
JAVASCRIPT
www.postparaprogramadores.com

Contenido

Objetos JavaScript

Javascript - Objetos

Javascript - Número

Javascript - Booleano

Javascript - Cadenas

Javascript - Matrices

Javascript - Fecha

Javascript - Matemáticas

Javascript - RegExp

Javascript - HTML DOM

JavaScript avanzado

Javascript - Manejo de errores

Javascript - Validaciones

Javascript - Animación

Javascript - Multimedia

Javascript - Depuración

Javascript - Mapa de imagen

Javascript - Navegadores

JavaScript - Descripción general

¿Qué es JavaScript?

JavaScript es un lenguaje dinámico de programación de computadoras. Es liviano y se usa más comúnmente como parte de las páginas web, cuyas implementaciones permiten que el script del lado del cliente interactúe con el usuario y cree páginas dinámicas. Es un lenguaje de programación interpretado con capacidades orientadas a objetos.

JavaScript se conoció por primera vez como **LiveScript,** pero Netscape cambió su nombre a JavaScript, posiblemente debido a la emoción que genera Java. JavaScript hizo su primera aparición en Netscape 2.0 en 1995 con el nombre **LiveScript** . El núcleo de uso general del lenguaje se ha incrustado en Netscape, Internet Explorer y otros navegadores web.

La especificación ECMA-262 definió una versión estándar del lenguaje JavaScript central.

- JavaScript es un lenguaje de programación ligero e interpretado.
- Diseñado para crear aplicaciones centradas en la red.
- Complementario e integrado con Java.
- Complementario e integrado con HTML.
- Abierto y multiplataforma

JavaScript del lado del cliente

JavaScript del lado del cliente es la forma más común del lenguaje. El script debe estar incluido o referenciado en un documento HTML para que el código sea interpretado por el navegador.

Significa que una página web no necesita ser un HTML estático, sino que puede incluir programas que interactúan con el usuario, controlan el navegador y crean dinámicamente contenido HTML.

El mecanismo del lado del cliente de JavaScript ofrece muchas ventajas sobre los scripts tradicionales del lado del servidor CGI. Por ejemplo, puede usar JavaScript para verificar si el usuario ha ingresado una dirección de correo electrónico válida en un campo de formulario.

El código JavaScript se ejecuta cuando el usuario envía el formulario, y solo si todas las entradas son válidas, se enviarán al servidor web.

JavaScript se puede utilizar para atrapar eventos iniciados por el usuario, como clics de botones, navegación de enlaces y otras acciones que el usuario inicia explícita o implícitamente.

Ventajas de JavaScript

Los méritos de usar JavaScript son:

- **Menos interacción con el servidor** : puede validar la entrada del usuario antes de enviar la página al servidor. Esto ahorra tráfico del servidor, lo que significa menos carga en su servidor.
- **Comentarios inmediatos a los visitantes** : no tienen que esperar a que se vuelva a cargar la página para ver si se han olvidado de ingresar algo.
- **Mayor interactividad** : puede crear interfaces que reaccionan cuando el usuario pasa el mouse sobre ellas o las activa a través del teclado.
- **Interfaces más ricas** : puede usar JavaScript para incluir elementos tales como componentes y controles deslizantes de arrastrar y soltar para proporcionar una interfaz enriquecida a los visitantes de su sitio.

Limitaciones de JavaScript

No podemos tratar a JavaScript como un lenguaje de programación completo. Carece de las siguientes características importantes:

- JavaScript del lado del cliente no permite la lectura o escritura de archivos. Esto se ha mantenido por razones de seguridad.
- JavaScript no se puede usar para aplicaciones de red porque no hay tal soporte disponible.
- JavaScript no tiene ninguna capacidad de subprocesamiento múltiple o multiprocesador.

Una vez más, JavaScript es un lenguaje de programación ligero e interpretado que le permite crear interactividad en páginas HTML estáticas.

Herramientas de desarrollo de JavaScript

Una de las principales fortalezas de JavaScript es que no requiere herramientas de desarrollo costosas. Puede comenzar con un editor de texto simple como el Bloc de notas. Dado que es un lenguaje interpretado dentro del contexto de un navegador web, ni siquiera necesita comprar un compilador.

Para simplificar nuestra vida, varios proveedores han creado herramientas de edición de JavaScript muy buenas. Algunos de ellos se enumeran aquí:

- **Microsoft FrontPage** : Microsoft ha desarrollado un popular editor HTML llamado FrontPage. FrontPage también proporciona a los desarrolladores web una serie de herramientas de JavaScript para ayudar en la creación de sitios web interactivos.
- **Macromedia Dreamweaver MX** : Macromedia Dreamweaver MX es un editor HTML y JavaScript muy popular entre la multitud de profesionales del desarrollo web. Proporciona varios componentes JavaScript preconstruidos útiles, se integra bien con las bases de datos y cumple con los nuevos estándares como XHTML y XML.
- **Macromedia HomeSite 5** : HomeSite 5 es un editor de HTML y JavaScript muy popular de Macromedia que se puede utilizar para administrar sitios web personales de manera efectiva.

¿Dónde está JavaScript hoy?

El estándar ECMAScript Edition 5 será la primera actualización que se lanzará en más de cuatro años. JavaScript 2.0 cumple con la Edición 5 del estándar ECMAScript, y la diferencia entre los dos es extremadamente menor.

La especificación para JavaScript 2.0 se puede encontrar en el siguiente sitio: http://www.ecmascript.org/

Hoy, el JavaScript de Netscape y el JScript de Microsoft se ajustan al estándar ECMAScript, aunque ambos lenguajes aún admiten las características que no forman parte del estándar.

JavaScript: sintaxis

JavaScript se puede implementar utilizando declaraciones de JavaScript que se colocan dentro de las etiquetas HTML **<script> ... </script>** en una página web.

Puede colocar las etiquetas **<script>** , que contienen su JavaScript, en cualquier lugar dentro de su página web, pero normalmente se recomienda que lo mantenga dentro de las etiquetas **<head>** .

La etiqueta <script> alerta al programa del navegador para comenzar a interpretar todo el texto entre estas etiquetas como un script. Una sintaxis simple de su JavaScript aparecerá de la siguiente manera.

```
<script ...>
   JavaScript code
</script>
```

La etiqueta de script toma dos atributos importantes:

- **Idioma** : este atributo especifica qué lenguaje de script está utilizando. Por lo general, su valor será javascript. Aunque las versiones recientes de HTML (y XHTML, su sucesor) han eliminado el uso de este atributo.
- **Tipo** : este atributo es lo que ahora se recomienda para indicar el lenguaje de secuencias de comandos en uso y su valor debe establecerse en "texto / javascript".

Entonces su segmento de JavaScript se verá así:

```
<script language = "javascript" type = "text/javascript">
   JavaScript code
</script>
```

Tu primer código JavaScript

Tomemos un ejemplo de muestra para imprimir "Hello World". Agregamos un comentario HTML opcional que rodea nuestro código JavaScript. Esto es para guardar nuestro código desde un navegador que no admite JavaScript. El comentario termina con un "// ->". Aquí "//" significa un comentario en JavaScript, por lo que lo agregamos para evitar que un navegador lea el final del comentario HTML como un fragmento de código JavaScript. A continuación, llamamos a una función **document.write** que escribe una cadena en nuestro documento HTML.

Esta función se puede usar para escribir texto, HTML o ambos. Echa un vistazo al siguiente código.

```
<html>
   <body>
      <script language = "javascript" type =
"text/javascript">
         <!--
            document.write("Hello World!")
         //-->
      </script>
   </body>
</html>
```

Este código producirá el siguiente resultado:

```
Hello World!
```

Espacios en blanco y saltos de línea

JavaScript ignora los espacios, las pestañas y las nuevas líneas que aparecen en los programas de JavaScript. Puede usar espacios, pestañas y líneas nuevas libremente en su programa y puede formatear e sangrar sus

programas de una manera ordenada y coherente que hace que el código sea fácil de leer y comprender.

Los punto y coma son opcionales

Las declaraciones simples en JavaScript generalmente van seguidas de un punto y coma, al igual que en C, C ++ y Java. Sin embargo, JavaScript le permite omitir este punto y coma si cada una de sus declaraciones se coloca en una línea separada. Por ejemplo, el siguiente código podría escribirse sin punto y coma.

```
<script language = "javascript" type = "text/javascript">
   <!--
      var1 = 10
      var2 = 20
   //-->
</script>
```

Pero cuando se formatea en una sola línea de la siguiente manera, debe usar punto y coma:

```
<script language = "javascript" type = "text/javascript">
   <!--
      var1 = 10; var2 = 20;
   //-->
</script>
```

Nota : es una buena práctica de programación usar punto y coma.

Sensibilidad del caso

JavaScript es un lenguaje sensible a mayúsculas y minúsculas. Esto significa que las palabras clave del lenguaje, las variables, los nombres de las funciones y cualquier otro identificador siempre deben escribirse con mayúsculas y minúsculas.

Por lo tanto, los identificadores **Time** y **TIME** transmitirán diferentes significados en JavaScript.

NOTA - Se debe tener cuidado al escribir nombres de variables y funciones en JavaScript.

Comentarios en JavaScript

JavaScript admite los comentarios de estilo C y C ++, por lo tanto,

- Cualquier texto entre un // y el final de una línea se trata como un comentario y JavaScript lo ignora.
- Cualquier texto entre los caracteres / * y * / se trata como un comentario. Esto puede abarcar varias líneas.

- JavaScript también reconoce la secuencia de apertura de comentarios HTML <! - . JavaScript trata esto como un comentario de una sola línea, tal como lo hace con el // comentario.
- La secuencia de cierre de comentarios HTML -> no es reconocida por JavaScript, por lo que debe escribirse como // ->.

Ejemplo

El siguiente ejemplo muestra cómo usar los comentarios en JavaScript.

```
<script language = "javascript" type = "text/javascript">
   <!--
      // This is a comment. It is similar to comments in C++

      /*
      * This is a multi-line comment in JavaScript
      * It is very similar to comments in C Programming
      */
   //-->
</script>
```

Habilitar JavaScript en los navegadores

Todos los navegadores modernos vienen con soporte incorporado para JavaScript. Con frecuencia, es posible que deba habilitar o deshabilitar este soporte manualmente. Este capítulo explica el procedimiento para habilitar y deshabilitar la compatibilidad con JavaScript en sus navegadores: Internet Explorer, Firefox, Chrome y Opera.

JavaScript en Internet Explorer

Aquí hay pasos simples para activar o desactivar JavaScript en su Internet Explorer:

- Siga **Herramientas** → **Opciones de Internet** desde el menú.
- Seleccione **la** pestaña **Seguridad** en el cuadro de diálogo.
- Haga clic en el botón **Nivel personalizado** .
- Desplácese hacia abajo hasta encontrar la opción de **secuencias de comandos** .
- Seleccione *Habilitar* botón de **opción** en **Active scripting** .
- Finalmente haga clic en Aceptar y salga

Para deshabilitar la compatibilidad con JavaScript en su Internet Explorer, debe seleccionar el botón de opción **Deshabilitar** en **Scripting activo** .

JavaScript en Firefox

Estos son los pasos para activar o desactivar JavaScript en Firefox:

- Abra una nueva pestaña → escriba **about: config** en la barra de direcciones.

- Luego encontrará el cuadro de diálogo de advertencia. Seleccione ¡ **Tendré cuidado, lo prometo!**
- Luego encontrará la lista de **opciones** de **configuración** en el navegador.
- En la barra de búsqueda, escriba **javascript.enabled** .
- Allí encontrará la opción para habilitar o deshabilitar javascript haciendo clic derecho en el valor de esa opción → **seleccione alternar** .

Si javascript.enabled es verdadero; se convierte en falso al hacer clic en **toogle** . Si javascript está deshabilitado; se habilita al hacer clic en alternar.

JavaScript en Chrome

Estos son los pasos para activar o desactivar JavaScript en Chrome:

- Haz clic en el menú de Chrome en la esquina superior derecha de tu navegador.
- Selecciona **Configuración** .
- Haga clic en **Mostrar configuración avanzada** al final de la página.
- En la sección **Privacidad** , haga clic en el botón Configuración de contenido.
- En la sección "Javascript", seleccione "No permitir que ningún sitio ejecute JavaScript" o "Permitir que todos los sitios ejecuten JavaScript (recomendado)".

JavaScript en Opera

Estos son los pasos para activar o desactivar JavaScript en Opera:

- Siga **Herramientas** → **Preferencias** desde el menú.
- Seleccione la opción **Avanzada** del cuadro de diálogo.
- Seleccione **Contenido** de los elementos enumerados.
- Seleccione la casilla de verificación **Habilitar JavaScript** .
- Finalmente haga clic en Aceptar y salga.

Para deshabilitar la compatibilidad con JavaScript en su Opera, no debe seleccionar la **casilla de verificación Habilitar JavaScript** .

Advertencia para navegadores que no son JavaScript

Si tiene que hacer algo importante usando JavaScript, puede mostrar un mensaje de advertencia al usuario usando etiquetas **<noscript>** .

Puede agregar un bloque **noscript** inmediatamente después del bloque script de la siguiente manera:

```
<html>
   <body>
      <script language = "javascript" type =
"text/javascript">
         <!--
            document.write("Hello World!")
```

```
         //-->
      </script>

      <noscript>
         Sorry...JavaScript is needed to go ahead.
      </noscript>
   </body>
</html>
```

Ahora, si el navegador del usuario no admite JavaScript o JavaScript no está habilitado, el mensaje de </noscript> se mostrará en la pantalla.

JavaScript - Colocación en archivo HTML

Existe una flexibilidad para incluir código JavaScript en cualquier parte de un documento HTML. Sin embargo, las formas más preferidas de incluir JavaScript en un archivo HTML son las siguientes:

- Script en la sección <head> ... </head>.
- Script en la sección <body> ... </body>.
- Script en las secciones <body> ... </body> y <head> ... </head>.
- Script en un archivo externo y luego incluir en la sección <head> ... </head>.

En la siguiente sección, veremos cómo podemos colocar JavaScript en un archivo HTML de diferentes maneras.

JavaScript en la sección <head> ... </head>

Si desea que se ejecute un script en algún evento, como cuando un usuario hace clic en algún lugar, colocará ese script en la cabeza de la siguiente manera:

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function sayHello() {
               alert("Hello World")
            }
         //-->
      </script>
   </head>

   <body>
      <input type = "button" onclick = "sayHello()" value =
"Say Hello" />
   </body>
</html>
```

Este código producirá los siguientes resultados:

JavaScript en la sección <body> ... </body>

Si necesita que se ejecute una secuencia de comandos a medida que se carga la página para que la secuencia de comandos genere contenido en la página, la secuencia de comandos va en la parte <body> del documento. En este caso, no tendría ninguna función definida usando JavaScript. Echa un vistazo al siguiente código.

```
<html>
   <head>
   </head>

   <body>
      <script type = "text/javascript">
         <!--
            document.write("Hello World")
         //-->
      </script>

      <p>This is web page body </p>
   </body>
</html>
```

Este código producirá los siguientes resultados:

JavaScript en las secciones <body> y <head>

Puede poner su código JavaScript en la sección <head> y <body> del siguiente modo:

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function sayHello() {
               alert("Hello World")
            }
         //-->
      </script>
   </head>

   <body>
      <script type = "text/javascript">
         <!--
            document.write("Hello World")
         //-->
      </script>

      <input type = "button" onclick = "sayHello()" value =
"Say Hello" />
```

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

Este código producirá el siguiente resultado:

JavaScript en archivo externo

A medida que comience a trabajar más extensamente con JavaScript, es probable que encuentre casos en los que está reutilizando código JavaScript idéntico en varias páginas de un sitio.

No está restringido a mantener un código idéntico en varios archivos HTML. La etiqueta de **script** proporciona un mecanismo que le permite almacenar JavaScript en un archivo externo y luego incluirlo en sus archivos HTML.

Aquí hay un ejemplo para mostrar cómo puede incluir un archivo JavaScript externo en su código HTML usando la etiqueta de **script** y su atributo **src** .

```
<html>
   <head>
      <script type = "text/javascript" src = "filename.js"
></script>
   </head>

   <body>
      .......
   </body>
</html>
```

Para usar JavaScript desde una fuente de archivo externa, debe escribir todo su código fuente de JavaScript en un archivo de texto simple con la extensión ".js" y luego incluir ese archivo como se muestra arriba.

Por ejemplo, puede mantener el siguiente contenido en el archivo **filename.js** y luego puede usar la función **sayHello** en su archivo HTML después de incluir el archivo filename.js.

```
function sayHello() {
   alert("Hello World")
}
```

JavaScript - Variables

Tipos de datos de JavaScript

Una de las características más fundamentales de un lenguaje de programación es el conjunto de tipos de datos que admite. Estos son el tipo de valores que se pueden representar y manipular en un lenguaje de programación.

JavaScript le permite trabajar con tres tipos de datos primitivos:

- **Números,** por ej. 123, 120.50 etc.

- **Cadenas** de texto, por ejemplo, "Esta cadena de texto", etc.
- **Booleano,** por ejemplo, verdadero o falso.

JavaScript también define dos tipos de datos triviales, **nulos** e **indefinidos,** cada uno de los cuales define un solo valor. Además de estos tipos de datos primitivos, JavaScript admite un tipo de datos compuesto conocido como **objeto** . Cubriremos objetos en detalle en un capítulo separado.

Nota : JavaScript no distingue entre valores enteros y valores de punto flotante. Todos los números en JavaScript se representan como valores de punto flotante. JavaScript representa números usando el formato de coma flotante de 64 bits definido por el estándar IEEE 754.

Variables JavaScript

Como muchos otros lenguajes de programación, JavaScript tiene variables. Las variables se pueden considerar como contenedores con nombre. Puede colocar datos en estos contenedores y luego hacer referencia a los datos simplemente nombrando el contenedor.

Antes de usar una variable en un programa de JavaScript, debe declararla. Las variables se declaran con la palabra clave **var de la** siguiente manera.

```
<script type = "text/javascript">
   <!--
      var money;
      var name;
   //-->
</script>
```

También puede declarar múltiples variables con la misma palabra clave **var de la** siguiente manera:

```
<script type = "text/javascript">
   <!--
      var money, name;
   //-->
</script>
```

El almacenamiento de un valor en una variable se llama **inicialización de variable** . Puede hacer la inicialización de la variable en el momento de la creación de la variable o en un momento posterior cuando necesite esa variable.

Por ejemplo, puede crear una variable llamada **dinero** y asignarle el valor 2000.50 más tarde. Para otra variable, puede asignar un valor en el momento de la inicialización de la siguiente manera.

```
<script type = "text/javascript">
   <!--
      var name = "Ali";
      var money;
```

```
      money = 2000.50;
   //-->
</script>
```

Nota : utilice la palabra clave **var** solo para declaración o inicialización, una vez durante la vida útil de cualquier nombre de variable en un documento. No debe volver a declarar la misma variable dos veces.

JavaScript es un lenguaje sin **tipo** . Esto significa que una variable de JavaScript puede contener un valor de cualquier tipo de datos. A diferencia de muchos otros idiomas, no tiene que decirle a JavaScript durante la declaración de la variable qué tipo de valor tendrá la variable. El tipo de valor de una variable puede cambiar durante la ejecución de un programa y JavaScript se encarga de ello automáticamente.

Alcance variable de JavaScript

El alcance de una variable es la región de su programa en la que se define. Las variables de JavaScript tienen solo dos ámbitos.

- **Variables globales** : una variable global tiene un alcance global, lo que significa que se puede definir en cualquier parte de su código JavaScript.
- **Variables locales** : una variable local será visible solo dentro de una función donde se define. Los parámetros de la función siempre son locales para esa función.

Dentro del cuerpo de una función, una variable local tiene prioridad sobre una variable global con el mismo nombre. Si declara una variable local o parámetro de función con el mismo nombre que una variable global, efectivamente oculta la variable global. Eche un vistazo al siguiente ejemplo.

```
<html>
   <body onload = checkscope();>
      <script type = "text/javascript">
         <!--
            var myVar = "global";      // Declare a global
variable
            function checkscope( ) {
               var myVar = "local";    // Declare a local
variable
               document.write(myVar);
            }
         //-->
      </script>
   </body>
</html>
```

Esto produce el siguiente resultado:

```
local
```

Nombres de variables de JavaScript

Mientras nombra sus variables en JavaScript, tenga en cuenta las siguientes reglas.

- No debe usar ninguna de las palabras clave reservadas de JavaScript como nombre de variable. Estas palabras clave se mencionan en la siguiente sección. Por ejemplo, **ruptura** o **booleanas** nombres de variables no son válidos.
- Los nombres de variables de JavaScript no deben comenzar con un número (0-9). Deben comenzar con una letra o un carácter de subrayado. Por ejemplo, **123test** es un nombre de variable no válido pero **_123test** es válido.
- Los nombres de variables de JavaScript distinguen entre mayúsculas y minúsculas. Por ejemplo, **Nombre** y **nombre** son dos variables diferentes.

Palabras reservadas de JavaScript

En la siguiente tabla se incluye una lista de todas las palabras reservadas en JavaScript. No se pueden usar como variables de JavaScript, funciones, métodos, etiquetas de bucle o cualquier nombre de objeto.

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var

debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

JavaScript: operadores

¿Qué es un operador?

Tomemos una expresión simple **4 + 5 es igual a 9** . Aquí 4 y 5 se llaman **operandos** y '+' se llama **operador** . JavaScript admite los siguientes tipos de operadores.

- Operadores aritméticos
- Operadores de comparación
- Operadores lógicos (o relacionales)
- Operadores de Asignación
- Operadores condicionales (o ternarios)

Echemos un vistazo a todos los operadores uno por uno.

Operadores aritméticos

JavaScript admite los siguientes operadores aritméticos:

Suponga que la variable A tiene 10 y la variable B tiene 20, entonces -

No Señor.	Operador y Descripción
1	**+ (Adición)** Agrega dos operandos **Ej:** A + B dará 30
2	**- (Resta)** Resta el segundo operando del primero **Ej:** A - B dará -10
3	*** (Multiplicación)** Multiplica ambos operandos **Ej:** A * B dará 200
4 4	**/ (División)** Divide el numerador entre el denominador **Ej:** B / A dará 2
5 5	**% (Módulo)** Produce el resto de una división entera **Ej:** B% A dará 0
6 6	**++ (Incremento)** Aumenta un valor entero en uno **Ej:** A ++ dará 11
7 7	**- (Decremento)** Disminuye un valor entero en uno **Ej:** A-- dará 9

Nota : el operador de adición (+) funciona tanto para numérico como para cadenas. por ejemplo, "a" + 10 dará "a10".

Ejemplo

El siguiente código muestra cómo usar operadores aritméticos en JavaScript.

```
<html>
   <body>

      <script type = "text/javascript">
         <!--
            var a = 33;
            var b = 10;
            var c = "Test";
            var linebreak = "<br />";

            document.write("a + b = ");
            result = a + b;
            document.write(result);
            document.write(linebreak);

            document.write("a - b = ");
            result = a - b;
            document.write(result);
            document.write(linebreak);

            document.write("a / b = ");
            result = a / b;
            document.write(result);
            document.write(linebreak);

            document.write("a % b = ");
            result = a % b;
            document.write(result);
            document.write(linebreak);

            document.write("a + b + c = ");
            result = a + b + c;
            document.write(result);
            document.write(linebreak);

            a = ++a;
            document.write("++a = ");
            result = ++a;
            document.write(result);
            document.write(linebreak);

            b = --b;
            document.write("--b = ");
            result = --b;
            document.write(result);
            document.write(linebreak);
         //-->
      </script>

      Set the variables to different values and then try...
```

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

Salida

```
a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
++a = 35
--b = 8
Set the variables to different values and then try...
```

Operadores de comparación

JavaScript admite los siguientes operadores de comparación:

Suponga que la variable A tiene 10 y la variable B tiene 20, entonces -

No Señor.	Operador y Descripción
1	**= = (Igual)** Comprueba si el valor de dos operandos es igual o no, en caso afirmativo, la condición se vuelve verdadera. **Ej:** (A == B) no es cierto.
2	**! = (No es igual)** Comprueba si el valor de dos operandos es igual o no, si los valores no son iguales, entonces la condición se vuelve verdadera. **Ej:** (A! = B) es cierto.
3	**> (Mayor que)** Comprueba si el valor del operando izquierdo es mayor que el valor del operando derecho; en caso afirmativo, la condición se vuelve verdadera. **Ej:** (A> B) no es cierto.
4 4	**<(Menos de)** Comprueba si el valor del operando izquierdo es menor que el valor del operando derecho; en caso afirmativo, la condición se vuelve verdadera. **Ej:** (A <B) es cierto.

5 5	**> = (Mayor o igual que)** Comprueba si el valor del operando izquierdo es mayor o igual que el valor del operando derecho, en caso afirmativo, la condición se vuelve verdadera. **Ej:** (A> = B) no es cierto.
6 6	**<= (Menor o igual que)** Comprueba si el valor del operando izquierdo es menor o igual que el valor del operando derecho; en caso afirmativo, la condición se vuelve verdadera. **Ej:** (A <= B) es verdadero.

Ejemplo

El siguiente código muestra cómo usar operadores de comparación en JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = 10;
            var b = 20;
            var linebreak = "<br />";

            document.write("(a == b) => ");
            result = (a == b);
            document.write(result);
            document.write(linebreak);

            document.write("(a < b) => ");
            result = (a < b);
            document.write(result);
            document.write(linebreak);

            document.write("(a > b) => ");
            result = (a > b);
            document.write(result);
            document.write(linebreak);

            document.write("(a != b) => ");
            result = (a != b);
            document.write(result);
            document.write(linebreak);

            document.write("(a >= b) => ");
            result = (a >= b);
            document.write(result);
            document.write(linebreak);
```

```
            document.write("(a <= b) => ");
            result = (a <= b);
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      Set the variables to different values and different
operators and then try...
   </body>
</html>
```

Salida

```
(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
a <= b) => true
Set the variables to different values and different operators
and then try...
```

Operadores logicos

JavaScript admite los siguientes operadores lógicos:

Suponga que la variable A tiene 10 y la variable B tiene 20, entonces -

No Señor.	Operador y Descripción
1	**&& (Lógico Y)** Si ambos operandos son distintos de cero, la condición se vuelve verdadera. **Ej:** (A y B) es cierto.
2	**|| (O lógico)** Si alguno de los dos operandos no es cero, entonces la condición se vuelve verdadera. **Ej:** (A || B) es cierto.
3	**! (Lógico NO)** Invierte el estado lógico de su operando. Si una condición es verdadera, entonces el operador lógico NO la hará falsa. **Ej** :! (A y B) es falso.

Ejemplo

Pruebe el siguiente código para aprender a implementar operadores lógicos en JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = true;
            var b = false;
            var linebreak = "<br />";

            document.write("(a && b) => ");
            result = (a && b);
            document.write(result);
            document.write(linebreak);

            document.write("(a || b) => ");
            result = (a || b);
            document.write(result);
            document.write(linebreak);

            document.write("!(a && b) => ");
            result = (!(a && b));
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      <p>Set the variables to different values and different
operators and then try...</p>
   </body>
</html>
```

Salida

```
(a && b) => false
(a || b) => true
!(a && b) => true
Set the variables to different values and different operators
and then try...
```

Operadores bit a bit

JavaScript admite los siguientes operadores bit a bit:

Suponga que la variable A contiene 2 y la variable B contiene 3, entonces -

No	Operador y Descripción

Señor.	
1	**& (Bitwise Y)** Realiza una operación booleana AND en cada bit de sus argumentos enteros. **Ej:** (A y B) es 2.
2	**El | (BitWise O)** Realiza una operación OR booleana en cada bit de sus argumentos enteros. **Ej:** (A | B) es 3.
3	**^ (Bitwise XOR)** Realiza una operación OR exclusiva booleana en cada bit de sus argumentos enteros. OR exclusivo significa que el operando uno es verdadero o el operando dos es verdadero, pero no ambos. **Ej:** (A ^ B) es 1.
4 4	**~ (No bit a bit)** Es un operador unario y opera invirtiendo todos los bits en el operando. **Ej:** (~ B) es -4.
5 5	**<< (desplazamiento a la izquierda)** Mueve todos los bits en su primer operando a la izquierda por el número de lugares especificados en el segundo operando. Los bits nuevos se rellenan con ceros. Cambiar un valor a la izquierda por una posición equivale a multiplicarlo por 2, cambiar dos posiciones equivale a multiplicar por 4, y así sucesivamente. **Ej:** (A << 1) es 4.
6 6	**>> (Desplazamiento a la derecha)** Operador binario de desplazamiento a la derecha. El valor del operando izquierdo se mueve hacia la derecha por la cantidad de bits especificados por el operando derecho. **Ej:** (A >> 1) es 1.
7 7	**>>> (Desplazamiento a la derecha con cero)** Este operador es igual que el operador >>, excepto que los bits desplazados a la izquierda son siempre cero. **Ej:** (A >>> 1) es 1.

Ejemplo

Pruebe el siguiente código para implementar el operador Bitwise en JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = 2; // Bit presentation 10
            var b = 3; // Bit presentation 11
            var linebreak = "<br />";

            document.write("(a & b) => ");
            result = (a & b);
            document.write(result);
            document.write(linebreak);

            document.write("(a | b) => ");
            result = (a | b);
            document.write(result);
            document.write(linebreak);

            document.write("(a ^ b) => ");
            result = (a ^ b);
            document.write(result);
            document.write(linebreak);

            document.write("(~b) => ");
            result = (~b);
            document.write(result);
            document.write(linebreak);

            document.write("(a << b) => ");
            result = (a << b);
            document.write(result);
            document.write(linebreak);

            document.write("(a >> b) => ");
            result = (a >> b);
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      <p>Set the variables to different values and different
operators and then try...</p>
   </body>
</html>
```

```
(a & b) => 2
(a | b) => 3
(a ^ b) => 1
(~b) => -4
```

```
(a << b) => 16
(a >> b) => 0
Set the variables to different values and different operators
and then try...
```

Operadores de Asignación

JavaScript admite los siguientes operadores de asignación:

No Señor.	Operador y Descripción
1	**= (Asignación simple)** Asigna valores del operando del lado derecho al operando del lado izquierdo **Ej:** C = A + B asignará el valor de A + B a C
2	**+ = (Agregar y asignación)** Agrega el operando derecho al operando izquierdo y asigna el resultado al operando izquierdo. **Ej:** C + = A es equivalente a C = C + A
3	**- = (Restar y Asignación)** Resta el operando derecho del operando izquierdo y asigna el resultado al operando izquierdo. **Ej:** C - = A es equivalente a C = C - A
4 4	*** = (Multiplicar y Asignación)** Multiplica el operando derecho con el operando izquierdo y asigna el resultado al operando izquierdo. **Ej:** C * = A es equivalente a C = C * A
5 5	**/ = (División y asignación)** Divide el operando izquierdo con el operando derecho y asigna el resultado al operando izquierdo. **Ej:** C / = A es equivalente a C = C / A
6 6	**% = (Módulos y asignación)** Toma módulo usando dos operandos y asigna el resultado al operando izquierdo. **Ej:** C% = A es equivalente a C = C% A

Nota : La misma lógica se aplica a los operadores de Bitwise, por lo que se convertirán en << =, >> =, >> =, & =, | = y ^ =.

Ejemplo

Pruebe el siguiente código para implementar el operador de asignación en JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = 33;
            var b = 10;
            var linebreak = "<br />";

            document.write("Value of a => (a = b) => ");
            result = (a = b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a += b) => ");
            result = (a += b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a -= b) => ");
            result = (a -= b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a *= b) => ");
            result = (a *= b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a /= b) => ");
            result = (a /= b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a %= b) => ");
            result = (a %= b);
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      <p>Set the variables to different values and different
operators and then try...</p>
   </body>
</html>
```

Salida

```
Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0
Set the variables to different values and different operators
and then try...
```

Operador misceláneo

Discutiremos aquí dos operadores que son bastante útiles en JavaScript: el **operador condicional** (? :) y el **operador typeof** .

Operador condicional (? :)

El operador condicional primero evalúa una expresión para un valor verdadero o falso y luego ejecuta una de las dos declaraciones dadas dependiendo del resultado de la evaluación.

No Señor.	Operador y Descripción
1	**? : (Condicional)** Si la condición es verdadera? Entonces valor X: de lo contrario, valor Y

Ejemplo

Pruebe el siguiente código para comprender cómo funciona el operador condicional en JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = 10;
            var b = 20;
            var linebreak = "<br />";

            document.write ("((a > b) ? 100 : 200) => ");
            result = (a > b) ? 100 : 200;
            document.write(result);
            document.write(linebreak);

            document.write ("((a < b) ? 100 : 200) => ");
```

```
            result = (a < b) ? 100 : 200;
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      <p>Set the variables to different values and different
operators and then try...</p>
   </body>
</html>
```

Salida

```
((a > b) ? 100 : 200) => 200
((a < b) ? 100 : 200) => 100
Set the variables to different values and different operators
and then try...
```

tipo de operador

El operador **typeof** es un operador unario que se coloca antes de su único operando, que puede ser de cualquier tipo. Su valor es una cadena que indica el tipo de datos del operando.

El operador *typeof se* evalúa como "número", "cadena" o "booleano" si su operando es un número, cadena o valor booleano y devuelve verdadero o falso en función de la evaluación.

He aquí una lista de los valores de retorno de la **typeof** Operador.

Tipo	Cadena devuelta por typeof
Número	"número"
Cuerda	"cuerda"
Booleano	booleano
Objeto	"objeto"
Función	"función"
Indefinido	"indefinido"

Nulo	"objeto"

Ejemplo

El siguiente código muestra cómo implementar el operador **typeof** .

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = 10;
            var b = "String";
            var linebreak = "<br />";

            result = (typeof b == "string" ? "B is String" :
"B is Numeric");
            document.write("Result => ");
            document.write(result);
            document.write(linebreak);

            result = (typeof a == "string" ? "A is String" :
"A is Numeric");
            document.write("Result => ");
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      <p>Set the variables to different values and different
operators and then try...</p>
   </body>
</html>
```

Salida

```
Result => B is String
Result => A is Numeric
Set the variables to different values and different operators
and then try...
```

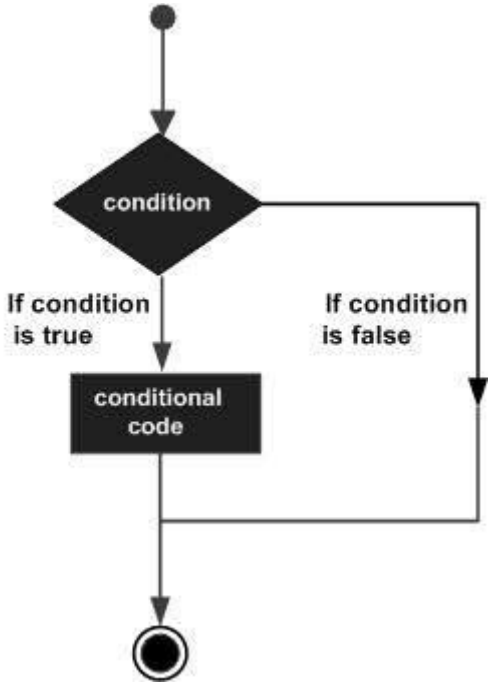
JavaScript - if ... else Declaración

Mientras escribe un programa, puede haber una situación en la que necesite adoptar uno de un conjunto determinado de rutas. En tales casos, debe usar declaraciones condicionales que le permitan a su programa tomar decisiones correctas y realizar acciones correctas.

JavaScript admite declaraciones condicionales que se utilizan para realizar diferentes acciones en función de diferentes condiciones. Aquí explicaremos la declaración **if..else** .

Diagrama de flujo de if-else

El siguiente diagrama de flujo muestra cómo funciona la instrucción if-else.



JavaScript admite las siguientes formas de declaración **if..else** :

- si la declaración
- si ... otra declaración
- si ... más si ... declaración.

si la declaración

La declaración **if** es la declaración de control fundamental que permite que JavaScript tome decisiones y ejecute declaraciones condicionalmente.

Sintaxis

La sintaxis para una instrucción if básica es la siguiente:

```
if (expression) {
   Statement(s) to be executed if expression is true
}
```

Aquí se evalúa una expresión de JavaScript. Si el valor resultante es verdadero, se ejecutan las declaraciones dadas. Si la expresión es falsa, entonces no se ejecutará ninguna declaración. La mayoría de las veces, utilizará operadores de comparación mientras toma decisiones.

Ejemplo

Pruebe el siguiente ejemplo para comprender cómo funciona la instrucción **if** .

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var age = 20;

            if( age > 18 ) {
               document.write("<b>Qualifies for
driving</b>");
            }
         //-->
      </script>
      <p>Set the variable to different value and then
try...</p>
   </body>
</html>
```

Salida

```
Qualifies for driving
Set the variable to different value and then try...
```

si ... otra declaración

La instrucción **'if ... else'** es la siguiente forma de instrucción de control que permite que JavaScript ejecute instrucciones de una manera más controlada.

Sintaxis

```
if (expression) {
   Statement(s) to be executed if expression is true
} else {
   Statement(s) to be executed if expression is false
}
```

Aquí se evalúa la expresión de JavaScript. Si el valor resultante es verdadero, se ejecutan las declaraciones dadas en el bloque 'si'. Si la expresión es falsa, se ejecutan las declaraciones dadas en el bloque else.

Ejemplo

Pruebe el siguiente código para aprender cómo implementar una instrucción if-else en JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
```

```
            var age = 15;

            if( age > 18 ) {
               document.write("<b>Qualifies for
driving</b>");
            } else {
               document.write("<b>Does not qualify for
driving</b>");
            }
         //-->
      </script>
      <p>Set the variable to different value and then
try...</p>
   </body>
</html>
```

Salida

```
Does not qualify for driving
Set the variable to different value and then try...
```

si ... más si ... declaración

La declaración **if ... else if ...** es una forma avanzada de **if ... else** que permite a JavaScript tomar una decisión correcta de varias condiciones.

Sintaxis

La sintaxis de una instrucción if-else-if es la siguiente:

```
if (expression 1) {
   Statement(s) to be executed if expression 1 is true
} else if (expression 2) {
   Statement(s) to be executed if expression 2 is true
} else if (expression 3) {
   Statement(s) to be executed if expression 3 is true
} else {
   Statement(s) to be executed if no expression is true
}
```

No hay nada especial sobre este código. Es solo una serie de declaraciones **if** , donde cada **if** es parte de la cláusula **else** de la declaración anterior. Las declaraciones se ejecutan en función de la condición verdadera, si ninguna de las condiciones es verdadera, se ejecuta el bloque **else** .

Ejemplo

Pruebe el siguiente código para aprender a implementar una instrucción if-else-if en JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var book = "maths";
            if( book == "history" ) {
               document.write("<b>History Book</b>");
            } else if( book == "maths" ) {
               document.write("<b>Maths Book</b>");
            } else if( book == "economics" ) {
               document.write("<b>Economics Book</b>");
            } else {
               document.write("<b>Unknown Book</b>");
            }
         //-->
      </script>
      <p>Set the variable to different value and then
try...</p>
   </body>
<html>
```

Salida

Maths Book
Set the variable to different value and then try...

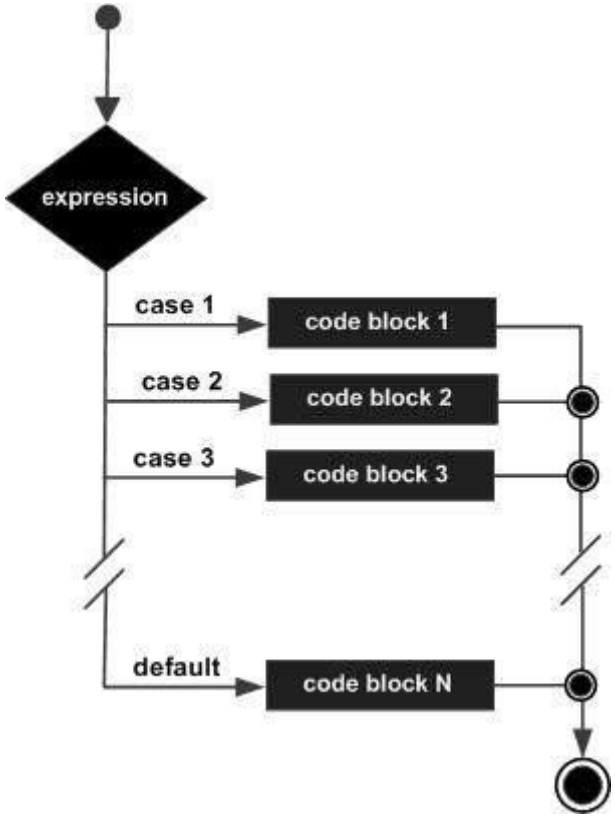
JavaScript - Cambiar caso

Puede usar múltiples declaraciones **if ... else ... if** , como en el capítulo anterior, para realizar una rama de múltiples vías. Sin embargo, esta no siempre es la mejor solución, especialmente cuando todas las ramas dependen del valor de una sola variable.

Comenzando con JavaScript 1.2, puede usar una instrucción **switch** que maneja exactamente esta situación, y lo hace de manera más eficiente que repetir las declaraciones **if ... else if** .

Diagrama de flujo

El siguiente diagrama de flujo explica que funciona una declaración de cambio de mayúsculas y minúsculas.



Sintaxis

El objetivo de una declaración de **cambio** es dar una expresión para evaluar y varias declaraciones diferentes para ejecutar en función del valor de la expresión. El intérprete verifica cada **caso** con el valor de la expresión hasta que se encuentre una coincidencia. Si nada coincide, se utilizará una condición **predeterminada** .

```
switch (expression) {
   case condition 1: statement(s)
   break;

   case condition 2: statement(s)
   break;
   ...

   case condition n: statement(s)
   break;

   default: statement(s)
}
```

Las declaraciones de **ruptura** indican el final de un caso particular. Si se omitieran, el intérprete continuaría ejecutando cada declaración en cada uno de los siguientes casos.

Explicaremos la declaración de **ruptura** en **el** capítulo **Control de bucle** .

Ejemplo

Pruebe el siguiente ejemplo para implementar la instrucción switch-case.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var grade = 'A';
            document.write("Entering switch block<br />");
            switch (grade) {
               case 'A': document.write("Good job<br />");
               break;

               case 'B': document.write("Pretty good<br />");
               break;

               case 'C': document.write("Passed<br />");
               break;

               case 'D': document.write("Not so good<br />");
               break;

               case 'F': document.write("Failed<br />");
               break;

               default:  document.write("Unknown grade<br
/>")
            }
            document.write("Exiting switch block");
         //-->
      </script>
      <p>Set the variable to different value and then
try...</p>
   </body>
</html>
```

Salida

```
Entering switch block
Good job
Exiting switch block
Set the variable to different value and then try...
```

Las declaraciones de ruptura juegan un papel importante en las declaraciones de cambio de caso. Pruebe el siguiente código que usa la instrucción switch-case sin ninguna instrucción break.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var grade = 'A';
```

```
            document.write("Entering switch block<br />");
            switch (grade) {
               case 'A': document.write("Good job<br />");
               case 'B': document.write("Pretty good<br />");
               case 'C': document.write("Passed<br />");
               case 'D': document.write("Not so good<br />");
               case 'F': document.write("Failed<br />");
               default: document.write("Unknown grade<br />")
            }
            document.write("Exiting switch block");
         //-->
      </script>
      <p>Set the variable to different value and then
try...</p>
   </body>
</html>
```

Salida

```
Entering switch block
Good job
Pretty good
Passed
Not so good
Failed
Unknown grade
Exiting switch block
Set the variable to different value and then try...
```

JavaScript: bucles While

Mientras escribe un programa, puede encontrarse con una situación en la que necesita realizar una acción una y otra vez. En tales situaciones, necesitaría escribir declaraciones de bucle para reducir el número de líneas.

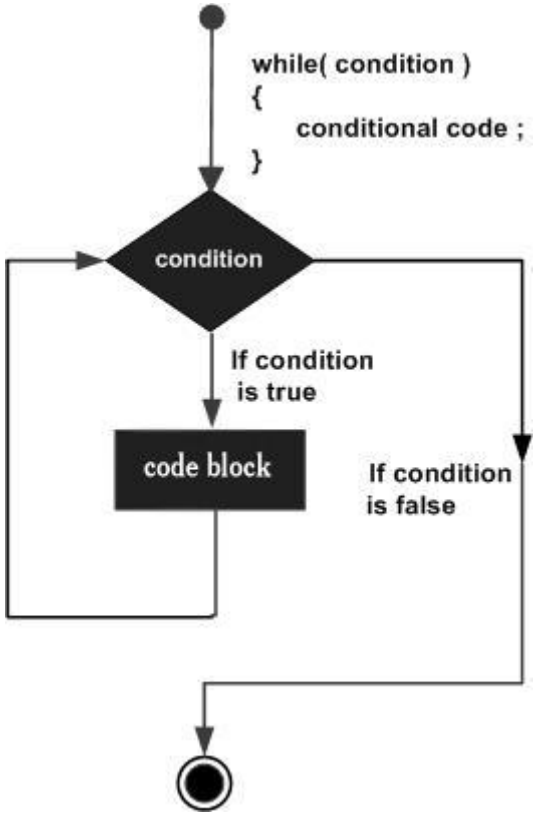
JavaScript admite todos los bucles necesarios para aliviar la presión de la programación.

El bucle while

El bucle más básico en JavaScript es el **tiempo** de bucle que se discute en este capítulo. El propósito de un **tiempo** de bucle es ejecutar una instrucción o bloque de código en varias ocasiones, siempre que una **expresión** es verdadera. Una vez que la expresión se vuelve **falsa,** el ciclo termina.

Diagrama de flujo

El diagrama de flujo del **ciclo while** tiene el siguiente aspecto:



Sintaxis

La sintaxis del **ciclo while** en JavaScript es la siguiente:

```
while (expression) {
   Statement(s) to be executed if expression is true
}
```

Ejemplo

Pruebe el siguiente ejemplo para implementar el bucle while.

```
<html>
   <body>

      <script type = "text/javascript">
         <!--
            var count = 0;
            document.write("Starting Loop ");

            while (count < 10) {
               document.write("Current Count : " + count +
"<br />");
               count++;
            }

            document.write("Loop stopped!");
         //-->
```

```
      </script>

      <p>Set the variable to different value and then
try...</p>
   </body>
</html>
```

Salida

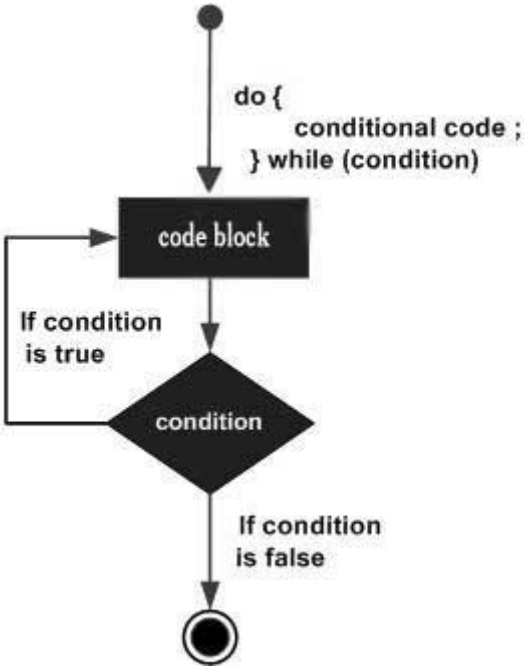
```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
Set the variable to different value and then try...
```

The do ... while Loop

El **do ... while** bucle es similar al **tiempo** de bucle excepto que la comprobación del estado ocurre al final del bucle. Esto significa que el bucle siempre se ejecutará al menos una vez, incluso si la condición es **falsa** .

Diagrama de flujo

El diagrama de flujo de un ciclo **do-** while sería el siguiente:



Sintaxis

La sintaxis para el bucle **do-** while en JavaScript es la siguiente:

```
do {
   Statement(s) to be executed;
} while (expression);
```

Nota : No se pierda el punto y coma utilizado al final del ciclo **do ...** while.

Ejemplo

Pruebe el siguiente ejemplo para aprender a implementar un bucle **do-** while en JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var count = 0;

            document.write("Starting Loop" + "<br />");
            do {
               document.write("Current Count : " + count +
"<br />");
               count++;
            }

            while (count < 5);
            document.write ("Loop stopped!");
         //-->
      </script>
      <p>Set the variable to different value and then
try...</p>
   </body>
</html>
```

Salida

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop Stopped!
Set the variable to different value and then try...
```

JavaScript: para bucle

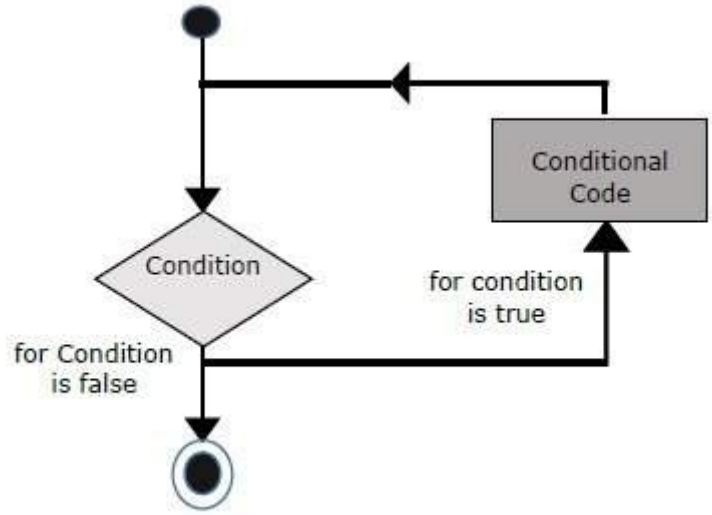
El bucle ' **for** ' es la forma más compacta de bucle. Incluye las siguientes tres partes importantes:

- La **inicialización del ciclo** donde inicializamos nuestro contador a un valor inicial. La instrucción de inicialización se ejecuta antes de que comience el ciclo.
- La **declaración de prueba** que probará si una condición dada es verdadera o no. Si la condición es verdadera, se ejecutará el código dado dentro del bucle; de lo contrario, el control saldrá del bucle.
- La **declaración de iteración** donde puede aumentar o disminuir su contador.

Puede poner las tres partes en una sola línea separada por punto y coma.

Diagrama de flujo

El diagrama de flujo de un bucle **for** en JavaScript sería el siguiente:



Sintaxis

La sintaxis de **for** loop es JavaScript:

```
for (initialization; test condition; iteration statement) {
   Statement(s) to be executed if test condition is true
}
```

Ejemplo

Pruebe el siguiente ejemplo para aprender cómo funciona un bucle **for** en JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var count;
            document.write("Starting Loop" + "<br />");

            for(count = 0; count < 10; count++) {
```

```
               document.write("Current Count : " + count );
               document.write("<br />");
            }
            document.write("Loop stopped!");
         //-->
      </script>
      <p>Set the variable to different value and then
try...</p>
   </body>
</html>
```

Salida

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
Set the variable to different value and then try...
```

JavaScript *para ... en* bucle

El bucle **for ... in** se usa para recorrer las propiedades de un objeto. Como todavía no hemos discutido los Objetos, es posible que no te sientas cómodo con este ciclo. Pero una vez que comprenda cómo se comportan los objetos en JavaScript, encontrará este bucle muy útil.

Sintaxis

La sintaxis de 'for..in' loop es -

```
for (variablename in object) {
   statement or block to execute
}
```

En cada iteración, una propiedad del **objeto** se asigna a **variablename** y este ciclo continúa hasta que se agoten todas las propiedades del objeto.

Ejemplo

Pruebe el siguiente ejemplo para implementar el bucle 'for-in'. Imprime el objeto **Navigator** del navegador web .

```
<html>
   <body>
```

```
      <script type = "text/javascript">
         <!--
            var aProperty;
            document.write("Navigator Object Properties<br />
");
            for (aProperty in navigator) {
               document.write(aProperty);
               document.write("<br />");
            }
            document.write ("Exiting from the loop!");
         //-->
      </script>
      <p>Set the variable to different object and then
try...</p>
   </body>
</html>
```

Salida

```
Navigator Object Properties
serviceWorker
webkitPersistentStorage
webkitTemporaryStorage
geolocation
doNotTrack
onLine
languages
language
userAgent
product
platform
appVersion
appName
appCodeName
hardwareConcurrency
maxTouchPoints
vendorSub
vendor
productSub
cookieEnabled
mimeTypes
plugins
javaEnabled
getStorageUpdates
getGamepads
webkitGetUserMedia
vibrate
getBattery
sendBeacon
registerProtocolHandler
unregisterProtocolHandler
Exiting from the loop!
```

```
Set the variable to different object and then try...
```

JavaScript - Control de bucle

JavaScript proporciona control total para manejar bucles y cambiar declaraciones. Puede haber una situación en la que necesite salir de un bucle sin llegar al fondo. También puede haber una situación en la que desee omitir una parte de su bloque de código y comenzar la próxima iteración del bucle.

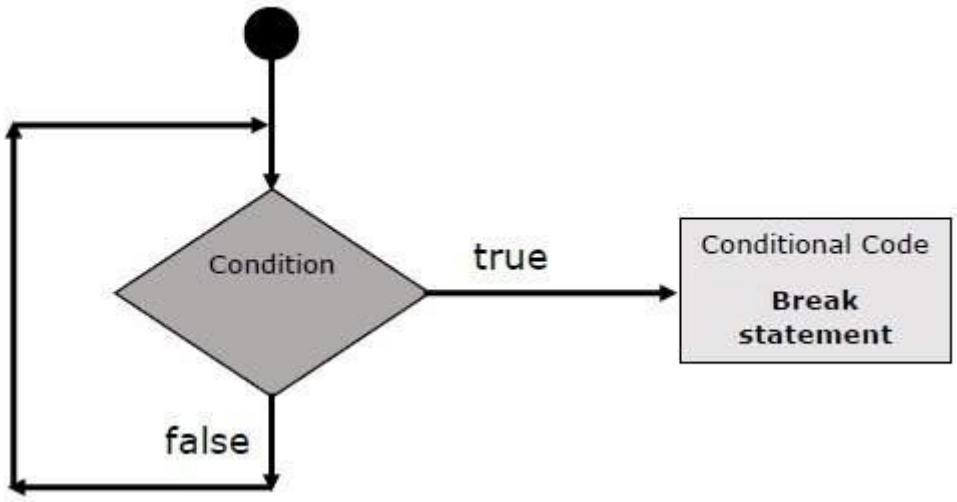
Para manejar todas estas situaciones, JavaScript proporciona declaraciones de **interrupción** y **continuación** . Estas declaraciones se utilizan para salir inmediatamente de cualquier bucle o para iniciar la próxima iteración de cualquier bucle, respectivamente.

La declaración de descanso

La declaración de **interrupción** , que se introdujo brevemente con la declaración de *cambio* , se usa para salir de un bucle temprano, separándose de las llaves.

Diagrama de flujo

El diagrama de flujo de una declaración de ruptura se vería de la siguiente manera:



Ejemplo

El siguiente ejemplo ilustra el uso de una instrucción **break** con un ciclo while. Observe cómo el ciclo se rompe temprano una vez que **x** alcanza 5 y llega a la declaración **document.write (..)** justo debajo de la llave de cierre -

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
```

```
         var x = 1;
         document.write("Entering the loop<br /> ");

         while (x < 20) {
            if (x == 5) {
               break;   // breaks out of loop completely
            }
            x = x + 1;
            document.write( x + "<br />");
         }
         document.write("Exiting the loop!<br /> ");
         //-->
      </script>

      <p>Set the variable to different value and then
try...</p>
   </body>
</html>
```

Salida

```
Entering the loop
2
3
4
5
Exiting the loop!
Set the variable to different value and then try...
```

Ya hemos visto el uso de la declaración **break** dentro de **una** declaración **switch** .

La declaración continua

La instrucción **continue** le dice al intérprete que inicie inmediatamente la próxima iteración del bucle y omita el bloque de código restante. Cuando se encuentra una instrucción de **continuación** , el flujo del programa se mueve a la expresión de verificación de bucle de inmediato y si la condición sigue siendo verdadera, entonces comienza la siguiente iteración, de lo contrario el control sale del bucle.

Ejemplo

Este ejemplo ilustra el uso de una instrucción **continue** con un ciclo while. Observe cómo se usa la instrucción **continue** para omitir la impresión cuando el índice contenido en la variable **x** alcanza 5 -

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

```
         <!--
            var x = 1;
            document.write("Entering the loop<br /> ");

            while (x < 10) {
               x = x + 1;

               if (x == 5) {
                  continue;   // skip rest of the loop body
               }
               document.write( x + "<br />");
            }
            document.write("Exiting the loop!<br /> ");
         //-->
      </script>
      <p>Set the variable to different value and then
try...</p>
   </body>
</html>
```

Salida

```
Entering the loop
2
3
4
6
7
8
9
10
Exiting the loop!
Set the variable to different value and then try...
```

Usar etiquetas para controlar el flujo

A partir de JavaScript 1.2, se puede usar una etiqueta con **pausa** y **continuar** controlando el flujo con mayor precisión. Una **etiqueta** es simplemente un identificador seguido de dos puntos (:) que se aplica a una declaración o un bloque de código. Veremos dos ejemplos diferentes para entender cómo usar etiquetas con pausa y continuar.

Nota : No se permiten saltos de línea entre la instrucción **'continuar'** o **'salto'** y su nombre de etiqueta. Además, no debe haber ninguna otra declaración entre un nombre de etiqueta y un bucle asociado.

Pruebe los siguientes dos ejemplos para comprender mejor las etiquetas.

Ejemplo 1

El siguiente ejemplo muestra cómo implementar Label con una instrucción break.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            document.write("Entering the loop!<br /> ");
            outerloop:        // This is the label name
            for (var i = 0; i < 5; i++) {
               document.write("Outerloop: " + i + "<br />");
               innerloop:
               for (var j = 0; j < 5; j++) {
                  if (j > 3 ) break ;           // Quit the
innermost loop
                  if (i == 2) break innerloop;  // Do the
same thing
                  if (i == 4) break outerloop;  // Quit the
outer loop
                  document.write("Innerloop: " + j + " <br
/>");
               }
            }
            document.write("Exiting the loop!<br /> ");
         //-->
      </script>
   </body>
</html>
```

Salida

```
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 2
Outerloop: 3
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 4
Exiting the loop!
```

Ejemplo 2

```
<html>
   <body>

      <script type = "text/javascript">
         <!--
         document.write("Entering the loop!<br /> ");
         outerloop:     // This is the label name

         for (var i = 0; i < 3; i++) {
            document.write("Outerloop: " + i + "<br />");
            for (var j = 0; j < 5; j++) {
               if (j == 3) {
                  continue outerloop;
               }
               document.write("Innerloop: " + j + "<br />");
            }
         }

         document.write("Exiting the loop!<br /> ");
         //-->
      </script>

   </body>
</html>
```

Salida

```
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 2
Innerloop: 0
Innerloop: 1
Innerloop: 2
Exiting the loop!
```

JavaScript - Funciones

Una función es un grupo de código reutilizable al que se puede llamar en cualquier parte de su programa. Esto elimina la necesidad de escribir el mismo código una y otra vez. Ayuda a los programadores a escribir códigos modulares. Las funciones permiten que un programador divida un gran programa en una serie de funciones pequeñas y manejables.

Al igual que cualquier otro lenguaje de programación avanzado, JavaScript también admite todas las características necesarias para escribir código modular utilizando funciones. Debe haber visto funciones como **alert ()** y **write ()** en los capítulos anteriores. Estábamos usando estas funciones una y otra vez, pero solo se habían escrito en JavaScript central una vez.

JavaScript nos permite escribir nuestras propias funciones también. Esta sección explica cómo escribir sus propias funciones en JavaScript.

Definición de función

Antes de usar una función, necesitamos definirla. La forma más común de definir una función en JavaScript es mediante el uso de la palabra clave de **función** , seguida de un nombre de función único, una lista de parámetros (que puede estar vacía) y un bloque de enunciado rodeado de llaves.

Sintaxis

La sintaxis básica se muestra aquí.

```
<script type = "text/javascript">
   <!--
      function functionname(parameter-list) {
         statements
      }
   //-->
</script>
```

Ejemplo

Prueba el siguiente ejemplo. Define una función llamada sayHello que no toma parámetros:

```
<script type = "text/javascript">
   <!--
      function sayHello() {
         alert("Hello there");
      }
   //-->
</script>
```

Llamar a una función

Para invocar una función en algún lugar posterior en el script, simplemente necesitará escribir el nombre de esa función como se muestra en el siguiente código.

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

```
         function sayHello() {
            document.write ("Hello there!");
         }
      </script>

   </head>

   <body>
      <p>Click the following button to call the function</p>
      <form>
         <input type = "button" onclick = "sayHello()" value
= "Say Hello">
      </form>
      <p>Use different text in write method and then
try...</p>
   </body>
</html>
```

Salida

Parámetros de funciones

Hasta ahora, hemos visto funciones sin parámetros. Pero hay una facilidad para pasar diferentes parámetros mientras se llama a una función. Estos parámetros pasados se pueden capturar dentro de la función y cualquier manipulación se puede hacer sobre esos parámetros. Una función puede tomar múltiples parámetros separados por comas.

Ejemplo

Prueba el siguiente ejemplo. Hemos modificado nuestra función **sayHello** aquí. Ahora se necesitan dos parámetros.

```
<html>
   <head>
      <script type = "text/javascript">
         function sayHello(name, age) {
            document.write (name + " is " + age + " years
old.");
         }
      </script>
   </head>

   <body>
      <p>Click the following button to call the function</p>
      <form>
         <input type = "button" onclick = "sayHello('Zara',
7)" value = "Say Hello">
      </form>
```

```
      <p>Use different parameters inside the function and
then try...</p>
   </body>
</html>
```

Salida

La declaración de devolución

Una función de JavaScript puede tener una declaración de **devolución** opcional . Esto es necesario si desea devolver un valor de una función. Esta declaración debería ser la última declaración en una función.

Por ejemplo, puede pasar dos números en una función y luego puede esperar que la función devuelva su multiplicación en su programa de llamadas.

Ejemplo

Prueba el siguiente ejemplo. Define una función que toma dos parámetros y los concatena antes de devolver el resultante en el programa de llamada.

```
<html>
   <head>
      <script type = "text/javascript">
         function concatenate(first, last) {
            var full;
            full = first + last;
            return full;
         }
         function secondFunction() {
            var result;
            result = concatenate('Zara', 'Ali');
            document.write (result );
         }
      </script>
   </head>

   <body>
      <p>Click the following button to call the function</p>
      <form>
         <input type = "button" onclick = "secondFunction()"
value = "Call Function">
      </form>
      <p>Use different parameters inside the function and
then try...</p>
  </body>
</html>
```

Salida

Hay mucho que aprender sobre las funciones de JavaScript, sin embargo, hemos cubierto los conceptos más importantes en este tutorial.

- Funciones anidadas de JavaScript
- Constructor de funciones JavaScript ()
- Literales de funciones de JavaScript

JavaScript - Eventos

¿Qué es un evento?

La interacción de JavaScript con HTML se maneja a través de eventos que ocurren cuando el usuario o el navegador manipula una página.

Cuando se carga la página, se llama evento. Cuando el usuario hace clic en un botón, ese clic también es un evento. Otros ejemplos incluyen eventos como presionar cualquier tecla, cerrar una ventana, cambiar el tamaño de una ventana, etc.

Los desarrolladores pueden usar estos eventos para ejecutar respuestas codificadas en JavaScript, lo que hace que los botones cierren ventanas, se muestren mensajes a los usuarios, se validen los datos y prácticamente cualquier otro tipo de respuesta imaginable.

Los eventos son parte del Nivel 3 del Modelo de objetos de documento (DOM) y cada elemento HTML contiene un conjunto de eventos que pueden activar el código JavaScript.

Siga este pequeño tutorial para comprender mejor la Referencia de eventos HTML . Aquí veremos algunos ejemplos para comprender una relación entre Evento y JavaScript:

onclick Tipo de evento

Este es el tipo de evento más utilizado que ocurre cuando un usuario hace clic en el botón izquierdo de su mouse. Puede poner su validación, advertencia, etc., contra este tipo de evento.

Ejemplo

Prueba el siguiente ejemplo.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function sayHello() {
               alert("Hello World")
            }
         //-->
```

```
      </script>
   </head>

   <body>
      <p>Click the following button and see result</p>
      <form>
         <input type = "button" onclick = "sayHello()" value
= "Say Hello" />
      </form>
   </body>
</html>
```

Salida

Tipo de evento de envío

onsubmit es un evento que ocurre cuando intentas enviar un formulario. Puede poner su validación de formulario contra este tipo de evento.

Ejemplo

El siguiente ejemplo muestra cómo usar onsubmit. Aquí estamos llamando a una función **validate ()** antes de enviar los datos de un formulario al servidor web. Si la función **validate ()** devuelve verdadero, se enviará el formulario; de lo contrario, no se enviarán los datos.

Prueba el siguiente ejemplo.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function validation() {
               all validation goes here
               .........
               return either true or false
            }
         //-->
      </script>
   </head>

   <body>
      <form method = "POST" action = "t.cgi" onsubmit =
"return validate()">
         .......
         <input type = "submit" value = "Submit" />
      </form>
   </body>
</html>
```

onmouseover y onmouseout

Estos dos tipos de eventos te ayudarán a crear buenos efectos con imágenes o incluso con texto. El evento **onmouseover** se dispara cuando coloca el mouse sobre cualquier elemento y el **onmouseout se** dispara cuando mueve el mouse fuera de ese elemento. Prueba el siguiente ejemplo.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function over() {
               document.write ("Mouse Over");
            }
            function out() {
               document.write ("Mouse Out");
            }
         //-->
      </script>
   </head>

   <body>
      <p>Bring your mouse inside the division to see the
result:</p>
      <div onmouseover = "over()" onmouseout = "out()">
         <h2> This is inside the division </h2>
      </div>
   </body>
</html>
```

Salida

Eventos estándar de HTML 5

Los eventos HTML 5 estándar se enumeran aquí para su referencia. Aquí el script indica una función de Javascript que se ejecutará contra ese evento.

Atributo	Valor	Descripción
Offline	guión	Se activa cuando el documento se desconecta
Onabort	guión	Activadores en un evento de aborto
onafterprint	guión	Se activa después de que se imprime el documento
onbeforeonload	guión	Se dispara antes de que se cargue el documento

onbeforeprint	guión	Se activa antes de que se imprima el documento
en la falta de definición	guión	Se dispara cuando la ventana pierde el foco
oncanplay	guión	Se activa cuando los medios pueden comenzar a reproducirse, pero es posible que deba detenerse para almacenar en búfer
oncanplaythrough	guión	Se dispara cuando los medios se pueden reproducir hasta el final, sin detenerse para el almacenamiento en búfer
onchange	guión	Se dispara cuando un elemento cambia
al hacer clic	guión	Se activa con un clic del mouse
oncontextmenu	guión	Se dispara cuando se activa un menú contextual
ondblclick	guión	Activa el doble clic del mouse
ondrag	guión	Se dispara cuando se arrastra un elemento
ondragend	guión	Se dispara al final de una operación de arrastre
ondragenter	guión	Se dispara cuando un elemento ha sido arrastrado a un objetivo de caída válido
ondragleave	guión	Se dispara cuando un elemento se arrastra sobre un objetivo de caída válido
resaca	guión	Se activa al comienzo de una operación de arrastre
ondragstart	guión	Se activa al comienzo de una operación de arrastre
ondrop	guión	Se activa cuando se suelta el elemento arrastrado

ondurationchange	guión	Se activa cuando se cambia la longitud de los medios
unmptied	guión	Se dispara cuando un elemento de recursos de medios de repente se vacía.
perseguido	guión	Se activa cuando los medios llegan al final
onerror	guión	Se dispara cuando ocurre un error
enfocado	guión	Se dispara cuando la ventana se enfoca
onformchange	guión	Se dispara cuando un formulario cambia
onforminput	guión	Se activa cuando un formulario recibe la entrada del usuario
onhaschange	guión	Se dispara cuando el documento ha cambiado
entrada	guión	Se dispara cuando un elemento recibe la entrada del usuario
no válido	guión	Se dispara cuando un elemento no es válido
onkeydown	guión	Se dispara cuando se presiona una tecla
onkeypress	guión	Se activa cuando se presiona y suelta una tecla
onkeyup	guión	Se dispara cuando se suelta una tecla
onload	guión	Se dispara cuando se carga el documento
datos cargados	guión	Se dispara cuando se cargan datos multimedia
metadatos cargados	guión	Se activa cuando se carga la duración y otros datos multimedia de un elemento multimedia

onloadstart	guión	Se activa cuando el navegador comienza a cargar los datos multimedia
mensaje	guión	Se dispara cuando se dispara el mensaje
onmousedown	guión	Se dispara cuando se presiona un botón del mouse
onmousemove	guión	Se dispara cuando se mueve el puntero del mouse
onmouseout	guión	Se dispara cuando el puntero del mouse se mueve fuera de un elemento
el ratón por encima	guión	Se dispara cuando el puntero del mouse se mueve sobre un elemento
onmouseup	guión	Se dispara cuando se suelta un botón del mouse
onmousewheel	guión	Se dispara cuando se gira la rueda del mouse
en línea	guión	Se activa cuando el documento se desconecta
onoine	guión	Se activa cuando el documento entra en línea
en línea	guión	Se activa cuando el documento entra en línea
onpagehide	guión	Se dispara cuando la ventana está oculta
onpageshow	guión	Se dispara cuando la ventana se hace visible
en pausa	guión	Se dispara cuando los datos multimedia están en pausa
onplay	guión	Se dispara cuando los datos multimedia van a comenzar a reproducirse
reproducción	guión	Se activa cuando los datos multimedia comienzan a reproducirse

onpopstate	guión	Se activa cuando cambia el historial de la ventana
en progreso	guión	Se activa cuando el navegador recupera los datos multimedia
onratechange	guión	Se activa cuando la velocidad de reproducción de los datos multimedia ha cambiado
onreadystatechange	guión	Se dispara cuando cambia el estado de preparación
onredo	guión	Se dispara cuando el documento realiza una rehacer
onresize	guión	Se dispara cuando la ventana cambia de tamaño
enrollarse	guión	Se dispara cuando se desplaza la barra de desplazamiento de un elemento
buscado	guión	Se activa cuando el atributo de búsqueda de un elemento multimedia ya no es verdadero y la búsqueda ha finalizado
en búsqueda	guión	Se activa cuando el atributo de búsqueda de un elemento multimedia es verdadero y la búsqueda ha comenzado
onselect	guión	Se dispara cuando se selecciona un elemento
instalado	guión	Se dispara cuando hay un error al recuperar datos multimedia
almacenamiento	guión	Se dispara cuando se carga un documento
presentar	guión	Se activa cuando se envía un formulario
gasto	guión	Se activa cuando el navegador ha estado recuperando datos multimedia, pero se detuvo antes de recuperar todo el archivo multimedia
ontimeupdate	guión	Se activa cuando los medios cambian su posición de reproducción

onundo	guión	Se desencadena cuando un documento realiza una acción de deshacer
onunload	guión	Se activa cuando el usuario abandona el documento.
onvolumechange	guión	Se activa cuando los medios cambian el volumen, también cuando el volumen está configurado en "silencio"
en espera	guión	Se activa cuando los medios han dejado de reproducirse, pero se espera que se reanuden

JavaScript y cookies

¿Qué son las cookies?

Los navegadores y servidores web utilizan el protocolo HTTP para comunicarse y HTTP es un protocolo sin estado. Pero para un sitio web comercial, se requiere mantener la información de la sesión entre diferentes páginas. Por ejemplo, el registro de un usuario finaliza después de completar muchas páginas. Pero cómo mantener la información de la sesión de los usuarios en todas las páginas web.

En muchas situaciones, el uso de cookies es el método más eficiente para recordar y rastrear preferencias, compras, comisiones y otra información requerida para una mejor experiencia del visitante o estadísticas del sitio.

Cómo funciona ?

Su servidor envía algunos datos al navegador del visitante en forma de cookie. El navegador puede aceptar la cookie. Si lo hace, se almacena como un registro de texto sin formato en el disco duro del visitante. Ahora, cuando el visitante llega a otra página de su sitio, el navegador envía la misma cookie al

servidor para su recuperación. Una vez recuperado, su servidor sabe / recuerda lo que se almacenó anteriormente.

Las cookies son un registro de datos de texto sin formato de 5 campos de longitud variable:

- **Caduca** : la fecha de caducidad de la cookie. Si está en blanco, la cookie caducará cuando el visitante cierre el navegador.
- **Dominio** : el nombre de dominio de su sitio.
- **Ruta** : la ruta al directorio o página web que configura la cookie. Esto puede estar en blanco si desea recuperar la cookie de cualquier directorio o página.
- **Seguro** : si este campo contiene la palabra "seguro", la cookie solo se puede recuperar con un servidor seguro. Si este campo está en blanco, no existe tal restricción.
- **Nombre = Valor** : las cookies se configuran y recuperan en forma de pares clave-valor

Las cookies fueron diseñadas originalmente para la programación CGI. Los datos contenidos en una cookie se transmiten automáticamente entre el navegador web y el servidor web, por lo que los scripts CGI en el servidor pueden leer y escribir valores de cookies que están almacenados en el cliente.

JavaScript también puede manipular cookies utilizando la propiedad de **cookies** del objeto **Documento** . JavaScript puede leer, crear, modificar y eliminar las cookies que se aplican a la página web actual.

Almacenar Cookies

La forma más sencilla de crear una cookie es asignar un valor de cadena al objeto document.cookie, que se ve así.

```
document.cookie = "key1 = value1;key2 = value2;expires =
date";
```

Aquí el atributo **caduca** es opcional. Si proporciona este atributo con una fecha u hora válida, la cookie caducará en una fecha u hora determinada y, a partir de entonces, el valor de las cookies no será accesible.

Nota : Los valores de cookies no pueden incluir punto y coma, comas o espacios en blanco. Por esta razón, es posible que desee utilizar la función JavaScript **escape ()** para codificar el valor antes de almacenarlo en la cookie. Si hace esto, también tendrá que usar la función **unescape ()** correspondiente cuando lea el valor de la cookie.

Ejemplo

Intenta lo siguiente. Establece un nombre de cliente en una cookie de entrada.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
```

```
            function WriteCookie() {
               if( document.myform.customer.value == "" ) {
                  alert("Enter some value!");
                  return;
               }
               cookievalue =
escape(document.myform.customer.value) + ";";
               document.cookie = "name=" + cookievalue;
               document.write ("Setting Cookies : " + "name="
+ cookievalue );
            }
         //-->
      </script>
   </head>

   <body>
      <form name = "myform" action = "">
         Enter name: <input type = "text" name = "customer"/>
         <input type = "button" value = "Set Cookie" onclick
= "WriteCookie();"/>
      </form>
   </body>
</html>
```

Salida

Ahora su máquina tiene una cookie llamada **nombre** . Puede configurar varias cookies utilizando múltiples pares clave = valor separados por comas.

Cookies de lectura

Leer una cookie es tan simple como escribir una, porque el valor del objeto document.cookie es la cookie. Por lo tanto, puede usar esta cadena cuando quiera acceder a la cookie. La cadena document.cookie mantendrá una lista de pares nombre = valor separados por punto y coma, donde **nombre** es el nombre de una cookie y valor es su valor de cadena.

Puede usar la función **split () de** cadenas para dividir una cadena en clave y valores de la siguiente manera:

Ejemplo

Pruebe el siguiente ejemplo para obtener todas las cookies.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function ReadCookie() {
```

```
                var allcookies = document.cookie;
                document.write ("All Cookies : " + allcookies
);

                // Get all the cookies pairs in an array
                cookiearray = allcookies.split(';');

                // Now take key value pair out of this array
                for(var i=0; i<cookiearray.length; i++) {
                   name = cookiearray[i].split('=')[0];
                   value = cookiearray[i].split('=')[1];
                   document.write ("Key is : " + name + " and
Value is : " + value);
                }
             }
          //-->
      </script>
   </head>

   <body>
      <form name = "myform" action = "">
         <p> click the following button and see the
result:</p>
         <input type = "button" value = "Get Cookie" onclick
= "ReadCookie()"/>
      </form>
   </body>
</html>
```

Nota : Aquí la **longitud** es un método de la clase **Array** que devuelve la longitud de una matriz. Discutiremos las matrices en un capítulo separado. Para ese momento, por favor intente digerirlo.

Nota : Puede haber algunas otras cookies ya configuradas en su máquina. El código anterior mostrará todas las cookies establecidas en su máquina.

Configuración de la fecha de vencimiento de las cookies

Puede extender la vida útil de una cookie más allá de la sesión actual del navegador estableciendo una fecha de vencimiento y guardando la fecha de vencimiento dentro de la cookie. Esto se puede hacer estableciendo el atributo **'caduca'** en una fecha y hora.

Ejemplo

Prueba el siguiente ejemplo. Ilustra cómo extender la fecha de vencimiento de una cookie en 1 mes.

```
<html>
   <head>
```

```
      <script type = "text/javascript">
         <!--
            function WriteCookie() {
               var now = new Date();
               now.setMonth( now.getMonth() + 1 );
               cookievalue =
escape(document.myform.customer.value) + ";"

               document.cookie = "name=" + cookievalue;
               document.cookie = "expires=" +
now.toUTCString() + ";"
               document.write ("Setting Cookies : " + "name="
+ cookievalue );
            }
         //-->
      </script>
   </head>

   <body>
      <form name = "myform" action = "">
         Enter name: <input type = "text" name = "customer"/>
         <input type = "button" value = "Set Cookie" onclick
= "WriteCookie()"/>
      </form>
   </body>
</html>
```

Salida

Eliminar una cookie

A veces querrás eliminar una cookie para que los intentos posteriores de leer la cookie no devuelvan nada. Para hacer esto, solo necesita establecer la fecha de vencimiento en un momento en el pasado.

Ejemplo

Prueba el siguiente ejemplo. Ilustra cómo eliminar una cookie estableciendo su fecha de vencimiento a un mes de la fecha actual.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function WriteCookie() {
               var now = new Date();
               now.setMonth( now.getMonth() - 1 );
               cookievalue =
escape(document.myform.customer.value) + ";"
```

```
                document.cookie = "name=" + cookievalue;
                document.cookie = "expires=" +
now.toUTCString() + ";"
                document.write("Setting Cookies : " + "name="
+ cookievalue );
             }
          //-->
      </script>
   </head>

   <body>
      <form name = "myform" action = "">
         Enter name: <input type = "text" name = "customer"/>
         <input type = "button" value = "Set Cookie" onclick
= "WriteCookie()"/>
      </form>
   </body>
</html>
```

Salida

JavaScript - Redirección de página

¿Qué es la redirección de página?

Es posible que haya encontrado una situación en la que hizo clic en una URL para llegar a una página X pero internamente fue dirigido a otra página Y. Esto ocurre debido a la **redirección de la página** . Este concepto es diferente de la actualización de la página de JavaScript .

Puede haber varias razones por las que le gustaría redirigir a un usuario desde la página original. Estamos enumerando algunas de las razones:

- No le gustó el nombre de su dominio y se está mudando a uno nuevo. En tal escenario, es posible que desee dirigir a todos sus visitantes al nuevo sitio. Aquí puede mantener su dominio anterior, pero poner una sola página con una redirección de página para que todos los visitantes de su dominio anterior puedan ingresar a su nuevo dominio.
- Ha acumulado varias páginas en función de las versiones del navegador o sus nombres o puede estar basado en diferentes países, luego, en lugar de utilizar la redirección de la página del lado del servidor, puede utilizar la redirección de la página del lado del cliente para que sus usuarios accedan a la página correspondiente.
- Es posible que los motores de búsqueda ya hayan indexado sus páginas. Pero mientras se muda a otro dominio, no le gustaría perder a sus visitantes que llegan a través de los motores de búsqueda. Entonces puede usar la redirección de página del lado del cliente. Pero tenga en cuenta que esto no debe hacerse para engañar al motor de búsqueda, podría hacer que su sitio sea bloqueado.

¿Cómo funciona la redirección de página?

- Las implementaciones de redirección de página son las siguientes.

- Ejemplo 1
- Es bastante simple hacer una redirección de página usando JavaScript en el lado del cliente. Para redirigir a los visitantes de su sitio a una nueva página, solo necesita agregar una línea en su sección principal de la siguiente manera.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function Redirect() {
               window.location =
"https://www.postparaprogramadores.com";
            }
         //-->
      </script>
   </head>

   <body>
      <p>Click the following button, you will be
redirected to home page.</p>

      <form>
         <input type = "button" value = "Redirect Me"
onclick = "Redirect();" />
      </form>

   </body>
</html>
```

- Salida
- Ejemplo 2
- Puede mostrar un mensaje apropiado a los visitantes de su sitio antes de redirigirlos a una nueva página. Esto necesitaría un poco de retraso para cargar una nueva página. El siguiente ejemplo muestra cómo implementar lo mismo. Aquí **setTimeout ()** es una función de JavaScript incorporada que se puede utilizar para ejecutar otra función después de un intervalo de tiempo determinado.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function Redirect() {
               window.location =
"https://www.postparaprogramadores.com";
            }
            document.write("You will be redirected to
main page in 10 sec.");
            setTimeout('Redirect()', 10000);
         //-->
```

```
      </script>
   </head>

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

- Salida

```
You will be redirected to postparaprogramadores.com main page in 10 seconds!
```

- Ejemplo 3
- El siguiente ejemplo muestra cómo redirigir a los visitantes de su sitio a una página diferente en función de sus navegadores.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            var browsername = navigator.appName;
            if( browsername == "Netscape" ) {
               window.location = "http://www.location.com/ns.htm";
            } else if ( browsername =="Microsoft Internet Explorer") {
               window.location = "http://www.location.com/ie.htm";
            } else {
               window.location = "http://www.location.com/other.htm";
            }
         //-->
      </script>
   </head>

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

JavaScript: cuadros de diálogo

- JavaScript admite tres tipos importantes de cuadros de diálogo. Estos cuadros de diálogo se pueden usar para generar y alertar, o para obtener confirmación sobre cualquier entrada o para tener un tipo de entrada de los usuarios. Aquí discutiremos cada cuadro de diálogo uno por uno.

Cuadro de diálogo de alerta

- Un cuadro de diálogo de alerta se usa principalmente para dar un mensaje de advertencia a los usuarios. Por ejemplo, si un campo de entrada requiere ingresar texto pero el usuario no proporciona ninguna entrada, entonces, como parte de la validación, puede usar un cuadro de alerta para dar un mensaje de advertencia.

- No obstante, todavía se puede usar un cuadro de alerta para mensajes más amigables. El cuadro de alerta le da a un solo botón "Aceptar" para seleccionar y continuar.
- Ejemplo

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function Warn() {
               alert ("This is a warning message!");
               document.write ("This is a warning
message!");
            }
         //-->
      </script>
   </head>

   <body>
      <p>Click the following button to see the result:
</p>
      <form>
         <input type = "button" value = "Click Me"
onclick = "Warn();" />
      </form>
   </body>
</html>
```

- Salida

Cuadro de diálogo de confirmación

- Un cuadro de diálogo de confirmación se utiliza principalmente para obtener el consentimiento del usuario en cualquier opción. Muestra un cuadro de diálogo con dos botones: **Aceptar** y **Cancelar** .
- Si el usuario hace clic en el botón Aceptar, el método de ventana **confirmar ()** devolverá verdadero. Si el usuario hace clic en el botón Cancelar, **confirme ()** devuelve falso. Puede usar un cuadro de diálogo de confirmación de la siguiente manera.
- Ejemplo

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function getConfirmation() {
               var retVal = confirm("Do you want to
continue ?");
               if( retVal == true ) {
                  document.write ("User wants to
continue!");
                  return true;
               } else {
```

```
                    document.write ("User does not want to
continue!");
                    return false;
                }
            }
         //-->
      </script>
   </head>

   <body>
      <p>Click the following button to see the result:
</p>
      <form>
         <input type = "button" value = "Click Me"
onclick = "getConfirmation();" />
      </form>
   </body>
</html>
```

- Salida

Cuadro de diálogo rápido

El cuadro de diálogo de solicitud es muy útil cuando desea abrir un cuadro de texto para obtener la entrada del usuario. Por lo tanto, le permite interactuar con el usuario. El usuario debe completar el campo y luego hacer clic en Aceptar.

Este cuadro de diálogo se muestra usando un método llamado **prompt ()** que toma dos parámetros: (i) una etiqueta que desea mostrar en el cuadro de texto y (ii) una cadena predeterminada para mostrar en el cuadro de texto.

Este cuadro de diálogo tiene dos botones: **Aceptar** y **Cancelar** . Si el usuario hace clic en el botón Aceptar, el **indicador del** método de **ventana ()** devolverá el valor ingresado desde el cuadro de texto. Si el usuario hace clic en el botón Cancelar, el **indicador del** método de **ventana ()** devuelve **nulo** .

- Ejemplo

El siguiente ejemplo muestra cómo usar un cuadro de diálogo de solicitud:

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function getValue() {
               var retVal = prompt("Enter your name : ",
"your name here");
               document.write("You have entered : " +
retVal);
            }
         //-->
      </script>
   </head>
```

```
   <body>
      <p>Click the following button to see the result:
   </p>
      <form>
         <input type = "button" value = "Click Me"
   onclick = "getValue();" />
      </form>
   </body>
</html>
```

- Salida

JavaScript: palabra clave nula

- **void** es una palabra clave importante en JavaScript que se puede usar como operador unario que aparece antes de su único operando, que puede ser de cualquier tipo. Este operador especifica una expresión para ser evaluada sin devolver un valor.

Sintaxis

La sintaxis de **void** puede ser cualquiera de los dos siguientes:

```
<head>
   <script type = "text/javascript">
      <!--
         void func()
         javascript:void func()
         or:
         void(func())
         javascript:void(func())
      //-->
   </script>
</head>
```

Ejemplo 1

El uso más común de este operador es en un *javascript:* URL del lado del cliente , donde le permite evaluar una expresión para sus efectos secundarios sin que el navegador muestre el valor de la expresión evaluada.

Aquí se evalúa la **alerta de** expresión **('¡Advertencia!')** Pero no se vuelve a cargar en el documento actual:

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
         //-->
      </script>
   </head>

   <body>
      <p>Click the following, This won't react at all...</p>
```

```
      <a href = "javascript:void(alert('Warning!!!'))">Click
me!</a>
   </body>
</html>
```

Salida

Ejemplo 2

Eche un vistazo al siguiente ejemplo. El siguiente enlace no hace nada porque la expresión "0" no tiene ningún efecto en JavaScript. Aquí se evalúa la expresión "0", pero no se vuelve a cargar en el documento actual.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
         //-->
      </script>
   </head>

   <body>
      <p>Click the following, This won't react at all...</p>
      <a href = "javascript:void(0)">Click me!</a>
   </body>
</html>
```

Salida

Ejemplo 3

Otro uso de **void** es generar deliberadamente el valor **indefinido de la** siguiente manera.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function getValue() {
               var a,b,c;

               a = void ( b = 5, c = 7 );
               document.write('a = ' + a + ' b = ' + b +' c =
' + c );
            }
         //-->
      </script>
   </head>
```

```
   <body>
      <p>Click the following to see the result:</p>
      <form>
         <input type = "button" value = "Click Me" onclick =
"getValue();" />
      </form>
   </body>
</html>
```

Salida

JavaScript - Impresión de página

Muchas veces le gustaría colocar un botón en su página web para imprimir el contenido de esa página web a través de una impresora real. JavaScript le ayuda a implementar esta funcionalidad utilizando la función de **impresión** del objeto de **ventana** .

La función de impresión de JavaScript **window.print ()** imprime la página web actual cuando se ejecuta. Puede llamar a esta función directamente usando el evento **onclick** como se muestra en el siguiente ejemplo.

Ejemplo

Prueba el siguiente ejemplo.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
         //-->
      </script>
   </head>

   <body>
      <form>
         <input type = "button" value = "Print" onclick =
"window.print()" />
      </form>
   </body>
<html>
```

Salida

Aunque sirve para obtener una copia impresa, no es una forma recomendada. Una página para imprimir es realmente solo una página con texto, sin imágenes, gráficos o publicidad.

Puede hacer que una página sea fácil de imprimir de las siguientes maneras:

- Haga una copia de la página y omita el texto y los gráficos no deseados, luego enlace a la página para imprimir del original. Verifique el ejemplo .
- Si no desea conservar una copia adicional de una página, puede marcar su texto imprimible con los comentarios adecuados, como <! - IMPRIME COMIENZA AQUÍ -> ..... <! - IMPRIMIR TERMINA AQUÍ -> y luego puede usar PERL o cualquier otro script en segundo plano para purgar el texto imprimible y mostrarlo para la impresión final. Nosotros en Postparaprogramadores utilizamos este método para proporcionar servicios de impresión a los visitantes de nuestro sitio.

¿Cómo imprimir una página?

Si no encuentra las instalaciones anteriores en una página web, puede usar la barra de herramientas estándar del navegador para imprimir la página web. Sigue el enlace de la siguiente manera.

```
File →  Print → Click OK  button.
```

JavaScript: descripción general de los objetos

JavaScript es un lenguaje de programación orientada a objetos (OOP). Un lenguaje de programación puede llamarse orientado a objetos si proporciona cuatro capacidades básicas para los desarrolladores:

- **Encapsulación** : la capacidad de almacenar información relacionada, ya sean datos o métodos, en un objeto.
- **Agregación** : la capacidad de almacenar un objeto dentro de otro objeto.
- **Herencia** : la capacidad de una clase de confiar en otra clase (o número de clases) para algunas de sus propiedades y métodos.
- **Polimorfismo** : la capacidad de escribir una función o método que funciona en una variedad de formas diferentes.

Los objetos están compuestos de atributos. Si un atributo contiene una función, se considera un método del objeto; de lo contrario, el atributo se considera una propiedad.

Propiedades del objeto

Las propiedades de los objetos pueden ser cualquiera de los tres tipos de datos primitivos, o cualquiera de los tipos de datos abstractos, como otro objeto. Las propiedades de los objetos suelen ser variables que se utilizan internamente en los métodos del objeto, pero también pueden ser variables visibles globalmente que se utilizan en toda la página.

La sintaxis para agregar una propiedad a un objeto es:

```
objectName.objectProperty = propertyValue;
```

Por ejemplo : el siguiente código obtiene el título del documento utilizando la propiedad **"título"** del objeto del **documento** .

```
var str = document.title;
```

Métodos de objeto

Los métodos son las funciones que permiten que el objeto haga algo o que se le haga algo. Hay una pequeña diferencia entre una función y un método: en una función hay una unidad de enunciados independiente y un método se adjunta a un objeto y puede ser referenciado por **esta** palabra clave.

Los métodos son útiles para todo, desde mostrar el contenido del objeto en la pantalla hasta realizar operaciones matemáticas complejas en un grupo de propiedades y parámetros locales.

Por ejemplo : el siguiente es un ejemplo simple para mostrar cómo usar el método **write ()** del objeto de documento para escribir cualquier contenido en el documento.

```
document.write("This is test");
```

Objetos definidos por el usuario

Todos los objetos definidos por el usuario y los objetos integrados son descendientes de un objeto llamado **Objeto** .

El nuevo operador

El **nuevo** operador se utiliza para crear una instancia de un objeto. Para crear un objeto, el **nuevo** operador es seguido por el método del constructor.

En el siguiente ejemplo, los métodos de construcción son Object (), Array () y Date (). Estos constructores son funciones de JavaScript incorporadas.

```
var employee = new Object();
var books = new Array("C++", "Perl", "Java");
var day = new Date("August 15, 1947");
```

El constructor de objetos ()

Un constructor es una función que crea e inicializa un objeto. JavaScript proporciona una función de constructor especial llamada **Object ()** para construir el objeto. El valor de retorno del constructor **Object ()** se asigna a una variable.

La variable contiene una referencia al nuevo objeto. Las propiedades asignadas al objeto no son variables y no están definidas con la palabra clave **var** .

Ejemplo 1

Pruebe el siguiente ejemplo; demuestra cómo crear un objeto.

```
<html>
   <head>
```

```
      <title>User-defined objects</title>
      <script type = "text/javascript">
         var book = new Object();   // Create the object
         book.subject = "Perl";     // Assign properties to
the object
         book.author  = "Mohtashim";
      </script>
   </head>

   <body>
      <script type = "text/javascript">
         document.write("Book name is : " + book.subject +
"<br>");
         document.write("Book author is : " + book.author +
"<br>");
      </script>
   </body>
</html>
```

Salida

```
Book name is : Perl
Book author is : Mohtashim
```

Ejemplo 2

Este ejemplo muestra cómo crear un objeto con una función definida por el usuario. Aquí **esta** palabra clave se usa para referirse al objeto que se ha pasado a una función.

```
<html>
   <head>
   <title>User-defined objects</title>
      <script type = "text/javascript">
         function book(title, author) {
            this.title = title;
            this.author  = author;
         }
      </script>
   </head>

   <body>
      <script type = "text/javascript">
         var myBook = new book("Perl", "Mohtashim");
         document.write("Book title is : " + myBook.title +
"<br>");
         document.write("Book author is : " + myBook.author +
"<br>");
      </script>
   </body>
</html>
```

Salida

```
Book title is : Perl
Book author is : Mohtashim
```

Definición de métodos para un objeto

Los ejemplos anteriores demuestran cómo el constructor crea el objeto y asigna propiedades. Pero necesitamos completar la definición de un objeto asignándole métodos.

Ejemplo

Pruebe el siguiente ejemplo; muestra cómo agregar una función junto con un objeto.

```
<html>

   <head>
   <title>User-defined objects</title>
      <script type = "text/javascript">
         // Define a function which will work as a method
         function addPrice(amount) {
            this.price = amount;
         }

         function book(title, author) {
            this.title = title;
            this.author  = author;
            this.addPrice = addPrice;  // Assign that method
as property.
         }
      </script>
   </head>

   <body>
      <script type = "text/javascript">
         var myBook = new book("Perl", "Mohtashim");
         myBook.addPrice(100);

         document.write("Book title is : " + myBook.title +
"<br>");
         document.write("Book author is : " + myBook.author +
"<br>");
         document.write("Book price is : " + myBook.price +
"<br>");
      </script>
   </body>
</html>
```

Salida

```
Book title is : Perl
Book author is : Mohtashim
Book price is : 100
```

La palabra clave 'con'

La palabra clave **'con'** se usa como una forma abreviada para hacer referencia a las propiedades o métodos de un objeto.

El objeto especificado como argumento con **with se** convierte en el objeto predeterminado durante la duración del bloque que sigue. Las propiedades y métodos para el objeto se pueden usar sin nombrar el objeto.

Sintaxis

La sintaxis para con objeto es la siguiente:

```
with (object) {
   properties used without the object name and dot
}
```

Ejemplo

Prueba el siguiente ejemplo.

```
<html>
   <head>
   <title>User-defined objects</title>
      <script type = "text/javascript">
         // Define a function which will work as a method
         function addPrice(amount) {
            with(this) {
               price = amount;
            }
         }
         function book(title, author) {
            this.title = title;
            this.author = author;
            this.price = 0;
            this.addPrice = addPrice;  // Assign that method
as property.
         }
      </script>
   </head>

   <body>
      <script type = "text/javascript">
         var myBook = new book("Perl", "Mohtashim");
         myBook.addPrice(100);
```

```
         document.write("Book title is : " + myBook.title +
"<br>");
         document.write("Book author is : " + myBook.author +
"<br>");
         document.write("Book price is : " + myBook.price +
"<br>");
      </script>
   </body>
</html>
```

Salida

```
Book title is : Perl
Book author is : Mohtashim
Book price is : 100
```

Objetos nativos de JavaScript

JavaScript tiene varios objetos integrados o nativos. Se puede acceder a estos objetos desde cualquier lugar de su programa y funcionarán de la misma manera en cualquier navegador que se ejecute en cualquier sistema operativo.

Aquí está la lista de todos los objetos nativos importantes de JavaScript:

- Objeto de número de JavaScript
- Objeto booleano de JavaScript
- Objeto de cadena de JavaScript
- Objeto de matriz de JavaScript
- Objeto de fecha de JavaScript
- Objeto matemático JavaScript
- Objeto RegExp de JavaScript

JavaScript - El objeto Number

El objeto Number representa la fecha numérica, enteros o números de punto flotante. En general, usted no necesita preocuparse por los objetos Número ya que el navegador convierte automáticamente literales numéricas para las instancias de la clase número.

Sintaxis

La sintaxis para crear un objeto de número es el siguiente -

```
val var = new Number (número);
```

En el lugar del número, si se proporciona ningún argumento que no sea el número, entonces el argumento no se puede convertir en un número, devuelve NaN (Not-a-Number).

propiedades de los números

Aquí está una lista de cada propiedad y su descripción.

No Señor.	Descripción de propiedad
1	VALOR MÁXIMO El número de un posible valor más grande de JavaScript puede tener 1.7976931348623157E + 308
2	MIN_VALUE El número de un posible valor más pequeño en JavaScript puede tener 5E-324
3	Yaya Igual a un valor que no es un número.
4	NEGATIVE_INFINITY Un valor que es menor que MIN_VALUE.
5	POSITIVE_INFINITY Un valor que es mayor que MAX_VALUE
6	prototipo Una propiedad estática del objeto Number. Utilice la propiedad prototipo de nuevas propiedades y métodos de asignar al objeto Number en el documento actual
7	constructor Devuelve la función que crea la instancia de este objeto. Por defecto, este es el objeto Number.

En las siguientes secciones, tomaremos algunos ejemplos para demostrar las propiedades de los números.

Métodos numéricos

El objeto Number contiene sólo los métodos predeterminados que son parte de la definición de cada objeto.

No Señor.	Método y Descripción
1	toExponential ()

	Fuerzas un número para mostrar en notación exponencial, incluso si el número está en el rango en el que JavaScript normalmente usa la notación estándar.
2	toFixed () Formato a un número con un número específico de dígitos a la derecha del punto decimal.
3	toLocaleString () Devuelve una versión valor de cadena del número actual en un formato que puede variar según la configuración local de un navegador.
4	toPrecision () Define el número de dígitos en total (incluyendo los dígitos a la izquierda y la derecha del punto decimal) para mostrar de un número.
5	Encadenar() Devuelve la representación de cadena del valor del número.
6	valor de() Devuelve el valor del número.

En las siguientes secciones, vamos a tener un par de ejemplos para explicar los métodos de Número.

JavaScript - El objeto booleano

El objeto Boolean representa dos valores, ya sea "verdadero" o "falso". Si se omite el parámetro valor o es 0, -0, null, falso, NaN, sin definir, o la cadena vacía (""), el objeto tiene un valor inicial de falsa.

Sintaxis

Utilice la siguiente sintaxis para crear un objeto booleano.

```
val var = new Boolean (valor);
```

Propiedades de Boole

Aquí está una lista de las propiedades del objeto Boolean -

No Señor.	Descripción de propiedad
1	constructor Devuelve una referencia a la función booleana que creó el objeto.

2	prototipo La propiedad prototipo permite agregar propiedades y métodos a un objeto.

En las siguientes secciones, vamos a tener unos pocos ejemplos para ilustrar las propiedades de objeto Boolean.

Métodos de Boole

Aquí está una lista de los métodos de objeto booleano y su descripción.

Sr.No.	Método y Descripción
1	a la fuente() Devuelve una cadena que contiene la fuente del objeto Boolean; puede utilizar esta secuencia para crear un objeto equivalente.
2	Encadenar() Devuelve una serie de "true" o "false" dependiendo del valor del objeto.
3	valor de() Devuelve el valor simple del objeto Boolean.

En las siguientes secciones, vamos a tener unos pocos ejemplos para demostrar el uso de los métodos booleanos.

JavaScript - El objeto Cuerdas

El objeto String le permite trabajar con una serie de personajes; se envuelve cadena de tipo de datos primitivo de Javascript con una serie de métodos de ayuda.

Como JavaScript convierte automáticamente entre las primitivas de cuerda y objetos de cadena, puede llamar a cualquiera de los métodos de ayuda del objeto String en una cadena primitiva.

Sintaxis

Utilice la siguiente sintaxis para crear un objeto String -

```
val var = new String (cadena);
```

El parámetro de cadena es una serie de caracteres que se ha codificado correctamente.

Propiedades de cuerda

Aquí está una lista de las propiedades del objeto String y su descripción.

No Señor.	Descripción de propiedad
1	constructor Devuelve una referencia a la función de cadena que crea el objeto.
2	longitud Devuelve la longitud de la cadena.
3	prototipo La propiedad prototipo permite agregar propiedades y métodos a un objeto.

En las siguientes secciones, vamos a tener unos pocos ejemplos para demostrar el uso de propiedades de Cuerda.

Métodos de cuerda

Aquí está una lista de los métodos disponibles en el objeto String junto con su descripción.

No Señor.	Método y Descripción
1	charAt () Devuelve el carácter en el índice especificado.
2	charCodeAt () Devuelve un número que indica el valor Unicode del carácter en el índice especificado.
3	concat () Combina el texto de dos cadenas y devuelve una nueva cadena.
4	índice de() Devuelve el índice dentro del objeto String llamar de la primera aparición del valor especificado, o -1 si no se encuentra.
5	lastIndexOf () Devuelve el índice dentro del objeto String llamando de la última aparición del valor especificado, o -1 si no se encuentra.
6	localeCompare ()

	Devuelve un número que indica si una cadena de referencia viene antes o después, o es la misma que la cadena dada en el orden de clasificación.
7	partido() Se utiliza para que coincida con una expresión regular con una cadena.
8	reemplazar() Se utiliza para encontrar una coincidencia entre una expresión regular y una cadena, y sustituir la subcadena coincidente con una nueva subcadena.
9	buscar() Ejecuta la búsqueda de una coincidencia entre una expresión regular y una cadena especificada.
10	rebanada() Extrae una sección de una cadena y devuelve una nueva cadena.
11	división() Divide un objeto String en una matriz de cadenas mediante la separación de la cadena en subcadenas.
12	substr () Devuelve los caracteres en una cadena que comienza en la posición especificada por el número de caracteres especificado.
13	subcadena () Devuelve los caracteres de una cadena entre dos índices en la cadena.
14	toLocaleLowerCase () Los caracteres dentro de una cadena se convierten a minúsculas en el respeto de la localización actual.
15	toLocaleUpperCase () Los caracteres dentro de una cadena se convierten en mayúsculas en el respeto de la localización actual.
dieciséis	toLowerCase () Devuelve el valor de cadena llamando convertidos a minúsculas.
17	Encadenar() Devuelve una cadena que representa el objeto especificado.
18	toUpperCase ()

	Devuelve el valor de cadena llamando convierte a mayúsculas.
19	valor de() Devuelve el valor simple del objeto especificado.

Envolturas de cadena HTML

He aquí una lista de los métodos que devuelven una copia de la cadena envuelta dentro de una etiqueta HTML correspondiente.

No Señor.	Método y Descripción
1	ancla() Crea un ancla HTML que se utiliza como un objetivo de hipertexto.
2	grande() Crea una cadena que se mostrará en una fuente grande como si se tratara de una etiqueta <big>.
3	parpadeo() Crea una cadena para abrir y cerrar como si se tratara de una etiqueta <blink>.
4	negrita() Crea una cadena que se mostrará en negrita como si se tratara de una etiqueta <b>.
5	fijo() Provoca que una cadena se mostrará en la fuente de paso fijo como si se tratara de una etiqueta <tt>
6	color de fuente() Provoca una cadena que se mostrará en el color especificado como si fuera en un <font color = "color"> etiqueta.
7	tamaño de fuente() Provoca una cadena que se mostrará en el tamaño de la fuente especificada como si fuera en un <font size = "tamaño"> etiqueta.
8	cursiva() Causa que una cadena sea cursiva, como si se tratara de una etiqueta <i>.

9	enlace() Crea un enlace de hipertexto HTML que solicita otra URL.
10	pequeño() Provoca una cadena que se mostrará en una pequeña fuente, como si se tratara de una etiqueta <small>.
11	Huelga() Provoca una cadena que se mostrará como texto tachado, como si se tratara de una etiqueta <strike>.
12	sub() Provoca que una cadena se muestre como subíndice, como si se tratara de una etiqueta <sub>
13	cenar() Provoca que una cadena se muestre como un exponente, como si se tratara de una etiqueta <sup>

En las siguientes secciones, vamos a tener unos pocos ejemplos para demostrar el uso de métodos de String.

JavaScript - El matrices de objetos

El objeto Array le permite almacenar varios valores en una sola variable. Se almacena una colección secuencial de tamaño fijo de elementos del mismo tipo. Una matriz se utiliza para almacenar un conjunto de datos, pero a menudo es más útil pensar en una matriz como una colección de variables del mismo tipo.

Sintaxis

Utilice la siguiente sintaxis para crear un objeto Array -

```
frutas var = new Array ( "manzana", "naranja", "mango");
```

El parámetro de matriz es una lista de cadenas o enteros. Cuando se especifica un parámetro numérico con el constructor de Array, se especifica la longitud inicial de la matriz. La longitud máxima permitida para una matriz es 4294967295.

Se puede crear una matriz mediante una simple asignación de los valores de la siguiente manera -

```
var frutas = [ "manzana", "naranja", "mango" ];
```

Que va a utilizar los números ordinales para el acceso y para valores establecidos dentro de una matriz de la siguiente manera.

```
frutas [0] es el primer elemento
```

```
frutas [1] es el segundo elemento
frutas [2] es el tercer elemento
```

propiedades de la matriz

Aquí está una lista de las propiedades del objeto Array junto con su descripción.

No Señor.	Descripción de propiedad
1	constructor Devuelve una referencia a la función de matriz que creó el objeto.
2	**índice** La propiedad representa el índice de base cero de la coincidencia en la cadena
3	**entrada** Esta propiedad sólo está presente en las matrices creadas por coincidencias de expresiones regulares.
4	longitud Refleja el número de elementos de una matriz.
5	prototipo La propiedad prototipo permite agregar propiedades y métodos a un objeto.

En las siguientes secciones, vamos a tener algunos ejemplos para ilustrar el uso de propiedades de la matriz.

Métodos de matriz

He aquí una lista de los métodos del objeto de matriz junto con su descripción.

No Señor.	Método y Descripción
1	concat () Devuelve una nueva matriz compuesta de esta matriz se unió a otros array (s) y / o el valor (s).
2	cada()

	Devuelve TRUE si cada elemento en esta matriz satisface la función de control definidos.
3	filtrar() Crea una nueva matriz con todos los elementos de esta matriz para el que la función de filtrado proporcionada devuelve verdadero.
4	para cada() Llama a una función para cada elemento de la matriz.
5	índice de() Devuelve el primer (menos) de índice de un elemento dentro de la matriz igual al valor especificado, o -1 si no se encuentra ninguno.
6	unirse() Se une a todos los elementos de una matriz en una cadena.
7	lastIndexOf () Devuelve el último índice (mayor) de un elemento dentro de la matriz igual al valor especificado, o -1 si no se encuentra ninguno.
8	mapa() Crea una nueva matriz con los resultados de llamar a una función proporcionada en cada elemento de este vector.
9	popular() Elimina el último elemento de una matriz y devuelve dicho elemento.
10	empujar() Añade uno o más elementos al final de una matriz y devuelve la nueva longitud de la matriz.
11	reducir() Aplicar una función simultáneamente contra dos valores de la matriz (de izquierda a derecha) como para reducirla a un único valor.
12	reduceRight () Aplicar una función simultáneamente contra dos valores de la matriz (de derecha a izquierda) como para reducirla a un único valor.
13	marcha atrás() Invierte el orden de los elementos de una matriz - la primera se convierte en el pasado, y el último se convierte en el primero.

14	cambio() Elimina el primer elemento de una matriz y devuelve dicho elemento.
15	rebanada() Extrae una sección de una matriz y devuelve una nueva matriz.
dieciséis	algunos() Devuelve true si al menos uno de los elementos de esta matriz satisface la función de control definidos.
17	a la fuente() Representa el código fuente de un objeto
18	ordenar() Ordena los elementos de una matriz
19	empalme() elementos agrega y / o elimina de una matriz.
20	Encadenar() Devuelve una cadena que representa la matriz y sus elementos.
21	unshift () Añade uno o más elementos en la parte delantera de una matriz y devuelve la nueva longitud de la matriz.

En las siguientes secciones, vamos a tener unos pocos ejemplos para demostrar el uso de los métodos de matriz.

JavaScript - La Fecha de objetos

El objeto Date es un tipo de datos integrado en el lenguaje JavaScript. Fecha objetos son creados con la nueva fecha () como se muestra a continuación.

Una vez que se crea un objeto Date, una serie de métodos le permiten operar en él. La mayoría de los métodos simplemente le permiten obtener y establecer el año, mes, día, hora, minuto, segundo y milisegundo campos del objeto, utilizando el tiempo, ya sea local o la hora UTC (Universal, o GMT).

El estándar ECMAScript requiere el objeto Date a ser capaz de representar cualquier fecha y hora, a una precisión de milisegundos, a menos de 100 millones de días antes o después de 1/1/1970. Se trata de una gama de más o menos 273,785 años, por lo que puede representar JavaScript fecha y hora hasta el año 275755.

Sintaxis

Puede utilizar cualquiera de las siguientes sintaxis para crear un objeto Date utilizando Fecha (constructor).

```
new Date ()
nuevos Fecha (milisegundos)
nuevos Fecha (DateString)
nueva fecha (año, mes, fecha [, horas, minutos, segundos,
milisegundos])
```

Nota- parámetros entre paréntesis son siempre opcionales.

He aquí una descripción de los parámetros -

- **ningún argumento**- Sin argumentos, el constructor Date () crea un conjunto objeto Date con la fecha y hora actual.
- **milisegundos**- Cuando se pasa un argumento numérico, se toma como la representación numérica interna de la fecha en milisegundos, como devuelto por el método getTime (). Por ejemplo, pasando el argumento 5000 crea una fecha que representa cinco segundos después de la medianoche en 1/1/70.
- **DateString**- Cuando se pasa un argumento de serie, se trata de una representación de cadena de una fecha, en el formato aceptado por el método () Date.parse.
- **7 agruments**- Para utilizar la última forma del constructor se muestra arriba. He aquí una descripción de cada argumento -
    - **año**- Valor entero que representa el año. Por razones de compatibilidad (con el fin de evitar el problema Y2K), siempre debe especificar el año en su totalidad; utilizar 1998, en lugar de 98.
    - **mes**- Valor entero que representa el mes, empezando por 0 para enero al 11 de diciembre.
    - **fecha**- Valor entero que representa el día del mes.
    - **hora**- valor entero que representa la hora del día (escala de 24 horas).
    - **minuto**- valor entero que representa el segmento de minutos de un tiempo de lectura.
    - **segundo**- valor entero que representa el segundo segmento de un tiempo de lectura.
    - **milisegundo**- valor entero que representa el segmento de milisegundo de un tiempo de lectura.

fecha Propiedades

Aquí está una lista de las propiedades del objeto Date, junto con su descripción.

No Señor.	Descripción de propiedad
1	constructor Especifica la función que crea un prototipo de un objeto.

2	prototipo La propiedad prototipo permite agregar propiedades y métodos a un objeto

En las siguientes secciones, vamos a tener un par de ejemplos para demostrar el uso de diferentes propiedades de fecha.

fecha Métodos

Aquí está una lista de los métodos utilizados con la fecha y su descripción.

No Señor.	Método y Descripción
1	Fecha() Devuelve la fecha y la hora actual
2	obtener la fecha() Devuelve el día del mes de la fecha especificada según la hora local.
3	getDay () Devuelve el día de la semana para la fecha especificada según la hora local.
4	getFullYear () Devuelve el año de la fecha especificada según la hora local.
5	getHours () Devuelve la hora en la fecha especificada según la hora local.
6	getMilliseconds () Devuelve los milisegundos de la fecha especificada según la hora local.
7	getMinutes () Devuelve los minutos de la fecha especificada según la hora local.
8	getMonth () Devuelve el mes de la fecha especificada según la hora local.
9	getSeconds () Devuelve los segundos de la fecha especificada según la hora local.
10	consigue tiempo()

	Devuelve el valor numérico de la fecha especificada como el número de milisegundos desde el 1 de enero de 1970 00:00:00 GMT.
11	getTimezoneOffset () Devuelve el desplazamiento en minutos para la localización actual de zona horaria.
12	getUTCDate () Devuelve el día (fecha) del mes de la fecha especificada según la hora universal.
13	getUTCDay () Devuelve el día de la semana en la fecha especificada según la hora universal.
14	getUTCFullYear () Devuelve el año en la fecha especificada según la hora universal.
15	getUTCHours () Devuelve las horas en la fecha especificada según la hora universal.
dieciséis	getUTCMilliseconds () Devuelve los milisegundos de la fecha especificada según la hora universal.
17	getUTCMinutes () Devuelve los minutos de la fecha especificada según la hora universal.
18	getUTCMonth () Devuelve el mes de la fecha especificada según la hora universal.
19	getUTCSeconds () Devuelve los segundos de la fecha especificada según la hora universal.
20	getYear () **Obsoleto**- Devuelve el año en la fecha especificada según la hora local. Uso getFullYear lugar.
21	define la fecha() Establece el día del mes de la fecha especificada según la hora local.
22	setFullYear () Establece el año completo para una fecha especificada según la hora local.

23	setHours () Establece las horas de la fecha especificada según la hora local.
24	setMilliseconds () Establece los milisegundos de la fecha especificada según la hora local.
25	setMinutes () Establece los minutos de la fecha especificada según la hora local.
26	setMonth () Establece el mes de la fecha especificada según la hora local.
27	setSeconds () Establece los segundos de la fecha especificada según la hora local.
28	fijar tiempo() Establece el objeto Fecha al tiempo representado por un número de milisegundos desde el 1 de enero de 1970, 00:00:00 UTC.
29	setUTCDate () Establece el día del mes de la fecha especificada según la hora universal.
30	setUTCFullYear () Establece el año completo para la fecha especificada según la hora universal.
31	setUTCHours () Ajusta la hora en la fecha especificada según la hora universal.
32	setUTCMilliseconds () Establece los milisegundos de la fecha especificada según la hora universal.
33	setUTCMinutes () Establece los minutos de la fecha especificada según la hora universal.
34	setUTCMonth () Establece el mes de la fecha especificada según la hora universal.
35	setUTCSeconds () Establece los segundos de la fecha especificada según la hora universal.
36	setYear ()

	Obsoleto -Establece el año de la fecha especificada según la hora local. Uso setFullYear lugar.
37	toDateString () Devuelve la parte de "fecha" de la fecha como una cadena legible.
38	toGMTString () **Obsoleto -**Convierte una fecha en una cadena, usando las convenciones de Internet GMT. Uso toUTCString lugar.
39	toLocaleDateString () Devuelve la parte de "fecha" de la fecha como una cadena, usando reglas de la localización actual.
40	toLocaleFormat () Convierte una fecha en una cadena, usando una cadena de formato.
41	toLocaleString () Convierte una fecha en una cadena, usando reglas de la localización actual.
42	toLocaleTimeString () Devuelve la parte de "tiempo" de la fecha como una cadena, usando reglas de la localización actual.
43	a la fuente() Devuelve una cadena que representa el origen de un objeto Date equivalente; puede utilizar este valor para crear un nuevo objeto.
44	Encadenar() Devuelve una cadena que representa el objeto Date especificado.
45	toTimeString () Devuelve la parte de "tiempo" de la fecha como una cadena legible.
46	toUTCString () Convierte una fecha en una cadena, usando la convención de tiempo universal.
47	valor de() Devuelve el valor primitivo de un objeto Date.

Convierte una fecha en una cadena, usando la convención de tiempo universal.

Fecha métodos estáticos

Además de los muchos métodos de instancia enumerados anteriormente, el objeto Date también define dos métodos estáticos. Estos métodos se invocan a través de la constructor Date () en sí.

No Señor.	Método y Descripción
1	Date.parse () Analiza una cadena que representa una fecha y hora y devuelve la representación interna de milisegundos de esa fecha.
2	Date.UTC () Devuelve la representación de milisegundos de la fecha especificada y la hora UTC.

En las siguientes secciones, vamos a tener un par de ejemplos para demostrar los usos de la fecha de métodos estáticos.

JavaScript - El objeto Math

El objeto de matemáticas le proporciona propiedades y métodos para las constantes y funciones matemáticas. A diferencia de otros objetos globales, matemáticas no es un constructor. Todas las propiedades y métodos de Math son estáticos y pueden ser llamados mediante el uso de matemáticas como un objeto sin crearlo.

Por lo tanto, se hace referencia a la constante pi como Math.PI y se llama a la función seno como Math.sin (x), donde x es el argumento del método.

Sintaxis

La sintaxis para llamar a las propiedades y métodos de Math son como sigue

```
var pi_val = Math.PI;
sine_val var = Math.sin (30);
```

Propiedades matemáticas

Aquí está una lista de todas las propiedades de las matemáticas y su descripción.

No Señor.	Descripción de propiedad

1	E \ constante de Euler y la base de los logaritmos naturales, aproximadamente 2.718.
2	LN2 logaritmo natural de 2, aproximadamente 0.693.
3	LN10 logaritmo natural de 10, aproximadamente 2,302.
4	LOG2E Logaritmo en base 2 de E, de aproximadamente 1.442.
5	LOG10E Logaritmo en base 10 de E, de aproximadamente 0,434.
6	Pi Relación de la circunferencia de un círculo a su diámetro, aproximadamente 3,14159.
7	SQRT1_2 raíz cuadrada de 1/2; equivalentemente, 1 sobre la raíz cuadrada de 2, aproximadamente 0.707.
8	SQRT2 raíz cuadrada de 2, aproximadamente 1.414.

En las siguientes secciones, vamos a tener unos pocos ejemplos para demostrar el uso de las propiedades matemáticas.

Métodos matemáticos

Aquí es una lista de los métodos asociados con objeto Math y su descripción

No Señor.	Método y Descripción
1	abdominales() Devuelve el valor absoluto de un número.
2	acos ()

	Devuelve el arco coseno (en radianes) de un número.
3	como en() Devuelve el arco seno (en radianes) de un número.
4	un bronceado() Devuelve el arco tangente (en radianes) de un número.
5	atan2 () Devuelve el arcotangente del cociente de sus argumentos.
6	fortificar techo() Devuelve el número entero más pequeño mayor o igual que un número.
7	cos () Devuelve el coseno de un número.
8	Exp() Las devoluciones E^{norte}, Donde N es el argumento, y E es la constante de Euler, la base del logaritmo natural.
9	piso() Devuelve el mayor entero menor o igual a un número.
10	Iniciar sesión() Devuelve el logaritmo natural (base e) de un número.
11	max () Devuelve el mayor de cero o más números.
12	min () Devuelve el más pequeño de cero o más números.
13	pow () Las devoluciones de base a la potencia de exponente, es decir, exponente de base.
14	aleatorio() Devuelve un número pseudo-aleatorio entre 0 y 1.
15	redondo()

	Devuelve el valor de un número redondeado al entero más cercano.
dieciséis	pecado() Devuelve el seno de un número.
17	sqrt () Devuelve la raíz cuadrada de un número.
18	bronceado() Devuelve la tangente de un número.
19	a la fuente() Devuelve la cadena "Matemáticas".

En las siguientes secciones, vamos a tener unos pocos ejemplos para demostrar el uso de los métodos asociados con Math.

Las expresiones regulares y RegExp objeto

Una expresión regular es un objeto que describe un patrón de caracteres.

La clase RegExp JavaScript representa expresiones regulares, y las dos cuerdas y RegExp definir métodos que utilizan expresiones regulares para realizar poderosos de patrones y buscar y reemplazar las funciones de texto.

Sintaxis

Una expresión regular podría definirse con el constructor RegExp (), como sigue -

```
patrón var = new RegExp (patrón, atributos);
o simplemente
var pattern = / atributos patrón /;
```

Aquí está la descripción de los parámetros -

- **modelo**- Una cadena que especifica el patrón de la expresión regular u otra expresión regular.
- **atributos**- Una cadena opcional que contiene cualquiera de los "g", "i", y "m" atributos que especifican partidos globales, mayúsculas y minúsculas, y de varias líneas, respectivamente.

Soportes

Los corchetes ([]) tienen un especial significado cuando se utiliza en el contexto de las expresiones regulares. Se utilizan para encontrar un rango de caracteres.

No Señor.	Expresión y Descripción
1	**[...]** Cualquier carácter entre los corchetes.
2	**[^ ...]** Cualquier carácter no entre los corchetes.
3	**[0-9]** Coincide con cualquier dígito decimal de 0 a 9.
4	**[Arizona]** Coincide con cualquier carácter de una minúscula través de minúsculas z.
5	**[ARIZONA]** Coincide con cualquier carácter de la A a la Z. mayúscula mayúsculas
6	**[Arizona]** Coincide con cualquier carácter de una minúscula mayúscula a la Z.

Los rangos mostrados anteriormente son en general; también se podría utilizar el rango [0-3] para combinar con cualquier dígito decimal que va de 0 a 3, o el rango [bv] para que coincida con cualquier carácter en minúscula que van desde B a través v.

cuantificadores

La frecuencia o la posición de las secuencias de caracteres entre corchetes y los caracteres individuales se pueden denotan por un carácter especial. Cada personaje tiene una connotación especial específico. El +, *,?, Y banderas $ todo sigue una secuencia de caracteres.

No Señor.	Expresión y Descripción
1	**p +** Coincide con cualquier cadena que contenga uno o más de p.

2	**pag*** Coincide con cualquier cadena que contiene cero o más de p.
3	**¿pag?** Coincide con cualquier cadena que contenga como máximo un p.
4	**p {N}** Se coincide con cualquier cadena que contiene una secuencia de N p de
5	**p {2,3}** Coincide con cualquier cadena que contiene una secuencia de dos o tres de p.
6	**p {2,}** Coincide con cualquier cadena que contiene una secuencia de al menos dos de p.
7	**p $** Coincide con cualquier cadena con p al final de la misma.
8	**p ^** Coincide con cualquier cadena con p al comienzo de la misma.

Ejemplos

Los siguientes ejemplos explican más sobre caracteres coincidentes.

No Señor.	Expresión y Descripción
1	**[^ A-zA-Z]** Se coincide con cualquier cadena que no contiene ninguno de los caracteres que van desde A a la Z y A a la Z.
2	**páginas** Coincide con cualquier cadena que contiene p, seguido de cualquier carácter, a su vez, seguido de otro pág.

3	**^. {2} $** Coincide con cualquier cadena que contenga exactamente dos caracteres.
4	**<B> (. *) </ B>** Que coincide con cualquier cadena encerrada dentro de <b> y </ b>.
5	**p (CV) *** Se coincide con cualquier cadena que contiene ap seguido de cero o más instancias de la secuencia de hp.

Los caracteres literales

No Señor.	Descripción del personaje
1	**Alfanumérico** Sí mismo
2	**\ 0** El carácter NUL (\ u0000)
3	**\ t** Pestaña (\ u0009
4	**\norte** Nueva línea (\ u000A)
5	**\ v** pestaña vertical (\ u000B)
6	**\F** Avance de página (\ u000C)
7	**\ r** retorno de carro (\ u000d)

8	**\ xnn** El carácter latino especificado por el número hexadecimal nn; por ejemplo, \ x0A es el mismo que \ n
9	**\ uxxxx** El carácter Unicode especificado por el número hexadecimal xxxx; por ejemplo, \ u0009 es el mismo que \ t
10	**\ cX** El carácter de control ^ X; por ejemplo, \ cJ es equivalente al carácter de nueva línea \ n

metacaracteres

Un meta-carácter es simplemente un carácter alfabético precedido por una barra invertida que actúa para dar a la combinación de un significado especial.

Por ejemplo, se puede buscar una gran suma de dinero a través de la '\ d' metacarácter: / ([\ d] +) 000 /, aquí \ d buscará cualquier cadena de caracteres numéricos.

La siguiente tabla muestra un conjunto de meta-caracteres que se pueden utilizar en Perl estilo expresiones regulares.

No Señor.	Descripción del personaje
1	**.** un solo carácter
2	**\ s** un carácter de espacio en blanco (espacio, tabulación, nueva línea)
3	**\ S** no está en blanco
4	**\re** un dígito (0-9)
5	**\RE**

	un no-dígitos
6	**\ w** un carácter de palabra (az, AZ, 0-9, _)
7	**\ W** un carácter no-palabra
8	**[\si]** un retroceso literal (caso especial).
9	**[Aeiou]** coincide con un carácter único en el conjunto dado
10	**[^ Aeiou]** coincide con un solo carácter fuera del conjunto dado
11	**(Foo | bar | Baz)** coincide con cualquiera de las alternativas especificadas

modificadores

Varios modificadores están disponibles que pueden simplificar la forma de trabajar con las expresiones regulares, como mayúsculas y minúsculas, buscando en múltiples líneas, etc.

No Señor.	Modificador y Descripción
1	**yo** Realizar caso-insensible a juego.
2	**metro** Especifica que si la cadena tiene caracteres de nueva línea o retorno de carro, el ^ $ y los operadores serán ahora partido contra un límite de nueva línea, en lugar de un límite cuerdas

3	**sol** Realiza una matchthat mundial es, encontrar todos los partidos en lugar de detenerse después de que el primer partido.

Propiedades RegExp

Aquí está una lista de las propiedades asociadas con RegExp y su descripción.

No Señor.	Descripción de propiedad
1	constructor Specifies the function that creates an object's prototype.
2	global Especifica si el "g" modificador está establecido.
3	ignorar caso Especifica si se establece el "i" modificador.
4	lastIndex El índice en el cual para iniciar el próximo partido.
5	multilínea Especifica si el modificador de "m" está establecido.
6	fuente El texto del patrón.

En las siguientes secciones, vamos a tener unos pocos ejemplos para demostrar el uso de propiedades RegExp.

Métodos RegExp

Aquí está una lista de los métodos asociados con la expresión regular junto con su descripción.

No Señor.	Método y Descripción

1	exec () Ejecuta una búsqueda de una coincidencia en su parámetro de cadena.
2	prueba() Las pruebas para un partido en su parámetro de cadena.
3	a la fuente() Devuelve un literales objeto que representa el objeto especificado; puede utilizar este valor para crear un nuevo objeto.
4	Encadenar() Devuelve una cadena que representa el objeto especificado.

En las siguientes secciones, vamos a tener unos pocos ejemplos para demostrar el uso de los métodos de RegExp.

JavaScript - Documento Modelo de objetos o DOM

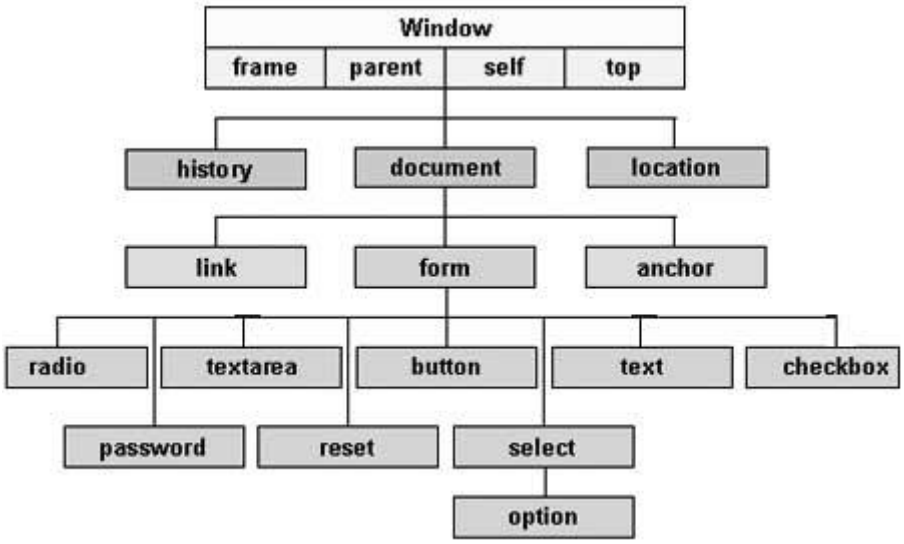
Cada página Web reside dentro de una ventana del navegador que puede ser considerado como un objeto.

Un objeto de documento representa el documento HTML que se muestra en esa ventana. El objeto del documento tiene varias propiedades que hacen referencia a otros objetos que permiten el acceso a y modificación de contenido del documento.

La forma en que se accede a un contenido del documento y modificado se llama el modelo de objetos de documento, o DOM. Los objetos se organizan en una jerarquía. Esta estructura jerárquica se aplica a la organización de los objetos en un documento Web.

- **objeto de la ventana**- Parte superior de la jerarquía. Es el elemento más exterior de la jerarquía de objetos.
- **objeto de documento**- Cada documento HTML que se carga en una ventana se convierte en un objeto de documento. El documento contiene el contenido de la página.
- **objeto de formulario**- Todo lo incluido en las etiquetas <form> ... </ form> conjuntos de la forma del objeto.
- **elementos de control de formulario**- El objeto formulario contiene todos los elementos definidos para ese objeto, tales como campos de texto, botones, botones de radio, casillas de verificación y.

Aquí es un simple jerarquía de algunos objetos importantes -



Hay varios departamentos de ultramar en existencia. Las siguientes secciones explican cada uno de estos departamentos de ultramar en detalle y describen cómo se les puede utilizar para acceder y modificar el contenido del documento.

- El legado de DOM- Este es el modelo que se introdujo en las primeras versiones del lenguaje JavaScript. Es bien soportado por todos los navegadores, pero permite el acceso sólo a ciertas partes clave de documentos, tales como formularios, elementos de formulario, e imágenes.
- El DOM W3C- Este modelo de objetos de documento permite el acceso y la modificación de todo el contenido del documento y está estandarizado por el Consorcio de la World Wide Web (W3C). Este modelo se apoya en casi todos los navegadores modernos.
- El IE4 DOM- Este modelo de objeto de documento se introdujo en la versión 4 del navegador de Microsoft Internet Explorer. IE 5 y versiones posteriores incluyen soporte para la mayoría de las características básicas de DOM W3C.

compatibilidad DOM

Si desea escribir un guión con la flexibilidad de utilizar ya sea W3C DOM o IE 4 DOM en función de su disponibilidad, a continuación, puede utilizar un enfoque de pruebas de capacidad que busca en primer lugar la existencia de un método o una propiedad para determinar si el navegador tiene la capacidad que usted desea. Por ejemplo -

```
si (document.getElementById) {
 // Si existe el método del W3C, lo utilizan
} Else if (document.all) {
 // Si existe el todo [] array, usarlo
} Else {
 // De lo contrario usar el DOM legado
}
```

JavaScript - Errores y manejo de excepciones

Hay tres tipos de errores en la programación: (a) errores de sintaxis, (b) los errores de ejecución, y (c) los errores lógicos.

Los errores de sintaxis

Los errores de sintaxis, también llamados errores de análisis, se producen en tiempo de compilación en lenguajes de programación tradicionales y, al interpretar el tiempo en JavaScript.

Por ejemplo, la siguiente línea hace que un error de sintaxis porque falta un paréntesis de cierre.

```
<Script type = "text / javascript">
 <! -
 window.print (;
 // ->
</ Script>
```

Cuando se produce un error de sintaxis en JavaScript, sólo el código de contenido dentro de la misma rosca que se ve afectado el error de sintaxis y el resto del código en otros hilos es ejecutado suponiendo nada en ellos depende del código que contiene el error.

Los errores de tiempo de ejecución

los errores de ejecución, también llamadas excepciones, ocurrir durante la ejecución (después de la compilación y / o interpretación).

Por ejemplo, la siguiente línea hace que un error de ejecución porque aquí la sintaxis es correcta, pero en tiempo de ejecución, que está tratando de llamar a un método que no existe.

```
<Script type = "text / javascript">
 <! -
 window.printme ();
 // ->
</ Script>
```

Excepciones también afectan a la rosca en el que se producen, lo que permite otros hilos de JavaScript para continuar la ejecución normal.

Los errores lógicos

Los errores lógicos pueden ser del tipo difícil la mayor parte de los errores de rastrear. Estos errores no son el resultado de un error de sintaxis o de tiempo de ejecución. En su lugar, se producen cuando se comete un error en la lógica que impulsa su guión y que no obtienen el resultado que se esperaba.

No se puede coger esos errores, porque depende de sus necesidades de negocio de qué tipo de lógica que desea poner en su programa.

El try ... catch ... finally

Las últimas versiones de JavaScript añaden capacidades de manejo de excepciones. implementos de JavaScript del try ... catch ... finally constructo, así como el operador de tiro a excepciones manejar.

Puede capturar las excepciones programador-generado y tiempo de ejecución, pero no se puede detectar errores de sintaxis de JavaScript.

Aquí está el try ... catch ... finally sintaxis -

```
<script tipo = "Text / javascript">
 <! -
 tratar {
 // código se ejecute
 [descanso;]
 }

 captura ( mi ) {
 // Código que se ejecuta si se produce una excepción
 [descanso;]
 }

 [ finalmente {
 // Código que se ejecuta siempre, independientemente de
 // una excepción que ocurre
 }]
 // ->
</ Script>
```

El bloque try debe ir seguido de exactamente un bloque catch o un bloque finally (o uno de los dos). Cuando se produce una excepción en el bloque try, la excepción se coloca en e y se ejecuta el bloque catch. El bloque finally se ejecuta opcional incondicionalmente después de try / catch.

Ejemplos

Aquí está un ejemplo en el que estamos tratando de llamar a una función no existente, que a su vez está levantando una excepción. Veamos cómo se comporta sin probar ... Catch

```
<Html>
 <Head>
 <script tipo = "Text / javascript">
 <! -
 función myFunc() {
 var un = 100;
 alerta("El valor de una variable es:" + un );
 }
 // ->
 </ Script>
 </ Head>

 <Body>
 <P>Haga clic en el siguiente para ver el resultado:</ P>
```

```
 <Form>
 <input tipo = "botón" valor = "Haz click en mi" al hacer
clic = "myFunc();" />
 </ Form>
 </ Body>
</ Html>
```

Salida

Ahora vamos a tratar de detectar esta excepción utilizando try ... catch y mostrar un mensaje fácil de usar. También puede suprimir este mensaje, si desea ocultar este error de un usuario.

```
<Html>
 <Head>

 <script tipo = "Text / javascript">
 <! -
 función myFunc() {
 var un = 100;
 tratar {
 alerta("El valor de una variable es:" + un );
 }
 captura ( mi ) {
 alerta("Error:" + mi.descripción);
 }
 }
 // ->
 </ Script>

 </ Head>
 <Body>
 <P>Haga clic en el siguiente para ver el resultado:</ P>

 <Form>
 <input tipo = "botón" valor = "Haz click en mi" al hacer
clic = "myFunc();" />
 </ Form>

 </ Body>
</ Html>
```

Salida

Se puede utilizar por último bloque que siempre se ejecutará incondicionalmente después del try / catch. Aquí hay un ejemplo.

```
<Html>
 <Head>
```

```
 <script tipo = "Text / javascript">
 <! -
 función myFunc() {
 var un = 100;

 tratar {
 alerta("El valor de una variable es:" + un );
 }
 captura ( mi ) {
 alerta("Error:" + mi.descripción);
 }
 finalmente {
 alerta("Por último bloque se ejecutará siempre!" );
 }
 }
 // ->
 </ Script>

 </ Head>
 <Body>
 <P>Haga clic en el siguiente para ver el resultado:</ P>

 <Form>
 <input tipo = "botón" valor = "Haz click en mi" al hacer
clic = "myFunc();" />
 </ Form>

 </ Body>
</ Html>
```

Salida

La sentencia throw

Se puede utilizar instrucción throw para elevar sus excepciones incorporado o sus excepciones personalizadas. Más tarde, estas excepciones pueden ser capturados y se puede tomar una acción apropiada.

Ejemplo

El siguiente ejemplo muestra cómo utilizar una instrucción throw.

```
<Html>
 <Head>

 <script tipo = "Text / javascript">
 <! -
 función myFunc() {
 var un = 100;
```

```
 var si = 0;

 tratar {
 Si ( si == 0 ) {
 lanzar( "Error de división por cero." );
 } más {
 var C = un / si;
 }
 }
 captura ( mi ) {
 alerta("Error:" + mi );
 }
 }
 // ->
 </ Script>

 </ Head>
 <Body>
 <P>Haga clic en el siguiente para ver el resultado:</ P>

 <Form>
 <input tipo = "botón" valor = "Haz click en mi" al hacer
clic = "myFunc();" />
 </ Form>

 </ Body>
</ Html>
```

Salida

Puede provocar una excepción en una función usando una cadena, entero, booleano o un objeto y luego se puede capturar esa excepción, ya sea en la misma función como lo hicimos anteriormente, o en otra función utilizando un bloque try ... catch.

El onerror () Método

El controlador de eventos onerror fue la primera característica para facilitar el manejo de errores en JavaScript. El evento de error se dispara en el objeto ventana cada vez que se produce una excepción en la página.

```
<Html>
 <Head>

 <script tipo = "Text / javascript">
 <! -
 ventana.onerror = función () {
 alerta("Ocurrió un error.");
 }
 // ->
 </ Script>
```

```
 </ Head>
 <Body>
 <P>Haga clic en el siguiente para ver el resultado:</ P>

 <Form>
 <input tipo = "botón" valor = "Haz click en mi" al hacer
clic = "myFunc();" />
 </ Form>

 </ Body>
</ Html>
```

Salida

El controlador de eventos onerror ofrece tres tipos de información para identificar la naturaleza exacta del error -

- **Mensaje de error**- El mismo mensaje que el navegador podría mostrar para el error dado
- **URL**- El archivo en el que se produjo el error
- **Número de línea**- El número de línea en la URL dada que provocó el error

Aquí está el ejemplo para mostrar cómo extraer esta información.

Ejemplo

```
<Html>
 <Head>

 <script tipo = "Text / javascript">
 <! -
 ventana.onerror = función (msg, url, línea) {
 alerta("Mensaje:" + msg );
 alerta("Url:" + url );
 alerta("Número de línea : " + línea );
 }
 // ->
 </ Script>

 </ Head>
 <Body>
 <P>Haga clic en el siguiente para ver el resultado:</ P>

 <Form>
 <input tipo = "botón" valor = "Haz click en mi" al hacer
clic = "myFunc();" />
 </ Form>

 </ Body>
```

```
</ Html>
```

Salida

Se puede visualizar la información extraída de la manera que usted piensa que es mejor.

Se puede utilizar un método onerror, como se muestra a continuación, para mostrar un mensaje de error en caso de que haya algún problema en la carga de una imagen.

```
<img src="Mimagen.gif" onerror="alerta('Ha ocurrido un error
al cargar la imagen.)" />
```

Puede utilizar onerror con muchas etiquetas HTML para mostrar los mensajes apropiados en caso de errores.

JavaScript - Formulario de Validación

la validación de formularios utiliza normalmente para producir en el servidor, después de que el cliente había introducido todos los datos necesarios y luego pulsa el botón Enviar. Si los datos introducidos por un cliente era incorrecta o simplemente faltaba, el servidor tendría que enviar toda la parte trasera de datos al cliente y solicitar que se vuelva a presentar el formulario con la información correcta. Esto fue realmente un largo proceso que se usa para poner una gran cantidad de carga en el servidor.

JavaScript proporciona una manera de validar los datos del formulario en la computadora del cliente antes de enviarlo al servidor web. la validación de formularios generalmente realiza dos funciones.

- **Validación básica**- En primer lugar, la forma debe ser comprobado para asegurarse de que todos los campos obligatorios se rellenan Se requeriría sólo un bucle a través de cada campo en el formulario y comprobar si hay datos..
- **Formato de datos de validación**- En segundo lugar, los datos que se introducen deben ser revisados por la forma y el valor correcto. El código debe incluir lógica apropiada a la corrección de los datos de prueba.

Ejemplo

Vamos a tomar un ejemplo para entender el proceso de validación. Aquí es una forma sencilla en formato html.

```
<Html>
 <Head>
 <Title>Validación de formularios</ Title>
 <script tipo = "Text / javascript">
 <! -
 // código de validación Formulario vendrá aquí.
 // ->
 </ Script>
 </ Head>
```

```
 <Body>
 <form acción = "/Cgi-bin/test.cgi" nombre = "MyForm"
onsubmit = "regreso(validar());">
 <table cellspacing = "2" cellpadding = "2" frontera = "1">

 <Tr>
 <td alinear = "Derecha">Nombre</ Td>
 <Td> <input tipo = "texto" nombre = "Nombre" /> </ Td>
 </ Tr>

 <Tr>
 <td alinear = "Derecha">Correo electrónico</ Td>
 <Td> <input tipo = "texto" nombre = "Correo electrónico" />
</ Td>
 </ Tr>

 <Tr>
 <td alinear = "Derecha">Código postal</ Td>
 <Td> <input tipo = "texto" nombre = "Código Postal" /> </
Td>
 </ Tr>

 <Tr>
 <td alinear = "Derecha">País</ Td>
 <Td>
 <seleccione nombre = "País">
 <opción valor = "-1" seleccionado>[Elegir el suyo]</ Option>
 <opción valor = "1">Estados Unidos</ Option>
 <opción valor = "2">Reino Unido</ Option>
 <opción valor = "3">INDIA</ Option>
 </ Select>
 </ Td>
 </ Tr>

 <Tr>
 <td alinear = "Derecha"> </ Td>
 <Td> <input tipo = "enviar" valor = "Enviar" /> </ Td>
 </ Tr>

 </ Table>
 </ Form>
 </ Body>
</ Html>
```

Salida

Validación de formularios básico

En primer lugar vamos a ver cómo hacer una validación de forma básica. En la forma anterior, estamos llamando validate () a los datos de validación cuando

se está produciendo evento onsubmit. El código siguiente muestra la implementación de esta función validate ().

```
<script tipo = "Text / javascript">
 <! -
 // código de validación Formulario vendrá aquí.
 función validar() {

 Si( documento.myForm.Nombre.valor == "" ) {
 alerta( "Por favor proporcione su nombre!" );
 documento.myForm.Nombre.atención() ;
 regreso falso;
 }
 Si( documento.myForm.Correo electrónico.valor == "" ) {
 alerta( "Por favor, proporcione su correo electrónico!" );
 documento.myForm.Correo electrónico.atención() ;
 regreso falso;
 }
 Si( documento.myForm.Código Postal.valor == "" || isNaN(
documento.myForm.Código Postal.valor ) ||
 documento.myForm.Código Postal.valor.longitud ! = 5 ) {

 alerta( "Proporciona una postal en el formato #####". );
 documento.myForm.Código Postal.atención() ;
 regreso falso;
 }
 Si( documento.myForm.País.valor == "-1" ) {
 alerta( "Por favor, proporcione su país!" );
 regreso falso;
 }
 regreso( cierto );
 }
 // ->
</ Script>
```

Formato de datos de validación

Ahora vamos a ver cómo podemos validar nuestros datos de formulario introducidos antes de enviarlo al servidor web.

El siguiente ejemplo muestra cómo validar una dirección de correo electrónico introducida. Una dirección de correo electrónico debe contener al menos un signo '@' y un punto (.). Además, la '@' no debe ser el primer carácter de la dirección de correo electrónico, y el último punto imprescindible al menos sea un carácter después del signo '@'.

Ejemplo

Pruebe el siguiente código de validación de correo electrónico.

```
<script tipo = "Text / javascript">
 <! -
```

```
 función validar correo electrónico() {
 var identificación de correo = documento.myForm.Correo
electrónico.valor;
 ATPOS = identificación de correo.índice de("@");
 dotpos = identificación de correo.lastIndexOf("");

 Si (ATPOS < 1 || ( dotpos - ATPOS < 2 )) {
 alerta("Por favor, introduzca el ID de correo electrónico
correcta")
 documento.myForm.Correo electrónico.atención() ;
 regreso falso;
 }
 regreso( cierto );
 }
 // ->
</ Script>
```

JavaScript - Animación

Se puede utilizar JavaScript para crear una animación compleja que tiene, pero no limitado a, los siguientes elementos -

- Fuegos artificiales
- Efecto de desvanecimiento
- Roll-in o el despliegue de
- Página de entrada o de salida página
- movimientos de objetos

Quizás se encuentre interesado en la biblioteca de animación basada en JavaScript existente:Script.Aculo.us.

Este tutorial proporciona una comprensión básica de cómo utilizar JavaScript para crear una animación.

JavaScript se puede utilizar para mover un número de elementos DOM (<img />, o cualquier otro elemento HTML) alrededor de la página de acuerdo con algún tipo de patrón determinado por una ecuación o función lógica.

JavaScript proporciona las siguientes dos funciones que se utilizan con frecuencia en los programas de animación.

- **setTimeout (función, duración)**- Esta función llama a la función después de milisegundos de duración a partir de ahora.
- **setInterval (función, duración)**- Esta función llama a la función después de cada milisegundos de duración.
- **clearTimeout (setTimeout_variable)**- Esta función llama borra cualquier conjunto por el temporizador () funciones setTimeout.

JavaScript también puede establecer una serie de atributos de un objeto DOM incluyendo su posición en la pantalla. Puede configurar superior e izquierda atributo de un objeto para colocarlo en cualquier lugar de la pantalla. Aquí su sintaxis.

```
// conjunto de distancia desde el borde izquierdo de la
pantalla.
object.style.left = distancia en píxeles o puntos;

o

// conjunto de distancia desde el borde superior de la
pantalla.
object.style.top = distancia en píxeles o puntos;
```

Animación Manual

Así que vamos a implementar una animación simple usando las propiedades del objeto DOM y funciones de JavaScript de la siguiente manera. La siguiente lista contiene diferentes métodos DOM.

- Estamos utilizando la función de JavaScript getElemenById () para obtener un objeto DOM y después asignarlo a un imgObj variable global.
- Hemos definido una función init inicialización () para inicializar imgObj donde nos hemos fijado su posición y atributos izquierda.
- Estamos llamando a la función de inicialización en el momento de la carga de la ventana.
- Por último, estamos llamando a la función MoverDerecha () para aumentar la distancia izquierda por 10 píxeles. También se podría establecer en un valor negativo para moverlo hacia el lado izquierdo.

Ejemplo

El siguiente ejemplo muestra.

```
<Html>
 <Head>
 <Title>JavaScript Animación</ Title>
 <script tipo = "Text / javascript">
 <! -
 var imgObj = nulo;

 función en eso() {
 imgObj = documento.getElemenById('Mi imagen');
 imgObj.estilo.posición= 'relativo';
 imgObj.estilo.izquierda = '0px';
 }
 función mover a la derecha() {
 imgObj.estilo.izquierda = parseInt(imgObj.estilo.izquierda)
+ 10 + 'Px';
 }

 ventana.onload = en eso;
 // ->
 </ Script>
 </ Head>
```

```
 <Body>
 <Form>
 <img carné de identidad = "Mi imagen" src =
"/Images/html.gif" />
 <P>Haga clic en botón de abajo para mover la imagen a la
derecha</ P>
 <input tipo = "botón" valor = "Haz click en mi" al hacer
clic = "mover a la derecha();" />
 </ Form>
 </ Body>
</ Html>
```

Salida

automatizado de Animación

En el ejemplo anterior, vimos cómo una imagen se mueve hacia la derecha con cada clic. Podemos automatizar este proceso mediante el uso de la función setTimeout () de JavaScript de la siguiente manera -

Aquí hemos añadido más métodos. Así que vamos a ver lo que es nuevo aquí -

- La función MoveRight () está llamando a la función setTimeout () para establecer la posición de imgObj.
- Hemos añadido una nueva función stop () para borrar el conjunto del temporizador setTimeout () y para establecer el objeto a su posición inicial.

Ejemplo

Pruebe el siguiente código de ejemplo.

```
<Html>
 <Head>
 <Title>JavaScript Animación</ Title>
 <script tipo = "Text / javascript">
 <! -
 var imgObj = nulo;
 var animar ;

 función en eso() {
 imgObj = documento.getElemenById('Mi imagen');
 imgObj.estilo.posición= 'relativo';
 imgObj.estilo.izquierda = '0px';
 }
 función mover a la derecha() {
 imgObj.estilo.izquierda = parseInt(imgObj.estilo.izquierda)
+ 10 + 'Px';
 animar = setTimeout(mover a la derecha,20); // MoverDerecha
llamada en 20msec
```

```
 }
 función detener() {
 clearTimeout(animar);
 imgObj.estilo.izquierda = '0px';
 }

 ventana.onload = en eso;
 // ->
 </ Script>
 </ Head>

 <Body>
 <Form>
 <img carné de identidad = "Mi imagen" src =
"/Images/html.gif" />
 <P>Haga clic en los botones de abajo para la animación
mango</ P>
 <input tipo = "botón" valor = "Comienzo" al hacer clic =
"mover a la derecha();" />
 <input tipo = "botón" valor = "Detener" al hacer clic =
"detener();" />
 </ Form>
 </ Body>
</ Html>
```

Vuelco con un evento de ratón

Aquí está un ejemplo simple que muestra vuelco imagen con un evento de ratón.

A ver que estamos utilizando en el siguiente ejemplo -

- En el momento de carga de esta página, los 'si' comprobaciones de los estados de la existencia del objeto de la imagen. Si el objeto de la imagen no está disponible, no se ejecutará este bloque.
- La imagen () crea y precarga un nuevo objeto imagen llamada imagen1.
- La propiedad src se le asigna el nombre del archivo de imagen externo llamado /images/html.gif.
- Del mismo modo, hemos creado objeto imagen2 y asignado /images/http.gif en este objeto.
- El # (almohadilla) desactiva el enlace para que el navegador no trata de ir a una dirección URL cuando se hace clic. Este enlace es una imagen.
- El controlador de eventos onMouseOver se activa cuando el usuario del ratón se mueve sobre el enlace, y el controlador de eventos onMouseOut se activa cuando el usuario del ratón se aleja del enlace (imagen).
- Cuando se mueve el ratón sobre la imagen, la imagen cambia HTTP desde la primera imagen a la segunda. Cuando el ratón se mueve fuera de la imagen, se muestra la imagen original.
- Cuando el ratón se mueve lejos del enlace, el html.gif imagen inicial volverá a aparecer en la pantalla.

```
<Html>

 <Head>
 <Title>Vuelco con un ratón Eventos</ Title>

 <script tipo = "Text / javascript">
 <! -
 Si(documento.imágenes) {
 var imagen1 = nuevo Imagen(); // una imagen de precarga
 imagen1.src = "/Images/html.gif";
 var imagen2 = nuevo Imagen(); // segunda imagen de precarga
 imagen2.src = "/Images/http.gif";
 }
 // ->
 </ Script>
 </ Head>

 <Body>
 <P>Mover el puntero del ratón sobre la imagen para ver el
resultado</ P>

 <a href = "#" el ratón por encima = "documento.Mi imagen.src
= imagen2.src;"
 onMouseOut = "documento.Mi imagen.src = imagen1.src;">
 <img nombre = "Mi imagen" src = "/Images/html.gif" />
 </a>
 </ Body>
</ Html>
```

JavaScript - Multimedia

El navegador de objetos JavaScript incluye un objeto hijo llamado plugins. Este objeto es una matriz, con una entrada para cada plug-in instalado en el navegador. El objeto navigator.plugins sólo es compatible con Netscape, Firefox y Mozilla solamente.

Ejemplo

He aquí un ejemplo que muestra cómo se listan abajo todo el plug-on instalado en su navegador -

```
<Html>
 <Head>
 <Title>Lista de los plug-ins</ Title>
 </ Head>

 <Body>
 <table frontera = "1">
 <Tr>
 <Th>Plug-in Nombre</ Th>
 <Th>Nombre del archivo</ Th>
```

```
 <Th>Descripción</ Th>
 </ Tr>

 <script idioma = "JavaScript" tipo = "Text / javascript">
 para (yo = 0; yo<navegador.plugins.longitud; yo++) {
 documento.escribir("<Tr> <td>");
 documento.escribir(navegador.plugins[yo].nombre);
 documento.escribir("</ Td> <td>");
 documento.escribir(navegador.plugins[yo].nombre del
archivo);
 documento.escribir("</ Td> <td>");
 documento.escribir(navegador.plugins[yo].descripción);
 documento.escribir("</ Td> </ tr>");
 }
 </ Script>
 </ Table>
 </ Body>
</ Html>
```

Salida

Comprobación de los plug-ins

Cada plug-in tiene una entrada en la matriz. Cada entrada tiene las siguientes propiedades -

- **nombre**- es el nombre del plug-in.
- **nombre del archivo**- es el archivo ejecutable que se cargó para instalar el plug-in.
- **descripción**- es una descripción del plug-in, suministrado por el desarrollador.
- **mimeTypes**- es una matriz con una entrada para cada tipo MIME soportado por el plug-in.

Puede utilizar estas propiedades en una secuencia de comandos para averiguar los plug-ins instalados y haga uso de JavaScript, puede reproducir archivos multimedia apropiado. Echar un vistazo al siguiente ejemplo.

```
<Html>
 <Head>
 <Title>El uso de plug-ins</ Title>
 </ Head>

 <Body>
 <script idioma = "JavaScript" tipo = "Text / javascript">
 medios de comunicación = navegador.mimeTypes["Video /
quicktime"];

 Si (medios de comunicación) {
 documento.escribir("<Embed src = 'quick.mov' height = 100
width = 100>");
 } más {
```

```
 documento.escribir("< 'Quick.gif' altura img src = = 100
width = 100>");
 }
 </ Script>
 </ Body>
</ Html>
```

Salida

NOTA- Aquí estamos utilizando HTML <embed> para incrustar un archivo multimedia.

Control de Multimedia

Tomemos un ejemplo real, que funciona en casi todos los navegadores -

```
<Html>
 <Head>
 <Title>El uso de objetos incrustado</ Title>

 <script tipo = "Text / javascript">
 <! -
 función jugar() {
 Si (!documento.manifestación.Está jugando()) {
 documento.manifestación.Jugar();
 }
 }
 función detener() {
 Si (documento.manifestación.Está jugando()) {
 documento.manifestación.StopPlay();
 }
 }
 función rebobinar() {
 Si (documento.manifestación.Está jugando()) {
 documento.manifestación.StopPlay();
 }
 documento.manifestación.Rebobinar();
 }
 // ->
 </ Script>
 </ Head>

 <Body>
 <embed carné de identidad = "manifestación" nombre =
"manifestación"
 src = "Http://www.amrood.com/games/kumite.swf"
 anchura = "318" altura = "300" jugar = "falso" lazo =
"falso"
 pluginspage = "Http://www.macromedia.com/go/getflashplayer"
 swLiveConnect = "cierto">
```

```
 <form nombre = "formar" carné de identidad = "formar" acción
= "#" método = "obtener">
 <input tipo = "botón" valor = "Comienzo" al hacer clic =
"jugar();" />
 <input tipo = "botón" valor = "Detener" al hacer clic =
"detener();" />
 <input tipo = "botón" valor = "Rebobinar" al hacer clic =
"rebobinar();" />
 </ Form>
 </ Body>
</ Html>
```

Salida

Si está utilizando Mozilla, Firefox o Netscape, a continuación,

JavaScript - Depuración

De vez en cuando, los desarrolladores de cometer errores mientras que la codificación. Un error en un programa o una secuencia de comandos que se conoce como un error.

El proceso de encontrar y corregir errores que se llama depuración y es una parte normal del proceso de desarrollo. Esta sección cubre las herramientas y técnicas que le pueden ayudar con la depuración de tareas ..

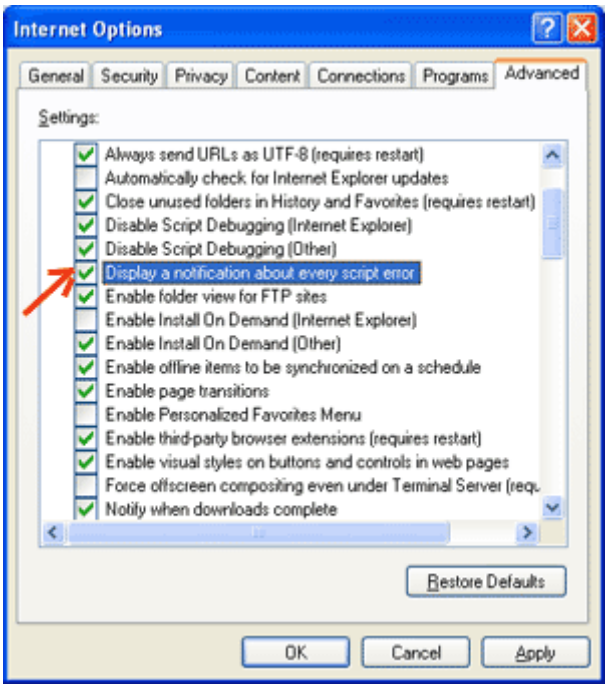
Mensajes de error en el IE

La forma más básica para el diagnóstico de problemas es mediante la activación de información de error en su navegador. De forma predeterminada, Internet Explorer muestra un icono de error en la barra de estado cuando se produce un error en la página.

Al hacer doble clic en este icono le lleva a un cuadro de diálogo que muestra información sobre el error específico que se produjo.

Desde este icono es fácil pasar por alto, Internet Explorer le da la opción para mostrar automáticamente el cuadro de diálogo de error cuando se produce un error.

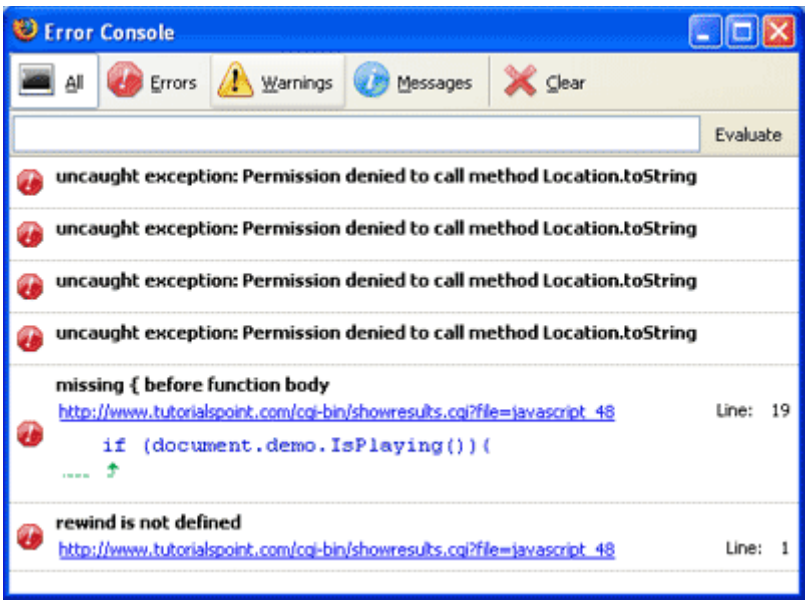
Para activar esta opción, seleccione Herramientas → Opciones de Internet → pestaña Avanzado. y, finalmente, comprobar la opción del cuadro "Mostrar una notificación sobre cada error de secuencias de comandos", como se muestra a continuación -



Mensajes de error en Firefox o Mozilla

Otros navegadores como Firefox, Netscape, Mozilla y mensajes de error de envío a una ventana especial llamada la consola JavaScript o Error Consol. Para ver la consola, seleccione Herramientas → Error Consol o desarrollo web.

Por desgracia, ya que estos navegadores no dan ninguna indicación visual cuando se produce un error, se debe mantener la consola abierta y esté pendiente de errores como sus ejecuta scripts.



Notificaciones de error

notificaciones de error que aparecen en la consola oa través de cuadros de diálogo de Internet Explorer son el resultado de dos errores de sintaxis y de tiempo de ejecución. Estos notificación de error incluye el número de línea en la que se produjo el error.

Si está usando Firefox, a continuación, puede hacer clic en el error disponibles en la consola de errores para ir a la línea exacta en el error de script que tiene.

Cómo depurar una secuencia de comandos

Hay varias maneras de depurar JavaScript -

Utilizar un validador Javascript

Una forma de comprobar su código JavaScript para insectos extraños es ejecutarlo a través de un programa que comprueba para asegurarse de que es válida y que sigue las reglas oficiales de sintaxis de la lengua. Estos programas se denominan analizadores validar o validadores, sólo para abreviar, y, a menudo vienen con HTML comercial y editores de JavaScript.

El validador más conveniente para JavaScript es de Douglas Crockford Javascript Lint, que está disponible durante al libre JavaScript pelusa de Douglas Crockford.

Sólo tiene que visitar la página web, pegue el código en el área de texto proporcionado JavaScript (JavaScript), y haga clic en el botón JSLint. Este programa va a analizar a través de su código JavaScript, lo que garantiza que todas las definiciones de variables y funciones siguen la sintaxis correcta. También comprobará sentencias JavaScript, como si, y cuando, para asegurarse de que ellos también siguen el formato correcto

Agregar código de depuración para sus programas

Puede utilizar la alerta () o document.write () en su programa para depurar el código. Por ejemplo, es posible escribir algo de la siguiente manera -

```
var depuración = cierto;
var cuál imagen = "Widget";

Si( depuración )
 alerta( "Las llamadas swapImage () con el argumento:" + cuál
imagen );
 var swapStatus = swapImage( cuál imagen );

Si( depuración )
 alerta( "Salidas swapImage () con swapStatus =" + swapStatus
);
```

Al examinar el contenido y el orden de la alerta (), tal como aparecen, se puede examinar la salud de su programa muy fácilmente.

Utilizar un depurador de JavaScript

Un depurador es una aplicación que coloca todos los aspectos de la ejecución del script bajo el control del programador. Depuradores proporcionan un control preciso sobre el estado de la secuencia de comandos a través de una interfaz que le permite examinar y valores establecidos, así como el control del flujo de ejecución.

Una vez que un script se ha cargado en un depurador, se puede ejecutar una línea a la vez o instrucciones para detenerse en ciertos puntos de interrupción. Una vez que se detiene la ejecución, el programador puede examinar el estado de la secuencia de comandos y sus variables con el fin de determinar si algo está mal. También puede ver las variables de los cambios en sus valores.

La última versión del depurador de JavaScript de Mozilla (Venkman nombre en código) tanto para los navegadores Mozilla y Netscape se puede descargar enhttp://www.hacksrus.com/~ginda/venkman

Consejos útiles para Desarrolladores

Usted puede mantener los siguientes consejos en mente para reducir el número de errores en los scripts y simplificar el proceso de depuración -

- Use un montón de comentarios. Comentarios le permiten explicar por qué escribió el guión de la manera que lo hizo y para explicar las secciones particularmente difíciles de código.
- Siempre use sangría para que su código fácil de leer. Sangrado de las declaraciones también hace que sea más fácil para usted para que coincida con las etiquetas de inicio y fin, llaves y otros elementos HTML y script.
- Escribir código modular. Siempre que sea posible, el grupo de las sentencias en funciones. Funciones que permiten declaraciones relacionados con el grupo, y porciones de ensayo y la reutilización de código con el mínimo esfuerzo.
- Sea consistente en la forma de nombrar las variables y funciones. Trate de usar los nombres que son el tiempo suficiente para que tenga sentido y que describen el contenido de la variable o el propósito de la función.
- Utilice la sintaxis coherente al asignar nombres a las variables y funciones. En otras palabras, los mantienen en minúsculas o en mayúsculas; si lo prefiere lomo de camello notación, utilizar de manera consistente.
- **Probar guiones largos**de forma modular. En otras palabras, no se trata de escribir el guión completo antes de probar cualquier porción de él. Escribir una pieza y conseguir que funcione antes de añadir la siguiente porción de código.
- Utilice nombres de variables y funciones descriptivas y evitar el uso de nombres de un solo carácter.
- **Mire sus comillas**. Recuerde que la cita marcas se usan en pares alrededor de cadenas y que ambas comillas deben ser del mismo estilo (simple o doble).
- **Mire sus signos iguales**. Usted no debe usa un solo = con fines comparativos.
- las variables declarar explícitamente el uso de la palabra clave var.

Se puede utilizar para crear JavaScript del lado del cliente mapa de imagen. mapas de imagen del lado del cliente están habilitadas por el atributo usemap para la etiqueta <img /> y definidos por especial <map> y <area> etiquetas de extensión.

La imagen que se va a formar el mapa se inserta en la página utilizando la etiqueta <img /> elemento de forma normal, excepto que lleva un atributo extra llamado usemap. El valor del atributo usemap es el valor del atributo de nombre en el elemento <map>, que está a punto de cumplir, precedido por un signo de almohadilla.

La <map> en realidad crea el mapa de la imagen y por lo general sigue directamente después de la etiqueta <img /> elemento. Actúa como un contenedor para los </ área> elementos que realmente definen las zonas activas. La <map> lleva sólo un atributo, el atributo de nombre, que es el nombre que identifica el mapa. Así es como el <img /> elemento sabe elemento que <map> para su uso.

El elemento <area> especifica la forma y las coordenadas que definen los límites de cada punto de acceso puede hacer clic.

Los siguientes mapas de imágenes y cosechadoras de código JavaScript para producir un mensaje en un cuadro de texto cuando el ratón se mueve sobre diferentes partes de una imagen.

```
<Html>
 <Head>
 <Title>Uso de JavaScript mapa de imagen</ Title>

 <script tipo = "Text / javascript">
 <! -
 función Tutorial de presentación(nombre) {
 documento.myform.etapa.valor = nombre
 }
 // ->
 </ Script>
 </ Head>

 <Body>
 <form nombre = "Myform">
 <input tipo = "texto" nombre = "etapa" Talla = "20" />
 </ Form>

 <! - crear asignaciones ->
 <img src = "/Images/usemap.gif" alt = "HTML Mapa" frontera =
"0" usemap = "#tutorials"/>

 <mapa nombre = "tutoriales">
 <area forma="escuela politécnica"
 coords = "74,0,113,29,98,72,52,72,38,27"
 href = "/Perl/index.htm" alt = "Tutorial de Perl"
 objetivo = "_yo"
 el ratón por encima = "Tutorial de presentación('Perl')"
```

```
 onMouseOut = "Tutorial de presentación('')"/>

 <area forma = "Rect"
 coords = "22,83,126,125"
 href = "/Html/index.htm" alt = "Tutorial HTML"
 objetivo = "_yo"
 el ratón por encima = "Tutorial de presentación('Html')"
 onMouseOut = "Tutorial de presentación('')"/>

 <area forma = "circulo"
 coords = "73,168,32"
 href = "/Php/index.htm" alt = "Tutorial PHP"
 objetivo = "_yo"
 el ratón por encima = "Tutorial de presentación('Php')"
 onMouseOut = "Tutorial de presentación('')"/>
 </ Map>
 </ Body>
</ Html>
```

Salida

Se puede sentir el concepto de mapa colocando el cursor del ratón sobre el objeto de la imagen.

JavaScript - Navegadores de compatibilidad

Es importante entender las diferencias entre los diferentes navegadores con el fin de manejar cada uno en la forma en que se espera. Por lo tanto, es importante saber qué navegador su página web se ejecuta en.

Para obtener información sobre el navegador de su página web se está ejecutando actualmente, utilice la incorporada en el navegador de objetos.

Propiedades del guía

Hay varias propiedades relacionadas Navigator que se pueden utilizar en su página Web. La siguiente es una lista de los nombres y descripciones de cada uno.

No Señor.	Descripción de propiedad
1	**appCodeName** Esta propiedad es una cadena que contiene el nombre de código del navegador, Netscape para Netscape y Microsoft Internet Explorer para Internet Explorer.
2	**version de aplicacion** Esta propiedad es una cadena que contiene la versión del navegador, así como

	otra información útil, como su lengua y su compatibilidad.
3	**idioma** Esta propiedad contiene la abreviatura de dos letras para el idioma que se utiliza por el navegador. Netscape solamente.
4	**mimTypes []** Esta propiedad es una matriz que contiene todos los tipos MIME soportados por el cliente. Netscape solamente.
5	**plataforma[]** Esta propiedad es una cadena que contiene la plataforma para la que se compiló el navegador. "Win32" para los sistemas operativos Windows de 32 bits
6	**plugins []** Esta propiedad es una matriz que contiene todos los plug-ins que se han instalado en el cliente. Netscape solamente.
7	**agente de usuario[]** Esta propiedad es una cadena que contiene el nombre de código y versión del navegador. Este valor se envía al servidor de origen para identificar al cliente.

Métodos Navigator

Hay varios métodos Navigator-específicos. Aquí está una lista de sus nombres y descripciones.

No Señor.	Descripción
1	**javaEnabled ()** Este método determina si JavaScript está habilitado en el cliente. Si JavaScript está activado, este método devuelve true; de lo contrario, devuelve falso.
2	**plugings.refresh** Este método hace que acaba de instalar plugins disponibles y llena la matriz con todos los plugins nuevos nombres enchufables. Netscape solamente.
3	**preferencia (nombre, valor)**

	Este método permite una escritura firmada para obtener y definir algunas preferencias de Netscape. Si se omite el segundo parámetro, este método devolverá el valor de la preferencia especificada; de lo contrario, se establece el valor. Netscape solamente.
4	**taintEnabled ()** Este método devuelve verdadero si adulteración de datos está habilitada; en caso contrario.

Detección del navegador

No es un simple JavaScript que se puede utilizar para averiguar el nombre de un navegador y luego en consecuencia una página HTML puede ser servido al usuario.

```
<Html>
 <Head>
 <Title>Ejemplo de detección del explorador</ Title>
 </ Head>

 <Body>
 <script tipo = "Text / javascript">
 <! -
 var agente de usuario = navegador.agente de usuario;
 var ópera = (agente de usuario.índice de('Ópera') ! = -1);
 var es decir = (agente de usuario.índice de('MSIE') ! = -1);
 var geco = (agente de usuario.índice de('Geco') ! = -1);
 var Netscape = (agente de usuario.índice de('Mozilla') ! = -
1);
 var versión = navegador.version de aplicacion;

 Si (ópera) {
 documento.escribir("Basado en navegador Opera");
 // Mantenga su dirección URL específica de opera aquí.
 } más Si (geco) {
 documento.escribir("Navegador basado en Mozilla");
 // Mantenga su dirección URL específica gecko aquí.
 } más Si (es decir) {
 documento.escribir("Basado en navegador IE");
 // Mantenga su dirección URL específica IE aquí.
 } más Si (Netscape) {
 documento.escribir("Basado en navegador Netscape");
 // Mantenga su dirección URL específica Netscape aquí.
 } más {
 documento.escribir("Navegador Desconocido");
 }

 // Puede incluir la versión a lo largo de con cualquier
condición anterior.
```

```
 documento.escribir("<br /> información de la versión del
navegador:" + versión );
 // ->
 </ Script>
 </ Body>
</ Html>
```