrollLeft;
        coordenadaY = evento.clientY + document.body.scrollTop;
      }
      else {
        coordenadaX = evento.pageX;
        coordenadaY = evento.pageY;
      }

      alert("Has pulsado el ratón en la posición: " +
            coordenadaX + ", " + coordenadaY + " respecto de la página web");
    }

    document.onclick = muestraInformacion;
  </script>
</head>
<body>
  Pulsa para ver la posición del ratón
  <p> </p><p> </p><p> </p><p> </p><p> </p>
   <p> </p><p> </p><p> </p><p> </p><p> </p>
    <p> </p><p> </p><p> </p><p> </p><p> </p>
     <p> </p><p> </p><p> </p><p> </p><p> </p>
       <p> </p><p> </p><p> </p><p> </p><p> </p>
         <p> </p><p> </p><p> </p><p> </p><p> </p>
</body>
</html>
```

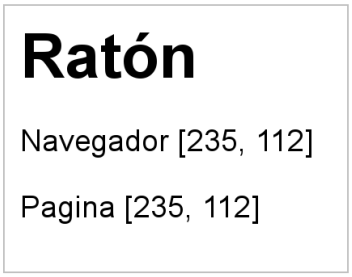
La variable ie vale true si el navegador en el que se ejecuta el script es de tipo Internet Explorer (cualquier versión) y vale false en otro caso. Para el resto de navegadores, las coordenadas respecto del origen de la página se obtienen mediante las propiedades pageX y pageY. En el caso de Internet Explorer, se obtienen sumando la posición respecto de la ventana del navegador (clientX, clientY) y el desplazamiento que ha sufrido la página (document.body.scrollLeft, document.body.scrollTop).

6.4 Ejercicios

Ejercicio 15

Crear el código JavaScript necesario para realizar las siguientes acciones sobre una caja cuyo id es info. Es decir, el código del body será:

```
<body>
<div id="info"></div>
</body>
```

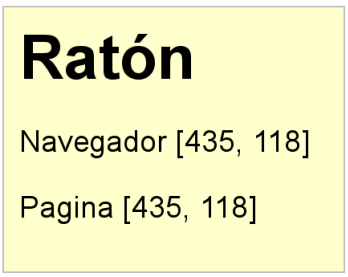


1. Al mover el ratón en cualquier punto de la ventana del navegador, se muestre la posición del puntero respecto del navegador y respecto de la página:

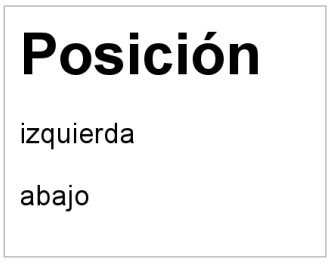
2. Al pulsar cualquier tecla, el mensaje mostrado debe cambiar para indicar el nuevo evento y su información asociada:

3. Añadir la siguiente característica al script: cuando se pulsa un botón del ratón, el color de fondo del cuadro de mensaje debe ser amarillo (#FFFFCC) y cuando se pulsa una tecla, el color de fondo debe ser azul (#CCE6FF). Al volver a mover el ratón, el color de fondo vuelve a ser blanco.

Es decir, quedará el mismo DIV de este modo:



Ejercicio 16



Crear un script que informe al usuario en que zona de la pantalla ha pulsado el ratón. Las zonas definidas son las siguientes: izquierda arriba, izquierda abajo, derecha arriba y derecha abajo. Para determinar el tamaño de la ventana del navegador, utilizar la función tamanoVentanaNavegador() proporcionada.

7 FORMULARIOS

La programación de aplicaciones que contienen formularios web siempre ha sido una de las tareas fundamentales de JavaScript. De hecho, una de las principales razones por las que se inventó el lenguaje de programación JavaScript fue la necesidad de validar los datos de los formularios directamente en el navegador del usuario. De esta forma, se evitaba recargar la página cuando el usuario cometía errores al rellenar los formularios.

No obstante, la aparición de las aplicaciones AJAX ha relevado al tratamiento de formularios como la principal actividad de JavaScript. Ahora, el principal uso de JavaScript es el de las comunicaciones asíncronas con los servidores y el de la manipulación dinámica de las aplicaciones. De todas formas, el manejo de los formularios sigue siendo un requerimiento imprescindible para cualquier programador de JavaScript.

7.1 Propiedades básicas de formularios y elementos

JavaScript dispone de numerosas propiedades y funciones que facilitan la programación de aplicaciones que manejan formularios. En primer lugar, cuando se carga una página web, el navegador crea automáticamente un array llamado forms y que contiene la referencia a todos los formularios de la página.

Para acceder al array forms, se utiliza el objeto document, por lo que document.forms es el array que contiene todos los formularios de la página. Como se trata de un array, el acceso a cada formulario se realiza con la misma sintaxis de los arrays. La siguiente instrucción accede al primer formulario de la página:

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

Además del array de formularios, el navegador crea automáticamente un array llamado elements por cada uno de los formularios de la página. Cada array elements contiene la referencia a todos los elementos (cuadros de texto, botones, listas desplegables, etc.) de ese formulario. Utilizando la sintaxis de los arrays, la siguiente instrucción obtiene el primer elemento del primer formulario de la página:

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

La sintaxis de los arrays no siempre es tan concisa. El siguiente ejemplo muestra cómo obtener directamente el último elemento del primer formulario de la página:

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

Aunque esta forma de acceder a los formularios es rápida y sencilla, tiene un inconveniente muy grave. ¿Qué sucede si cambia el diseño de la página y en el código HTML se cambia el orden de los formularios originales o se añaden nuevos formularios? El problema es que "el primer formulario de la página" ahora podría ser otro formulario diferente al que espera la aplicación.

En un entorno tan cambiante como el diseño web, es muy difícil confiar en que el orden de los formularios se mantenga estable en una página web. Por este motivo, siempre debería evitarse el acceso a los formularios de una página mediante el array document.forms.

Una forma de evitar los problemas del método anterior consiste en acceder a los formularios de una página a través de su nombre (atributo name) o a través de su atributo id.

El objeto document permite acceder directamente a cualquier formulario mediante su atributo name:

```
var formularioPrincipal = document.formulario;
var formularioSecundario = document.otro_formulario;

<form name="formulario" >
  ...
</form>

<form name="otro_formulario" >
  ...
</form>
```

Accediendo de esta forma a los formularios de la página, el script funciona correctamente aunque se reordenen los formularios o se añadan nuevos formularios a la página. Los elementos de los formularios también se pueden acceder directamente mediante su atributo name:

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

<form name="formulario">
  <input type="text" name="elemento" />
</form>
```

Esta será la opción mas usada. Veamos un ejemplo en el que usaremos la propiedad value para observar el valor de un elemento de un formulario:

Ej07.01a-FormularioCCC.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej07.01a-FormularioCCC.html -->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <style type="text/css">
  </style>
   <script language="javascript">
      function validar() {
         if (document.form1.ccc.value.length != 20) {
            alert (" El código NO es correcto!");
         }
         else {
            alert (" El código Es correcto!");
         }
      }
   </script>
</head>
<body>
   <form action="#" method="post" name="form1">
      Código Cuenta Corriente:
      <input type="text" name="ccc" />
      <input type="button" value="validar" onclick="validar();" />
   </form>
</body>
</html>
```

Obviamente, también se puede acceder a los formularios y a sus elementos utilizando las funciones DOM de acceso directo a los nodos. El siguiente ejemplo utiliza la habitual función document.getElementById() para acceder de forma directa a un formulario y a uno de sus elementos:

```
var formularioPrincipal = document.getElementById("formulario");
var primerElemento = document.getElementById("elemento");
<form name="formulario" id="formulario" >
  <input type="text" name="elemento" id="elemento" />
</form>
```

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

- **type**: indica el tipo de elemento que se trata. Para los elementos de tipo <input> (text, button, checkbox, etc.) coincide con el valor de su atributo type. Para las listas desplegables normales (elemento <select>) su valor es select-one, lo que permite diferenciarlas de las listas que permiten seleccionar varios elementos a la vez y cuyo tipo es select-multiple. Por último, en los elementos de tipo <textarea>, el valor de type es textarea.

- **form**: es una referencia directa al formulario al que pertenece el elemento. Así, para acceder al formulario de un elemento, se puede utilizar document.getElementById("id_del_elemento").form

- **name**: obtiene el valor del atributo name de XHTML. Solamente se puede leer su valor, por lo que no se puede modificar.

- **value**: permite leer y modificar el valor del atributo value de XHTML. Para los campos de texto (<input type="text"> y <textarea>) obtiene el texto que ha escrito el usuario. Para los botones obtiene el texto que se muestra en el botón. Para los elementos checkbox y radiobutton no es muy útil, como se verá más adelante

Por último, los eventos más utilizados en el manejo de los formularios son los siguientes:

- **onclick**: evento que se produce cuando se pincha con el ratón sobre un elemento. Normalmente se utiliza con cualquiera de los tipos de botones que permite definir XHTML (<input type="button">, <input type="submit">, <input type="image">).

- **onchange**: evento que se produce cuando el usuario cambia el valor de un elemento de texto (<input type="text"> o <textarea>). También se produce cuando el usuario selecciona una opción en una lista desplegable (<select>). Sin embargo, el evento sólo se produce si después de realizar el cambio, el usuario pasa al siguiente campo del formulario, lo que técnicamente se conoce como que "el otro campo de formulario ha perdido el foco".

- **onfocus**: evento que se produce cuando el usuario selecciona un elemento del formulario.
- **onblur**: evento complementario de onfocus, ya que se produce cuando el usuario ha deseleccionado un elemento por haber seleccionado otro elemento del formulario. Técnicamente, se dice que el elemento anterior "ha perdido el foco".

7.2 Utilidades básicas para formularios

7.2.1 Obtener el valor de los campos de formulario

La mayoría de técnicas JavaScript relacionadas con los formularios requieren leer y/o modificar el valor de los campos del formulario. Por tanto, a continuación se muestra cómo obtener el valor de los campos de formulario más utilizados.

7.2.1.1 TextArea.

El valor del texto mostrado por estos elementos se obtiene y se establece directamente mediante la propiedad value. Tenemos dos alternativas:
a) Usar, mediante DOM, el método getElementById.
b) Emplear la ruta del elemento: <document>.<nombre_formulario>.<nombre_input>.value

Ej07.02a-InputTexto.html

```
<!DOCTYPE html>
<html lang="es">        <!-Ej07.02a-InputTexto.html-->
<head>
  <meta charset="UTF-8" />
  <script language="javascript">
    function info(){
      var texto = document.getElementById("texto").value;
      alert (texto);
      var parrafo = document.formulario.parrafo.value;
      alert (parrafo);
    }
  </script>
</head>
<body>
  <form name="formulario">
    Usuario: <input type="text" id="texto" />
    Comentarios: <textarea id="parrafo"></textarea>
    <input type="button" value="Ver Inputs" onClick="info();" />
  </form>
</body> </html>
```

7.2.1.2 Radiobutton

Cuando se dispone de un grupo de radiobuttons, generalmente no se quiere obtener el valor del atributo value de alguno de ellos, sino que lo importante es conocer cuál de todos los radiobuttons se ha seleccionado. La propiedad checked devuelve true para el radiobutton seleccionado y false en cualquier otro caso. Si por ejemplo se dispone del siguiente grupo de radiobuttons:

Ej07.02b-InputRadio.html

```
<!DOCTYPE html>
<html lang="es">        <!-Ej07.02b-InputRadio.html-->
<head>
  <meta charset="UTF-8" />
  <script language="javascript">
    function ejecutar(){
      var i=0;    var sexoElegido;
      var inputs = document.getElementsByName("sexo");
      for(i=0; i<inputs.length; i++) {
        if (inputs[i].checked)
          sexoElegido = inputs[i].value;
      }
      alert(" Sexo Elegido: "+sexoElegido);    }
  </script>
<body>
  <form name="formulario"> Especifique Sexo: <br />
    <input type="radio" value="mujer" name="sexo" /> Mujer <br />
    <input type="radio" value="hombre" name="sexo" /> Hombre <br />
    <input type="radio" value="indeterminado" name="sexo" /> La Veneno <br />
    <input type="button" value="Ver Sexo" onClick="ejecutar();" />
  </form>
</body> </html>
```

7.2.1.3 Checkbox

Los elementos de tipo checkbox son muy similares a los radiobutton, salvo que en este caso se debe comprobar cada checkbox de forma independiente del resto. El motivo es que los grupos de radiobutton son mutuamente excluyentes y sólo se puede seleccionar uno de ellos cada vez. Por su parte, los checkbox se pueden seleccionar de forma independiente respecto de los demás.

Ej07.02c-InputCheckBox.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej07.02c-InputCheckBox.html -->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <style type="text/css">
  </style>
   <script language="javascript">
      function info() {
         var inputCondiciones = document.getElementById ("condiciones");
         var inputPrivacidad = document.getElementById ("privacidad");

         if (inputCondiciones.checked && inputPrivacidad.checked) {
            alert (" Puede Continuar...");
         }
         else {
            alert (" Ya no puedes continuar!!");
         }
      }
   </script>
</head>
<body>
   <form action="#" method="post" name="form1">
      Terminos del Contrato: <br />
      <input type="checkbox" value="condiciones" id="condiciones" />
      Acepta las Condiciones del Contrato. <br />
      <input type="checkbox" value="privacidad" id="privacidad" />
      He leido la politica de Privacidad y estoy de acuerdo.
      <input type="button" value="Ver " onclick="info();" />
   </form>
</body>
</html>
```

7.2.1.4 Select

Las listas desplegables (<select>) son los elementos en los que es más difícil obtener su valor. Si se dispone de una lista desplegable como la siguiente:

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

En general, lo que se requiere es obtener el valor del atributo value de la opción (<option>) seleccionada por el usuario. Obtener este valor no es sencillo, ya que se deben realizar una serie de pasos. Además, para obtener el valor seleccionado, deben utilizarse las siguientes propiedades:

options, es un array creado automáticamente por el navegador para cada lista desplegable y que contiene la referencia a todas las opciones de esa lista. De esta forma, la primera opción de una lista se puede obtener mediante document.getElementById("id_de_la_lista").options[0].

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

```
// Obtener la referencia a la lista
var lista = document.getElementById("opciones");

// Obtener el índice de la opción que se ha seleccionado
var indiceSeleccionado = lista.selectedIndex;
// Con el índice y el array "options", obtener la opción seleccionada
var opcionSeleccionada = lista.options[indiceSeleccionado];

// Obtener el valor y el texto de la opción seleccionada
var textoSeleccionado = opcionSeleccionada.text;
var valorSeleccionado = opcionSeleccionada.value;

alert("Opción seleccionada: " + textoSeleccionado + "\n Valor de la opción: " +
valorSeleccionado);
```

Como se ha visto, para obtener el valor del atributo value correspondiente a la opción seleccionada por el usuario, es necesario realizar varios pasos. No obstante, normalmente se abrevian todos los pasos necesarios en una única instrucción:

```
var lista = document.getElementById("opciones");
// Obtener el valor de la opción seleccionada
var valorSeleccionado = lista.options[lista.selectedIndex].value;
// Obtener el texto que muestra la opción seleccionada
var valorSeleccionado = lista.options[lista.selectedIndex].text;
```

Lo más importante es no confundir el valor de la propiedad selectedIndex con el valor correspondiente a la propiedad value de la opción seleccionada. En el ejemplo anterior, la primera opción tiene un value igual a 1. Sin embargo, si se selecciona esta opción, el valor de selectedIndex será 0, ya que es la primera opción del array options (y los arrays empiezan a contar los elementos en el número 0).

Ej07.02d-InputSelect.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej07.02d-InputSelect.html -->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <style type="text/css">
  </style>
   <script language="javascript">
      function ejecutar() {
         var lista = new Array();
         lista = document.getElementById ("opciones");
         var valorElegido = lista.options[lista.selectedIndex].value;
         var textoElegido = lista.options[lista.selectedIndex].text;
         alert("Opción Elegida: "+textoElegido+ "\n"+
              "El Value es: "+valorElegido);
      }
   </script>
</head>
<body>
   <form action="#" method="post" id="form1">
      Ciudad Elegida: <br />
      <select id="opciones" name="opciones">
         <option value="sevilla">Sevilla</option>
         <option value="malaga">Málaga</option>
         <option value="cordoba">Córdoba</option>
         <option value="granada">Granada</option>
      </select>
      <input type="button" value="Ver Ciudad" onclick="ejecutar();" />
   </form>
</body>
</html>
```

7.2.2 Establecer el foco en un elemento

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

Si un cuadro de texto de un formulario tiene el foco, el usuario puede escribir directamente en el sin necesidad de pinchar previamente con el ratón en el interior del cuadro. Igualmente, si una lista desplegable tiene el foco, el usuario puede seleccionar una opción directamente subiendo y bajando con las flechas del teclado.

Al pulsar repetidamente la tecla TABULADOR sobre una página web, los diferentes elementos (enlaces, imágenes, campos de formulario, etc.) van obteniendo el foco del navegador (el elemento seleccionado cada vez suele mostrar un pequeño borde punteado). Si en una página web el formulario es el elemento más importante, como por ejemplo en una página de búsqueda o en una página con un formulario para registrarse, se considera una buena práctica de usabilidad el asignar automáticamente el foco al primer elemento del formulario cuando se carga la página.

Para asignar el foco a un elemento de XHTML, se utiliza la función focus(). El siguiente ejemplo asigna el foco a un elemento de formulario cuyo atributo id es igual a primero:

```
document.getElementById("primero").focus();
<form id="formulario" action="#">
  <input type="text" id="primero" />
</form>
```

Ampliando el ejemplo anterior, se puede asignar automáticamente el foco del programa al tercer elemento del segundo formulario de la página, independientemente del id del formulario y de los elementos. De paso controlamos que dicho campo NO sea del tipo oculto (hidden)

Ej07.02e-InputFocus.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej07.02d-InputSelect.html -->
<head>
  <meta charset="UTF-8" />
  <style type="text/css">
   input:focus {  background: khaki; }
  </style>
   <script language="javascript">
      window.onload = function () {
         if (document.forms.length>0) {      // Controlo que haya al menos un formulario
            if (document.forms[1].elements.length>0) { // E2° Formulario tiene un campo...
               var campoFoco = document.forms[1].elements[2];
               if (campoFoco.type!="hidden") {      // El Campo del Foco NO esté oculto
                  campoFoco.focus();
               }
            }
         }
      }
   </script>
</head>
<body>
   <h3> Primer formulario</h3>
   <form action="#" method="post" id="form1">
      Campo1 <input type="text" /> <br />          Campo2 <input type="text" /> <br />
      Campo3 <input type="text" /> <br />          Campo4 <input type="text" /> <br />
   </form>
   <p> </p>
   <h3> Segundo formulario</h3>
   <form action="#" method="post" id="form2">
      Campo1 <input type="text" /> <br />          Campo2 <input type="text" /> <br />
      Campo3 <input type="text" /> <br />          Campo4 <input type="text" /> <br />
   </form>
</body>
</html>
```

7.2.3 Evitar el envío duplicado de un formulario

Uno de los problemas habituales con el uso de formularios web es la posibilidad de que el usuario pulse dos veces seguidas sobre el botón "Enviar". Si la conexión del usuario es demasiado lenta o la respuesta del servidor se hace esperar, el formulario original sigue mostrándose en el navegador y por ese motivo, el usuario tiene la tentación de volver a pinchar sobre el botón de "Enviar".

En la mayoría de los casos, el problema no es grave e incluso es posible controlarlo en el servidor, pero puede complicarse en formularios de aplicaciones importantes como las que implican transacciones económicas.

Por este motivo, una buena práctica en el diseño de aplicaciones web suele ser la de deshabilitar el botón de envío después de la primera pulsación. El siguiente ejemplo muestra el código necesario:

Ej07.02f-FormDuplicado.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej07.02f-FormDuplicado.html -->
<head>
  <meta charset="UTF-8" />
  <style type="text/css">
   input:focus {  background: khaki;
                  color: black;}
  </style>
   <script language="javascript">
      function bloqueaBoton(boton) {
         boton.disabled=true;
         boton.value='Se está enviando...';
         document.form[0].submit();
      }
   </script>
</head>
<body>
   <h3> Primer formulario</h3>
   <form action="maneja.php" method="post" id="form1">
      Campo1 <input type="text" /> <br />
      Campo2 <input type="text" /> <br />
      Campo3 <input type="text" /> <br />
      <input type="button" value="Enviar Formulario"
            onclick="bloqueaBoton(this);" />
            <!-- También es válido poner (quitando la parte en negrita del Script):
            onclick="this.disabled=true;
                     this.value='Se está enviando...';
                     this.form.submit();" />
            -->
   </form>
</body>
</html>
```

Cuando se pulsa sobre el botón de envío del formulario, se produce el evento onclick sobre el botón y por tanto, se ejecutan las instrucciones JavaScript contenidas en el atributo onclick:

- En primer lugar, se deshabilita el botón mediante la instrucción this.disabled = true;. Esta es la única instrucción necesaria si sólo se quiere deshabilitar un botón.
- A continuación, se cambia el mensaje que muestra el botón. Del original "Enviar" se pasa al más adecuado "Enviando..."
- Por último, se envía el formulario mediante la función submit() en la siguiente instrucción: this.form.submit()

El botón del ejemplo anterior está definido mediante un botón de tipo <input type="button" />, ya que el código JavaScript mostrado no funciona correctamente con un botón de tipo <input type="submit" />. Si se utiliza un botón de tipo submit, el botón se deshabilita antes de enviar el formulario y por tanto el formulario acaba sin enviarse.

7.2.4 Limitar el tamaño de caracteres de un textarea

La carencia más importante de los campos de formulario de tipo textarea es la imposibilidad de limitar el máximo número de caracteres que se pueden introducir, de forma similar al atributo maxlength de los cuadros de texto normales.

JavaScript permite añadir esta característica de forma muy sencilla. En primer lugar, hay que recordar que con algunos eventos (como onkeypress, onclick y onsubmit) se puede evitar su comportamiento normal si se devuelve el valor false.

Evitar el comportamiento normal equivale a modificar completamente el comportamiento habitual del evento. Si por ejemplo se devuelve el valor false en el evento onkeypress, la tecla pulsada por el usuario no se tiene en cuenta. Si se devuelve false en el evento onclick de un elemento como un enlace, el navegador no carga la página indicada por el enlace.

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

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

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

Sin embargo, en el siguiente ejemplo:

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

Como el valor devuelto por el evento onkeypress es igual a false, el navegador no ejecuta el comportamiento por defecto del evento, es decir, la tecla presionada no se transforma en ningún carácter dentro del textarea. No importa las veces que se pulsen las teclas y no importa la tecla pulsada, ese textarea no permitirá escribir ningún carácter.

Aprovechando esta característica, es sencillo limitar el número de caracteres que se pueden escribir en un elemento de tipo textarea: se comprueba si se ha llegado al máximo número de caracteres permitido y en caso afirmativo se evita el comportamiento habitual del evento y por tanto, los caracteres adicionales no se añaden al textarea:

Ej07.02g-FormMaxCaracteres.html

```
<!DOCTYPE html>
<html lang="es">        <!-Ej07.02g-FormMaxCaracteres.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <style type="text/css">
    textarea {    width: 150px;      height: 100px;
              border: 2px solid blue;
    }
  </style>
  <script language="javascript">
    function limita(maximoCaracteres) {
      var elemento = document.getElementById("texto");
      if(elemento.value.length >= maximoCaracteres )
        return false;
      else
        return true;
    }
  </script>
<body>
  <form id="formulario" action="#">
    <textarea id="texto" onkeypress="return limita(50);"></textarea>
</form>
</body>
</html>
```

En el ejemplo anterior, con cada tecla pulsada se compara el número total de caracteres del textarea con el máximo número de caracteres permitido. Si el número de caracteres es igual o mayor que el límite, se devuelve el valor false y por tanto, se evita el comportamiento por defecto de onkeypress y la tecla no se añade.

Ejercicio 17

Mejorar el ejemplo anterior indicando en todo momento al usuario el número de caracteres que aún puede escribir. Además, se debe permitir pulsar las teclas Backspace, Supr. y las flechas horizontales cuando se haya llegado al máximo número de caracteres.

7.2.5 Restringir los caracteres permitidos en un cuadro de texto

En ocasiones, puede ser útil bloquear algunos caracteres determinados en un cuadro de texto. Si por ejemplo un cuadro de texto espera que se introduzca un número, puede ser interesante no permitir al usuario introducir ningún carácter que no sea numérico.

Igualmente, en algunos casos puede ser útil impedir que el usuario introduzca números en un cuadro de texto. Utilizando el evento onkeypress y unas cuantas sentencias JavaScript, el problema se resuelve fácilmente: **Ej07.02h-RestringeValores.html**

```
<!DOCTYPE html>
<html lang="es">                    <!-Ej07.02h-RestringeValores.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <script language="javascript">
  function permite(elEvento, permitidos) {
    // Variables que definen los caracteres permitidos
    var numeros = "0123456789";
    var caracteres = " abcdefghijklmnñopqrstuvwxyzABCDEFGHIJKLMNÑOPQRSTUVWXYZ";
    var numeros_caracteres = numeros + caracteres;

    var teclas_especiales = [8, 37, 39, 46];
    // 8 = BackSpace, 46 = Supr, 37 = flecha izquierda, 39 = flecha derecha


    // Seleccionar los caracteres a partir del parámetro de la función
    switch(permitidos) {
      case 'num':
        permitidos = numeros;
        break;
      case 'car':
        permitidos = caracteres;
        break;
      case 'num_car':
        permitidos = numeros_caracteres;
        break;
    }

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

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

```
    // Comprobar si la tecla pulsada se encuentra en los caracteres permitidos
    // o si es una tecla especial
    return permitidos.indexOf(caracter) != -1 || tecla_especial;
}
  </script>
<body>
  <form id="formulario" action="#">
    Input Sólo números
    <input type="text" id="texto" onkeypress="return permite(event, 'num')" /> <br />

    Input Sólo letras
    <input type="text" id="texto" onkeypress="return permite(event, 'car')" /> <br />

    Input Sólo letras o números
    <input type="text" id="texto" onkeypress="return permite(event, 'num_car')" />
</form>
</body>
</html>
```

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

Si el carácter se encuentra dentro de los caracteres permitidos, se devuelve true y por tanto el comportamiento de onkeypress es el habitual y la tecla se escribe. Si el carácter no se encuentra dentro de los caracteres permitidos, se devuelve false y por tanto se impide el comportamiento normal de onkeypress y la tecla no llega a escribirse en el input.

Además, el script anterior siempre permite la pulsación de algunas teclas especiales. En concreto, las teclas BackSpace y Supr para borrar caracteres y las teclas Flecha Izquierda y Flecha Derecha para moverse en el cuadro de texto siempre se pueden pulsar independientemente del tipo de caracteres permitidos. Os paso el ejemplo anterior simplificado (sin caracteres especiales):

Ej07.02h-RestringeValores_OpcionB.html

```
<!DOCTYPE html>
...
   <script language="javascript">
      function permite(evento,valoresPermitidos) {
         var numeros = "0123456789";
         var caracteres = "ABCDEFGHIJKLMNOPQRSTUVWXYZ ";
         var num_caracteres = numeros+caracteres;
         var permitidos = "";

         switch (valoresPermitidos) {
            case "letras":
               permitidos = caracteres;     break;
            case "num":
               permitidos = numeros;        break;
            case "letras_num":
               permitidos = num_caracteres; break;
         }
         var codigoCaracter = evento.charCode || evento.keyCode;
         var caracterPulsado = String.fromCharCode (codigoCaracter);

         // Busco en Permitidos el caracterPulsado; Si lo encuentra devuelve 0; si NO, -1
         var encontrado = permitidos.indexOf(caracterPulsado);
         if (encontrado==-1) {
            return false;
         }
         else {
            return true;
         }
      }
   </script>
</head>
...
</html>
```

7.3 Validación

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

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

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

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

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

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

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

```
function validacion() {
  if (condicion que debe cumplir el primer campo del formulario) {
    // Si no se cumple la condicion...
    alert('[ERROR] El campo debe tener un valor de...');
    return false;
  }
  ...
  else if (condicion que debe cumplir el último campo del formulario) {
    // Si no se cumple la condicion...
    alert('[ERROR] El campo debe tener un valor de...');
    return false;
  }
  // Si el script ha llegado a este punto, todas las condiciones
  // se han cumplido, por lo que se devuelve el valor true
  return true;
}
```

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

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

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

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

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

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

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

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

7.3.1 Validar un campo de texto obligatorio

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

Ej07.03a-ValidaCampoTexto.html

```
<!DOCTYPE html>
<html lang="es">       <!--Ej07.03a-ValidaCampoTexto.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      function valida() {
         var inputs = new Array ();
         var i;
         var resultado = true;
         // Defino una expresión regular para obligar a escribir algo menos espacios...
         var expresion = /^\s+$/;

         inputs = document.getElementsByClassName ("acceso");
         for (i=0; i<inputs.length; i++) {
            // Este if controla que el campo esté VACÍO
            if (inputs[i].value=="") {
               resultado = false;
            }
            // Este if valida que el campo NO esté lleno de espacios...
            if (expresion.test(inputs[i].value)==true) {
               resultado = false;
            }
         }
         return resultado;
      }
   </script>
</head>
<body>
   <h3> Acceso Página:</h3>
   <form action="maneja.php" method="post" id="form1" onsubmit="return valida();">
      Usuario: <input type="text" name="usuario" class="acceso" /> <br />
      Contraseña: <input type="password" name="clave" class="acceso" /> <br />
      <input type="submit" value="Comprobar Usuario" />
   </form>
</body>
</html>
```

Para que se de por completado un campo de texto obligatorio, se comprueba que el valor introducido sea válido, que el número de caracteres introducido sea mayor que cero y que no se hayan introducido sólo espacios en blanco.

La primera parte de la condición obliga a que el texto introducido tenga una longitud superior a cero caracteres, esto es, que no sea un texto vacío.

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

7.3.2 Validar un campo de texto con valores numéricos

Se trata de obligar al usuario a introducir un valor numérico en un cuadro de texto.

Ej07.03b-ValidaCampoNum.html

```
<!DOCTYPE html>
<html lang="es">                    <!-Ej07.03b-ValidaCampoNum.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <script language="javascript">
    //valida que el campo no este vacio y no tenga solo espacios en blanco
    function valida() {
      var valorCampo = document.getElementById ("nombre").value;

      if( isNaN(valorCampo) || valorCampo == "") {
        alert (" El campo tiene caracteres NO numéricos");
        return false;
      }
      else
      {
        alert (" Todo Correcto!");
        return true;
      }
    }
  </script>
<body>
  <form id="formulario" action="#" onsubmit="return valida()">
    Nombre<input type="text" id="nombre"/> <br />
    <input type="submit" value="Enviar" />
</form>
</body>
</html>
```

Si el contenido de la variable valorCampo no es un número válido, no se cumple la condición. La ventaja de utilizar la función interna isNaN() es que simplifica las comprobaciones, ya que JavaScript se encarga de tener en cuenta los decimales, signos, etc.

7.3.3 Validar que se ha seleccionado una opción de una lista

Se trata de obligar al usuario a seleccionar un elemento de una lista desplegable. El siguiente código JavaScript permite conseguirlo: **Ej07.03c-ValidaLista.html**

```
<!DOCTYPE html>
<html lang="es">                    <!-Ej07.03c-ValidaLista.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <script language="javascript">
    function valida () {
      var opcion = document.getElementById("opciones").selectedIndex;
      if( opcion  == null || opcion  == 0 ) {
        alert (" Ninguna Opción Seleccionada!")
        return false;
      }
      else {
        alert (" Todo Correcto!");
        return true;
      }
    }
  </script>
<body>
  <form id="formulario" action="#" onsubmit="return valida()">
    <select id="opciones" name="opciones">
      <option value="">- Seleccionar Ciudad -</option>
      <option value="sevilla">Sevilla</option>
      <option value="malaga">Málaga</option>
      <option value="granada">Granada</option>
    </select>
    <input type="submit" value="Enviar" />
  </form>
</body> </html>
```

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

7.3.4 Validar una dirección de email

Se trata de obligar al usuario a introducir una dirección de email con un formato válido. Por tanto, lo que se comprueba es que la dirección parezca válida, ya que no se comprueba si se trata de una cuenta de correo electrónico real y operativa. La condición JavaScript consiste en:
Ej07.03d-ValidaCorreo.html

```
<!DOCTYPE html>
<html lang="es">                    <!-Ej07.03d-ValidaCorreo.html-->
<head>
  <meta charset="UTF-8" />
  <script language="javascript">
    function valida () {
      var resultado = true;
      valorInput = document.getElementById("correo").value;
      alert (valorInput);

      var expresion = /^\w+([\.\-\_]?\w+)*@\w+([\.\-]?\w+)*(\.\D{2,4})+$/;

      if (expresion.test(valorInput)){
        alert("La dirección de email " + valorInput + " es correcta.");
      }
      else {
        alert("La dirección de email es incorrecta.");
      }
      return resultado;
    }
  </script>
```

```
<body>
  <form id="formulario" action="#" onsubmit="return valida()">
    Correo Electrónico: <input type="text" id="correo" />
    <input type="submit" value="Enviar" />
  </form>
</body>
</html>
```

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

Toda la implementación del ejemplo estará en montar la expresión regular. Para ello debemos de tener en cuenta varias cosas.

En primer lugar La expresion regular delimita su inicio con /^ y su fin con $/. Por lo que una expresión regular tendría la forma:
`/^ expr regular $/`

El email se compone de tres partes:
`nombre usuario + @ + servidor + dominio`

Veamos que consideraciones debemos de tener con cada una de esas partes.

Nombre

- Debe de empezar por letra o numero. Al menos tiene una letra o número. La letra o número se expresa mediante el carácter \w. Para asegurarnos de que la letra o número aparece al menos una vez utilizaremos el modificador +.
- Puede contener puntos (\.) y guiones (_ y \-) además de las letras y números. Esta combinación podrá aparecer, es por ello que utilizaremos el modificador * (cero o varias veces). Insertaremos toda la combinación entre paréntesis.

Su expresión regular, para ambos casos, será la siguiente:
`\w+([\.\-\_]?\w+)*`

Servidor

- Debe aparece luego la @ seguido del nombre del servidor que puede contener puntos o guiones normales seguidos de otras combinaciones de letras/números.
  Por ejemplo: yahoo-server2 o miservidor-2

La expresión regular resultante es:
`@\w+([\.-]?\w+)*`

Dominio

Irá al final, detrás de un punto. Podrá tener dos (.es, .fr, .it,...) o tres letras (.com, .net, .org,..) o cuatro (.mobi, info,...). Si queremos indicar un intervalor concreto de caracteres lo expresamos con el número entre los operadores { y }, en este caso de 2 a 4 caracteres.

Además podemos tener varios dominios seguidos (.com.ar, .com.uk,....) es por ello que deberemos de usar el modificador +. Ya que el dominio podrá aparecer varias veces.
Mediante la D mayúscula indicamos que va sólo letras...
Su expresión regular queda de este modo:
`(\.\D{2,4})+`

7.3.5 Validar una fecha

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

Ej07.03e-ValidaFecha.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej07.03e-ValidaFecha.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <style type="text/css">
      #d, #m { width: 35px; }
      #a { width: 70px; }
   </style>
   <script language="javascript">
      function valida() {
         var d = parseInt(document.getElementById("d").value);
         var m = parseInt(document.getElementById("m").value-1);
         var a = parseInt(document.getElementById("a").value);
         var fecha = new Date(a,m,d);

         if (isNaN(fecha)) {
            alert (" ERROR, no es una fecha válida!");
            return false;
         }
         if (d<1 || d>31 || m<0 || m>11 || a<1900 || a>2013) {
            alert (" ERROR, Se excede del Rango de Fechas!");
            return false;
         }
         if (m==1 && d>29) {
            alert (" ERROR, Febrero llega a 29 Días!");
            return false;
         }
         else {
            return true;
         }
      }
   </script>
</head>
<body>
   <form action="maneja.php" method="post" id="form1" onsubmit="return valida();">
      Especifique Fecha de Nacimiento: <br />
      Día <input type="text" maxlength="2" id="d" /> <br />
      Mes <input type="text" maxlength="2" id="m" /> <br />
      Año <input type="text" maxlength="4" id="a" /> <br />
      <input type="submit" value="Comprobar" />
   </form>
</body>
</html>
```

La función Date(a, m, d) es una función interna de JavaScript que permite construir fechas a partir del año, el mes y el día de la fecha. Es muy importante tener en cuenta que el número de mes se indica de 0 a 11, siendo 0 el mes de Enero y 11 el mes de Diciembre. Los días del mes siguen una numeración diferente, ya que el mínimo permitido es 1 y el máximo 31.

La validación consiste en intentar construir una fecha con los datos proporcionados por el usuario. Si los datos del usuario no son correctos, la fecha no se puede construir correctamente y por tanto la validación del formulario no será correcta.

7.3.6 Validar un número de DNI

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

Ej07.03f-ValidaDNI.html

```
<!DOCTYPE html>
<html lang="es">                    <!-Ej07.03d-ValidaDNI.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <style type="text/css">
    #nif {  width: 140px; }
  </style>
  <script language="javascript">
    function valida () {
      var resultado = true;

      var valorInput = document.getElementById("nif").value;

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

      var expresion = /^\d{8}[A-Z]$/;
      var numDNI = valorInput.substring(0, 8);
      var letraNIF = valorInput.charAt(8);
      var indice = numDNI % 23;

      if( !(expresion.test(valorInput)) ) {
        alert (" Formato DNI Incorrecto!");
        resultado =  false;
      }
      else {
        if( letraNIF != letras[indice] ) {
          alert (" Letra Incorrecta!");
          resultado = false;
        }
        else {
          alert (" Todo Correcto!");
        }
      }

      return resultado;
    }
  </script>
<body>
  <form id="formulario" action="#" onsubmit="return valida()">
    NIF: <input type="text" maxlength="9" id="nif" />
    <input type="submit" value="Enviar" />
  </form>
</body>
</html>
```

7.3.7 Validar un número de teléfono

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

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

Ej07.03g-ValidaTfno.html

```
<!DOCTYPE html>
<html lang="es">                    <!-Ej07.03f-ValidaTfno.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <style type="text/css">
    #tfno {  width: 140px; }
  </style>
  <script language="javascript">
    function valida () {
      var resultado = true;
      valorInput = document.getElementById("tfno").value;
      var expresion = /^\d{9}$/;

      if( !(expresion.test(valorInput)) ) {
        alert (" Teléfono Incorrecto!");
        resultado = false;
      }
      else{
        alert (" Todo Correcto...");
      }
      return resultado;
    }
  </script>
<body>
  <form id="formulario" action="#" onsubmit="return valida()">
    Teléfono: <input type="text" maxlength="9" id="tfno" />
    <input type="submit" value="Enviar" />
  </form>
</body>
</html>
```

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

Número	Expresión regular	Formato
900900900	`/^\d{9}$/`	9 cifras seguidas
900-900-900	`/^\d{3}-\d{3}-\d{3}$/`	9 cifras agrupadas de 3 en 3 y separadas por guiones
900 900900	`/^\d{3}\s\d{6}$/`	9 cifras, las 3 primeras separadas por un espacio
900 90 09 00	`/^\d{3}\s\d{2}\s\d{2}\s\d{2}$/`	9 cifras, las 3 primeras separadas por un espacio, las siguientes agrupadas de 2 en 2
(900) 900900	`/^\(\d{3}\)\s\d{6}$/`	9 cifras, las 3 primeras encerradas por paréntesis y un espacio de separación respecto del resto
+34 900900900 0034 900900900	`/^\+\d{2,4}\s\d{9}$/` `-----------`	Prefijo internacional (+ seguido de 2 o 3 cifras), espacio en blanco y 9 cifras consecutivas

Podéis encontrar un documento excelente que explica las expresiones regulares aquí:
https://sites.google.com/site/dwebtodojs/referencia/expresiones-regulares

8 OTRAS UTILIDADES

Además de los formularios y de todas las funciones y utilidades de JavaScript que se han visto, existen muchas otras utilidades que es necesario conocer para desarrollar aplicaciones completas. Como no es posible estudiar todas las herramientas que se pueden crear con JavaScript, a continuación se muestran algunas de las utilidades básicas más comunes.

8.1 Relojes, contadores e intervalos de tiempo

En ocasiones, algunas páginas web muestran un reloj con la hora actual. Si el reloj debe actualizarse cada segundo, no se puede mostrar la hora directamente en la página HTML generada por el servidor. En este caso, aunque existen alternativas realizadas con Java y con Flash, la forma más sencilla de hacerlo es mostrar la hora del ordenador del usuario mediante JavaScript.

Para crear y mostrar un reloj con JavaScript, se debe utilizar el objeto interno Date() para crear fechas/horas y las utilidades que permiten definir contadores, para actualizar el reloj cada segundo.

El objeto Date() es una utilidad que proporciona JavaScript para crear fechas y horas. Una vez creado un objeto de tipo fecha, es posible manipularlo para obtener información o realizar cálculos con las fechas. Para obtener la fecha y hora actuales, solamente es necesario crear un objeto Date() sin pasar ningún parámetro:

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

Utilizando el código anterior, se puede construir un reloj muy básico que no actualiza su contenido:

Ej08.01a-HoraLocal.html

```
<!DOCTYPE html>
<html lang="es">                    <!-Ej08.01a-HoraLocal.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <style type="text/css">
    #reloj {  width: 200px; height: 100px;
              color: blue;
              font: Arial 15px bold;
              border: 2px solid red;}
  </style>
  <script language="javascript">
    function verHora () {
      var fechaHora = new Date();
      document.getElementById("reloj").innerHTML = fechaHora;
    }
  </script>
<body>
  <div id="reloj"></div>
  <input type="button" value="Ver Hora" onclick="verHora();" />
</body>
</html>
```

Este primer reloj construido presenta muchas diferencias respecto al reloj que se quiere construir. En primer lugar, muestra más información de la necesaria. Además, su valor no se actualiza cada segundo, por lo que no es un reloj muy práctico.

El objeto Date() proporciona unas funciones muy útiles para obtener información sobre la fecha y la hora. Concretamente, existen funciones que devuelven la hora, los minutos y los segundos:

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

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

<div id="reloj" />
```

Utilizando las funciones getHours(), getMinutes() y getSeconds() del objeto Date, el reloj solamente muestra la información de la hora. El resultado del ejemplo anterior sería un reloj como el siguiente:

```
20:9:21
```

Si la hora, minuto o segundo son menores que 10, JavaScript no añade el 0 por delante, por lo que el resultado no es del todo satisfactorio. El siguiente código soluciona este problema añadiendo un 0 cuando sea necesario:

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

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

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

<div id="reloj" />
```

Ahora el reloj muestra correctamente la hora:

```
20:14:03
```

Si se quiere mostrar el reloj con un formato de 12 horas en vez de 24, se puede utilizar el siguiente código:

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

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

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

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

<div id="reloj" />
```

Utilizando el código anterior, el reloj ya no muestra la hora como 20:14:03, sino que la muestra en formato de 12 horas y con el sufijo adecuado:

```
08:14:03 pm
```

En la página siguiente está el código completo para ser probado:

Ej08.01b-HoraCompleta.html

```
<!DOCTYPE html>
<html lang="es">                    <!-Ej08.01b-HoraCompleta.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <style type="text/css">
    #reloj {  width: 100px;   height: 30px;
              color: blue;    font: Arial 15px bold;
              border: 2px solid red;}
  </style>
  <script language="javascript">
    function verHora () {
      var caja = document.getElementById("reloj");
     var fechaHora = new Date();
      var horas = fechaHora.getHours();
      var minutos = fechaHora.getMinutes();
      var segundos = fechaHora.getSeconds();

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

      if(horas < 10) { horas = '0' + horas; }
      if(minutos < 10) { minutos = '0' + minutos; }
      if(segundos < 10) { segundos = '0' + segundos; }
      caja.innerHTML = horas+':'+minutos+':'+segundos +sufijo;
    }
  </script>
<body>
  <div id="reloj"></div>
  <input type="button" value="Ver Hora" onclick="verHora();" />
</body>
</html>
```

Para completar el reloj, sólo falta que se actualice su valor cada segundo. Para conseguirlo, se deben utilizar unas funciones especiales de JavaScript que permiten ejecutar determinadas instrucciones cuando ha transcurrido un determinado espacio de tiempo.

La función setTimeout() permite ejecutar una función una vez que haya transcurrido un periodo de tiempo indicado. La definición de la función es:

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

La función que se va a ejecutar se debe indicar mediante su nombre sin paréntesis y el tiempo que debe transcurrir hasta que se ejecute se indica en milisegundos. De esta forma, si se crea una función encargada de mostrar la hora del reloj y se denomina muestraReloj(), se puede indicar que se ejecute dentro de 1 segundo mediante el siguiente código:

```
function muestraReloj() {
  var fechaHora = new Date();
  var horas = fechaHora.getHours();
  var minutos = fechaHora.getMinutes();
  var segundos = fechaHora.getSeconds();

  if(horas < 10) { horas = '0' + horas; }
  if(minutos < 10) { minutos = '0' + minutos; }
  if(segundos < 10) { segundos = '0' + segundos; }
  document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;
}
setTimeout(muestraReloj, 1000);
...
<div id="reloj" />
```

No obstante, el código anterior simplemente muestra el contenido del reloj 1 segundo después de que se pulse el botón, por lo que no es muy útil. Para ejecutar una función de forma periódica, se utiliza una función de JavaScript muy similar a setTimeout() que se denomina setInterval(). Su definición es:

```
setInterval(nombreFuncion, milisegundos);
```

La definición de esta función es idéntica a la función setTimeout(), salvo que en este caso, la función programada se ejecuta infinitas veces de forma periódica con un lapso de tiempo entre ejecuciones de tantos milisegundos como se hayan establecido.

Así, para construir el reloj completo, se establece una ejecución periódica de la función muestraReloj() cada segundo (esta vez el reloj aparece al cargar la página):

Ej08.01c-Hora.html

```
<!DOCTYPE html>
<html lang="es">                    <!--Ej08.01c-Hora.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <style type="text/css">
      #reloj {        width: 100px;         height: 40px;
               color: blue;          font: Arial 15px bold;
               border: 2px solid red;
      }
   </style>
   <script language="javascript">
      function hora() {
         setInterval(verHora,1);
      }

      function verHora() {
         var fechaHora = new Date ();
         var caja = document.getElementById ("reloj");

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

         // Para cambiar a formato de 12 Horas
         var sufijo = " AM";
         if (horas>12) {
            horas = horas - 12;
            sufijo = " PM";
         }

         if (horas<10)               { horas = '0'+horas;          }
         if (minutos<10)      { minutos = '0'+minutos;      }
         if (segundos<10)     { segundos = '0'+segundos;    }
         caja.innerHTML =
            horas + ":" + minutos + ":" + segundos + ":" + milisegundos + sufijo;
      }
   </script>
</head>
<body>
   <div id="reloj"></div>
   <input type="button" value="Ver Hora" onclick="hora();" />
</body>
</html>
```

Empleando el objeto Date y sus funciones, es posible construir "cuentras atrás", es decir, relojes que muestran el tiempo que falta hasta que se produzca un evento. Además, las funciones setTimeout() y setInterval() pueden resultar muy útiles en otras técnicas de programación.

Ejercicio 17B

Crear un cronometro que arranque y pare mediante sendos botones.

8.1.1 Métodos del Objeto Date

El objeto Date proporciona métodos para procesar fechas y horas.
El formato tendrá dos constructores:

- new Date ( );
- new Date (año, mes, dia, [, horas, [, minutos [, segundos [, milisegundos]]]])

Para este segundo caso si queremos meter el 2001/11/4 a las 14:20:25.750 seria
ATENCIÓN: El mes se pone con una unidad menos, de Enero (0) a Diciembre (12)

```
var unaFecha = new Date ( 2001, 10, 6, 14, 20 , 25, 750);
```

- **getDate( )** → Devuelve el día del mes.
- **getDay( )** → Devuelve el día de la semana (Domingo → 0; Sábado → 6)
- **getHours( )** → Retorna 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.

También podemos ver los milisegundos entre dos fechas:
ms=fecha1.getTime() – fech2.gettime ( )

- **getYear ( )** → Retorna el año, al que se le ha restado 1900. Por ejemplo, para el 1995 retorna 95, para el 2005 retorna 105. Este método está obsoleto en Netscape a partir de la versión 1.3 de Javascript y ahora se utiliza getFullYear().
- **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.
- **setTime( )** → Actualiza la fecha completa. Recibe un número de milisegundos desde el 1 de enero de 1970.

- **setYear ( )** → Cambia el año recibe un número, al que le suma 1900 antes de colocarlo como año de la fecha. Por ejemplo, si recibe 95 colocará el año 1995. Este método está obsoleto a partir de Javascript 1.3 en Netscape. Ahora se utiliza setFullYear(), indicando el año con todos los dígitos.
- **setFullYear ( )** → Cambia el año de la fecha al número que recibe por parámetro. El número se indica completo ej: 2005 o 1995. Utilizar este método para estar seguros que todo funciona para fechas posteriores a 2000.
- **getTimeZoneOffset** → Expresa la diferencia horaria entre la hora del Sistema y la hora Oficial UTC (la del Meridiano de Greenwich).
- **toUTCString** → Devuelve la fecha en cadena de caracteres según la Hora del Tiempo Universal (ITC).
- **toLocaleString** → Devuelve la Hora del Sistema.

Mas información: http://www.cristalab.com/tutoriales/metodos-del-objeto-date-de-javascript-c105282l/

Ejercicio 18

Usando el método getTime, pedir al usuario 2 fechas (dia, mes y año) mediante un formulario (incluir 3 inputs para Fecha1 y 3 inputs para Fecha2) y devolver el número de días transcurridos.

8.2 El Objeto Math

El objeto Math se crea por el motor de la secuencia de comandos al ser cargado. No necesita New para ser instanciado (aunque se recomienda usarlo para ver los métodos en el IDE).

8.2.1 Métodos del Objeto Math.

- **abs** → Devuelve el valor absoluto de una expresión. Ejemplo:
```
var numero = - 75;
alert ("El valor absoluto es: " + Math.abs (numero));
```

- **acos** → Presenta el arcocoseno
- **asin** → arcoseno
- **atan** → arcotangente
- **atan2** → arcotangente al cuadrado
- **cos** → coseno
- **sin** → seno
- **tan** → tangente.
  Los ángulos especificados en los parámetros o devueltos por los métodos son radianes.
```
var angulo = 1;     // En Radianes
alert ("El seno es: " + Math.sin (angulo));
```

- **ceil** → Redondea por exceso una expresión // 120.8 → 121
- **floor** → Redondea por defecto (trunca) una expresión // 120.8 → 120
- **round** → Redondea un número decimal. // 120.8 → 121 (auto)
- **max** → Devuelve el valor mayor de dos expresiones
- **min** → Devuelve el valor menor de dos expresiones.
- **sqrt** → Devuelve la raiz cuadradada del argumento.
- **pow** → Realiza la potenciación. Argumentos: base y exponente.
- **exp** → Realiza el cálculo del número e elevado a exponente.
- **log** → Devuelve el logaritimo natural (en base e) del argumento.

Ej08.02a-MinMax.html

```
<!DOCTYPE html>
<html lang="es">                    <!-Ej08.02a-MinMax.html-->
<head>
  <meta charset="UTF-8" />
  <style type="text/css">
  </style>
  <script language="javascript">
    function ejecutar () {
      var valor1 = document.getElementById ("v1").value;
      var valor2 = document.getElementById ("v2").value;
      minmax (valor1,valor2);
    }

    function minmax (num1, num2) {
      var mayor = Math.max(num1,num2);
      var menor = Math.min(num1,num2);
      alert (" Mayor: "+mayor+"; Menor: "+menor);
    }
  </script> </head>
<body>
    <form id="formulario" action="#">
    Valor1: <input type="text" maxlength="9" id="v1" />
    Valor2: <input type="text" maxlength="9" id="v2" />
    <input type="submit" value="Enviar" onclick="ejecutar();" />
  </form>
</body>
</html>
```

- **random** → Devuelve un pseudoaletorio entre 0 y 1

Ej08.02b-Aleatorios.html

```
<!DOCTYPE html>
<html lang="es">       <!--Ej08.02b-Aleatorios.html-->
<head>
  <meta charset="UTF-8" />
   <script language="javascript">
      function ejecutar() {
         var inferior = parseInt (document.getElementById ("num1").value);       // 101
         var superior = parseInt (document.getElementById ("num2").value)+1;     // 200+1
         if (inferior<=superior) {
            aleatorio (inferior,superior);
         }
         else {
            alert ("ERROR! Limite Inferior debe ser menor que Límite Superior");
         }
      }

      // Parámetros: valor1 -> Limite Inferior; valor2 -> Límite Superior
      function aleatorio(valor1, valor2) {
         var aleatorio = 0;
         var resultado;

         var posibilidades = valor2 - valor1;       // (10+1) - 1 = 10 posibilidades
            l (Math.random() * posibilidades);
         resultado = aleatorio + (valor1-1);              // Debe salir el límite inferior
         alert (resultado);
      }
   </script>
</head>
<body>
   Generador de Números Aleatorios: <br />
   Limite inferior: <input type="text" id="num1" /> <br />
   Limite Superior: <input type="text" id="num2" /> <br />
   <input type="button" value="Generar Aleatorio" onclick="ejecutar();" />
</body>
</html>
```

Ejercicio 18 Especial

A partir del código anterior de los números aleatorios realizar la siguiente comprobación (para ver que está bien implementado): Asignando por código los valores inferior y superior (por ejemplo 1 y 5) ver el resultado de 1000 peticiones de números aleatorios entre ambos valores y comprobar las veces que sale cada uno de los números. Sacar los porcentajes de salida de los mismos y comprobar que son parecidos.

Ejercicio 18 B (opcional)

Crear una aplicación que será EL BINGO (80 posibilidades).
Previamente hay que crear los cartones que tienen 24 números.

8.2.2 Propiedades del Objeto Math.

Las propiedades del objeto Math son las constantes matemáticas. Son estás:

Constante	Definición	Valor aproximado
Math.PI	p, relación entre circunferencia y diámetro de un círculo	3.14159265359
Math.E	e, constante de Euler (base de los logaritmos naturales)	2,71828182846
Math.LN2	Logaritmo natural de 2	0,69314718056
Math.LOG2E	Logaritmo en base 2 de e, constante de Euler	1,44269504089
Math.LN10	Logaritmo natural de 10	2,30258509299
Math.LOG10E	Logaritmo en base 10 de e, constante de Euler	0,43429448190
Math.SQRT2	Raíz cuadrada de 2	1,41421356237
Math.SQRT1_2	Raíz cuadrada de 1/2	0,70710678119

8.3 Tooltip

Los tooltips son pequeños recuadros de información que aparecen al posicionar el ratón sobre un elemento. Normalmente se utilizan para ofrecer información adicional sobre el elemento seleccionado o para mostrar al usuario pequeños mensajes de ayuda. Los tooltips son habituales en varios elementos de los sistemas operativos:

Realizar un tooltip completo mediante JavaScript es una tarea muy compleja, sobre todo por la dificultad de mostrar el mensaje correctamente en función de la posición del ratón. Afortunadamente, existen scripts que ya están preparados para generar cualquier tipo de tooltip. La librería que se va a utilizar se denomina overLIB, y se puede descargar desde su página web principal:
http://www.bosrup.com/web/overlib/
El enlace directo a la descarga actual es:http://sourceforge.net/projects/overlib/files/overlib/overlib421/

La descarga incluye todos los scripts necesarios pero no su documentación, que se puede consultar en: http://www.bosrup.com/web/overlib/?Documentation
La referencia de todos los comandos disponibles se puede consultar en:
http://www.bosrup.com/web/overlib/?Command_Reference

A continuación se indican los pasos necesarios para incluir un tooltip básico en cualquier página web:

1) Descargar el ZIP y descomprimirlo en la carpeta de la práctica, dentro de una subcarpeta. Por ejemplo, dicha subcarpeta se llamará scripts
2) Enlazar los archivos JavaScript requeridos

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

Los tooltips sólo requieren que se enlace un único archivo JavaScript (overlib.js). El comentario que incluye el código XHTML es el aviso de copyright de la librería, que es obligatorio incluirlo para poder usarla.

3) Definir el texto que activa el tooltip y su contenido:

```
<p>Lorem ipsum dolor sit amet, <a href="#" onmouseover="return overlib('Prueba de
un tooltip básico y muy sencillo.');" onmouseout="return nd();">consectetuer
adipiscing elit</a>. Etiam eget metus. Proin varius auctor tortor. Cras augue
neque, porta vitae, vestibulum nec, pulvinar id, nibh. Fusce in arcu. Duis vehicula
nonummy orci.</p>
```

Los tooltip se incluyen como enlaces de tipo <a> para los que se definen los eventos onmouseover y onmouseout. Cuando el ratón se pasa por encima del enlace anterior, se muestra el tooltip con el aspecto por defecto:

La librería overLIB permite configurar completamente el aspecto y comportamiento de cada tooltip. Las opciones se indican en cada tooltip y se incluyen como parámetros al final de la llamada a la función overlib():

```
<a href="#" onmouseover="return overlib('Prueba de un tooltip básico y muy
sencillo.', CENTER);" onmouseout="return nd();">consectetuer adipiscing elit</a>
```

NOTA: Para desactivar el tooltip hay que incluir la función nd() (por ejemplo en onmouseout).

El parámetro CENTER indica que el tooltip se debe mostrar centrado respecto de la posición del ratón. Otra opción que se puede controlar es la anchura del tooltip.
Por defecto, su anchura es de 200px, pero la opción WIDTH permite modificarla:

```
<a href="#" onmouseover="return overlib('Prueba de un tooltip básico y muy
sencillo.', CENTER, WIDTH, 120);" onmouseout="return nd();">consectetuer adipiscing
elit</a>
```

El uso de los parámetros de configuración en la propia llamada a la función que muestra los tooltips no es recomendable cuando el número de parámetros empieza a ser muy numeroso. Afortunadamente, overLIB permite realizar una única configuración que se utilizará para todos los tooltips de la página.

La configuración global consiste en llamar a la función **overlib_pagedefaults()** con todos los parámetros de configuración deseados. Por tanto, el código JavaScript necesario para realizar los cambios configurados hasta el momento sería:

```
<script type="text/javascript" src="js/overlib.js">
//overLIB (c) Erik Bosrup
</script>
<script type="text/javascript">
overlib_pagedefaults(CENTER, WIDTH, 120);
</script>
...
<p>Lorem ipsum dolor sit amet, <a href="#" onmouseover="return overlib('Prueba de
un tooltip básico y muy sencillo.');" onmouseout="return nd();">consectetuer
adipiscing elit</a>. Etiam eget metus. Proin varius auctor tortor. Cras augue
neque, porta vitae, vestibulum nec, pulvinar id, nibh. Fusce in arcu. Duis vehicula
nonummy orci.</p>
```

En el ejemplo anterior se ha modificado el tamaño y tipo de letra y el color de fondo del tooltip mediante la siguiente configuración:

```
overlib_pagedefaults(WIDTH,150,FGCOLOR,'#ffffcc',BGCOLOR,'#666666',TEXTFONT,"Arial,
Helvetica, Verdana",TEXTSIZE,".8em");
```

Ahora veamos un ejemplo completo: **Ej08.03a-Tooltip.html**

```
<!DOCTYPE html>
<html lang="es">                    <!--Ej08.03a-Tooltip.html -->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
  <style type="text/css">
    .campos { width: 100px; }
    .campos:focus { background: khaki;  }
  </style>
  <script type="text/javascript" src="scripts/overlib.js">
    // overLIB (c) Erik Bosrup
  </script>
 <script language="JavaScript">
    function ejecutar() {}
    overlib_pagedefaults(CENTER, WIDTH,150, TEXTCOLOR, 'red',
                         FGCOLOR,'yellow', BGCOLOR,'blue',
                         TEXTFONT,"Verdana",TEXTSIZE,"10px");
  </script>
</head>
<body>
  Usuario <input class="campos" type="text" name="Usuario"
          onmouseover="return overlib('Escriba Usuario sin espacios');"/> <br />
  Contraseña <input class="campos" type="text" name="clave"
          onmouseover="return overlib('Escriba clave sin espacios');"/> <br />
  <input type="button" value="Ver Ancho" onClick="ejecutar();" />
</body>
</html>
```

Ejercicio 18C

Mejorar el tooltip propuesto añadiendo las siguientes características:

- El tooltip se mostrará con un retraso de un cuarto de segundo (DELAY).
- Incluir una separación horizontal de 15 píxel entre el puntero del ratón y el tooltip (OFFSETX)
- Que el tooltip se muestre en la parte superior del puntero del ratón (pista: ABOVE)
- Cambiar colores de Fondo, letra y del marco al gusto.

8.4 Menú desplegable

La navegación de algunas páginas web se realiza mediante menús desplegables, que disponen de varios niveles jerárquicos que se despliegan a medida que el usuario pasa el puntero del ratón por encima de cada elemento. Aunque CSS permite realizar menús desplegables horizontales y verticales de cualquier nivel de profundidad, los navegadores de la familia Internet Explorer versión 6.0 y anteriores no disponen del suficiente soporte de CSS como para crear este tipo de menús.

De esta forma, la única forma de crear un menú desplegable que funcione correctamente en cualquier navegador consiste en utilizar JavaScript. A pesar de que realizar un menú desplegable sencillo con JavaScript no es una tarea muy compleja, se va a utilizar un componente ya realizado que pertenece a la librería de desarrollo web de Yahoo.

La Yahoo UI Library, conocida como YUI, y que se puede traducir como "Librería de componentes de interfaz de usuario de Yahoo" es un conjunto de utilidades y controles escritos en JavaScript para el desarrollo rápido y sencillo de aplicaciones web complejas.

La librería completa está dividida en módulos y componentes relacionados con CSS, DOM, eventos, AJAX, etc. Entre las utilidades disponibles se encuentran calendarios, tooltips, cuadros que autocompletan el texto, árboles jerárquicos, etc. Además de esas utilidades, la YUI incluye un completo módulo de menús que permite realizar decenas de tipos de menús diferentes: horizontales, verticales, desplegables, estáticos, menús contextuales, menús de aplicación, menús realizados con XHTML o con JavaScript, etc.

La librería YUI (actualmente la versión 3.9) se puede descargar gratuitamente en:
http://developer.yahoo.com/yui/
Los tutoriales sobre los menús (y mucho mas) se pueden encontrar en:
http://yuilibrary.com/yui/docs/tutorials/
Se pueden ver decenas de ejemplos en
http://yuilibrary.com/yui/docs/examples/

Veamos un ejemplo completo: **Ej08.04a-MenuYUI.html**

```
<!DOCTYPE html>
<html lang="en" class="yui3-loading">
<head>
    <meta charset="utf-8">
    <title>Ejemplo Menú Desplegable YUI</title>
    <link rel="stylesheet" href="http://yui.yahooapis.com/combo?3.9.0/build/cssreset/reset-
min.css&amp;3.9.0/build/cssfonts/fonts-min.css&amp;3.9.0/build/cssgrids/grids-
min.css&amp;3.9.0/build/cssbase/base-min.css">
    <link rel="stylesheet" href="../assets/node-menunav/node-menunav-examples-base.css">
    <script src="http://yui.yahooapis.com/3.9.0/build/yui/yui-min.js"></script>
    <script>
    // Para cargar el plugin y sus dependencias...
    YUI().use('node-menunav', function(Y) {

    // Creo el menú y le asigno una ID
    var menu = Y.one("#menudesplegable");
    menu.plug(Y.Plugin.NodeMenuNav);

    //  Se muestra el menú
    menu.get("ownerDocument").get("documentElement").removeClass("yui3-loading");

    });

</script>
</head>
<body class="yui3-skin-sam">
<h1>Ejemplo Menú Vertical</h1>
<div class="yui3-u" id="main">
```

```
<div id="menudesplegable" class="yui3-menu yui3-menu-horizontal yui3-menubuttonnav">
     <div class="yui3-menu-content">
         <ul class="first-of-type">
           <li class="yui3-menuitem">
              <a class="yui3-menuitem-content" href="#">Portada</a>
            </li>

           <li>
             <a class="yui3-menu-label" href="#web"><em>Diseño Web</em></a>
             <div id="web" class="yui3-menu">
                 <div class="yui3-menu-content">
                   <ul>
                     <li class="yui3-menuitem">
                     <a class="yui3-menuitem-content" href="#">Introducción</a></li>
                     <li class="yui3-menuitem">
                     <a class="yui3-menuitem-content" href="#">Proyectos</a></li>
                     <li>
                         <a class="yui3-menu-label" href="#html">HTML</a>
                         <div id="html" class="yui3-menu">
                           <div class="yui3-menu-content">
                               <ul>
                                   <li class="yui3-menuitem">
                              <a class="yui3-menuitem-content" href="#">Etiquetas</a></li>
                                   <li class="yui3-menuitem">
                              <a class="yui3-menuitem-content" href="#">Atributos</a></li>
                               </ul>
                               <ul>
                                   <li class="yui3-menuitem">
                              <a class="yui3-menuitem-content" href="#">Texto</a></li>
                                   <li class="yui3-menuitem">
                              <a class="yui3-menuitem-content" href="#">Enlaces</a></li>
                                   <li class="yui3-menuitem">
                              <a class="yui3-menuitem-content" href="#">Listas</a></li>
                               </ul>

                               <ul>
                                   <li class="yui3-menuitem">
                              <a class="yui3-menuitem-content" href="#">Imágenes</a></li>
                                   <li class="yui3-menuitem">
                              <a class="yui3-menuitem-content" href="#">Tablas</a></li>
                                   <li class="yui3-menuitem">
                              <a class="yui3-menuitem-content" href="#">Formularios</a></li>
                               </ul>
                            </div>
                         </div>
                     </li>

                     <li class="yui3-menuitem">
                        <a class="yui3-menuitem-content" href="#">CSS</a> </li>
                     <li class="yui3-menuitem">
                        <a class="yui3-menuitem-content" href="#">JavaScript 1.8</a></li>
                     <li class="yui3-menuitem">
                        <a class="yui3-menuitem-content" href="#">MySQL</a></li>
                     <li class="yui3-menuitem">
                        <a class="yui3-menuitem-content" href="#">PHP</a></li>
                   </ul>
                 </div>
             </div>
           </li>
```

```
            <li>
                <a class="yui3-menu-label" href="#find-info"><em>Aplicaciones</em></a>
                <div id="find-info" class="yui3-menu">
                    <div class="yui3-menu-content">
                        <ul>
                            <li class="yui3-menuitem">
                            <a class="yui3-menuitem-content" href="#">C++</a></li>
                            <li class="yui3-menuitem">
                            <a class="yui3-menuitem-content" href="#">Java 5</a></li>
                            <li class="yui3-menuitem">
                            <a class="yui3-menuitem-content" href="#">Android 2/3</a></li>
                        </ul>
                    </div>
                </div>
            </li>

            <li>
                <a class="yui3-menu-label" href="#find-info"><em>Acerca de</em></a>
                <div id="find-info" class="yui3-menu">
                    <div class="yui3-menu-content">
                        <ul>
                            <li class="yui3-menuitem">
                            <a class="yui3-menuitem-content" href="#">Sobre mi</a></li>
                            <li class="yui3-menuitem">
                             <a class="yui3-menuitem-content" href="#">Mis Enlaces </a></li>
                            <li class="yui3-menuitem">
                             <a class="yui3-menuitem-content" href="#">Bibliografía</a></li>
                        </ul>
                    </div>
                </div>
            </li>
          </ul>
        </div>
    </div>
</body>
</html>
```

8.5 Galerías de imágenes (Lightbox)

Muchos sitios web utilizan galerías de imágenes para mostrar sus productos y servicios. Este tipo de galerías muestran una serie de miniaturas de imágenes que se amplían al pinchar sobre cada imagen. Hasta hace poco, la técnica más utilizada para construir una galería consistía en incluir las miniaturas en la página y abrir cada imagen grande en una ventana emergente (o pop-up) o en una nueva página.

El uso de técnicas basadas en JavaScript ha supuesto una revolución en la creación de las galerías de imágenes y ha permitido mejorar la experiencia de navegación del usuario. La técnica se conoce como Lightbox y fue creada originalmente por Lokesh Dhakar. Lightbox es una técnica muy sencilla de utilizar, funciona correctamente en todos los navegadores y permite mantener la semántica del documento (no ensucia la página con código JavaScript).

El código de la versión más reciente se puede descargar gratuitamente en:
http://lokeshdhakar.com/projects/lightbox2/
La descarga incluye el código fuente del script, todos los archivos JavaScript externos necesarios, archivos CSS e imágenes de prueba y una breve pero completa documentación.

A continuación se indican los pasos necesarios para incluir Lightbox en una página web:

1) Nos bajamos el ZIP con todo lo necesario:
   http://lokeshdhakar.com/projects/lightbox2/releases/lightbox2.51.zip
   Incluye Jquery, algunas fotos de ejemplo y el propio lightbox
2) Descomprimimos el ZIP por ejemplo en la carpeta
   /opt/lampp/htdocs/ejemplosJavaScript/tema08 (aparece la carpeta lightbox)
3) Abrimos el archivo lightbox.js y cambiamos las líneas 51,52, 55 y 56 por estas:
```
this.fileLoadingImage = 'lightbox/images/loading.gif';
this.fileCloseImage = 'lightbox/images/close.png';
this.labelImage = "Imagen";
this.labelOf = "de";
```

Por último, un código de ejemplo: **Ej08.05a-GaleriaFotos.html**

```
<!DOCTYPE html>
<html lang="es">        <!--Ej08.05a-GaleriaFotos.html-->
<head>
    <meta charset="utf-8">
   <link rel="stylesheet" href="lightbox/css/screen.css"
        type="text/css" media="screen" />

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

   <script src="lightbox/js/jquery-1.7.2.min.js"></script>
    <script src="lightbox/js/lightbox.js"></script>

</head>
<body>

  <h3>Ejemplo de Galería</h3>
<div class="imageRow">  <!-- Para presentalo en fila -->
  <div class="set">     <!-- Definimos un set de fotos -->

    <div class="single first">
        <a href="lightbox/images/examples/image-3.jpg"
         rel="lightbox[plantas]" title="Planta Naranja">
          <img src="lightbox/images/examples/thumb-3.jpg"
            alt="Plantas: Imagen 1 de 4" /></a>
      </div>
```

```
    <div class="single">
        <a href="lightbox/images/examples/image-4.jpg"
         rel="lightbox[plantas]" title="Una Fuente." >
          <img src="lightbox/images/examples/thumb-4.jpg"
            alt="Plantas: Imagen 2 de 4" /></a>
    </div>
    <div class="single">
        <a href="lightbox/images/examples/image-5.jpg"
         rel="lightbox[plantas]" title="Plantas Rojas.">
          <img src="lightbox/images/examples/thumb-5.jpg"
            alt="Plantas: Imagen 3 de 4" /></a>
    </div>

    <div class="single last">
        <a href="lightbox/images/examples/image-6.jpg"
         rel="lightbox[plantas]" title="Un Amanecer">
          <img src="lightbox/images/examples/thumb-6.jpg"
            alt="Plantas: Imagen 4 de 4" /></a>
    </div>

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

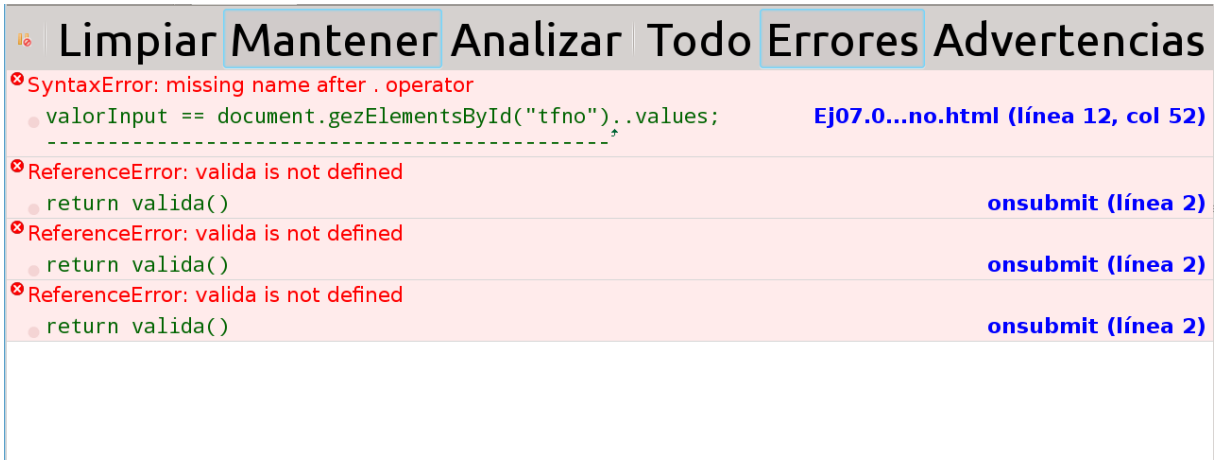
Ejercicio 18 B

Crear una galería de fotos dentro de una tabla de 4x4 (16 celdas = 16 Fotos).

8.6 Corrección de errores con Firefox

Depurar scripts utilizando Firefox es una experiencia completamente diferente y más sencilla que depurarlos con Internet Explorer. Firefox proporciona herramientas más útiles, activadas por defecto y que muestran más información y mucho más precisa.

Ademas, firefox permite instalar pequeñas mejoras y ampliaciones en el navegador, que se conocen con el nombre de extensiones. Una de las extensiones más interesantes para los desarrolladores de aplicaciones web es Firebug, que se puede descargar gratuitamente desde http://www.getfirebug.com/



Firebug incluye cientos de utilidades y herramientas necesarias para depurar aplicaciones web y para diseñar páginas web. Además, proporciona información detallada sobre XHTML, CSS, DOM y JavaScript. Toda la documentación, tutoriales y preguntas frecuentes sobre Firebug se pueden encontrar en http://www.getfirebug.com/docs.html

9 CANVAS

La API Canvas se hace cargo del aspecto gráfico y lo hace de una forma extremadamente efectiva. Canvas nos permite dibujar, presentar gráficos en pantalla, animar y procesar imágenes y texto, y trabaja junto con el resto de la especificación para crear aplicaciones completas e incluso video juegos en 2 y 3 dimensiones para la web. Ejemplos en Canvas:
http://net.tutsplus.com/articles/web-roundups/21-ridiculously-impressive-html5-canvas-experiments/

9.1 Preparando el lienzo

Lo primero que debemos hacer es incluir la etiqueta <canvas>. Este elemento genera un espacio rectangular vacío en la página web (lienzo) en el cual serán mostrados los resultados de ejecutar los métodos provistos por la API. Cuando escreado, produce sólo un espacio en blanco, como un elemento vacío, pero con un propósito totalmente diferente.

```
<canvas id="lienzo" width="500" height="300">
   Su navegador no soporta el elemento canvas
</canvas>
```

El contenido entre las etiquetas de apertura y cierre de canvas se mostrará si el API Canvas no está disponible en el navegador.

getContext()

Este método genera un contexto de dibujo que será asignado al lienzo. A través de la referencia que retorna podremos aplicar el resto de la API. El método puede tomar dos valores: 2d y 3d. Esto es, por supuesto, para ambientes de 2 dimensiones y 3 dimensiones. Por el momento solo el contexto 2d está disponible, pero serios esfuerzos están siendo volcados en el desarrollo de una API estable en 3 dimensiones.

El contexto de dibujo del lienzo será una grilla de pixeles listados en filas y columnas de arriba a abajo e izquierda a derecha, con su origen (el pixel 0,0) ubicado en la esquina superior izquierda del lienzo.

Veamos un ejemplo completo:

Ej09.11-LienzoCanvas.html

```
<!DOCTYPE html>
<html lang="es">
<head>
   <title>Canvas API</title>
   <script language="javascript">
      function iniciar(){
         // MUY IMPORTANTE: Hay que declarar elemento sin el var (dejándolo
         // como variable global). Si no se hace así, Komodo no saca los métodos.
        elemento=document.getElementById('lienzo');

        // Si no pongo var lienzo, esta variable se convierte en global
        lienzo=elemento.getContext('2d');
      }

      // Esto equivale al window.onload = function ()
      window.addEventListener("load", iniciar, false);

    </script>
</head>
<body>
   <section id="cajalienzo">
      <canvas id="lienzo" width="500" height="300">
         Su navegador no soporta el elemento canvas
      </canvas>
   </section>
</body>
</html>
```

9.2 Dibujando en el lienzo

Luego de que el elemento <canvas> y su contexto han sido inicializados podemos finalmente comenzar a crear y manipular gráficos. La lista de herramientas provista por la API para este propósito es extensa, desde la creación de simples formas y métodos de dibujo hasta texto, sombras o transformaciones complejas. Vamos a estudiarlas una por una.

9.2.1 Dibujando rectángulos.

Normalmente el desarrollador deberá preparar la figura a ser dibujada en el contexto (como veremos pronto), pero existen algunos métodos que nos permiten dibujar directamente en el lienzo, sin preparación previa. Estos métodos son específicos para formas rectangulares y son los únicos que generan una forma primitiva (para obtener otras formas tendremos que combinar otras técnicas de dibujo y trazados complejos). Los métodos disponibles son los siguientes:

- **fillRect(x, y, ancho, alto)** → Este método dibuja un rectángulo sólido. La esquina superior izquierda será ubicada en la posición especificada por los atributos x e y. Los atributos ancho y alto declaran el tamaño.
- **strokeRect(x, y, ancho, alto)** → Similar al método anterior, éste dibujará un rectángulo vacío (solo su contorno).
- **clearRect(x, y, ancho, alto)** → Esta método es usado para substraer pixeles del área especificada por sus atributos. Es un borrador rectangular.

Veamos un ejemplo (a partir del anterior)

Ej09.21-RectangulosCanvas.html

```
<!DOCTYPE html>
<html lang="es">           <!-- Ej09.021a-RectangulosCanvas.html-->
...
   <script language="javascript">
      function iniciar(){
        elemento=document.getElementById('lienzo');
        // Si no pongo var lienzo, esta variable se convierte en global
        lienzo=elemento.getContext('2d');

         lienzo.fillRect (50,50,100,50);

         // Dibujo un Rectángulo y otros 5 iguales
         lienzo.strokeRect (201,101,100,50);
         for (x=200,y=100, w=100, h=50;
             x<300,y<200, w<200, h<150;
             x=x+20, y=y+20, w=w+20, h=h+20) {
            lienzo.strokeRect (x,y,w,h);
         }

         // Con ClearRect borro 20x20px del 1er Rectángulo...
         lienzo.clearRect (50,50,20,20);
      }
...
</html>
```

Los rectángulos son dibujados en el lienzo en la posición declarada por los atributos x e y, y uno sobre el otro de acuerdo al orden en el código.

NOTA IMPORTANTE: Como se ha visto en los códigos anteriores, hay que definir tanto la variable elemento como lienzo como globales **DENTRO DE LA FUNCIÓN** iniciar ( ). Si no se hace así, puede suceder dos cosas:

1. Si se declara la variable elemento y lienzo global, no aparecen sus métodos (programar se vuelve fatigoso).
2. Si se declaran ambas variables como globales FUERA DE LA FUNCIÓN iniciar, la aplicación, sencillamente, deja de funcionar.

9.2.2 Colores.

Hasta el momento hemos usado el color otorgado por defecto, negro sólido, pero podemos especificar el color que queremos aplicar mediante sintaxis CSS utilizando las siguientes propiedades:

- **strokeStyle** → Esta propiedad declara el color para el contorno de la figura.
- **fillStyle** → Esta propiedad declara el color para el interior de la figura.
- **globalAlpha** → Esta propiedad no es para definir color sino transparencia. Especifica la transparencia para todas las figuras dibujadas en el lienzo.

```
lienzo.fillStyle="#000099";
lienzo.strokeStyle="#990000";
```

Hemos usados valores decimales, pero también podemos emplear rgb() o transparencias con rgba(). Estos métodos deben ser siempre escritos entre comillas. Ejemplo, `strokeStyle="rgba(255,165,0,1)")`.

Cuando un nuevo color es especificado se vuelve el color por defecto para el resto de los dibujos, a menos que volvamos a cambiarlo más adelante.

A pesar de que el uso de la función rgba() es posible, existe otra propiedad más específica para declarar el nivel de transparencia: globalAlpha. Su sintaxis es **globalAlpha=valor**, donde valor es un número entre 0.0 (totalmente opaco) y 1.0 (totalmente transparente).

Veamos un ejemplo (a partir de los anteriores)

Ej09.22-RectangulosColoresCanvas.html

```
<!DOCTYPE html>
<html lang="es">         <!-- Ej09.22-RectangulosColoresCanvas.html-->
<head>
   <title>Canvas API</title>
   <script language="javascript">
      window.onload = function () {
         iniciar();
      }

      function iniciar(){
        elemento=document.getElementById('lienzo');

        // Si no pongo var lienzo, esta variable se convierte en global
        lienzo=elemento.getContext('2d');

         // Propiedades (Colores)
         lienzo.fillStyle = "pink";  // Relleno
         lienzo.strokeStyle = "blue" // Contorno

         lienzo.fillRect (100,50,100,50);    // Relleno Rectángulo
         lienzo.strokeRect (100,50,100,50);  // Bordes Rectángulo

         lienzo.strokeStyle = "red"          // Cambio el color
         lienzo.clearRect (110,60,20,20);    // Quito un cacho...
         lienzo.strokeRect (110,60,20,20);   // Quito un cacho...
      }

    </script>
</head>
<body>
   <section id="cajalienzo">
      <canvas id="lienzo" width="500" height="300">
         Su navegador no soporta el elemento canvas
      </canvas>
   </section>
</body>
</html>
```

9.2.3 Gradientes

Los Gradientes son una herramienta esencial en cualquier programa de dibujo estos días, y esta API no es la excepción. Así como en CSS3, los gradientes en la API Canvas pueden ser lineales o radiales, y pueden incluir puntos de terminación para combinar colores. Métodos:

- **createLinearGradient(x1, y1, x2, y2)** → Este método crea un objeto que luego será usado para aplicar un gradiente lineal al lienzo.
- **createRadialGradient(x1, y1, r1, x2, y2, r2)** → Este método crea un objeto que luego será usado para aplicar un gradiente circular o radial al lienzo usando dos círculos. Los valores representan la posición del centro de cada círculo y sus radios.
- **addColorStop(posición, color)** → Este método especifica los colores a ser usados por el gradiente. El atributo posición es un valor entre 0.0 y 1.0 que determina dónde la degradación comenzará para ese color en particular.

Veamos un ejemplo completo: **Ej09.23-GradientesCanvas.html**

```
<!DOCTYPE html>
<html lang="es">          <!-- Ej09.23-GradientesCanvas.html-->
<head>
   <script language="javascript">
      function iniciar() {
        elemento=document.getElementById('lienzo');
        lienzo=elemento.getContext('2d');

        // Defino el Degradado Lineal
        var gradienteLineal = lienzo.createLinearGradient (100,100,250,200);
        gradienteLineal.addColorStop (1,"blue");
        gradienteLineal.addColorStop (0.3,"green");
        gradienteLineal.addColorStop (0.6,"pink");
        gradienteLineal.addColorStop (0,"red");

        // Defino el Degradado Radial
        var gradienteRadial = lienzo.createRadialGradient (300,100,50,450,200,50);
        gradienteRadial.addColorStop (0.5,"yellow");
        gradienteRadial.addColorStop (1,"deepPink");

        lienzo.fillStyle = gradienteLineal;
        lienzo.fillRect(100,100,150,100);

        lienzo.fillStyle = gradienteRadial;
        lienzo.fillRect(300,100,150,100);
     }

      // Esto equivale al window.onload = function ()
      window.addEventListener("load", iniciar, false);
    </script>
</head>
<body>
      <canvas id="lienzo" width="500" height="300">
         Su navegador no soporta el elemento canvas
      </canvas>
</body>
</html>
```

Creamos el objeto gradiente desde la posición 0,0 a la 10,100, otorgando una leve inclinación hacia la izquierda. Los colores fueron declarados por el método **addColorStop()** y el gradiente logrado fue finalmente aplicado a la propiedad fillStyle, como un color regular.

Las posiciones del gradiente son correspondientes al lienzo, no a las figuras que queremos afectar. El resultado es que si movemos los rectángulos dibujados al final de la función hacia una nueva posición, el gradiente para esos triángulos cambiará.
Por otro lado, para cambiar de gradiente lineal a radial, debemos hacer esto:

```
        //var gradiente=lienzo.createLinearGradient(0,0,10,100);
        var gradiente=lienzo.createRadialGradient(0,0,30,0,0,300)
```

9.2.4 Creando trazados.

Los métodos estudiados hasta el momento dibujan directamente en el lienzo, pero ese no es siempre el caso. Normalmente tendremos que procesar figuras en segundo plano y una vez que el trabajo esté hecho enviar el resultado al contexto para que sea dibujado. Con este propósito, API Canvas introduce varios métodos con los que podremos generar trazados.

Un trazado es como un mapa a ser seguido por el lápiz. Una vez declarado, el trazado será enviado al contexto y dibujado de forma permanente en el lienzo. El trazado puede incluir diferentes tipos de líneas, como líneas rectas, arcos, rectángulos, entre otros, para crear figuras complejas.

Existen dos métodos para comenzar y cerrar el trazado:

- **beginPath()** → Este método comienza la descripción de una nueva figura. Es llamado en primer lugar, antes de comenzar a crear el trazado.
- **closePath()** → Este método cierra el trazado generando una línea recta desde el último punto hasta el punto de origen. Puede ser ignorado cuando utilizamos el método fill() para dibujar el trazado en el lienzo.

También contamos con tres métodos para dibujar el trazado en el lienzo:

- **stroke()** → Este método dibuja el trazado como una figura vacía (solo el contorno).
- **fill()** → Este método dibuja el trazado como una figura sólida. Cuando usamos este método no necesitamos cerrar el trazado con closePath() , el trazado es automáticamente cerrado con una línea recta trazada desde el punto final hasta el origen.
- **clip()** → Este método declara una nueva área de corte para el contexto. Cuando el contexto es inicializado, el área de corte es el área completa ocupada por el lienzo. El método clip() cambiará el área de corte a una nueva forma creando de este modo una máscara. Todo lo que caiga fuera de esa máscara no será dibujado.

Para crear el trazado y la figura real que será enviada al contexto y dibujada en el lienzo, contamos con varios métodos disponibles:

- **moveTo(x, y)** → Este método mueve el lápiz a una posición específica para continuar con el trazado. Nos permite comenzar o continuar el trazado desde diferentes puntos, evitando líneas continuas.
- **lineTo(x, y)** → Este método genera una línea recta desde la posición actual del lápiz hasta la nueva declarada por los atributos x e y.
- **rect(x, y, an, al)**→ Este método genera un rectángulo. A diferencia de los métodos estudiados anteriormente, éste generará un rectángulo que formará parte del trazado (no directamente dibujado en el lienzo). Los atributos tienen la misma función.
- **arc(x, y, radio, ángulo inicio, ángulo final, dirección)** → Este método genera un arco o un círculo en la posición x e y , con un radio y desde un ángulo declarado por sus atributos. El último valor es un valor booleano (falso o verdadero) para indicar la dirección a favor o en contra de las agujas del reloj.

Ej09.24a-TrazadoCanvas.html

```
<!DOCTYPE html><html lang="es">          <!-- Ej09.24a-TrazadoCanvas.html-->
<head>
   <script language="javascript">
      function iniciar()   {
        elemento=document.getElementById('lienzo');
        lienzo=elemento.getContext('2d');

        lienzo.beginPath();          // Inicio Trazado
          lienzo.moveTo(100,100);    // Primer punto
          lienzo.lineTo(200,200);    // Línea al 2° Punto (hipotenusa)
          lienzo.lineTo(100,200);    // Línea al 3er Punto (Cateto1)
        lienzo.closePath();          // Cierro el triángulo...  (Cateto2)
        lienzo.stroke();             // Fin Trazado        }
```

```
      // Esto equivale al window.onload = function ()
      window.addEventListener("load", iniciar, false);
    </script>
</head>
<body>
   <section id="cajalienzo">
      <canvas id="lienzo" width="500" height="300">
         Su navegador no soporta el elemento canvas
      </canvas>
   </section>
</body>
</html>
```

Usando el método stroke() al final de nuestro trazado dibujamos un triángulo vacío en el lienzo. Para lograr una figura sólida, este método debe ser reemplazado por fill() :

Ej09.24b-TrazadoCanvas.html

```
<!DOCTYPE html>
...
         lienzo.closePath();          // Cierro el triángulo...  (Cateto2)
         lienzo.fill();               // Fin Trazado (con relleno)  }
...
```

Ahora la figura en la pantalla será un triángulo sólido. El método fill() cierra el trazado automáticamente, por lo que ya no tenemos que usar closePath() para lograrlo.

Uno de los métodos mencionados anteriormente para dibujar un trazado en el lienzo fue clip(). Este método en realidad no dibuja nada, lo que hace es crear una máscara con la forma del trazado para seleccionar qué será dibujado y qué no. Todo lo que caiga fuera de la máscara no se dibujará en el lienzo. Veamos un ejemplo:

Ej09.24c-MascaraCanvas.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej09.24c-TrazadoCanvas.html-->
<head>
    <meta charset="utf-8">
   <style type="text/css">
      canvas { background: PapayaWhip; }     </style>
   <script language="javascript">
      var x = 100;
      function dibujar() {
         elemento = document.getElementById ("lienzo");
         lienzo = elemento.getContext ("2d");

         lienzo.beginPath();
            lienzo.moveTo (x,100);
            lienzo.lineTo (x+100,200);
            lienzo.lineTo (100,200);
         lienzo.closePath();
         lienzo.clip();

         lienzo.strokeStyle = "blue";
         for (y=10; y<300; y=y+10) {
            lienzo.moveTo (0,y);
            lienzo.lineTo (500,y);
         }
         lienzo.stroke();            // Termino las líneas horizontales
         x=x+20;                     // Modifico la máscara cada vez que pulso en Dibujar...
      }
   </script>
</head>
<body>
   <canvas id="lienzo" width="500px" height="300px"></canvas> <br />
   <input type="button" value="Dibujar" onclick="dibujar();" />
</body> </html>
```

Para mostrar exactamente cómo funciona el método clip(), en el código anterior utilizamos un bucle for para crear líneas horizontales cada 10 pixeles. Estas líneas van desde el lado izquierdo al lado derecho del lienzo, pero solo las partes de las líneas que caen dentro de la máscara (el triángulo) serán dibujadas.

Ahora que ya sabemos cómo dibujar trazados, es tiempo de ver el resto de las alternativas con las que contamos para crearlos. Hasta el momento hemos estudiado cómo generar líneas rectas y formas rectangulares. Para figuras circulares, la API provee tres métodos: arc(), quadraticCurveTo() y bezierCurveTo(). El primero es relativamente sencillo y puede generar círculos parciales o completos, como mostramos en el siguiente ejemplo:

Ej09.24d-TrazadoCircularCanvas.html

```
<!DOCTYPE html>
<html lang="es">           <!-- Ej09.24d-TrazadoCircularCanvas.html-->
...
      function iniciar(){
        elemento=document.getElementById('lienzo');
        lienzo=elemento.getContext('2d');

         var misGrados = window.prompt (" Escriba los grados: ");
         var grados = Math.PI/180 * misGrados;
         lienzo.beginPath();
            lienzo.arc (200, 200, 50, 0, grados, false);
         lienzo.stroke();
      }
...
</body>
</html>
```

Lo primero que vemos en el método **arc()** en nuestro ejemplo es el uso del valor PI. Este método usa radianes en lugar de grados para los valores del ángulo. En radianes, el valor PI representa 180 grados, por lo que la formula PI*2 multiplica PI por 2 obteniendo un ángulo de 360 grados. Podemos probar a cambiar los grados que tendrá la circunferencia e incluso cambiar false (dibuja sólo lo indicado) por true (dibuja el círculo completo menos lo indicado.

Una cosa importante a considerar es que si continuamos construyendo el trazado luego del arco, el actual punto de comienzo será el final del arco. Si no deseamos que esto pase tendremos que usar el método moveTo() para cambiar la posición del lápiz, como hicimos anteriormente. Sin embargo, si la próxima figura es otro arco (por ejemplo, un círculo completo) siempre recuerde que el método moveTo() mueve el lápiz virtual hacia el punto en el cual el círculo comenzará a ser dibujado, no el centro del círculo. Digamos que el centro del círculo que queremos dibujar se encuentra en el punto 300,150 y su radio es de 50. El método moveTo() debería mover el lápiz a la posición 350,150 para comenzar a dibujar el círculo.

Además de arc(), existen dos métodos más para dibujar curvas, en este caso curvas complejas. El método quadraticCurveTo() genera una curva Bézier cuadrática, y el todo bezierCurveTo() es para curvas Bézier cúbicas. La diferencia entre estos dos métodos es que el primero cuenta con un solo punto de control y el segundo con dos, creando de este modo diferentes tipos de curvas.

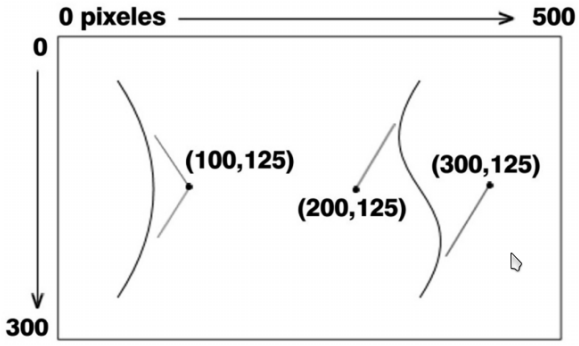
Un ejemplo: **Ej09.24e-CurvasCanvas.html**

```
<!DOCTYPE html>
<html lang="es">           <!-- Ej09.24e-CurvasCanvas.html-->
<head>
   <script language="javascript">
      function iniciar(){
        elemento=document.getElementById('lienzo');
        lienzo=elemento.getContext('2d');

        lienzo.beginPath();
          lienzo.moveTo(50,50);
          lienzo.quadraticCurveTo(100,125, 50,200);
          lienzo.moveTo(250,50);
          lienzo.bezierCurveTo(200,125, 300,125, 250,200);
        lienzo.stroke();
      }
...
</body>
</html>
```

Para la curva cuadrática movimos el lápiz virtual a la posición 50,50 y finalizamos la curva en el punto 50,200. El punto de control para esta curva fue ubicado en la posición 100,125.

La curva generada por el método bezierCurveTo() es un poco más compleja. Hay dos puntos de control para esta curva, el primero en la posición 200,125 y el segundo en la posición 300,125. Los valores de la imagen indican los puntos de control para las curvas. Moviendo estos puntos cambiamos la forma de la curva.



9.2.5 Estilos de línea

Hasta ahora hemos usado siempre los mismos estilos de líneas. El ancho, la terminación y otros aspectos de la línea pueden ser modificados para obtener exactamente el tipo de línea que necesitamos para nuestros dibujos.

Existen cuatro propiedades específicas para este propósito:

- **lineWidth** → Esta propiedad determina el grosor de la línea. Por defecto, 1.0 unidades.
- **lineCap** → Esta propiedad determina la forma de la terminación de la línea. Puede recibir uno de estos tres valores: butt (semicirculo), round (redondeado) y square (cuadrado).
- **lineJoin** → Esta propiedad determina la forma de la conexión entre dos líneas. Los valores posibles son: round (redondeado), bevel (cortado) y miter (puntiagudo) .
- **miterLimit** → Trabajando en conjunto con lineJoin, esta propiedad determina cuánto la conexión de dos líneas será extendida cuando la propiedad lineJoin es declarada con el valor miter.

Las propiedades afectarán el trazado completo. Cada vez que tenemos que cambiar las características de las líneas debemos crear un nuevo trazado. Veamos un ejemplo:

Ej09.25a-EstilosLineaCanvas.html

```
<!DOCTYPE html><html lang="es">          <!-- Ej09.25a-EstilosLineaCanvas.html-->
...
        elemento = document.getElementById ("lienzo");
        lienzo = elemento.getContext ("2d");

        lienzo.beginPath();
           lienzo.arc (200,150,50,0, Math.PI*2, false);
        lienzo.stroke();

        lienzo.lineWidth = 5;
        lienzo.lineCap = "round";
        lienzo.beginPath();
           lienzo.arc (200,150,30,0, Math.PI, false);
        lienzo.stroke();

        lienzo.lineWidth = 4;
        lienzo.lineJoin = "bevel";
        lienzo.beginPath();
           lienzo.moveTo (195,135);
           lienzo.lineTo (215,155);
           lienzo.lineTo (195,155);
        lienzo.stroke ();

        lienzo.beginPath();
           lienzo.arc (175,130,5,0,Math.PI*2,false);
        lienzo.stroke ();
        lienzo.beginPath();
           lienzo.arc (220,130,7,0,Math.PI*2,false);
        lienzo.fill ();
    } ...
```

9.2.6 Texto.

Escribir texto en el lienzo es tan simple como definir unas pocas propiedades y llamar al método apropiado. Tres propiedades son ofrecidas para configurar texto:

- **font** → Esta propiedad tiene una sintaxis similar a la propiedad font de CSS, y acepta los mismos valores.
- **textAlign** → Esta propiedad alinea el texto. Existen varios valores posibles: start (comienzo), end (final), left (izquierda), right (derecha) y center (centro).
- **textBaseline** → Esta propiedad es para alineamiento vertical. Establece diferentes posiciones para el texto (incluyendo texto Unicode). Los posibles valores son: top, hanging, middle, alphabetic, ideographic y bottom.

Dos métodos están disponibles para dibujar texto en el lienzo:

- **strokeText(texto, x, y)** → Del mismo modo que el método stroke() para el trazado, este método dibujará el texto especificado en la posición x,y como una figura vacía (solo los contornos). Puede también incluir un cuarto valor para declarar el tamaño máximo. Si el texto es más extenso que este último valor, será encogido para caber dentro del espacio establecido.
- **fillText(texto, x, y)** → Este método es similar al método anterior excepto que esta vez el texto dibujado será sólido (igual que la función para el trazado).

Veamos un ejemplo completo: **Ej09.26a-TextoCanvas.html**

```
<!DOCTYPE html>
<html lang="es">           <!-- Ej09.26a-TextoCanvas.html-->
<head>
   <title>Canvas API</title>
   <script language="javascript">
      window.onload = function () {
        iniciar();
      }

      function iniciar() {
        elemento=document.getElementById('lienzo');
        lienzo=elemento.getContext('2d');

        lienzo.font="bold 24px verdana, sans-serif";
        lienzo.textAlign="start";                  // El texto COMIENZA en 100,100
        lienzo.fillText("Hola Mundo!", 100,100);   // Dibujamos texto sólido; probar stroke
      }
    </script>
</head>
<body>
   <section id="cajalienzo">
      <canvas id="lienzo" width="500" height="300">
         Su navegador no soporta el elemento canvas
      </canvas>
   </section>
</body>
</html>
```

Además de los previamente mencionados, la API provee otro método importante para trabajar con texto:

- **measureText()** → Este método retorna información sobre el tamaño de un texto específico. Puede ser útil para combinar texto con otras formas en el lienzo y calcular posiciones o incluso colisiones en animaciones.

Un Ejemplo (a partir del anterior) **Ej09.26b-TextoRectanguloCanvas.html**

```
<!DOCTYPE html>
...
      function iniciar() {
        elemento=document.getElementById('lienzo');
        lienzo=elemento.getContext('2d');

            var texto = "Hola Curso!";
            lienzo.font="bold 20px Verdana";
            lienzo.textAlign="start";
            lienzo.textBaseline="bottom";
            lienzo.fillText(texto, 100,120);

            // ATENCIÓN: Usar este método JUSTO antes del rectángulo...
            var tam=lienzo.measureText(texto);             // Vemos el tamaño del texto
            lienzo.strokeRect(100,100,tam.width,20);       // Lo rodeamos con un rectángulo

            // Probar esto:
            // lienzo.strokeRect(90,90,tam.width+20,40);
      }
    </script>
</head>
<body>
      <canvas id="lienzo" width="500" height="300">
         Su navegador no soporta el elemento canvas
      </canvas>
</body> </html>
```

Vamos a explicar el ejemplo: En primer lugar ponemos el texto en la posición 100 (x), 120 (y). Como hemos definido un tamaño de 20px, la parte superior izquierda de la primera letra estará en 120-20 (y), es decir, 100, 100, donde comienza el rectángulo. Para el ancho y el alto del rectángulo definimos tamaño.width y 20 (que es el alto de letra). Ahora cambiamos:
`lienzo.strokeRect(100,100,tamano.width,20);`
Por: `lienzo.strokeRect(90,90,tamano.width+20,40);`

9.2.7 Sombras

Por supuesto, sombras son también una parte importante de Canvas API. Podemos generar sombras para cada trazado e incluso textos. La API provee cuatro propiedades para hacerlo:

- **shadowColor** → Esta propiedad declara el color de la sombra usando sintaxis CSS.
- **shadowOffsetX** → Esta propiedad recibe un número para determinar qué tan lejos la sombra estará ubicada del objeto (dirección horizontal) (positivo a la derecha)
- **shadowOffsetY** → Esta propiedad recibe un número para determinar qué tan lejos la sombra estará ubicada del objeto (dirección vertical) (positivo hacia abajo)
- **shadowBlur** → Esta propiedad produce un efecto de difuminación para la sombra.

Un ejemplo (basado en los anteriores): **Ej09.27a-SombrasCanvas.html**

```
<!DOCTYPE html>
<html lang="es">           <!-- Ej09.27a-SombrasCanvas.html-->
...
      function iniciar() {
        elemento=document.getElementById('lienzo');
        lienzo=elemento.getContext('2d');
        // IMPORTANTE: rgba debe estar PEGADO al paréntesis
        lienzo.shadowColor="rgba(0,0,255,0.5)";    // Azul con transparencia 50%
        lienzo.shadowOffsetX=4;                    // Desplazamiento X = 4px (NO PONER px!!)
        lienzo.shadowOffsetY=6;                    // Desplazamiento Y = 6px
        lienzo.shadowBlur=5;                       // Difuminado → 5px

        lienzo.font="bold 50px verdana";
        lienzo.fillText("Hola Mundo", 100,100);
      }
    </script>
...
```

Es muy importante, a estas alturas, empezar a mezclar elementos y, sobre todo, usar funciones y todas las herramientas que nos permite el lenguaje.
Por ejemplo, podemos poner un texto y un círculo con sendas sombras y luego quitar la sombra para dibujar los rectángulos del ejemplo **Ej09.22-RectangulosColoresCanvas.html**.
Un ejemplo completo: **Ej09.27b-SombrasVariadasCanvas.html**

```
<!DOCTYPE html>
<html lang="es">           <!-- Ej09.27b-SombrasVariadasCanvas.html-->
<head>
   <script language="javascript">
      window.onload = function () {
        iniciar();
      }

      function iniciar() {
        elemento=document.getElementById('lienzo');
        lienzo=elemento.getContext('2d');

        sombraAzul ();
        lienzo.font="bold 50px verdana";
        lienzo.fillText("Hola Mundo", 100,100);

        // Le aplico sombra verde a un círculo
        lienzo.beginPath();
          sombraVerde();
          lienzo.arc(200,150,30,0,Math.PI*2,false);
        lienzo.fill();

        // Quitamos la sombra para rectángulos...
        sinSombra();
        // Definimos los colores
        lienzo.fillStyle="#000099";   // Color Relleno
        lienzo.strokeStyle="#990000"; // Color Contorno

        // Definimos 3 Rectángulos
        lienzo.strokeRect(300,150,120,120);
        lienzo.fillRect(310,160,100,100);
        lienzo.clearRect(320,170,80,80);
      }

      function sombraAzul() {
        lienzo.shadowColor="rgba(0,0,255,0.5)";
        lienzo.shadowOffsetX=4;
        lienzo.shadowOffsetY=6;
        lienzo.shadowBlur=5;
      }

      function sombraVerde() {
        lienzo.shadowColor="rgba(0,255,0,0.5)";
        lienzo.shadowOffsetX=10;
        lienzo.shadowOffsetY=5;
        lienzo.shadowBlur=5;
      }

      function sinSombra() {
        lienzo.shadowOffsetX=0;
        lienzo.shadowOffsetY=0;
        lienzo.shadowBlur=0;
      }
    </script>
</head>
<body>
   <section id="cajalienzo">
      <canvas id="lienzo" width="500" height="300">
         Su navegador no soporta el elemento canvas
      </canvas>
   </section>
</body>
</html>
```

9.2.8 Transformaciones

LA API Canvas ofrece operaciones complejas que es posible aplicar sobre el lienzo para afectar los gráficos que luego son dibujados en él. Estas operaciones son realizadas utilizando cinco métodos de transformación diferentes, cada uno para un propósito específico.

- **translate(x, y)** → Este método de transformación es usado para mover el origen del lienzo. Cada lienzo comienza en el punto 0,0 localizado en la esquina superior izquierda, y los valores se incrementan en cualquier dirección dentro del lienzo. Valores negativos caen fuera del lienzo. A veces es bueno poder usar valores negativos para crear figuras complejas. El método translate() nos permite mover el punto 0,0 a una posición específica para usar el origen como referencia para nuestros dibujos o para aplicar otras transformaciones.
- **rotate(ángulo)** → Este método de transformación rotará el lienzo alrededor del origen tantos ángulos como sean especificados.
- **scale(x, y)** → Este método de transformación incrementa o disminuye las unidades de la grilla para reducir o ampliar todo lo que esté dibujado en el lienzo. La escala puede ser cambiada independientemente para el valor horizontal o vertical usando los atributos x e y. Los valores pueden ser negativos, produciendo un efecto de espejo. Por defecto los valores son iguales a 1.0. Por ejemplo, si pongo 1.1 el aumento es el 10%; si pongo 2 es 200%.
- **transform(m1, m2, m3, m4, dx, dy)** → El lienzo contiene una matriz de valores que especifican sus propiedades. El método transform() aplica una nueva matriz sobre la actual para modificar el lienzo.
- **setTransform(m1, m2, m3, m4, dx, dy)**→ Este método reinicializa la actual matriz de transformación y establece una nueva desde los valores provistos en sus atributos.

En la siguiente práctica usaremos translate para ir moviendo un texto:

Ej09.28a-TraslateCanvas.html

```
<!DOCTYPE html>
<html lang="es">           <!--Ej09.28a-TraslateCanvas.html-->
<head>
    <style type="text/css">
      #lienzo { background: aquamarine; }
    </style>
   <script language="javascript">
      var despX = 0;
      var despY = 0;

      window.onload = function () {
        iniciar();
      }

      function iniciar() {
        elemento=document.getElementById('lienzo');
        lienzo=elemento.getContext('2d');
        lienzo.font="bold 20px Verdana";
        lienzo.fillText("Hola Mundo!",50,50);
      }

      function desplazar() {
        elemento.width = elemento.width;
        despY += 20;
        lienzo.translate(despX,despY);
        lienzo.font="bold 20px Verdana";
        lienzo.fillText("Hola Mundo!",50,50);
      }
    </script>
</head>
<body>
   <canvas id="lienzo" width="500" height="300"></canvas>
   <input type="button" value="Desplazar" onclick="desplazar();" />
   <input type="button" value="animar" onclick="animarRebotar();" />
</body> </html>
```

Aprovechando el código anterior, podemos generar una animación:
Ej09.28b-AnimacionTraslateCanvas.html

```
<!DOCTYPE html>
...
function animarRebotar() {
        setInterval ("rebotar()",50);              }

      function rebotar() {
         elemento.width = elemento.width;

         if (despX==0) {      reboteX = true;       }
         if (despY==0) {      reboteY = true;       }
         if (despY==240) {    reboteY = false;      }
         if (despX==240 && despY==240) {            reboteX = false;      }

         if (reboteY==true) { despY += 20;          }
         if (reboteY==false && reboteX == true) {
            despY -= 20;   despX += 20;             }

         if (reboteX==false) {
            despX -= 20;   despY -= 20;             }

         lienzo.translate (despX,despY);
         lienzo.font = "bold 20px Verdana";
         lienzo.strokeText (texto,50,50);
      }
...
```

En este código se ha empleado una de las formas para borrar el contenido del lienzo. En concreto la menos intuitiva (aunque la mas efectiva): **elemento.width = elemento.width;**
Existen otras 2 técnicas:
Técnica 1: clearRect:
lienzo.clearRect(x, y, width, height);
Los parámetros son:

- **x** → Posición en el eje x donde empezar el área de limpiado.
- **y** → Posición en el eje y donde empezar el área de limpiado.
- **width** → ancho del área de limpiado.
- **height** → alto del área de limpiado.

Si quisiéramos limpiar toda la pantalla, la solución típica seria la siguiente:
lienzo.clearRect(0, 0, lienzo.width, lienzo.height);

Técnica 2: fillRect
Ésta técnica consiste en pintar todo el área de dibujo del color del fondo de nuestra aplicación.
lienzo.save();
lienzo.fillStyle = "#FFF";
lienzo.fillRect(x, y, width, height);
lienzo.restore();

Cómo veis, es algo más “compleja” que la anterior, ya que debemos guardar el estado del contexto, cambiar el color, pintar el rectángulo, y restaurar el estado del contexto.
El método fillRect espera exactamente los mismos parámetros que clearRect.
Volviendo a las transformaciones veamos la manera de usar rotate.
IMPORTANTE: Este código está basado en los anteriores. **Ej09.28c-RotateCanvas.html**

```
<!DOCTYPE html>...
function rotar() {
        elemento.width = elemento.width;
        lienzo.rotate(Math.PI/180*45);
        lienzo.font="bold 20px Verdana";
        lienzo.fillText("Hola Mundo!",50,50);
      }
...
<input type="button" value="rotar" onclick="rotar();" />
```

Ahora veamos otro ejemplo con rotate pero mucho mas "visual":
Ej09.28d-RotarColoresCanvas.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ej09.28d-RotarColoresCanvas.html-->
<head>
    <meta charset="utf-8">
   <style type="text/css">
      canvas { background: PapayaWhip; }     </style>
   <script language="javascript">
      function dibujar() {
         var i,j;
         var distancia = window.prompt (" Especifique distancia entre bolas: ");
         var k = 1;
         elemento = document.getElementById ("lienzo");
         lienzo = elemento.getContext ("2d");
         elemento.width = elemento.width;

         lienzo.translate (150,150);
         for (i=1; i<=5; i++) {
            k++;
            for (j=1; j<=i*6; j++) {
               lienzo.fillStyle = "rgb("+(255-j*k)+","+(j*k)+","+(255-j*k)+")";
               lienzo.rotate ( Math.PI*2 / (i*6) ); // Rotar 360° / nº de Bolitas
               lienzo.beginPath();
                  lienzo.arc (0, i*distancia, 5, 0, Math.PI*2, true);
               lienzo.fill();
            }
         }
      }
   </script>
</head>
<body>
   <canvas id="lienzo" width="500px" height="300px"></canvas> <br />
   <input type="button" value="Dibujar" onclick="dibujar();" />
</body> </html>
```

Seguimos por el escalado (basado en el anterior): **Ej09.28e-EscalarCanvas.html**

```
<!DOCTYPE html>...
function escalar() {
         var factorX = window.prompt (" Escriba Factor Escalar X: ");
         var factorY = window.prompt (" Escriba Factor Escalar Y: ");
         elemento.width = elemento.width;
         lienzo.scale (factorX,factorY);     // OJO,al escalar cambio el origen del dibujo
         lienzo.translate (150/factorX, 150/factorY);
         lienzo.font = "bold 20px Verdana";
         lienzo.fillText (texto,0,0);
      }
...
<input type="button" value="escalar" onclick="escalar();" />
```

Por último, vamos a ver como usar tranform (). Repasemos en primer lugar el concepto:
El método transform() puede realizar todo tipo de cambios en la forma del canvas, es decir traslación, rotación, escalado y sesgado. Para ello dispone de 6 parámetros:
lienzo.transform(a,b,c,d,e,f)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Los seis parámetros vienen determinados por una matriz de transformación, puedes ver su fórmula a la derecha.
Los siguientes valores son los que tendría por defecto el método, es decir al poner estos valores el canvas se queda igual que está:
lienzo.transform(1,0,0,1,0,0)
Resumiendo: **transform(m1, m2, m3, m4, dx, dy)**
m1 (escalaX), m2 (sesgadoHor), m3 (sesgadoVer), m4 (escalaY), dx (desplazX), dy (desplazY)
Mas info: http://aprendeweb.16mb.com/avanzados/canvasav/canvasav9.php

Veamos en primer lugar un ejemplo sencillo: **Ej09.28f-TransformCanvas.html**

```
<!DOCTYPE html>
<html lang="es">        <!--Ej09.28f-TransformCanvas.html-->
<head>
    <meta charset="utf-8">
   <style type="text/css">
      canvas { background: PapayaWhip; }     </style>
   <script language="javascript">
      window.onload = function () {
         iniciar();     }

      function iniciar() {
         elemento = document.getElementById ("lienzo");
         lienzo = elemento.getContext ("2d");
         texto = window.prompt (" Escriba el texto: ");
         posX = window.prompt (" Escriba Posición X: ");
         posY = window.prompt (" Escriba Posición Y: ");
         lienzo.font = "bold 20px Verdana";
         lienzo.fillText (texto,posX,posY);
      }

      function transformar() {
         elemento.width = elemento.width;
         var factorX = window.prompt (" Escriba Factor Escalar X: ");
         var factorY = window.prompt (" Escriba Factor Escalar Y: ");
         var despX = window.prompt (" Escriba Factor Desplazamiento X: ");
         var despY = window.prompt (" Escriba Factor Desplazamiento Y: ");
         var sesgX = window.prompt (" Escriba Sesgado Horizontal: ");
         var sesgY = window.prompt (" Escriba Sesgado Vertical: ");

         lienzo.font = "bold 20px Verdana";
         lienzo.transform (factorX,sesgX,sesgY,factorY,despX,despY);
         lienzo.fillText (texto, posX/factorX, posY/factorY);
      }
   </script>
...
   <input type="button" value="Transformar" onclick="transformar();" />
</body> </html>
```

Ahora otro ejemplo (sacado de MDN): **Ej09.28g-TransformCanvas.html**

```
<!DOCTYPE html>
<html lang="es">           <!--Ej09.28g-TransformCanvas.html-->
<head>
   <script language="javascript">
      function dibujar() {
        elemento=document.getElementById('lienzo');
        lienzo=elemento.getContext('2d');

        var i = 0;
        var sin = Math.sin(Math.PI/6);
        var cos = Math.cos(Math.PI/6);
        lienzo.translate(100, 100);
        var c = 0;
        for (i; i <= 12; i++) {
          c = Math.floor(255 / 12 * i);
          lienzo.fillStyle = "rgb(" + c + "," + c + "," + c + ")";
          lienzo.fillRect(0, 0, 100, 10);
          lienzo.transform(cos, sin, -sin, cos, 0, 0);
        }

        lienzo.setTransform(-1, 0, 0, 1, 200, 150);
        lienzo.fillStyle = "rgba(255, 0, 255, 1)";
        lienzo.fillRect(0, 50, 100, 100);
      }
    </script>
...
   <input type="button" value="Transformar" onclick="dibujar();" />
</body>
</html>
```

9.2.9 Restaurando el estado.

La acumulación de transformaciones hace realmente difícil volver a anteriores estados. Considerando situaciones como ésta, Canvas API provee dos métodos para grabar y recuperar el estado del lienzo.

- **save()** → Este método graba el estado del lienzo, incluyendo transformaciones ya aplicadas, valores de propiedades de estilo y la actual máscara (el área creada por el método clip(), si existe).
- **restore()** → Este método recupera el último estado grabado.

Veamos un ejemplo completo: **Ej09.29a-SaveRestoreCanvas.html**

```
<!DOCTYPE html>
<html lang="es">           <!--Ej09.29a-SaveRestoreCanvas.html-->
<head>
   <title>Canvas API</title>
    <style type="text/css">
      #lienzo { background: aquamarine; }
    </style>
   <script language="javascript">
      function dibujar() {
         elemento = document.getElementById ("lienzo");
         lienzo = elemento.getContext ("2d");

         lienzo.fillStyle = "blue";
         lienzo.font = "bold 30px Cursive";
         lienzo.fillText ("Hola Mundo",100,100);
         lienzo.save();    // Guardo las propiedades predeterminadas

         lienzo.fillStyle = "red";
         lienzo.font = "bold 20px Verdana";
         lienzo.translate (100,100);
         lienzo.fillText ("Otro Texto",100,100);

         lienzo.restore ();
         lienzo.fillText ("Texto en Comic Sans",0,50);
      }
    </script>
</head>
<body>
  <canvas id="lienzo" width="500" height="300"></canvas>
  <input type="button" value="Dibujar" onclick="dibujar();" />
</body>
</html>
```

Obsérvese que guardamos con el origen 0,0 para el lienzo y los valores predeterminados para las propiedades (incluido fillStyle que luego cambiamos a rojo). Por eso, al usar restore, volvemos al 0,0 y al color negro (a pesar de tener arriba el rojo).

9.2.10 globalCompositeOperation

Existe una propiedad para determinar cómo una figura es posicionada y combinada con figuras dibujadas previamente en el lienzo. La propiedad es globalCompositeOperation y su valor por defecto es **source-over**, lo que significa que la nueva figura será dibujada sobre las que ya existen en el lienzo. La propiedad ofrece 11 valores más:

- **source-in** → Solo la parte de la nueva figura que se sobrepone a las figuras previas es dibujada. El resto de la figura, e incluso el resto de las figuras previas, se vuelven transparentes.
- **source-out** → Solo la parte de la nueva figura que no se sobrepone a las figuras previas es dibujada. El resto de la figura, e incluso el resto de las figuras previas, se vuelven transparentes.
- **source-atop** → Solo la parte de la nueva figura que se superpone con las figuras previas es dibujada. Las figuras previas son preservadas, pero el resto de la nueva figura se vuelve transparente.
- **lighter** → Ambas figuras son dibujadas (nueva y vieja), pero el color de las partes que se superponen es obtenido adicionando los valores de los colores de cada figura.
- **xor** → Ambas figuras son dibujadas (nueva y vieja), pero las partes que se superponen se vuelven transparentes.
- **destination-over** → Este es el opuesto del valor por defecto. Las nuevas figuras son dibujadas detrás de las viejas que ya se encuentran en el lienzo.
- **destination-in** → Las partes de las figuras existentes en el lienzo que se superponen con la nueva figura son preservadas. El resto, incluyendo la nueva figura, se vuelven transparentes.
- **destination-out** → Las partes de las figuras existentes en el lienzo que no se superponen con la nueva figura son preservadas. El resto, incluyendo la nueva figura, se vuelven transparentes.
- **destination-atop** → Las figuras existentes y la nueva son preservadas solo en la parte en la que se superponen.
- **darker** → Ambas figuras son dibujadas, pero el color de las partes que se superponen es determinado substrayendo los valores de los colores de cada figura.
- **copy** → Solo la nueva figura es dibujada. Las ya existentes se vuelven transparentes.

Veamos un ejemplo: **Ej09.29b-globalCompositeOperation.html**

```
<!DOCTYPE html>
<html lang="es">           <!--Ej09.29b-globalCompositeOperation.html-->
<head>
   <script language="javascript">
      function dibujar() {
        elemento=document.getElementById('lienzo');
        lienzo=elemento.getContext('2d');

        lienzo.fillStyle="#990000";
        lienzo.fillRect(100,100,300,100);

        // Superponemos texto y dejamos la parte coincidente mas oscura
        lienzo.globalCompositeOperation="darker";
        // PROBAR MAS VALORES... EN CHROME!!

        lienzo.fillStyle="blue";
        lienzo.font="bold 50px verdana, sans-serif";
        lienzo.textAlign="center";
        lienzo.textBaseline="middle";
        lienzo.fillText("Hola Mundo!",250,110);
      }
    </script>
...
```

9.3 Procesando imágenes.

API Canvas no sería nada sin la capacidad de procesar imágenes. Pero incluso cuando las imágenes son un elemento tan importante para una aplicación gráfica, solo un método nativo fue provisto para trabajar con ellas.

9.3.1 drawImage()

El método drawImage() permite dibujar una imagen en el lienzo. Sin embargo, este método puede recibir un número de valores que producen diferentes resultados. Estudiemos estas posibilidades:

- **drawImage(imágen, x, y)** → Esta sintaxis es para dibujar una imagen en el lienzo en la posición declarada por x e y. El primer valor es una referencia a la imagen que será dibujada.
- **drawImage(imágen, x, y, ancho, alto)** → Esta sintaxis nos permite escalar la imagen antes de dibujarla en el lienzo, cambiando su tamaño con los valores de los atributos ancho y alto.

- **drawImage(imágen, x1, y1, ancho1, alto1, x2, y2, ancho2, alto2)** → Esta es la sintaxis más compleja. Hay dos valores para cada parámetro. El propósito es cortar partes de la imagen y luego dibujarlas en el lienzo con un tamaño y una posición específica. Los valores x1 e y1 declaran la esquina superior izquierda de la parte de la imagen que será cortada. Los valores ancho1 y alto1 indican el tamaño de esta pieza. El resto de los valores (x2, y2, ancho2 y alto2) declaran el lugar donde la pieza será dibujada en el lienzo y su nuevo tamaño (el cual puede ser igual o diferente al original).

En cada caso, el primer atributo puede ser una referencia a una imagen en el mismo documento generada por métodos como getElementById(), o creando un nuevo objeto imagen usando métodos regulares de Javascript. No es posible usar una URL o cargar un archivo desde una fuente externa directamente con este método.

Veamos un ejemplo completo: **Ej09.31a-drawImage.html**

```
<!DOCTYPE html>
<html lang="es">          <!--Ej09.31a-DrawImage.html-->
<head>
   <title>Canvas API</title>
    <style type="text/css">
      #lienzo { background: aquamarine; }
    </style>
   <script language="javascript">
      function verImagen(){
         elemento = document.getElementById ("lienzo");
         lienzo = elemento.getContext ("2d");

         foto = new Image();     // Creamos una instancia del objeto Imagen
         foto.src = "canvas.jpg";    // La imagen es de 500,335

         lienzo.drawImage (foto,0,0,350,250);
         lienzo.globalCompositeOperation = "xor";
         lienzo.drawImage (foto,100,100,450,350);
         lienzo.drawImage (foto,0,0,200,200,150,120,100,100);
      }
    </script>
</head>
<body>
  <canvas id="lienzo" width="500" height="300"></canvas>
  <input type="button" value="Ver Imagen" onclick="verImagen();" />
</body>
</html>
```

9.3.2 Datos de imágenes

Existen unos poderosos métodos para procesar imágenes en esta API que además pueden dibujarlas en el lienzo. ¿Pero por qué desearíamos procesar datos en lugar de imágenes?

Toda imagen puede ser representada por una sucesión de números enteros representando valores rgba (cuatro valores para cada píxel). Un grupo de valores con esta información resultará en un array unidimensional que puede ser usado luego para generar una imagen.

La API Canvas ofrece tres métodos para manipular datos y procesar imágenes de este modo:

- **getImageData(x, y, ancho, alto)** → Este método toma un rectángulo del lienzo del tamaño declarado por sus atributos y lo convierte en datos. Retorna un objeto que puede ser luego accedido por sus propiedades width, height y data.
- **putImageData(datosImagen, x, y)** → Este método convierte a los datos en datosImagen en una imagen y dibuja la imagen en el lienzo en la posición especificada por x e y. Este es el opuesto a getImageData().
- **createImageData(ancho, alto)** → Este método crea datos para representar una imagen vacía. Todos sus píxeles serán de color negro transparente. Puede también recibir datos como atributo (en lugar de los atributos ancho y alto) y utilizar las dimensiones tomadas de los datos provistos para crear la imagen.

La posición de cada valor en el array es calculada con la fórmula(ancho×4×y)+(x×4). Éste será el primer valor del píxel (rojo); para el resto tenemos que agregar 1 al resultado (por ejemplo, (ancho×4×y)+(x×4)+1 para verde, (ancho×4×y)+(x×4)+2 para azul, y (ancho×4×y)+(x×4)+3 para el valor alpha (transparencia). Veamos esto en práctica:
IMPORTANTE: Para que la práctica funcione debemos descargarnos la imagen (tomar la del ejercicio anterior). Veamos un ejemplo completo: **Ej09.32a-DatosImagen.html**

```
<!DOCTYPE html>
<html lang="es">           <!--Ej09.32a-DatosImagenes.html-->
<head>
    <style type="text/css">
      #lienzo { background: aquamarine; }     </style>
   <script language="javascript">
      function verImagen(){
        elemento=document.getElementById('lienzo');
        lienzo=elemento.getContext('2d');
        imagen=new Image();
        imagen.src="canvas.jpg";

        // OJO: la carga puede durar un poco...
        lienzo.drawImage(imagen,0,0);               // Pinto la imagen
        trozo=lienzo.getImageData(0,0,200,250);     // Recorto un trozo de 150x250
        var pos;

        for(x=0;x<=200;x++)      // El primer for es para la X
        {
          for(y=0;y<=250;y++)    // El segundo for es para la Y
          {                                 // Invierto los colores
            pos=(trozo.width*4*y)+(x*4);    // Para sacar la posición del pixel
            trozo.data[pos]=255-trozo.data[pos];           // Cambio el rojo
            trozo.data[pos+1]=255-trozo.data[pos+1];       // Cambio el verde
            trozo.data[pos+2]=255-trozo.data[pos+2];       // Cambio el azul
          }
        }
        lienzo.putImageData(trozo,0,0);         // Dibujo el trozo invertido
      }
    </script>
</head>
<body>
  <canvas id="lienzo" width="500" height="300"></canvas>
  <input type="button" value="Ver Imagen" onclick="verImagen();" />
</body> </html>
```

Existe otra manera de extraer datos del lienzo que retorna el contenido en una cadena de texto codificada en base64. Esta cadena puede ser usada luego como fuente para otro lienzo, como fuente de un elemento HTML (por ejemplo, <img>), o incluso ser enviado al servidor o grabado en un archivo. El siguiente es el método incluido con este fin:

- **toDataURL(tipo)** → El elemento <canvas> tiene dos propiedades, width y height, y dos métodos: getContext() y toDataURL(). Este último método retorna datos en el formato data:url conteniendo una representación del contenido del lienzo en formato PNG (o el formato de imagen especificado en el atributo tipo).

Vamos a ver un ejemplo completo donde usaremos el método anterior junto a las distintas transformaciones (de paso, pongo otra forma de borrar el lienzo).
Imagen a usar: http://s2.alt1040.com/files/2013/04/Paris-200x200.jpg

Ej09.32b-ExtraerImagen.html

```
<!DOCTYPE html>
<html lang="es">           <!--Ej09.32b-ExtraerImagen.html-->
<head>
    <meta charset="UTF-8" />
   <script language="javascript">
    window.onload = function(){
      trazarCanvas();
    }

    function trazarCanvas(){
      elemento = document.getElementById("canvas");
      lienzo = elemento.getContext("2d");

      imagen = new Image();
      imagen.src="paris.jpg";
      lienzo.drawImage(imagen,0,0);
    }

    function borrarCanvas(){
      lienzo.clearRect(0,0,elemento.width,elemento.height);
    }

    function escalarMas(){      // Aumento 20%
      borrarCanvas();
      lienzo.scale(1.2,1.2);
      trazarCanvas();         }

    function escalarMenos(){    // Decremento 20%
      borrarCanvas();
      lienzo.scale(0.8,0.8);
      trazarCanvas();         }

    function rotarMas(){        // Rota 10°
      borrarCanvas();
      lienzo.rotate(Math.PI/180 * 10);
      trazarCanvas();         }

    function rotarMenos(){      // Rota del revés 10°
      borrarCanvas();
      lienzo.rotate(-Math.PI/180 * 10);
      trazarCanvas();         }

    function trasladarMas(){     // Mueve hacia abajo, derecha 10px
      borrarCanvas();
      lienzo.translate(10,10);
      trazarCanvas();         }

    function trasladarMenos(){   // Sube hacia arriba, izquierda, 10px
      borrarCanvas();
      lienzo.translate(-10,-10);
      trazarCanvas();         }
```

```
    function extraerImagen(){
      var dataImage = lienzo.canvas.toDataURL("image/png");
      document.getElementById("datosImagen").value = dataImage;
      document.getElementById("imagenPNG").src = dataImage;
}
    </script>
</head>
<body>
  <canvas id="canvas" width="400" height="400" style="border:blue solid 1px;float:left;">
    No Soportas Canvas </canvas> <br/>
  <input type="button" value="Trasladar +" onclick="trasladarMas()"/>
  <input type="button" value="Trasladar -" onclick="trasladarMenos()"/><br/>
  <input type="button" value="Escalar +" onclick="escalarMas()"/>
  <input type="button" value="Escalar -" onclick="escalarMenos()"/><br/>
  <input type="button" value="Rotar +" onclick="rotarMas()"/>
  <input type="button" value="Rotar -" onclick="rotarMenos()"/><br/>
  <input type="button" value="Extraer imagen" onclick="extraerImagen()"/>

   <p> </p><p> </p><p> </p>
   <p> </p><p> </p><p> </p>  <p> </p>
  <h3> Estos serán los datos de la Imagen Codificados...</h3>
  <textarea rows="20" cols="20" id="datosImagen" style="width:100%"></textarea>

  <h3> Y está será la imagen convertida a PNG y mostrada</h3>
  <img id="imagenPNG" style="border:red dotted 1px" alt="Extraer imagen aquí"/></div>
</body>
</html>
```

9.3.3 Patrones

Los patrones son simples adiciones que pueden mejorar nuestros trazados. Con esta herramienta podemos agregar textura a nuestras figuras utilizando una imagen. El procedimiento es similar a la creación de gradientes; los patrones son creados por el método createPattern() y luego aplicados al trazado como si fuesen un color.

- **createPattern(imágen, tipo)** → El atributo imágen es una referencia a la imagen que vamos a usar como patrón, y tipo configura el patrón por medio de cuatro valores: repeat, repeat-x, repeat-y y no-repeat.

Veamos un ejemplo completo: **Ej09.33a-PatronImagen.html**

```
<!DOCTYPE html>
<html lang="es">           <!--Ej09.33a.PatronCanvas.html-->
<head>
    <meta charset="UTF-8" />
   <script language="javascript">
    function iniciar(){
      elemento = document.getElementById("canvas");
      lienzo = elemento.getContext("2d");

      var imagen=new Image();
      imagen.src="ladrillos.jpg";

      var patron=lienzo.createPattern(imagen,'repeat');
      lienzo.fillStyle=patron;
      lienzo.fillRect(0,0,500,300);
    }
    </script>
</head>
<body>
  <canvas id="canvas" width="500" height="300">
    No Soportas Canvas </canvas> <br/>
  <input type="button" value="Dibujar con Patrón" onclick="iniciar()"/>
</body>
</html>
```

9.4 Animaciones en el lienzo

Las animaciones son creadas por código Javascript convencional. No existen métodos para ayudarnos a animar figuras en el lienzo, y tampoco existe un procedimiento predeterminado para hacerlo. Básicamente, debemos borrar el área del lienzo que queremos animar, dibujar las figuras y repetir el proceso una y otra vez. Una vez que las figuras son dibujadas no se pueden mover. Solo borrando el área y dibujando las figuras nuevamente podemos construir una animación. Por esta razón, en juegos o aplicaciones que requieren grandes cantidades de objetos a ser animados, es mejor usar imágenes en lugar de figuras construidas con trazados complejos (por ejemplo, juegos normalmente utilizan imágenes PNG, que además son útiles por su capacidad de transparencia).

Vamos a ver un código curioso: mostrará dos ojos en pantalla que miran al puntero del ratón todo el tiempo. Para mover los ojos, debemos actualizar su posición cada vez que el ratón es movido. Por este motivo agregamos una escucha para el evento mousemove en la función iniciar(). Cada vez que el puntero del ratón cambia de posición, el evento es disparado y la función animacion() es llamada.

Ej09.41a-Animacion.html

```
<!DOCTYPE html>
<html lang="es">           <!--Ej09.33a.PatronCanvas.html-->
<head>
    <meta charset="UTF-8" />
   <script language="javascript">
      window.onload = function () {
        iniciar();
      }

    function iniciar(){
      elemento=document.getElementById('lienzo');
      lienzo=elemento.getContext('2d');
      window.addEventListener('mousemove', animacion, false);
    }

    function animacion(e){
      lienzo.clearRect(0,0,300,500);
      var xraton=e.clientX;   // Capturo la posición del ratón (x)
      var yraton=e.clientY;   // Capturo la posición del ratón (y)
      var xcentro=220;
      var ycentro=150;

      angulo=Math.atan2(xraton-xcentro,yraton-ycentro);
      var x=xcentro+Math.round(Math.sin(angulo)*10);       // 10 = radio /2
      var y=ycentro+Math.round(Math.cos(angulo)*10);

      // Dibujo los ojos
      lienzo.beginPath();        //radio = 20px
        lienzo.arc(xcentro,ycentro,20,0,Math.PI*2, false);  // Izquierdo
        lienzo.moveTo(xcentro+70,150);       // De un ojo a otro hay 50px + radio
        lienzo.arc(xcentro+50,150,20,0,Math.PI*2, false);   // Derecho a 50px
      lienzo.stroke();

      // Dibujo la pupila que se mueve
      lienzo.fillStyle = "blue";
      lienzo.beginPath();
        lienzo.moveTo(x+10,y);
        lienzo.arc(x,y,10,0,Math.PI*2, false);      // Pupila Izquierda
        lienzo.moveTo(x+60,y);
        lienzo.arc(x+50,y,10,0,Math.PI*2, false);   // Pupila Derecha
      lienzo.fill();
    }
   </script>
</head>
<body>
  <canvas id="lienzo" width="500" height="300">
    No Soportas Canvas </canvas> <br/>
</body>
</html>
```

9.5 Procesando vídeo en el lienzo.

Al igual que para animaciones, no hay ningún método especial para mostrar video en el elemento <canvas>. La única manera de hacerlo es tomando cada cuadro del video desde el elemento <video> y dibujarlo como una imagen en el lienzo usando drawImage(). Así que básicamente, el procesamiento de video en el lienzo es hecho con la combinación de técnicas ya estudiadas.
Veamos un ejemplo curioso y completo:
NOTA: Hay que bajarse el vídeo en formato MP4 (Chrome) u OGG (Firefox) desde aquí:
http://www.minkbooks.com/content/trailer2.mp4
http://www.minkbooks.com/content/trailer2.ogg

Ej09.51a-VideosCanvas.html

```
<!DOCTYPE html>
<html lang="es">           <!--Ej09.51a-VideosCanvas.html-->
<head>
  <meta charset="UTF-8" />
  <title>Canvas API</title>
  <style>
    .cajas{
      display: inline-block;
      margin: 10px;
      padding: 5px;
      border: 1px solid #999999;
    }
  </style>
 <script language="javascript">
      window.onload = function () {
        iniciar();
      }

    function iniciar(){
      elemento=document.getElementById('lienzo');
      lienzo=elemento.getContext('2d');

      video=document.getElementById('medio');
      // Cada vez que pulso pauso el vídeo
      video.addEventListener('click', presionar, false);
    }

    function presionar(){
      if(!video.paused && !video.ended){
        video.pause();
        // Para el procesado de imágenes
        window.clearInterval(bucle);
      }else{
        video.play();
        // Comienza el procesado de imágenes cada 33msg
        bucle=setInterval(procesarCuadros, 33);
      }
    }

    function procesarCuadros(){
      lienzo.drawImage(video,0,0);    // Tamaño vídeo 483px x 272px
      var info=lienzo.getImageData(0,0,483,272);

      var pos;
      var gris;
      for(x=0;x<=483;x++){      // For para píxeles horizontales
        for(y=0;y<=272;y++){    // For para píxeles verticales
          pos=(info.width*4*y)+(x*4);
          gris=parseInt(info.data[pos]*0.2989 +
                        info.data[pos+1]*0.5870 +
                        info.data[pos+2]*0.1140);
          info.data[pos]=gris;    // Pasamos el canal Rojo a Gris
          info.data[pos+1]=gris;  // Pasamos el canal Verde a Gris
          info.data[pos+2]=gris;  // Pasamos el canal Azul a Gris
        }
      }
```

```
      // Ponemos la imagen CAMBIADA en el lienzo (desde el origen 0,0)
      lienzo.putImageData(info,0,0);
    }
  </script>
</head>
<body>
  <section class="cajas">
    <video id="medio" width="483" height="272">
      <source src="trailer2.mp4">
    </video>
  </section>

  <section class="cajas">
    <canvas id="lienzo" width="483" height="272">
      Su navegador no soporta el elemento canvas
    </canvas>
  </section>
</body>
</html>
```

10 BIBLIOGRAFIA Y SOLUCIONES

10.1 Bibliografía y Enlaces

- http://librosweb.es/javascript/
- El gran libro de HTML5, CSS3 y Javascrip - Juan Diego Gauchat
  Editorial Marcombo
-

10.2 Práctica Final

Definir un formulario para el alta de usuario en el sistema.
Los campos a incluir serán los siguientes:

1. **Nombre de Usuario:**
   Validación: Campo No Vacío, inicio letras, mín. 6 caracteres; No caracteres especiales
2. **Contraseña:**
   Validación: Campo No Vacío, inicio letras, mín. 6 caracteres; Permite caracteres especiales.
3. **Repite Contraseña:**
   Validación: Campo No Vacío, inicio letras, mín. 6 caracteres; Permite caracteres especiales.
   Debe coincidir con el campo contraseña.
4. **Dirección: Tipo de Via**
   Select cuyos elementos se toman de un array (Ej: Calle, Avenida, Plaza, etc)
   Validación: Debe elegirse una opción.
5. **Dirección: Nombre**
   Validación: Campo No Vacío, inicio letras, mín. 6 caracteres; Permite caracteres especiales.
6. **Localidad:**
   Select cuyos elementos se toman de un array (Ej: Calle, Avenida, Plaza, etc)
   Validación: Debe elegirse una opción.
7. **Código Postal:**
   Validación: Campo No Vacío. Formato: +55555
8. **Teléfono Fijo:**
   Validación: Campo No Vacío. Formato: +34 555 555 555
9. **Teléfono Móvil:**
   Validación: Campo No Vacío. Formato: +34 555 555 555
10. **NIF:**
    Validación: Campo No Vacío. Letra Correcta: Formato: 55555555-K
11. **Captcha:**
    Saldrán cinco números escritos con letras.
    Escribir dichos números con dígitos AL REVÉS.

Con JavaScript hay que ponerle un fondo al input donde se encuentre el usuario (NO hacerlo con CSS). Además, al final, nos mostrará un mensaje en el propio documento, en rojo, con los errores cometidos junto al input correspondiente.

10.3 Soluciones Ejercicios

Ejercicio_01.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>JavaScript</title>
   <script language="javascript" src="codigo.js"> </script>
</head>
  <body>
   <noscript> Pecador no tienes el JavaScript activado!!</noscript>
  </body>
</html>
```

codigo.js

```
/* confirm  -> Incluye botones Aceptar y Cancelar
 * alert    -> Muestra el mensaje */
window.confirm ("¿Te ha gustado este script?");
window.alert ("Soy el primer script");
```

Ejercicio_02.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio_02.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript" src="Ejercicio_02.js"> </script>
</head>
  <body>
   <form name="form1">
      <input type="button"
            value="Mostrar Mensaje" onclick="mensaje ();" />
   </form>
  </body>
</html>
```

Ejercicio_.js

```
var mimensaje =
' Hola Mundo! \n Qué fácil es incluir \'comillas simples\' \n y "comillas dobles". ';
function mensaje (){
   window.alert (mimensaje);  }
```

Ejercicio_03.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio3.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      var meses = ["Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio",
       "Julio", "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"];

      function vermeses () {
         var todosLosMeses = meses[0]+"\n"+ meses[1]+"\n"+ meses[2]+"\n"+
                   meses[3]+"\n"+ meses[4]+"\n"+ meses[5]+"\n"+
                   meses[6]+"\n"+ meses[7]+"\n"+ meses[8]+"\n"+
                   meses[9]+"\n"+ meses[10]+"\n"+ meses[11];
         window.alert (todosLosMeses);
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="ver meses" onclick="vermeses ();" />
      </form>
  </body>
</html>
```

Ejercicio_04.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio04.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      var valores = [true, 5, false, "hola", "adios", 2];

      function compararTexto () {
         var cadena1 = "Hola Mundo";
         var cadena2 = "Hola Peña";
         var textomayor = cadena1 > cadena2;

         if (textomayor >= true) {
            window.alert ("Resultado: " + cadena1 + " es mayor que " + cadena2);
         }

         if (textomayor <= false) {
            window.alert ("Resultado: " + cadena1 + " es menor que " + cadena2);
         }
      }

      function cambiarBooleanos () {
         var booleano1 = valores [0];        // true
         var booleano2 = valores [2];        // false

         var combinarTrue = booleano1 || booleano2; // -> TRUE
         var combinarFalse = booleano1 && booleano2;        // -> FALSE

         window.alert (" Para que los valores " + booleano1 + " y " + booleano2 +
                    " salgan en TRUE o FALSE debemos usar: " + "\n" +
                    " OR -> " + combinarTrue + "\n" +
                    " AND -> " + combinarFalse);
      }

      function operacionesMates () {
         var num1 = valores [1];      // 5
         var num2 = valores [5];      // 2

         var suma = num1 + num2;
         var resta = num1 - num2;
         var producto = num1 * num2;
         var division = num1 / num2;
         var modulo = num1 % num2;

         window.alert ("La suma: " + num1 + "+" + num2 + "=" + suma);
         window.alert ("La resta: " + num1 + "-" + num2 + "=" + resta);
         window.alert ("El producto: " + num1 + "*" + num2 + "=" + producto);
         window.alert ("La division: " + num1 + "/" + num2 + "=" + division);
         window.alert ("El modulo: " + num1 + "%" + num2 + "=" + modulo);
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Comparar Texto" onclick="compararTexto ();" />
         <br /><br />
         <input type="button" value="Cambiar Booleanos" onclick="cambiarBooleanos ();" />
         <br /><br />
         <input type="button" value="Operaciones Mates" onclick="operacionesMates ();" />
      </form>
  </body>
</html>
```

Ejercicio_05.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio05.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
   var edad = 0;

   function preguntaEdad () {
      edad = window.prompt ("Introduzca su edad: ");
      if (edad==18){
         window.alert ("Enhorabuena PAVO");
      }
      else {
         if (edad >18) {
         window.alert ("Eres mayor de Edad");
         window.location.href ="http://www.meneame.net/";
         }
         else {
            window.alert ("Eres un pitufo");
            window.location.href ="http://www.danba.es/";
         }
      }
   }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Preguntar Edad" onclick="preguntaEdad ();" />
      </form>
  </body>
</html>
```

Ejercicio_06.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio 06.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      var letras = ['T', 'R', 'W', 'A', 'G',  'M', 'Y', 'F', 'P', 'D',
                    'X', 'B', 'N', 'J', 'Z',      'S', 'Q', 'V', 'H', 'L',
                    'C', 'K', 'E', 'T'];
      var letraresultado;

      function letraDNI() {
         var dni = window.prompt (" Escriba DNI: ");
         var letraindicada = window.prompt (" Escriba letra del DNI: ");

         if (dni<=0 || dni >99999999) {
            window.alert (" El número puesto NO es válido!");
         }
         else
         {
            var resto = dni % 23;
            letraresultado = letras[resto];
            window.alert (" Su letras es: "+letraresultado);

            if (letraindicada == letraresultado) {
            window.alert (" Has acertado!");
            }
            else
            {
            window.alert (" NO Has acertado!");
            }
         }
      }
   </script>
```

```
</head>
  <body>
      <form name="form1">
         <input type="button" value="Letra DNI" onclick="letraDNI ();" />
      </form>
  </body>
</html>
```

Ejercicio_07.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio 07.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">


      function multiplicar() {
         var numero;
         var i;

         numero = window.prompt (" Escriba número: ");
         var resultado;

         for (i=1; i<11; i++) {
            resultado = numero *i;
            document.write (numero + " x " +i+ " = " +resultado + "<br />")
         }
      }

      function multiplicarTodas() {
         var i, j;      // Índices de ambos FOR
         var resultado;

         document.write ("<link rel=StyleSheet href='estilo.css' TYPE='text/css' />");
         for (i=1; i<=10; i++) {
            document.write ("<div>");
            document.write (" TABLA DE MULTIPLICAR DEL "+i+ "<br />");

            for (j=1; j<=10; j++) {
               resultado = i*j;
               document.write (i+ "x" + j + " = " + resultado + "<br />");
            }
            document.write ("</div>");
         }
      }
</script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Tabla Multiplicar Elegida" onclick="multiplicar ();" />
         <br /><br />
         <input type="button" value="Tablas Multiplicar (Todas)" onclick="multiplicarTodas
();" />
      </form>
  </body>
</html>
```

estilos.css (para Ejercicio_07.html)

```
div {
   border: 2px solid red;
   background: pink;
   border-radius: 5px;
   display: block;
   float: left;
   text-align: center;
   padding: 10px;
   margin: 10px;
}
```

Ejercicio_07Especial.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio 07 Especial.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">

      var numero;
      var resultado;
      var i;

      function factorial () {
         resultado = 1;
         var numero = window.prompt (" Elija el valor para calcular Factorial: ");

         if (numero<0) {
            window.alert (" No se puede calcular el Factorial de un número negativo");
         }
         else
         {
            for (i=1; i<=numero; i++) {
               resultado = resultado * i;
            }
            window.alert (numero+"! = "+resultado);
         }
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Factorial" onclick="factorial ();" />
  </body>
</html>
```

Ejercicio_07c.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio7c.html -->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">

      // ######## Variables Globales ########

      // Variables para los menús
      var opcionPrincipal;
      var opcionIntroducir;
      var opcionVer;

      // Los arrays del Supermercado (Frutas, pescados, carnes y drogueria)
      var frutas  = new Array ();
      var pescados  = new Array ();
      var carnes  = new Array ();
      var drogueria  = new Array ();

      // Las variables que rellena el usuario
      var separador;
      var cadena = new String ();

      // El resto de variables
      var seguir = true;

      // Defino una Cadena Global para los submenús
      var menu = "\n"+ " 1. Frutas. " + "\n"+
                 " 2. Pescados. " + "\n"+
                 " 3. Carnes. " + "\n"+
                 " 4. Drogueria. " + "\n"+
                 " 5. Menú Principal. " + "\n";
```

```
// ########## MENU PRINCIPAL DE LA APLICACIÓN ##########
function ejecutar() {
   opcionPrincipal = window.prompt (" MENU PRINCIPAL " + "\n"+
                                    " 1. Introducir Listas. " + "\n"+
                                    " 2. Visualizar Listas. " + "\n");

   if (opcionPrincipal==1) {
      introducirListas();
   }

   if (opcionPrincipal==2) {
      verListas();
   }
}


// ########## SUBMENU INTRODUCIR LAS LISTAS ##########
function introducirListas() {
   opcionIntroducir = window.prompt (" MENU INTRODUCIR LISTAS" + menu);

   if (opcionIntroducir==1) {
      window.alert (" Va a introducir la lista de Frutas!!");

      if (frutas.length==0) {
         pedirdatos();
         frutas = cadena.split (separador);   // Convertir la cadena en Array
         ejecutar();                          // Vuelve al menú principal
      }
      else {
         window.alert (" La lista de Frutas NO está vacia!!");
         seguir = window.confirm (" Desea seguir (Aceptar, pierde lista anterior):  ");
         if (seguir==true) {
            pedirdatos();
            frutas = cadena.split (separador);
            ejecutar();
         }
         else {
            ejecutar();
         }
      }
   }


   if (opcionIntroducir==2) {
      window.alert (" Va a introducir la lista de Pescados!!");

      if (pescados.length==0) {
         pedirdatos();
         pescados = cadena.split (separador); // Convertir la cadena en Array
         ejecutar();                          // Vuelve al menú principal
      }
      else {
         window.alert (" La lista de Pescados NO está vacia!!");
         seguir = window.confirm (" Desea seguir (Aceptar, pierde lista anterior):  ");
         if (seguir==true) {
            pedirdatos();
            pescados = cadena.split (separador);
            ejecutar();
         }
         else {
            ejecutar();
         }
      }
   }
```

```
   if (opcionIntroducir==3) {
      window.alert (" Va a introducir la lista de Carnes!!");

      if (carnes.length==0) {
         pedirdatos();
         carnes = cadena.split (separador);   // Convertir la cadena en Array
         ejecutar();                          // Vuelve al menú principal
      }
      else {
         window.alert (" La lista de Carnes NO está vacia!!");
         seguir = window.confirm (" Desea seguir (Aceptar, pierde lista anterior):  ");
         if (seguir==true) {
            pedirdatos();
            carnes = cadena.split (separador);
            ejecutar();
         }
         else {
            ejecutar();
         }
      }
   }


   if (opcionIntroducir==4) {
      window.alert (" Va a introducir la lista de Drogueria!!");

      if (drogueria.length==0) {
         pedirdatos();
         drogueria = cadena.split (separador);// Convertir la cadena en Array
         ejecutar();                          // Vuelve al menú principal
      }
      else {
         window.alert (" La lista de Drogueria NO está vacia!!");
         seguir = window.confirm (" Desea seguir (Aceptar, pierde lista anterior):  ");
         if (seguir==true) {
            pedirdatos();
            drogueria = cadena.split (separador);
            ejecutar();
         }
         else {
            ejecutar();
         }
      }
   }

   if (opcionIntroducir==5) {
      ejecutar();
   }

}
```

```
      // ########## SUBMENU VISUALIZAR LAS LISTAS ##########
      function verListas() {
         opcionVer = window.prompt (" MENU VISUALIZAR LISTAS" + menu);

         if (opcionVer=="" || opcionVer<1 || opcionVer >5)
         {
            window.alert (" Atención, has cometido un ERROR!!");
            ejecutar();
         }
         else {
            if (opcionVer==1) {
               mostrarDatos (frutas);
               ejecutar();
            }

            if (opcionVer==2) {
               mostrarDatos (pescados);
               ejecutar();
            }

            if (opcionVer==3) {
               mostrarDatos (carnes);
               ejecutar();
            }

            if (opcionVer==4) {
               mostrarDatos (drogueria);
               ejecutar();
            }

            if (opcionVer==5) {
               ejecutar();
            }
         }
      }


      // Función para pedir los datos al usuario que se usa UNA Y OTRA VEZ
      function pedirdatos() {
         separador = window.prompt (" Especifique el separador de la cadena: ");
         cadena =
         window.prompt ( " Escriba la lista elegida separada por el caracter "+ separador);
      }

      // Función para leer los distintos arrays
      function mostrarDatos(cadenaPasada)
      {
         var cadenaLocal = cadenaPasada;
         var indice;
         var cadenaMostrar = "";

            for (indice in cadenaLocal) {
               cadenaMostrar = cadenaMostrar + cadenaLocal[indice] + "\n";
            }

            if (cadenaMostrar=="") {
               window.alert ("La lista está vacia!!");
            }
            else {
               window.alert (cadenaMostrar);
            }
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Ejecutar Programa" onclick="ejecutar ();" />
      </form>
  </body>
</html>
```

Ejercicio_03Especial.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio03Especial.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      var array1 = new Array();
      array1 = ["Enero", "Febrero", "Marzo",
              "Abril", "Mayo", "Junio",
              "Julio", "Agosto", "Septiembre",
              "Octubre", "Noviembre", "Diciembre"];

      function ejecutar (){
         var cadena = "";
         var indice;
         var array2 = new Array ();
         array2 = array1.reverse();

         for (indice in array2) {
            cadena = cadena + array2 [indice] + "\n";
         }
         window.alert (cadena);
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Ejecutar Programa" onclick="ejecutar ();" />
      </form>
  </body>
</html>
```

Ejercicio08.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio08.html -->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">

      function ejecutar (){
         var valor = parseInt(window.prompt (" Escriba valor a comprobar: "));
         espar (valor);
      }

      function espar(num) {
         var resto = num % 2;

         if (resto==0) {
            window.alert (" Es par!");
         }
         else {
            window.alert (" NO es par!");
         }
      }

   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Comprobar Número" onclick="ejecutar ();" />
      </form>
  </body>
</html>
```

Ejercicio09.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio09.html -->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">

      function ejecutar (){
         var cadena = new String ();
         var cadena = window.prompt (" Introduzca una cadena: ");
         var cadenaMayus = cadena.toUpperCase();
         var cadenaMinus = cadena.toLowerCase();

         if (cadena== cadenaMayus) {
            window.alert (" La cadena estaba en mayúsculas");
         }

         if (cadena== cadenaMinus) {
            window.alert (" La cadena estaba en minúsculas");
         }

         if (cadena!= cadenaMinus && cadena!= cadenaMayus ) {
            window.alert (" La cadena estaba mezclada");
         }
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Mayúsculas /Minúsculas" onclick="ejecutar ();" />
      </form>
  </body>
</html>
```

Ejercicio_10.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio10.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      function ejecutar (){
         var resultado; // Variable Booleana
         var cadenaPasada = window.prompt (" Escriba la cadena a comprobar: ");
         resultado = palindromo(cadenaPasada);
         if (resultado==true) {
            window.alert (" Es palíndromo!");
         }
         else {
            window.alert (" NO es palíndromo!");
         }
      }

      /* Función Palíndromos:
       * Argumentos: cadena pasada
       * Return: booleano (True si es palíndromo) */
      function palindromo(cadena) {
         // Variable que usaremos para saber si es palíndromo
         var iguales = true;

         // Pasar a minusculas la cadena
         var cadenaOriginal = cadena.toLowerCase();

         // Convertir la cadena en un array
         var letrasEspacios = cadenaOriginal.split("");

         // Eliminar los espacios en blanco
         var cadenaSinEspacios = "";
```

```
         for(i in letrasEspacios) {
            if(letrasEspacios[i] != " " &&
               letrasEspacios[i] != "," &&
               letrasEspacios[i] != ":") {
               cadenaSinEspacios += letrasEspacios[i];
            }
         }

         // Con la cadena en minúsculas y sin espacios, le damos la vuelta...
         var letras = cadenaSinEspacios.split("");
         var letrasReves = cadenaSinEspacios.split("").reverse();

         // Ahora comparamos ambas cadenas
         for(i in letras) {
            if(letras[i] != letrasReves[i]) {
               iguales = false;
               break;
            }
         }
         return iguales;
      }


      function palindromo_tere(cadena) {
         var palindromo = true;

         var cadenaInicial = new String();
         var cadenaRevertida = new String();

         var cadena1 = new String();
         var array1 = new Array();
         var array2 = new Array();
         var array3 = new Array();

         // La cadena en minusculas
         cadena1 = cadena.toLowerCase();

         // La cadena en minúsculas pasado a Array por palabras
         array1 = cadena1.split(" ");

         // Se quitan los espacios
         cadenaInicial = array1.join("");

         // La cadena se pasa a una Array por letras
         array2 = cadenaInicial.split("");

         // Se le da la vuelta
         array3 = array2.reverse();

         // Se unen los elementos del 2° Array en una cadena.
         cadenaRevertida = array3.join("");

         if(cadenaInicial != cadenaRevertida){
            palindromo = false;
         }

         return palindromo;
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Palíndromo" onclick="ejecutar ();" />
      </form>
  </body>
</html>
```

Ejercicio_10Especial.html

```
<!DOCTYPE html>
<html lang="es">       <!--Ejercicio10Especial.html-->
...
   <script language="javascript">
      function ejecutar () {
         var opcion = window.prompt
                  (  " >>MENU PRINCIPAL<<"   + "\n"+
                     " 1. Ec. 2°Grado. "     + "\n"+
                     " 2. Potencia. "        + "\n"+
                     " 3. Salir. "           + "\n"+
                     " Seleccione Opcion: ");

         if (opcion==1) {
            ecuacion2grado ();
         }

         if (opcion==2) {
            potencia ();
         }

         if (opcion==3) {
            salir ();
         }
      }

      function ecuacion2grado (){
         var a, b, c;
         var x1,x2;
         var numerador1;
         var numerador2;
         window.alert (" Programa Ecuación de Segundo grado: "+ "\n"+
                  " Por ejemplo x²-5x+6=0");
         a = parseInt (window.prompt (" Escriba valor para x²: "));
         b = parseInt (window.prompt (" Escriba valor para x: "));
         c = parseInt (window.prompt (" Escriba valor para término indep: "));

         var disc = b*b - (4*a*c);   // Para 1,-5,6 -> 1
         if (disc<0) {
            window.alert (" No hay solución");
         }
         else
         {
            numerador1 = -b + Math.sqrt (disc);
            numerador2 = -b - Math.sqrt (disc);

            x1 = numerador1 / 2*a;
            x2 = numerador2 / 2*a;

            window.alert (" Primera solución: "+x1);
            window.alert (" Segunda solución: "+x2);
         }
         ejecutar ();
      }

      function potencia() {
         var base;
         var exponente;
         var indice;
         var resultado = 1;
         base = window.prompt (" Escriba Base: ");
         exponente = window.prompt (" Escriba Exponente: ");

         /* Se obtiene el mismo resultado con un método de Math:
          * resultado = Math.pow (base,exponente)
          */
```

```
         if (exponente>=0) {
            for (indice=1; indice<=exponente; indice++) {
               resultado = resultado * base;
            }
         }
         else
         {
            for (indice=1; indice<= (-exponente); indice++) {
               resultado = resultado * base;
            }
            resultado = 1 / resultado;
         }

         window.alert (resultado);
      }

      function salir() {
         var seguir = false;
         seguir = window.confirm (" Desea Salir? ");
         if (seguir == true) {
            window.alert (" Gracias por usar este programa!");
         }
         else
         {
            ejecutar();
         }
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Iniciar Programa" onclick="ejecutar ();" />
      </form>
  </body>
</html>
```

Ejercicio_11.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio11.html-->
<head>
  <meta charset="UTF-8" />
  <title>Ejercicio 11 - DOM básico</title>
<script type="text/javascript">
   window.onload = function() {
      var enlaces = new Array ();
      var i;
      var contador=0;
      enlaces = document.getElementsByTagName ("a");
      for (i=0; i<enlaces.length; i++) {
         if (enlaces[i].href == "http://prueba/") {
            contador++;
         }
      }

      // N° de Enlaces (recorremos el array y se queda i=7)
      window.alert (" N° de Enlaces -> " + i);
      // Para sacar el penúltimo enlaces llamamos al penúltimo elemento del array
      window.alert (" Dirección del Penúltimo enlace: \n" + enlaces[enlaces.length-2].href);
      // Numero de enlaces que apuntan a http://prueba
      window.alert (" Enlaces que apuntan a Prueba: "+contador);

      // Numero de enlaces del tercer párrafo
      var parrafos = document.getElementsByTagName ("p");
      var enlaces = parrafos[2].getElementsByTagName ("a");
      alert (" Los enlaces del tercer párrafo son: " +enlaces.length);
   }
</script>
</head>
...
```

Ejercicio_11Especial.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio11Especial.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">
      var num;
      function ejecutar (){
         num = parseInt (window.prompt (     " Especifique la Tabla: "     + "\n"+
                                             " Si escribe 0, Sale del Programa"));
         if (num!=0) {
            multiplicar();
         }
         else        {
            window.alert (" Gracias por usar este Programa!");
         }
      }

      function multiplicar () {
         var i;
         var rdo;
         var cadena = " TABLA DE MULTIPLICAR "+num + "\n";

         if (num<1 || num >10) {
            ejecutar ();
         }
         else  {
            for (i=1; i<=10; i++) {
               rdo = num * i;
               cadena = cadena + num + " x " + i + " = " + rdo + "\n";
            }
            window.alert (cadena);
            seguir ();
         }
      }

      function seguir() {
         var continuar;
         if (num<=1) {
            continuar = window.confirm (" Aceptar, Tabla Posterior "+ (num+1) +"\n"+
                                        " Cancelar, Vuelve a Menú Principal");
         }
         if (num>=10) {
            continuar = window.confirm (" Aceptar, Menú Principal "+ "\n"+
                                        " Cancelar, Tabla Anterior "+ (num-1));
         }
         if (num>1 && num <10) {
            continuar = window.confirm (" Aceptar, Tabla Posterior "+ (num+1) +"\n"+
                                        " Cancelar, Tabla Anterior "+ (num-1));
         }

         if (continuar==true) {
            num++;
         }
         else
         {
            num--;
         }
         multiplicar();
      }
   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Programa Multiplicar" onclick="ejecutar ();" />
      </form>
  </body>
</html>
```

Ejercicio_11EspecialEspacial.html

```
<!DOCTYPE html>
<html lang="es">       <!--Ejercicio11_EspecialEspacial.html-->
<head>
  <meta charset="UTF-8" />
  <title>JavaScript</title>
   <script language="javascript">

      var n1, n2;
      var max_com_divisor = 0;

      function ejecutar() {
         var cadenaMenu = " MENU APLICACIÓN " + "\n" +
                      " 1. Ec. 2º Grado " + "\n" +
                      " 2. Factorial "       + "\n" +
                      " 3. ¿Es primo? "      + "\n" +
                      " 4. Fibonacci "       + "\n" +
                      " 5. Min Com Mult " + "\n" +
                      " 6. Max Com Div "  + "\n" +
                      " 7. Salir " + "\n";

         var opcion = parseInt (window.prompt (cadenaMenu));
         switch (opcion) {
            case 1: ecuacion2grado();break;
            case 2: factorial();            break;
            case 3: primo();                break;
            case 4: fibonacci();            break;
            case 5: mcm();                  break;
            case 6: mcd(6);                 break;
            default: salir ();              break;
         }
      }

      // Esta función ya estaba hecha
      function ecuacion2grado() {
      ...
         ejecutar ();
      }

      // Esta función ya estaba hecha
      function factorial() {
      ...
         ejecutar ();
      }

      function primo() {
         var divisores=0;
         var i;
         cadena = "";
         var num = parseInt (window.prompt (" Escriba valor a comprobar: "));

         // Dividimos entre todos los números entre 1 y el valor introducido
         for (i=1; i<=num; i++) {
            if (num %i ==0) {
               divisores++;
               cadena = cadena + i + " ";
            }
         }

         if (divisores>2) {
            alert (" NO es primo! \n" +" Divisores: "+ cadena );
         }
         else {
            alert (" Es primo! \n" +" Divisores: "+ cadena);
         }
         ejecutar();
      }
```

```
      function fibonacci() {
         //0 1 1 2 3 5 8 13 21 34...
         var cantidad = window.prompt (" Escriba N° Digitos Sucesión (min 3): ");
         var x = 0;
         var y = 1;
         var z;
         var cadena;
         cadena = x + " " + y + " ";

         var i;
         for (i=3; i<=cantidad; i++) {
            z = y + x;  // 1 = 1+0
            x = y;      // Ahora x vale el 1 de la y
            y = z;      // Ahora y vale el 1 de la z
            cadena = cadena + z + " ";
         }

         alert (cadena);
         ejecutar();
      }

      function mcd (opcion) {
         var numero1, numero2;

         var resto;
         numero1 = window.prompt (" Escriba el 1er número: ");     // 304
         numero2 = window.prompt (" Escriba el 2° número: ");      // 728

         n1 = numero1;
         n2 = numero2;

         // num1 = 728; num2 = 304 -> Algoritmo de Euclides
         // Controlo que num1 sea múltiplo de num2
         while (numero1 % numero2 != 0) {
            resto = numero1 % numero2;              // aux = 120
            numero1 = numero2;
            numero2 = resto;
         }

         max_com_divisor = numero2;

         if (opcion==6) {
            alert (" El Max Común Divisor de "
               + n1 + " y " + n2 + " es: "+ max_com_divisor);
            ejecutar();
         }

      }

      function mcm () {
         mcd(5);
         var min_com_multiplo = (n1 * n2) / max_com_divisor;
         alert (" El Min Común Múltiplo de " +
               n1 + " y " + n2 + " es: "+ min_com_multiplo);
         ejecutar();
      }

      function salir() {
         alert ("Adios!");
      }

   </script>
</head>
  <body>
      <form name="form1">
         <input type="button" value="Programa" onclick="ejecutar ();" />
      </form>
  </body>
</html>
```

Ejercicio_12.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio12.html-->
<head>
   <meta charset="UTF-8" />
   <title>Ejercicio 12 - DOM básico y atributos XHTML</title>
   <style type="text/css">
      .oculto { display: none; }
      .visible {  display: inline;}
   </style>
   <script type="text/javascript">
      function muestra() {
         var parrafo = document.getElementById ("adicional");
         parrafo.className = "visible";

         var enlace1 = document.getElementById ("enlace1");
         enlace1.className = "oculto";
         var enlace2 = document.getElementById ("enlace2");
         enlace2.className = "visible";
      }

      function oculta() {
         var parrafo = document.getElementById ("adicional");
         parrafo.className = "oculto";

         var enlace1 = document.getElementById ("enlace1");
         enlace1.className = "visible";
         var enlace2 = document.getElementById ("enlace2");
         enlace2.className = "oculto";
      }
   </script>
...
```

Ejercicio_13.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />                  <!--Ejercicio13.html-->
<title>Ejercicio 13 - DOM básico y atributos XHTML</title>

<script type="text/javascript">
   function anade() {
     var elemento = document.createElement("li");
     var texto = document.createTextNode("Elemento de prueba");
     elemento.appendChild(texto);

     var lista = document.getElementById("lista");
     lista.appendChild(elemento);

     var nuevoElemento = "<li>Texto de prueba</li>";
     lista.innerHTML = lista.innerHTML + nuevoElemento;
   }
</script>
</head>

<body>

<ul id="lista">
  <li>Lorem ipsum dolor sit amet</li>
  <li>Consectetuer adipiscing elit</li>
  <li>Sed mattis enim vitae orci</li>
  <li>Phasellus libero</li>
  <li>Maecenas nisl arcu</li>
</ul>

<input type="button" value="Añadir elemento" onclick="anade();">

</body>
</html>
```

Ejercicio_14.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
<title>Ejercicio 14 - DOM básico y atributos XHTML</title>
<script type="text/javascript">
function muestraOculta(num) {
   var parrafo = document.getElementById ("contenidos_"+num);
   var enlace = document.getElementById ("enlace_"+num);

   if (parrafo.style.display == "block" ||
      parrafo.style.display == "") {
      parrafo.style.display = "none";
      enlace.innerHTML = "Mostrar Contenido";
   }
   else
   {
      parrafo.style.display = "block";
      enlace.innerHTML = "Ocultar Contenido";
   }
}
</script>
</head>
<body>
<p id="contenidos_1">...</p>
<a id="enlace_1" href="#" onclick="muestraOculta('1');">Ocultar contenidos</a>
<br/>

<p id="contenidos_2">...</p>
<a id="enlace_2" href="#" onclick="muestraOculta('2');">Ocultar contenidos</a>
<br/>

<p id="contenidos_3">...</p>
<a id="enlace_3" href="#" onclick="muestraOculta('3');">Ocultar contenidos</a>
</body>
</html>
```

Ejercicio_15.html

```
<!DOCTYPE html> <html lang="es">     <!--Ejercicio15.html -->
<head>  <meta charset="UTF-8" />
  <style type="text/css">
   #info {  width: 200px;            height: 150px;
            border: 2px solid grey;  position: fixed;
   }
  </style>
   <script language="javascript">
      window.onload = function () {
         document.onmousemove = info;
         document.onkeypress = info;
         document.onclick = info;
      }

      function info(evento) {
         var micaja = document.getElementById ("info");
         var coordX, coordY;
         var coordPagX, coordPagY;
         var miArray = new Array ();

         // Type es el nombre del evento sin el on por delante
         switch (evento.type) {
            case 'mousemove':
               micaja.style.backgroundColor = "#FFFFFF";
               coordX = evento.clientX;
               coordY = evento.clientY;
               coordPagX = evento.pageX;
               coordPagY = evento.pageY;

               miArray =
                  ["Ratón",
                  "Navegador ("+coordX+","+coordY+")",
                  "Página ("+coordPagX+","+coordPagY+")" ];

               muestrainfo (miArray);
               break;

            case 'keypress':
               micaja.style.backgroundColor = "#CCE6FF";

               // Para sacar el código ASCII
               var codigo = evento.charCode || evento.keyCode;
               // Para convertir el ASCII en la letra
               var letra = String.fromCharCode (codigo);

               miArray = [    "Teclado",
                              "Caracter ("+letra+")",
                              "Codigo ASCII ("+codigo+")" ];
               muestrainfo (miArray);
               break;

            case 'click':
               micaja.style.backgroundColor = "#FFFFCC";
               break;
         }
      }

      function muestrainfo(mensaje) {
         var caja = document.getElementById ("info");
         caja.innerHTML = "<h3> " + mensaje[0] + "</h3>";
         caja.innerHTML += "<p> " + mensaje[1] + "</p>";
         caja.innerHTML += "<p> " + mensaje[2] + "</p>";
      }
   </script>
</head>
<body>
   <div id="info">...</div>
</body> </html>
```

Ejercicio_16.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio16.html -->
<head>
  <meta charset="UTF-8" />
  <style type="text/css">
   #info {  width: 200px;            height: 150px;
            border: 2px solid grey;  position: fixed;
   }
  </style>
   <script language="javascript">
      window.onload = function () {
         document.onclick = info;
      }

      // Manejador Semántico
      function info(evento) {
         var tamVentana = new Array ();
         tamVentana = tamVentanaNavegador();

         var anchoVentana = tamVentana [0];
         var altoVentana = tamVentana [1];
         var posHorizontal = "";
         var posVertical = "";
         var coordX = evento.pageX;
         var coordY = evento.pageY;

         if (coordX >= anchoVentana/2) {
            posHorizontal = "derecha";
         }
         else {   posHorizontal = "izquierda";
         }

         if (coordY >= altoVentana/2) {
            posVertical = "abajo";
         }
         else {   posVertical = "arriba";
         }

         muestrainfo ( [posHorizontal,posVertical] );

      }

      //screen.width -> ancho Pantalla
      //screen.height -> alto Pantalla
      //window.innerWidth -> ancho navegador (sin scroll)
      //window.innerheight -> alto navegador (sin scroll)
      //document.body.clientWidth -> ancho navegador (con scroll)
      //document.body.clientHeight -> alto navegador (con scroll)

      function tamVentanaNavegador() {
         var dimensiones = new Array ();
         dimensiones = [ document.body.clientWidth, document.body.clientHeight];
         return dimensiones;
      }

      function muestrainfo(mensaje) {
         var caja = document.getElementById ("info");
         caja.innerHTML =     "<h3> Posición </h3>";
         caja.innerHTML +=    "<p>" + mensaje[0] + "</p>";
         caja.innerHTML +=    "<p>" + mensaje[1] + "</p>";
      }
   </script>
</head>
<body>
   <div id="info">...</div>
   <p> </p><p> </p><p> </p><p> </p><p> </p>
   <p> </p><p> </p><p> </p><p> </p><p>FIN</p>
</body> </html>
```

Ejercicio 17

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio17.html-->
<head>
    <meta charset="utf-8">
   <title> Ejercicio 17</title>
   <style type="text/css">
      #info {  color: Maroon; }
      #texto   {  border: 2px solid blue;
               font: 15px cursive;
               color: MidnightBlue; }
   </style>
   <script language="javascript">
      function limita (miEvento,maxCaracteres) {
         campo = document.getElementById ("texto");
         var codCaracter = miEvento.charCode || miEvento.keyCode;  // ASCII

         if (     codCaracter == 8    || codCaracter==37
            ||    codCaracter==39     || codCaracter==46) {
            return true;
         }

         if (campo.value.length >= maxCaracteres) {
            return false;
         }
         else {
            return true;      }
      }

      function muestraInfo(maximoCaracteres) {
         var cajaInfo = document.getElementById ("info");
         if (campo.value.length >= maximoCaracteres) {
            cajaInfo.innerHTML = " Te has excedido del Tamaño";
         }
         else {
            cajaInfo.innerHTML =
            "Te quedan " + (maximoCaracteres-campo.value.length) + " caracteres";
         }
      }
   </script>
</head>
<body>
   <div id="info"> Máximo 100 Caracteres </div>
   <br />
   <textarea id="texto" rows="6" cols="30"
      onkeypress="return limita(event,100);"
      onkeyup="muestraInfo(100);"></textarea>
</body>
</html>
```

Ejercicio 18

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <style type="text/css">
    #d1, #m1, #d2, #m2 { width: 30px; }
    #a1, #a2 { width: 60px; }
  </style>
  <script type="text/javascript">
    var ms=0;
    var sg, min, h, d;

    function verDias ()
    {
        var d1 = document.getElementById("d1").value;
        var m1 = document.getElementById("m1").value;
        var a1= document.getElementById("a1").value;

        var d2= document.getElementById("d2").value;
        var m2= document.getElementById("m2").value;
        var a2= document.getElementById("a2").value;

        var fecha1 = new Date (a1,m1,d1);
        var fecha2 = new Date (a2,m2,d2);

        ms = fecha1.getTime() - fecha2.getTime();
        // hasta aqui está bien.

        sg = ms/1000;
        min = sg /60;
        h = min /60;
        d = h/24;
        alert (""+d+" Dias");
    }
</script>
<head>
<body>
<form name="formulario1"> Cálculo de Días entre 2 Fechas: <br />
    NOTA: La Fecha1 debe ser POSTERIOR a la fecha2 <br />
    Fecha1: <input type="text" maxlength="2" id="d1" />
            <input type="text" maxlength="2" id="m1" />
            <input type="text" maxlength="4" id="a1" /> <br />
    Fecha2: <input type="text" maxlength="2" id="d2" />
            <input type="text" maxlength="2" id="m2" />
            <input type="text" maxlength="4" id="a2" /> <br />
   <input type="button" name="boton1" value="clic" onClick=verDias()>
</form>
</body>
</html>
```

Ejercicio 18 Especial.html

```
<!DOCTYPE html>
<html lang="es">        <!--Ejercicio18 Especial.html-->
<head>
    <meta charset="utf-8">
   <title> Ejercicio 17</title>
   <style type="text/css">

   </style>
   <script language="javascript">
      function ejecutar() {
            var valor1 = parseInt (window.prompt (" Especifique el Limite Inferior: "));
            var valor2 = parseInt (window.prompt (" Especifique el Limite Superior: "));

            var numVeces = window.prompt (" Especifique el número de aleatorios: ");
         aleatorio (valor1, valor2, numVeces);
      }

      function aleatorio(inferior,superior,veces) {
         //Creo un array de apariciones de valores
         contadores = new Array ();
            var i, j;   // Indices de los For

            for (i=inferior; i<=superior; i++) {
              contadores[i] = 0;
            }

         var aleatorio;
         var i;
         var posibilidades;

         for (i=1; i<=veces; i++) {
            posibilidades = (superior+1) - inferior;       // 5 Posibilidades
            aleatorio = Math.random() * posibilidades;
            aleatorio = parseInt (Math.ceil (aleatorio));  // Genera el valor 2

            contadores[aleatorio]++; // Ese valor 2 lo metemos en contadores[2]
         }

         var caja = document.getElementById ("caja");
            caja.innerHTML = " Se ha generado " + veces + " aleatorios! <br />";

            // Recorro el array contadores para presentar los porcentajes
            for (j=inferior; j<=superior; j++) {
              caja.innerHTML += " % de " + j + ": " +
                                (contadores [j] * 100/ veces) + "% <br />";
            }
      }
   </script>
</head>
<body>
   <form id="formulario" action="#" method="post">
      Contador de Números Aleatorios: <br >
      <input type="button" value="Generar 1000 aleatorios!" onclick="ejecutar();" />
   </form>
   <br />
   <div id="caja">   </div>
</body>
</html>
```