Universidad Autónoma de Baja California

Aplicaciones web

JavaScript

Profesor: MC. Itzel Barriba Cázares

JavaScript

- Es un lenguaje de programación liviano, a menudo denominado lenguaje de secuencias de comandos.
- JavaScript brinda a los desarrolladores web un lenguaje de programación para usar en páginas web y les permite realizar tareas como las siguientes:
  - Leer elementos de documentos y escribir nuevos elementos y texto en documentos.
  - Manipular y mover texto
  - Crear ventanas emergentes
  - Realizar cálculos matemáticos sobre datos
  - Reaccionar a eventos, como cuando un usuarios pasa sobre una imagen o hace clic en un ratón.
  - Recuperar la fecha y hora actuales de la computadora de un usuarios o la ultima vez que se modificó un documento
  - Determinar el tamaño de pantalla del usuario, la versión del navegador o la resolución de pantalla
  - Realizar acciones basadas en condiciones, como alertar a los usuarios si ingresan información incorrecta en un formulario o si presionan un botón determinado.

¿De que se trata la programación?

- La programación consiste en gran medida en realizar cálculos a partir de datos. Ejemplos de tareas que se pueden realizar:
  - Cálculos matemáticos sobre números como suma, resta, multiplicación y división
  - Comprobar si un valor coincide con otro (si un usuario ingresa algún texto especifico o un número)
  - Encontrar una subsección de texto, como la tercera y cuarta letra de una palabra o la primera y ultima palabra de una oración.
  - Comprobar la longitud de un fragmento de texto o donde se encuentra la primera aparición de la letra “t” dentro de una sección de texto.
  - Comprobar si dos valores son diferentes, o si uno es más largo o más corto que el otro
  - Realizar diferentes acciones en función de si se cumple una condición (o una de varias condiciones).
  - Repetir una acción un cierto número de veces o hasta que se cumpla una condición

JavaScript

- Hay tres lugares donde puedes colocar tus JavaScripts:
  - En el <head> de la pagina: estos scripts se llamaran cuando un evento los active.
  - En la sección <body>: Estos scripts se ejecutaran a medida que se carga la pagina.
  - En un archivo externo: si el enlace se coloca dentro del elemento <head>, el script se trata de la misma manera que cuando el script se encuentra dentro del encabezado del documento esperando que un evento lo active, mientras que si se coloca en el <body> actuara como un script en la sección del body y se ejecutara a medida que se carga la pagina.

Comentarios

- Puede agregar comentarios al código JavaScript de dos maneras:

- Comentario de una sola linea:

```
// el comentario va aquí
```

- También puedes comentar varias líneas usando la siguiente sintaxis, manteniendo el comentario dentro un par de caracteres de apertura /* y de cierre*/

```
/* Toda esta sección está comentada por lo que no se trata
como parte del script. */
```

Declaración de Variables

- Las variables se utilizan para almacenar datos.

```
var nombreUsuario = "Bob Stewart"
```

- El nombre de la variable es nombreUsuario y el valor es Bob Stewart. Si no da ningún valor, entonces su valor no esta definido.
- Cuando usas una variable por primera vez, la estás creando.

Declaración de Variables

- El proceso de creación de una variable se conoce como declaración de la variable.
  - Un nombre de una variable se pueden incluir solo las letras a-z, A-Z, 0-9, el símbolo $ y el guion bajo _
  - No se permiten otros caracteres, como espacios o signos de puntuación.
  - El primer carácter puede ser solo una letra, $, o guion bajo (sin números).
  - Los nombres de las variables distinguen entre mayúsculas y minúsculas.
  - Evite dar el mismo nombre a dos variables dentro del mismo documento, ya que una podría anular el valor de la otra, creando un error.

Constante

- Una constante es una variable en la que el valor asociado no cambia.
- Un ejemplo de una constante sería un programa de cálculo de intereses para un préstamo bancario. Si bien el monto de un préstamo y la duración tendrían factores variables, la tasa de interés podría ser fija (es decir, constante) en un porcentaje determinado.
- Una constante no puedes declarar una variable sin valor.

Nombrar variables y constantes

- No hay limite establecido en la longitud de los nombres de las variables
- Intente utilizar nombres descriptivos para sus variables. Esto hace que su código sea más fácil de entender (y lo ayudará a depurarlo si hay algún problema con él).

Valid and Invalid Variable Names

Valid Example	Invalid Example	Explanation
`Number1Choice`	`#1Choice`	The invalid example begins with something other than a letter or underscore.
	`1Choice`	
`moonPhase`	`moon Phase`	The invalid example has a space between the words.
`_switch`	`switch`	The invalid example is a reserved word. The valid one, even though technically legal, isn't a good idea either, because it's just too close to the reserved word for comfort. A simple typographical error could easily change `_switch` into `switch`.

Palabras reservadas

- Restricciones para nombrar las variables:
  - No utilizar palabras reservada.
  - No contener espacios.
  - No puede iniciar el nombre de una variable con algo que no sea una letra o guion bajo (_).

Reserved Word	Reserved Word	Reserved Word
abstract	final	protected
as	finally	public
boolean	float	return
break	for	short
byte	function	static
case	goto	super
catch	if	switch
char	implements	synchronized
class	import	this
continue	in	throw
const	instanceof	throws
debugger	int	transient
default	interface	true
delete	is	try
do	long	typeof
double	namespace	use
else	native	var
enum	new	void
export	null	volatile
extends	package	while
false	private	with

Tipos de valores

- Un valor, ya sea literal o contenido en una variable, puede ser de varios tipos.
- Los tres más utilizados son:
  - Numéricos: números, como 8 o 37
  - Cadena: caracteres alfabéticos o, caracteres alfanuméricos.
  - Booleano: valores lógicos (verdadero/falso).

Asignar un valor a una variable

- Cuando se quiere dar un valor a una variable, primero debes declarar la variable, darle un nombre después asignarle un valor, esto se puede hacerlo de dos maneras:
- Declarando y asignando el valor:

```
var nombreUsuario = "Robert Stewart"
```

- O separado:

```
var nombreUsuario
nombreUsuario = "Robert Stewart"
```

- Donde nombreUsuario es ahora el equivalente de Robert Stewart.

Vida útil de una variable

- Cuando declaras una variable en una función, solo se puede acceder a ella en esa función. Una vez ejecutada la función, no podrá volver a llamar a la variable. Las variables en funciones se llaman variables locales.
- Una variable local funciona sólo dentro de una función, puede tener diferentes funciones que contengan variables con el mismo nombre (cada una es reconocida sólo por esa función).
- Si declara una variable fuera de una función, todas las funciones de su página pueden acceder a ella. La vida útil de estas variables comienza cuando se declaran y finaliza cuando se cierra la página.
- Las variables locales ocupan menos memoria y recursos que las variables a nivel de página porque solo requieren memoria durante el tiempo que se ejecuta la función, en lugar de tener que crearse y recordarse durante toda la vida de la página.

Strings

- Las variables de cadena (Strings) deben incluirse entre comillas simples o dobles:

```
greeting = "Hello there"
warning  = 'Be careful'
```

Variables numéricas

- Crear una variable numérica es tan sencillo como asignar un valor:

$$\text{count} = 42$$
$$\text{temperature} = 98.4$$

- Al igual que las cadenas, las variables numéricas se pueden leer y utilizar en expresiones y funciones.

Operadores

- El operador en sí es una palabra clave o símbolo que realiza una operación cuando se usa en una expresión.
- El operador se utiliza en una expresión con uno o dos valores y realiza un cálculo sobre los valores para generar un resultado.
- Una expresión es como una expresión matemática. Los valores se conocen como operandos. Los operadores que requieren solo un operando (o valor) a veces se denominan operadores unarios, mientras que los que requieren dos valores a veces se denominan operadores binarios.

Operadores

- Los diferentes tipos de operadores que verás en esta sección son:
  - Operadores aritméticos
  - Operadores de asignación
  - Operadores de comparación
  - Operadores lógicos
  - Operadores de cadena

Operadores aritméticos

- Los operadores aritméticos se utilizan para realizar operaciones matemáticas.

Operador	Descripción	Ejemplo
+	Adición	`j + 12`
-	Sustracción	`j - 22`
*	Multiplicación	`j * 7`
/	División	`j / 3.13`
%	Módulo (resto de la división)	`j % 6`
++	Incremento	`++j`
--	Decremento	`--j`

Operadores de asignación

- Los operadores de asignación se utilizan para asignar valores a variables.
- El operador de asignación se puede combinar con varios otros operadores para permitirle asignar un valor a una variable y realizar una operación en un solo paso.

Operador	Ejemplo	Equivalente a
=	j = 99	j = 99
+=	j += 2	j = j + 2
+=	j += 'string'	j = j + 'string'
-=	j -= 12	j = j - 12
*=	j *= 2	j = j * 2
/=	j /= 6	j = j / 6
%=	j %= 7	j = j % 7

Operadores de comparación

- Los operadores de comparación se utilizan generalmente dentro de un constructor, como en el caso de una sentencia en la que se deben comparar dos elementos y luego devuelve verdadero o falso según la comparación es verdadera o no.

Operador	Descripción	Ejemplo
==	Es *igual* a	`j == 42`
!=	*No es igual* a	`j != 17`
>	Es *mayor que*	`j > 0`
<	Es *menor que*	`j < 100`
>=	Es *mayor o igual* que	`j >= 23`
<=	Es *menor o igual* que	`j <= 13`
===	Es *igual* a (y del mismo tipo)	`j === 56`
!==	*No es igual* a (y del mismo tipo)	`j !== '1'`

Operadores booleanos o lógicos

- Los operadores lógicos o booleanos devuelven uno de dos valores: verdadero o falso.

Operador	Descripción	Ejemplo
`&&`	And	`j == 1 && k == 2`
`||`	Or	`j < 100 || j > 0`
`!`	Not	`! (j == k)`

Asignación creciente, decreciente y abreviada

- Las siguientes formas de post y pre-incremento y decremento:

```
++x
--y
x += 22
y -= 3
```

Concatenación de cadenas

- También puede agregar texto a cadenas usando el operador +.

```
name = "Bob"
lastName = "Stewart"
name = name + lastName
```

- El valor de la variable de name ahora seria Bob Stewart. El proceso de sumar dos cadenas se conoce como concatenación.

Declaraciones condicionales

- Las declaraciones condicionales le permiten realizar diferentes acciones dependiendo de diferentes declaraciones.
- Hay tres tipos de declaraciones condicionales que conocerá aquí:
  - sentencias if, que se utilizan cuando desea que el script se ejecute si una condición es verdadera
  - sentencias if...else, que se utilizan cuando se desea ejecutar un conjunto de código si se cumple una condición es verdadera y otra si es falsa
  - declaraciones switch, que se utilizan cuando desea seleccionar un bloque de código entre muchos dependiendo de una situación

If statements

- Las declaraciones if permiten que se ejecute código cuando la condición especificada es verdadera; Si la condición es verdadera, se ejecuta el código entre llaves.
- Aquí está la sintaxis de una declaración

```
if (condition) {
        //code to be executed if condition is true
}
```

- Si está ejecutando solo una declaración, las llaves no son estrictamente necesarias

If…else Statements

- Cuando tienes dos situaciones posibles y quieres reaccionar de manera diferente para cada una, puedes usar una declaración if...else.
- Esto significa: “Si se cumplen las condiciones especificadas, ejecute el primer bloque de código; de lo contrario, ejecute el segundo bloque”.
- La sintaxis es la siguiente:

```
if (condition) {
    //code to be executed if condition is true
}
else {
    //code to be executed if condition is false
}
```

- existen muchas posibilidades para utilizar declaraciones condicionales

Switch statement

- Una declaración de switch le permite lidiar con varios resultados de una condición. Tienes una única expresión, que suele ser una variable. Esto se evalúa inmediatamente. Luego, el valor de la expresión se compara con los valores de cada caso en la estructura. Si hay una coincidencia, se ejecutará el bloque de código.
- Aquí está la sintaxis de una declaración de switch:

```
switch (expression) {
    case option1:
        //code to be executed if expression is what is written in option1
        break;
    case option2:
        //code to be executed if expression is what is written in option2
        break;
    case option3:
        //code to be executed if expression is what is written in option3
        break;
    default:
        //code to be executed if expression is different from option1, option2,and option3
    }
```

- Utilice break para evitar que el código se ejecute automáticamente.

Operador condiciónal (o ternario)

- Un operador condicional (también conocido como operador ternario) asigna un valor a una variable según una condición:

```
nombreVariable = (condición)? valor1: valor2
```

Bucles

- Las sentencias de bucle se utilizan para ejecutar el mismo bloque de código un número específico de veces:
  - Un bucle while ejecuta el mismo bloque de código mientras o hasta que una condición sea verdadera.
  - Se ejecuta un bucle do while antes de comprobar la condición. Si la condición es verdadera, continuará ejecutándose hasta que la condición sea falsa. (La diferencia entre los bucles do y do while es que do while se ejecuta una vez independientemente de que se cumpla o no la condición).
  - Un bucle for ejecuta el mismo bloque de código un número específico de veces.

While

- En un bucle while, se ejecuta un bloque de código si una condición es verdadera y mientras esa condición siga siendo verdadera. La sintaxis es la siguiente:

```
while (condition) {
        //code to be executed
    }
```

Do…while

- Un bucle do... while ejecuta un bloque de código una vez y luego verifica una condición. Mientras la condición sea verdadera, el bucle continúa. Entonces, cualquiera que sea la condición, el bucle se ejecuta al menos una vez (como puede ver, la condición está después de las instrucciones).
- Aquí está la sintaxis:

```
do {
    //code to be executed
}
while (condition)
```

for

- La declaración for ejecuta un bloque de código un número específico de veces; lo usa cuando sabe cuántas veces desea que se ejecute el código (en lugar de ejecutarlo mientras una condición particular es verdadera/falsa). Primero, aquí está la sintaxis:

```
for (a; b; c)
{
    //code to be executed
}
```

- Ahora necesitas ver qué representan a, b y c:
- a se evalúa antes de ejecutar el ciclo y sólo se evalúa una vez. Es ideal para asignar un valor a una variable; por ejemplo, podrías usarlo para establecer un contador en 0 usando i=0.
- b debería ser una condición que indique si el bucle debe ejecutarse nuevamente; si devuelve verdadero, el ciclo se ejecuta nuevamente. Por ejemplo, podrías utilizar esto para comprobar si el contador es inferior a 11.
- c se evalúa después de que se haya ejecutado el bucle y puede contener múltiples expresiones separadas por una coma (por ejemplo, i++, j++;). Por ejemplo, podrías usarlo para incrementar el contador.

Bucles infinitos y break

- Tenga en cuenta que, si tiene una expresión que siempre se evalúa como verdadera en cualquier bucle, terminará con algo conocido como bucle infinito. Estos pueden consumir recursos del sistema e incluso bloquear la computadora, aunque algunos navegadores intentan detectar bucles infinitos y generalmente los detienen.

```
for (i=0; /* no condition here */ ; i++)
    {
        document.write(i + " x 3 = " + (i * 3) + "<br />" );
        if (i == 100) {
            break;
        }
    }
```

- Cuando el script llega a una declaración de interrupción, simplemente deja de ejecutarse. Esto evita efectivamente que un bucle se ejecute demasiadas veces.

Eventos

- Un evento ocurre cuando algo sucede.
- Hay dos tipos de eventos que se pueden utilizar para activar scripts:
  - ❑ Eventos de ventana, que ocurren cuando algo le sucede a una ventana. Por ejemplo, una página se carga o descarga (se reemplaza por otra página o se cierra) o el foco se mueve hacia o desde una ventana o marco.
  - ❑ Eventos de usuario, que ocurren cuando el usuario interactúa con elementos de la página usando un mouse (u otro dispositivo señalador) o un teclado.

Eventos

Event	Purpose	Applies To
onload	Document has finished loading (if used in a frameset, all frames have finished loading).	<body> <frameset>
onunload	Document is unloaded, or removed, from a window or frameset.	<body> <frameset>
onclick	Button on mouse (or other pointing device) has been clicked over the element.	Most elements
ondblclick	Button on mouse (or other pointing device) has been double-clicked over the element.	Most elements
onmousedown	Button on mouse (or other pointing device) has been depressed (but not released) over the element.	Most elements
onmouseup	Button on mouse (or other pointing device) has been released over the element.	Most elements
onmouseover	Button on mouse (or other pointing device) has been moved onto the element.	Most elements
onmousemove	Button on mouse (or other pointing device) has been moved while over the element.	Most elements
onmouseout	Button on mouse (or other pointing device) has been moved off the element.	Most elements
onkeypress	A key is pressed and released over the element.	Most elements
onkeydown	A key is held down over an element.	Most elements
onkeyup	A key is released over an element.	Most elements

Eventos

Event	Purpose	Applies To
onfocus	Element receives focus either by mouse (or other pointing device) clicking it, tabbing order giving focus to that element, or code giving focus to the element.	<a> <area> <button> <input> <label> <select> <textarea>
onblur	Element loses focus.	<a> <area> <button> <input> <label> <select> <textarea>
onsubmit	A form is submitted.	<form>
onreset	A form is reset.	<form>
onselect	User selects some text in a text field.	<input> <textarea>
onchange	A control loses input focus and its value has been changed since gaining focus.	<input> <select> <textarea>

Funciones

- Una función es un código que se ejecuta cuando se activa un evento o se realiza una llamada a esa función y, por lo general, una función contiene varias líneas de código.
- Las funciones están escritas en el elemento <head> y se pueden reutilizar en varios lugares dentro de la página, o en un archivo externo que está vinculado desde dentro del elemento <head>.

Definir una función

- Hay tres partes para crear o definir una función:
  - ❑ Defina un nombre para ello.
  - ❑ Indique cualquier valor que pueda ser necesario como argumento.
  - ❑ Agregue declaraciones.

```
function calculateArea(width, height) {
      area = width * height
      return area
  }
```

- Si una función no tiene argumentos, aún así debería tener paréntesis después de su nombre

Como llamar a una función

- Las funciones no hacen nada por si sola en el encabezado de un documento, hay que llamarla.

Crear un JavaScript externo

- Crear un archivo externo para escribir texto en la pagina deberá seguir los pasos:
  1. Abra un archivo y escriba el código en JavaScript
  2. Guarde el archivo nombreArchivo.js
  3. Debes incluir o importar el script dentro del body.
  4. Guarde el archivo y Abralo en el navegador.

Crear un JavaScript externo

- Al igual que las reglas CSS, JavaScript puede incrustarse en una página o colocarse en un archivo de script externo.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>JavaScript App</title>
        <link href="style.css" rel="stylesheet" type="text/css">
    </head>
    <body>
        <h1>Primer prueba con JavaScript</h1>
        <p>App Description</p>
        <script src="http://code.jquery.com/jquery-2.0.3.min.js"></script>
        <script src="js/app.js"></script>
    </body>
</html>
```

- Agrega scripts a su página dentro del elemento <script>. El atributo de tipo en la etiqueta <script> de apertura indica qué lenguaje de script se encontrará dentro del elemento.

Script

```
var main = function () {
    "use strict";

    $(".comment-input button").on("click",
function (event) {
        console.log("Hello World!");
    });
};
$(document).ready(main);
```

DOM

- El DOM conecta páginas web con scripts o lenguajes de programación representa la estructura de un documento (HTML) en la memoria.
- JavaScript por si solo no hace mucho más que permitirle realizar cálculos o trabajar con cadenas básicas. Para que un documento sea más interactivo, el script debe poder acceder al contenido del documentos y saber cuando el usuario está interactuando con él.
- Lo hace interactuando con el navegador mediante el uso de propiedades y métodos establecidos en una interfaz de programación de aplicaciones llamadas modelo de objeto de documento (DOM)
- El modelo de objetos de documento explica qué propiedades de un documento puede recuperar un script y cuáles puede modificar; también puede definir algunos métodos que se pueden llamar para realizar una acción en el documento.

DOM

- DOM representa un documento con un árbol lógico. Cada rama del árbol termina en un nodo y cada nodo contiene objetos. Los métodos DOM permiten el acceso programático del árbol. Con ellos puedes cambiar la estructura, el estilo o el contenido del documento.
- Los nodos también pueden tener controladores de eventos adjuntos. Una vez que se activa un evento, los controladores de eventos se ejecutan.

DOM

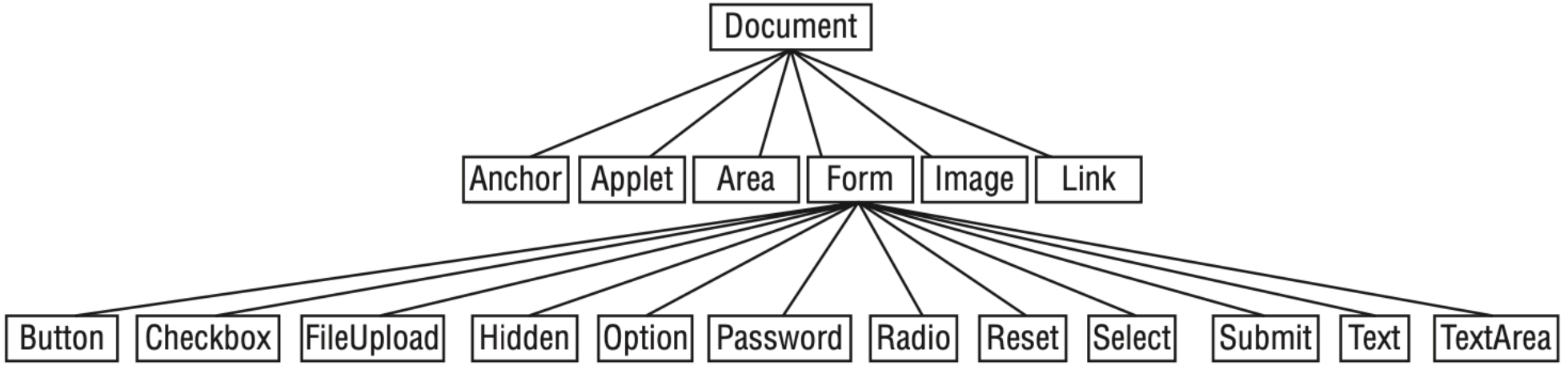

Figure 11-3

- La estructura muestra como los elementos de una pagina están disponibles en scripts como objetos programables.

DOM

- DOM es un modelo de objetos estándar y una interface de programación para HTML. Define:
  - Los elementos HTML como objetos.
  - Las propiedades de todos los elementos HTML.
  - Los métodos para acceder a todos los elementos HTML.
  - Los eventos para todos los elementos HTML.

DOM

- Entonces, el DOM especifica cómo se puede usar JavaScript (y otros lenguajes de programación) para acceder a la información sobre un documento.
- Luego poder realizar cálculos y decisiones basados en los valores recuperados del documento utilizando JavaScript.
- Sus script pueden incluso invocar métodos y cambiar algunas propiedades del documento. También puede utilizar eventos para activas ciertos scripts.

Objetos, Métodos y propiedades

- Cada objeto puede tener propiedades y métodos:
  - Una propiedad te dice algo sobre el objeto
  - Un método realiza una acción

Propiedades del objeto Document

- Muchas propiedades se pueden configurar (set) y leer. Si se puede establecer una propiedad, se conoce como propiedad de lectura/escritura (porque puede leerla o escribir en ella), mientras que las que sólo puede leer se conocen como de sólo lectura.

Propiedades del objeto Document

Property Name	Purpose	Read/Write
`alinkColor`	Specifies link colors. (Like the deprecated `alink` attribute on the `<body>` element.)	Read/write
`bgcolor`	Specifies background color. (Like the deprecated `bgcolor` attribute on the `<body>` element.)	Read/write
`fgcolor`	Foreground/text color. (Like the deprecated `text` attribute of the `<body>` element.)	Read/write
`lastModified`	The date the document was last modified. (This is usually sent by the web server in things known as HTTP headers that you do not see.)	Read only
`linkColor`	Specifies link colors. (Like the deprecated `link` attribute of the `<body>` element.)	Read/write

Propiedades del objeto Document

Property Name	Purpose	Read/Write
`alinkColor`	Specifies link colors. (Like the deprecated `alink` attribute on the `<body>` element.)	Read/write
`bgcolor`	Specifies background color. (Like the deprecated `bgcolor` attribute on the `<body>` element.)	Read/write
`fgcolor`	Foreground/text color. (Like the deprecated `text` attribute of the `<body>` element.)	Read/write
`lastModified`	The date the document was last modified. (This is usually sent by the web server in things known as HTTP headers that you do not see.)	Read only
`linkColor`	Specifies link colors. (Like the deprecated `link` attribute of the `<body>` element.)	Read/write
`referrer`	The URL of the XHTML page that users came from if they clicked a link. Empty if there is no referrer.	Read only
`title`	The `title` of the page in the `<title>` element.	Read only (until IE 5 and Netscape 6 and later versions)
`vlinkColor`	The `vlink` attribute of the `<body>` element (deprecated).	Read/write

Métodos del objeto Document

- Los métodos realizan acciones y siempre se escriben seguidos de un par de paréntesis. Dentro de los paréntesis, recibe parámetros o argumentos que pueden afectar la acción que realiza el método.

Method Name	Purpose
`write(string)`	Allows you to add text or elements into a document
`writeln(string)`	The same as `write()` but adds a new line at the end of the output (as if you had pressed the Enter key after you had finished what you were writing).

Formularios

- La colección de formularios contiene referencias correspondiente a cada uno de los elementos <form> de la página.
- Tener en cuenta si se tiene mas de un formulario: un formulario para inicio de sesión, otro para registro de nuevos usuarios y un cuadro de búsqueda en el mismo. En este caso se debe distinguir entre los diferentes formularios.
- Entonces, si el formulario de inicio de sesión es el primer formulario del documento y desea acceder a la propiedad de acción del formulario de inicio de sesión, puede usar el siguiente número de índice para seleccione el formulario apropiado y acceda a sus propiedades y métodos (recuerde que los números de índice comienzan en 0 para el primer formulario, 1 para el segundo formulario, 2 para el tercero, etc.):
- documento.formularios[0].acción

Formularios

- La colección de formularios contiene referencias correspondiente a cada uno de los elementos <form> de la página.

Properties of the Form Objects

The following table lists the properties of the form objects.

Property Name	Purpose	Read/Write
`action`	The `action` attribute of the `<form>` element	Read/write
`length`	Gives the number of form controls in the form	Read only
`method`	The `method` attribute of the `<form>` element	Read/write
`name`	The `name` attribute of the `<form>` element	Read only
`target`	The `target` attribute of the `<form>` element	Read/write

Formularios

Methods of the Form Objects

The following table lists the methods of the form objects.

Method Name	Purpose
`reset()`	Resets all form elements to their default values
`submit()`	Submits the form

Elementos de formulario

- Cuando accede a un formulario, normalmente desea acceder a uno o más de sus elementos. Cada elemento <form> tiene un objeto de colección elements[] como propiedad, que representa todos los elementos de ese formulario.
- Algunas de las cosas que quizás quieras hacer con los elementos de un formulario:
  - Campos de texto: lea los datos que un usuario ha ingresado o escriba texto nuevo en estos elementos.
  - Casillas de verificación y botones de radio: Pruebe si están marcados y márquelos o desmárquelos.
  - Botones: deshabilitarlos hasta que un usuario haya seleccionado una opción.
  - Seleccionar casillas: seleccione una opción o vea qué opción ha seleccionado el usuario.

Propiedades de los elementos

Properties of Form Elements

The following table lists the properties of form elements.

Property	Applies to	Purpose	Read/Write
`checked`	Checkboxes and radio buttons	Returns `true` when checked or `false` when not	Read/write
`disabled`	All except hidden elements	Returns `true` when disabled and user cannot interact with it (supported in IE4 and Netscape 6 and later versions only)	Read/write
`form`	All elements	Returns a reference to the form it is part of	Read only
`length`	Select boxes	Number of options in the `<select>` element	Read only
`name`	All elements	Accesses the name attribute of the element	Read only
`selectedIndex`	Select boxes	Returns the index number of the currently selected item	Read/write
`type`	All	Returns type of form control	Read only
`value`	All	Accesses the value attribute of the element or content of a text input	Read/write

Métodos de los elementos

Methods of Form Elements

The following table lists the methods of form elements.

Property Name	Applies to	Read/Write
`blur()`	All except hidden	Takes focus away from currently active element to next in tabbing order
`click()`	All except text	Simulates the user's clicking the mouse over the element
`focus()`	All except hidden	Gives focus to the element
`select()`	Text elements except hidden	Selects the text in the element