CAPÍTULO 5

JavaScript

5.1. Introducción

JavaScript es un lenguaje de programación que permite el script de eventos, clases y acciones para el desarrollo de aplicaciones Internet entre el cliente y el usuario. *JavaScript* permite con nuevos elementos dinámicos ir más allá de clicar y esperar en una página Web. Los usuarios no leerán únicamente las páginas sino que además las páginas ahora adquieren un carácter interactivo. Esta interacción permite cambiar las páginas dentro de una aplicación: poner botones, cuadros de texto, código para hacer una calculadora, un editor de texto, un juego, o cualquier otra cosa que pueda imaginarse.

Los navegadores interpretan las sentencias de *JavaScript* incluidas directamente en una página HTML, permitiendo la creación de aplicaciones similares a los CGI.

Aún no hay definición clara del *scripting language* ("lenguaje interpretado de comandos"). A veces el término se usa para distinguir este tipo de lenguaje de los lenguajes compilados como el *C++*. Quizá, algunos lenguajes como el *C* o *C++* puedan ser usados para *scripts* de aplicaciones. *JavaScript* es en muchos aspectos un lenguaje de programación parecido al *C* o *C++*.

Como otros lenguajes *script*, *JavaScript* extiende las capacidades de la aplicación con la que trabajan, así *JavaScript* extiende la página Web más allá de su uso normal. Hay numerosas maneras de dar vida al Web y dar flexibilidad al lenguaje. El único límite es la imaginación.

5.1.1. Propiedades del Lenguaje JavaScript

Las propiedades más importantes de *JavaScript* son las siguientes:

- Se interpreta por el ordenador que recibe el programa, no se compila.
- Tiene una programación orientada a objetos. El código de los objetos está predefinido y es expandible. No usa clases ni herencia.
- El código está integrado (incluido) en los documentos HTML.
- Trabaja con los elementos del HTML.
- No se declaran los tipos de variables.
- Ejecución dinámica: los programas y funciones no se chequean hasta que se ejecutan.
- Los programas de *JavaScript* se ejecutan cuando sucede algo, a ese algo se le llama evento.

5.1.2. El lenguaje JavaScript

JavaScript está basado en un modelo orientado al WWW. Elementos de una página como un botón o un cuadro de selección, pueden causar un evento que ejecutará una acción. Cuando ocurre alguno de estos eventos se ejecuta una función en *JavaScript*. Esta función está compuesta de varias sentencias que examinan o modifican el contenido de la página Web, o hacen otras tareas para dar respuesta de algún modo al evento.

Por lo general, los comandos de un programa en *JavaScript* se dividen en 5 categorías:

- Variables y sus valores.
- Expresiones, que manipulan los valores de las variables.
- Estructuras de control, que modifican cómo las sentencias son ejecutadas.
- Funciones, que ejecutan un bloque de sentencias

- Clases y arrays (vectores), que son maneras de agrupar datos.

A continuación se presenta una breve introducción sobre estas categorías.

5.1.3. Variables y valores

En *JavaScript* , a diferencia de la mayoría de los lenguajes de programación, no se debe especificar el tipo de datos. No hay manera de especificar que una variable representa un entero, una cadena de caracteres, un número con decimales (que se escriben con punto y no con coma), o un valor lógico booleano. De hecho, la misma variable puede ser interpretada de diferentes modos en diferentes momentos.

Todas las variables se declaran usando el comando ***var***. Una variable puede ser inicializada cuando se da un valor al ser declarada, o puede no ser inicializada. Además, varias variables pueden ser declaradas a la vez separadas por comas.

Ejemplo 1:

```
var variable1= "coche"
var cuaderno
var mi_variable = 123456, decimal =2342.89
var n_casas, n_habitaciones, n_cuadros, nombre = "Franklin"
```

5.1.4. Sentencias, Expresiones y Operadores

Como en la mayoría de los lenguajes de programación, la unidad básica de trabajo en *JavaScript* es la **sentencia.** Una sentencia de *JavaScript* hace que algo sea evaluado. Esto puede ser el resultado de dar valor a una variable, llamar a una función, etc. Cualquiera de las líneas del ejemplo 1 es una sentencia.

Los programas de *JavaScript* son un grupo de sentencias, normalmente organizadas en funciones que manipulan las variables y el entorno HTML en el cual el *script* trabaja.

Los operadores hacen que en una sentencia las variables sean evaluadas y se les asigne un valor o un resultado. Los operadores pueden actuar de distinto modo en diferentes situaciones. Algunos operadores de *JavaScript* pueden ser sobrecargados, es decir, pueden tener diversas interpretaciones según su modo de uso.

No hay ningún carácter especial o signo que marque el final de una sentencia en un programa. Por defecto se considera que una sentencia ha acabado cuando se llega al final de la línea, aunque se puede especificar el fin con el carácter punto y coma (;). Ésto hace posible poner varias sentencias en una sola línea separadas entre sí por un punto y coma.

Las dos siguientes sentencias realizan la misma operación, para *JavaScript* no tienen ninguna diferencia

Ejemplo 2:

```
var variable1= "coche"
var variable1= "coche";
```

y estos dos grupos de sentencias también:

Ejemplo 3:

```
var a = 0; a = 2+4; var c = a / 3 (;)
var a = 0
a = 2+4
var c = a / 3
```

Los segunda sentencia es una expresión.

5.1.5. Estructuras de Control.

Con lo explicado, aún no es posible escribir código para un programa completo; hay que conocer construcciones de nivel más elevado. Existen varios métodos para controlar el modo de ejecución de sentencias que se verán más adelante.

5.1.6. Funciones y Objetos

Las sentencias, expresiones y operadores básicos se agrupan en bloques más complejos dentro de un mismo programa llamadas funciones. El control de estructuras representa el siguiente nivel de organización de *JavaScript*. Las funciones y los objetos representan el nivel más alto de organización del lenguaje.

5.1.6.1. Funciones

Una función es un bloque de código con un nombre. Cada vez que se usa el nombre, se llama a la función y el código de la función es ejecutado. Las funciones pueden llamarse con valores, conocidos

como parámetros, que se usan en la función. Las funciones tienen dos objetivos: organización del programa (archivo o documento) y ejecución del código de la función. Al clicar con el ratón, apretar un botón, seleccionar texto y otras acciones pueden llamar a funciones.

El nombre de una función se escribe inmediatamente después del comando ***function***. Todos los nombres de funciones deben ser únicos y diferentes de los nombres de los comandos que usa *JavaScript*. No puede haber dos funciones con el mismo nombre. La lista de parámetros de una función se separa por comas. La función usa esos parámetros en las sentencias de su cuerpo que la configuran. Los argumentos que se le pasan a una función no pueden ser cambiados en su interior.

Ejemplo 4:

```
function valor_abs (num){
    if (num >= 0)
          return num
    else
          return -num
}
```

num es el argumento que se utiliza dentro de la función. El código de la función va entre llaves.

5.1.6.2. Objetos

Las funciones se usan para organizar el código. Los objetos tienen el mismo propósito pero con datos. Los tipos de datos conocidos hasta ahora son variables declaradas o inicializadas con **var.** Cada uno de estos tipos puede tener un solo valor. Los objetos permiten la capacidad de tener varios valores, de tal manera que un grupo de datos pueda estar relacionado con otro.

Lo que en *JavaScript* se llama objeto en otros lenguajes se llama estructura de datos o clase. Como las funciones, los objetos tienen 2 aspectos: cómo se crean y cómo se usan.

Al usar *JavaScript*, tenemos predefinidos una serie de objetos. Un objeto de *JavaScript* es un conjunto de componentes, llamados propiedades o miembros. Si se supone que se tiene un objeto llamado *cita* para organizar citas, éste tendrá las propiedades día, hora, con quién y para qué.

A cada una de estas propiedades del objeto se hace referencia con el operador punto (.).

Así, para referirse al mes de la cita se usa una_cita.mes mientras que una_cita.con_quien contendrá el nombre de con quién nos hemos citado.

Cada objeto puede contener todas las variables que nos interesen. Puede contener funciones que realicen algún trabajo. Puede incluso contener otros objetos, de tal manera que se pueden organizar los datos de modo jerárquico.

Más adelante se verán ejemplos al entrar en detalle con los objetos.

5.1.7. La TAG «Script».

En el sentido más general, cada página Web está hecha con sentencias de HTML que dividen la página en dos partes: el HEAD y el BODY.

La sección HEAD de un documento HTML es la que debe contener el código de *JavaScript* para los gestores de eventos. Aunque no es necesario que todo el código de *JavaScript* vaya en el HEAD, es importante que vaya en él para asegurar que todo el código de *JavaScript* haya sido definido antes del BODY del documento. En particular, si el documento tiene código para ejecutar un evento, y este evento se acciona antes de que el código se lea, podría ocurrir un error por que la función está sin definir.

En un documento HTML, el código de *JavaScript* se introduce mediante la TAG SCRIPT. Todo lo que haya entre <SCRIPT> y </SCRIPT> se considera como un tipo de código script, como *JavaScript*. La sintaxis para la TAG SCRIPT es:

```
<SCRIPT LANGUAGE="Nombre del lenguaje"  SRC="URL">
```

El elemento *Nombre del lenguaje* da el lenguaje que se usa en el subsiguiente script.

El atributo SRC es necesario cuando se quiere hacer referencia a un fichero que contiene el código del script. Para *JavaScript*, el fichero suele tener extensión *.js*. Si se usa el atributo SRC la TAG <SCRIPT> es inmediatamente seguida por </SCRIPT>. Por ejemplo un bloque <SCRIPT> que carga un código *JavaScript* del fichero click.js en el directorio relativo al documento, se haría del siguiente modo:

Ejemplo 5

```
<SCRIPT LANGUAGE="text/javascript" SRC="click.js"></SCRIPT>
```

Si no se pone el atributo SRC, entonces entre las TAGs <SCRIPT> y </SCRIPT> se escribe el código en el lenguaje indicado en la primera TAG. La estructura general del código es:

```
<SCRIPT LANGUAGE="text/javascript">
   Sentencias
</SCRIPT>
```

El siguiente programa usa la función del ejemplo 4. Con el formulario, <FORM>, nos pide un número que evaluamos con la función y devolvemos su valor absoluto. Puede que haya cosas que no queden claras, pero todo se explicará con más detalle a posteriori. La función de estos ejemplos previos no es sino la de dar una visión general del lenguaje.

Ejemplo 6:

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="text/javascript">
function valor_abs(form){
    var num = eval(form.expr.value)
    if (num >= 0)
          form.result.value = num
    else
          num = -num
          form.result.value = num
}
</SCRIPT>
<FORM>
  <SCRIPT>
  document.write("Introduce un número:")  //  Salida por pantalla
  </SCRIPT>
  <INPUT TYPE="text" NAME="expr" SIZE=15 ><BR>
  <INPUT TYPE="button" VALUE="Calcular" onClick="valor_abs(this.form)"><BR>
  <P ALIGN=LEFT>
    <SCRIPT>document.write("Valor Absoluto:")</SCRIPT>
    <INPUT TYPE="text" NAME="result" SIZE=15 >
  </P>
</FORM>
</BODY>
</HTML>
```

5.2. Activación de JavaScript: Eventos (Events).

El número de cosas que se pueden hacer en una página HTML es bastante limitado. La mayoría de los usuarios simplemente leen un texto, miran gráficos y como mucho escuchan sonidos. Para muchos, la experiencia del Web consiste en visitar una serie de páginas sin interaccionar prácticamente con ellas. La única interacción ocurre cuando el usuario selecciona un link o clica un mapa de imagen.

Los formularios de HTML han cambiado gradualmente este modelo para incrementar el nivel de interacción. Un formulario tiene varios modos de aceptar entradas. El usuario rellena el formulario y lo envía. Es difícil saber si el formulario ha sido rellenado correctamente y el tiempo de proceso del formulario es normalmente bastante largo. En el caso del HTML, este proceso ocurre porque el contenido del formulario tiene que ser enviado a través de la red a algún fichero en el servidor, donde se procesa y entonces se da una respuesta al usuario. Incluso el más simple error causa el rechazo del formulario, y por lo tanto que deba repetirse el proceso.

Uno de los objetivos de *JavaScript* es localizar la mayoría de estos procesos y mejorarlos dentro del browser del usuario. *JavaScript* es capaz de asegurarse que un formulario se rellene y envíe correctamente; evitando que el usuario tenga que repetir el formulario a causa de algún error.

JavaScript realiza esto mediante los gestores de eventos. Estos son sentencias de *JavaScript*, normalmente funciones, que se llaman cada vez que algo ocurre. Las funciones de *JavaScript* pueden ser llamadas cuando se envía un formulario o cuando el usuario usa campos del formulario.

5.2.1. Eventos y acciones

Para entender el modelo de gestores de eventos de *JavaScript*, hay que pensar primero sobre las cosas que pueden ocurrir actualmente en una página Web. Aunque algunas cosas se pueden hacer con el

browser, la mayoría de estas no tienen que ver con la navegación en el Web (salvar una página como texto, imprimirla, editar un bookmark, etc.).

Para entender qué acciones del browser corresponden a los eventos de *JavaScript* y cuales no, es importante distinguir aquellas acciones que causan algún cambio en la página Web cuando se muestra. De hecho, realmente hay sólo dos tipos de acciones: las que permiten que el usuario pueda navegar o las que hacen posible que el usuario pueda interactuar con un elemento de un formulario HTML.

5.2.1.1. Acciones de Navegación y Eventos

En la categoría de navegación se pueden distinguir las siguientes acciones:

- Seleccionar un link de hipertexto
- Mover hacia adelante o hacia atrás en la lista de Webs visitados.
- Abrir otro fichero.
- Cerrar el browser

En la mayoría de estos casos la página activa se descarga, y otra nueva se carga y se muestra en la ventana del browser. Pero cualquier persona que ha usado la WWW sabe que al seleccionar un link de hipertexto no siempre se tiene éxito, puesto que la máquina puede estar desconectada o inaccesible. El link puede haber muerto. Seleccionando un link muerto se descarga la página activa, y no se carga una nueva.

Dependiendo del tipo de error y del browser usado puede perderse la página activa. Estos eventos, cargar y descargar una página, son los únicos eventos que pueden ser manejados por *JavaScript* a nivel de los documentos. Esto significa que es posible escribir código *JavaScript* contenido dentro de la definición de HTML de una página, que se ejecutará cada vez que la página sea cargada o descargada.

5.2.2. Gestores de Eventos (Event Handlers)

5.2.2.1. Declaración

En la introducción se ha dicho que las funciones de *JavaScript* sólo se ejecutan en respuesta a eventos. Se sabe que los eventos ocurren cuando se produce alguna interacción o cambio en la página Web activa.

Las declaraciones de los gestores de eventos es muy similar a los atributos de HTML. Cada nombre del atributo empieza con la palabra *on* y sigue con el nombre del evento, así por ejemplo *onClick* es el atributo que se usaría para declarar un gestor de eventos para el evento Click (clicar un objeto).

La declaración de un gestor de eventos es: onEvent="Código_JS".

Normalmente, por convenio, se escribe on en minúscula y el nombre del evento con la letra inicial en mayúscula. Esto ayuda a distinguir éste de los demás atributos.

Los tipos de eventos y gestores de eventos son los siguientes:

Evento	Ocurre Cuando	Gestor
blur	El usuario quita el cursor de un elemento de formulario	onBlur
click	El usuario clica un link o un elemento de formulario	onClick
change	El usuario cambia el valor de un texto, un área de texto o selecciona un elemento.	onChange
focus	El usuario coloca el cursor en un elemento de formulario.	onFocus
load	El usuario carga una página en el Navegador	onLoad
Mouseover	El usuario mueve el ratón sobre un link	onMouseOver
Select	El usuario selecciona un campo del elemento de un formulario	onSelect
Submit	Se envía un formulario	onSubmit
Unload	Se descarga la página	onUnload

El valor del atributo es un conjunto de código *JavaScript* o una referencia a una función de *JavaScript*. El código o la función se ejecuta al activar el evento.

Ejemplo 7:

```
<INPUT TYPE="button" NAME="mycheck" VALUE="HA!"
   onClick="alert('Te he dicho que no me aprietes')">
```

Esta sentencia crea un botón (INPUT TYPE="button"). Al clicar el botón, el gestor de eventos onClick despliega una ventana con el mensaje que se pasa como argumento.

Normalmente una página HTML con programación en *JavaScript* tiene los siguientes componentes:

- Funciones *JavaScript* dentro de un bloque Script dentro del <HEAD> del documento.
- HTML no interactivo dentro del <BODY> del documento
- HTML interactivo con atributos gestores de eventos cuyos valores son funciones de *JavaScript*.

En general, ya sabemos declarar gestores de eventos. Ahora se verá qué gestores de eventos pueden asociarse con los TAGs específicos de HTML.

Los eventos de *JavaScript* suceden en tres niveles: a nivel del documento Web, a nivel de un formulario individual dentro del documento y a nivel de un campo de formulario.

5.2.2.2. Uso

5.2.2.2.1. Gestores a nivel de documento

La TAG HTML BODY contiene la descripción del contenido de la página HTML. La TAG BODY puede contener dos declaraciones de gestores de eventos usando los atributos onLoad y onUnload. Una declaración podría ser:

Ejemplo 8:

```
<BODY onLoad="cargarfuncion()" onUnload="descargarfuncion()">
```

El atributo onLoad="cargarfuncion()" declara un gestor de *JavaScript* que manejará la carga. El evento load se genera después de que el contenido de la página entre <BODY> y </BODY> se haya leído pero antes de que se haya mostrado. El gestor de evento onLoad es un buen lugar para mostrar el nombre de la compañía o la información de copyright, una ventana de seguridad preguntando el password de autorización, etc.

El atributo onUnload="descargarfuncion()" declara un gestor de eventos que se llama cada vez que la página se descarga. Esto ocurre cuando se carga una página nueva en la misma ventana, si una página no se carga con éxito y la página activa está aun descargada. El gestor de eventos onUnload puede servir para asegurarse de que no se ha perdido contacto con la página, por ejemplo si un usuario ha rellenado un formulario pero se ha olvidado de mandarlo.

Eventos aplicados a las TAGs de HTML:

- FOCUS, BLUR, CHANGE: campos de texto, áreas de texto y selecciones.
- CLICK: botones, botones de tipo radio, cajas de chequeo, botón de envío, botones de reset y links.
- SELECT: campos de texto, áreas de texto, cuadrso de selección.
- MOUSEOVER: links.

El evento ***focus*** se genera cuando el item de texto de un elemento de la lista obtiene el foco, normalmente como resultado de clicar con el ratón. El evento ***blur*** se genera cuando un ítem pierde el foco. El evento ***change*** se genera cada vez que un ítem sufre algún cambio. En un ítem de texto esto resulta cuando se introduce nuevo texto o el que existía se borra. En una lista de selección ocurre cada vez que una nueva selección se hace, incluso en una lista que permite múltiples selecciones. El evento ***select*** se genera cuando el usuario selecciona algún texto o hace una selección en el cuadro de selección.

Estos eventos pueden usarse para obtener un buen control sobre el contenido de un texto o una lista de selección de items. Las aplicaciones más comunes usan el evento ***change*** o ***blur*** para asegurarse de que el texto tiene el valor apropiado.

El argumento/comando especial **this:** Este comando se usa para referirse al objeto activo. Cuando la función a la que se le pasa el argumento ***this*** es llamada, el parámetro que usa la función en su definición se introduce con el objeto sobre el que se actúa.

Si nos fijamos en el ejemplo 9, la función `cambiar()` se activa cuando el formulario , en este caso un cuadro de selección, pierde el foco. A la función se le pasa como argumento el formulario mediante el comando ***this***. `form.cap.selectedIndex` es el índice de la selección escogida del formulario `(form)` llamado `cap`.

5.2.2.2.2. Gestores a nivel de formulario

La TAG FORM se usa para comenzar la definición de un formulario HTML. Incluye atributos como el METHOD usado para elegir el modo de envío del formulario, la acción que se debe cumplir (ACTION) y el atributo onSubmit. La sintaxis es como la que sigue:

```
<FORM NAME="nombre_del_formulario" ... onSubmit="función_o_sentencia">
```

El gestor onSubmit es llamado cuando el contenido del formulario se envía. También es posible especificar una acción onClick en un botón de envío. El uso común del gestor onSubmit es verificar el contenido del formulario: el envío continúa si el contenido es válido y se cancela si no lo es.

5.2.2.2.3. Gestores a nivel de elementos de formulario

Casi todos los elementos de un formulario tienen uno o más gestores de eventos. Los botones pueden generar eventos click, el texto y la selección de elementos pueden generar los eventos ***focus***, ***blur***, ***select*** y ***change***.

Hay dos excepciones a la regla que todos los elementos de un formulario pueden tener gestores de eventos. La primera excepción se aplica a los items ocultos, <INPUT TYPE= "hidden">. No se ven, no se pueden cambiar y no pueden generar eventos. La segunda se aplica a los elementos individuales OPTION dentro de una lista de selección (que se crean con la opción SELECT). La TAG SELECT puede tener atributos declarando gestores de eventos (***focus***, ***blur*** y ***change***), pero las OPTION no pueden generar eventos.

Los campos de texto (text fields) de HTML, <INPUT> con el atributo TYPE de texto "text" pueden declarar gestores de eventos como combinación de los 4 elementos de texto: ***focus***, ***blur***, ***change*** y ***select***. Con la TAG TEXTAREA se crea la entrada de texto en varias líneas y pueden generarse estos gestores de eventos. En cambio la selección de listas creadas con <SELECT> pueden generar todos los eventos menos el ***select***.

Estos eventos pueden usarse para obtener un buen control sobre el contenido de un texto o una lista de selección de ítems. Las aplicaciones más comunes usan el evento ***change*** o ***blur*** para asegurarse de que el campo tiene el valor apropiado.

Ejemplo 9:

```
<HTML><HEAD>
  <TITLE>EJEMPLO DEL COMANDO this</TITLE>
  <SCRIPT LANGUAGE="text/javascript">
  function cambiar(form){
    var indice = form.cap.selectedIndex
    if(indice==0){var archivo="cap1.htm"}
    if(indice==1){var archivo="cap2.htm"}
    if(indice==2){var archivo="cap3.htm"}
    window.open(archivo,'capitulos')
    window.open('marcador.htm','resultados')
  }
  </SCRIPT>
</HEAD>
<BODY>
<CENTER><FORM>
  <SELECT NAME="cap" SIZE=1 onBlur="cambiar(this.form)">
    <OPTION VALUE=1>1. HISTORIA Y CONCEPTOS DE LA AP</OPTION>
    <OPTION VALUE=2>2. MÉTODOS EN PATOLOGÍA</OPTION>
    <OPTION VALUE=3>3. PATOLOGÍA MOLECULAR</OPTION>
  <SELECT>
</FORM></CENTER>
</BODY></HTML>
```

5.3. Clases en JavaScript

Las clases en *JavaScript* se pueden agrupar en tres categorías:

- Clases Predefinidas, incluyen las clases Math, String y Date.
- Clases del Browser, tienen que ver con la navegación.

- Clases del HTML, están asociadas con cualquier elemento de una página Web (link, formulario, etc).

5.3.1. Clases Predefinidas (Built-In Objects).

5.3.1.1. Clase String

Cada vez que se asigna un valor string (cadena de caracteres a una variable o propiedad, se crea un objeto de la clase string. Al asignar un string a una variable no se usa el operador new.

Los objetos string tienen una propiedad, length (número de carácteres de la cadena), y varios métodos que manipulan la apariencia de la cadena (color, tamaño, etc.).

Métodos sobre el contenido: (recordar que las string tienen como base de índices el cero.)

- *charAt* (indice), muestra el carácter que ocupa la posición indice en la cadena.
- *indexOf* (caracter), muestra el primer índice del carácter.
- *lastIndexOf* (caracter), muestra el último carácter del índice.
- *subString* (primerindice, ultimoindice), muestra la cadena que hay que hay entre el primer índice (primerindice) y el último índice (ultimoindice) incluídos.
- *toLowerCase*( ), muestra todos los carácteres de la cadena en minúsculas.
- *toUpperCase*( ), muestra todos los carácteres de la cadena en mayúsculas.

Suponiendo que la variable `cadena` es un objeto de la clase string, el uso de los métodos se realiza de la siguiente manera: `cadena.método( )`.

Métodos sobre la apariencia:

- *big* ( ), muestra las letras más grandes.
- *blink* ( ), muestra texto intermitente (parpadeando).
- *bold* ( ), muestra las letras en negrita.
- *fixed* ( ), muestra el texto en paso fijo (letra Courier New).
- *fontcolor* (color), cambia el color de las letras.
- *fontsize* (size), cambia el tamaño de las letras.
- *italics* ( ), muestra en letra itálica.
- *small* ( ), muestra las letras más pequeñas.
- *strike* ( ), muestra las letras tachadas por una ralla.
- *sub* ( ), muestra la letra en subíndice.
- *sup* ( ), muestra la letra en superíndice.

Ejemplo 10:

```
var cadena = "Mira hacia aquí".
cadena.charAt ( 2 ) = "r"
cadena.indexOf ( i ) = 1
cadena.lastIndexOf (a ) = 11
cadena.subString ( 5, 9 )
cadena.toLowerCase( ) = "mira hacia aquí"
cadena.toUpperCase( ) = "MIRA HACIA AQUÍ"
```

Métodos sobre el HTML:

- *anchor* (nombre_string), este método crea un ancla, llamada nombre_string como valor para el atributo NAME.
- *link* (href_string), este método crea un link a un URL designado por el argumento href_string.

Ejemplo 11:

```
cadena.big() = "Mira hacia aquí".
cadena.blink() = "Mira hacia aquí". cadena.blink() = "                "
cadena.bold() = "Mira hacia aquí".
cadena.fixed() = "Mira hacia aquí". // Éste es el tipo de letra Courier New
cadena.fontcolor("red") = "Mira hacia aquí".
cadena.fontsize(3) = "Mira hacia aquí".
cadena.italics() = "Mira hacia aquí".
cadena.small() = "Mira hacia aquí".
cadena.strike()= "Mira hacia aquí".
cadena.sup()= "Mira hacia aquí".
cadena.sub()= "Mira hacia aquí".
```

5.3.1.2. Clase Math

La clase Math se usa para efectuar cálculos matemáticos. Contiene propiedades generales como pi = 3.14159…, y varios métodos que representan funciones trigonométricas y algebraicas. Todos los métodos de Math pueden trabajar con decimales. Los ángulos se dan en radianes, no en grados.

La clase Math es el primer ejemplo de clase estática (que no cambia). Todos sus argumentos son valores. Esta clase no permite crear objetos, por lo que hay que referirse directamente a la clase para usar los métodos.

Propiedades (se usan del modo `Math.propiedad`):

- *E*, número "e". Es un número tal que su logaritmo neperiano es 1, ln(e) = 1
- *LN10*, logaritmo neperiano del número 10.
- *LN2*, logaritmo neperiano del número 2.
- *PI*, número pi = 3.14159…
- *SQRT1_2*, raíz cuadradada de ½.
- *SQRT2*, raíz cuadrada de 2.

Métodos:

- *abs* (numero), calcula el número absoluto de numero.
- *acos* (numero), calcula el ángulo cuyo coseno es numero.
- *asin* (numero), calcula el ángulo cuyo seno es numero.
- *atan* (numero), calcula el ángulo cuya tangente es numero.
- *ceil* (numero), calcula el entero mayor o igual que numero.
- *cos* (angulo), calcula el coseno de angulo.
- *exp* (numero), calcula el número e elevado a la potencia numero.
- *floor* (numero), calcula el entero menor o igual que numero.
- *log* (numero), calcula el logaritmo natural de numero.
- *max* (numero1, numero2), calcula el máximo entre numero1y numero2.
- *min* (numero1, numero2), calcula el mínimo entre numero1y numero2.
- *pow* (numero1, numero2), calcula numero1 exponentado a numero2.
- *random* ( ), calcula un número decimal aleatorio entre 0 y 1, SÓLO PARA UNIX.
- *round* (numero), devuelve el entero más cercano a numero.
- *sin* (angulo), calcula el seno de angulo.
- *sqrt* (numero), calcula la raíz cuadrada de numero.
- *tan* (angulo), calcula la tangente de angulo.

Ejemplo 12:
```
Math.abs(-4) = 4
Math.abs(5) = 5
Math.max(2,9).=.9
Math.pow(3,2) = 9
Math.sqrt(144) = 12
```

5.3.1.3. Clase Date

Una de las cosas más complicadas de cualquier lenguaje es trabajar con fechas. Esto es porque hay gente que para representar fechas y horas toma un sistema no decimal (los meses en unidades sobre 12, las horas sobre 24 y los minutos y segundos sobre 60). Para el ordenador es ilógico trabajar con números bonitos y redondeados.

La clase date simplifica y automatiza la conversión entre las representaciones horarias del ordenador y la humana.

La clase date de *JavaScript* sigue el estándar de UNIX para almacenar los datos horarios como el número de milisegundos desde el día 1 de enero de 1970 a las 0:00. Esta fecha se denomina "la época".

Aunque la clase date no tiene propiedades, tiene varios métodos. Para usar la clase date hay que entender cómo construir un objeto de esta clase. Para eso hay tres métodos:

- *new Date*( ), inicializa un objeto con la hora y fecha actual.
- *new Date*(string_dato), inicializa un objeto con el argumento string_dato. El argumento debe ser de la forma "Mes día, año" como "Noviembre 29, 1990".
- *new Date*(año, mes, día), iniciliaza un objeto tomando 3 enteros que representan el año, mes y día. NOTA: los meses tienen como base el 0, lo que significa que 2 corresponde con el mes de marzo y 10 con el mes de noviembre.

Ejemplo 13:
```
var dato = new Date(90, 10, 23)
var dato = new Date(1990, 10, 23)
```

Estas dos declaraciones se refieren a la fecha del 23 de noviembre de 1990.

Hay un modo opcional para declarar la hora además de la fecha. Hay poner 3 argumentos más a la vez que se ponen los argumentos de la fecha.

Ejemplo 14:
```
var dato2 = new Date(90, 10, 23, 13,5,9)
```

Esta declaración se refiere a la 1:05:09 PM del día 23 de noviembre de 1990.

Métodos:

- *getDate* ( ), devuelve el número de día del mes (1-31).
- *getDay* ( ), devuelve el número de día de la semana (0-6).
- *getHours* ( ), devuelve el número de horas del día (0-23).
- *getMinutes* ( ), devuelve el número de minutos de la hora (0-59)
- *getMonth* ( ), devuelve el número de mes del año (0-11).
- *getSeconds* ( ), devuelve el número de segundos del minuto (0-59)
- *getTime* ( ), devuelve la hora.
- *getYear* ( ), devuelve el año.
- *setDate* ( ), fija la fecha.
- *setHours* ( ), fija el número de horas del día.
- *setMinutes* ( ), fija el número de segundos del minuto.

- *setMonth* ( ), fija el número de mes.
- *setSecond* ( ), fija el número de los segundos del minuto.
- *setTime* ( ), fija la hora.
- *setYear* ( ), fija el año.

Ejemplo 15:
```
var dato2 = new Date(90, 10, 23, 13,5,9)
dato2.getHours() = 13
dato2.getDay() = 0 (si es lunes),1 (si es martes), etc.
dato2.getSeconds = 9
dato2.setMonth() = 2 // el mes cambia a marzo
dato2.setYear() = 96
```

5.3.2. Funciones Predefinidas (Built-in Functions): eval, parseFloat, parseInt.

Además de los clases ***String***, ***Math*** y ***Date*** hay un pequeño conjunto de funciones predefinidas por *JavaScript*. No son métodos ni se aplican a los objetos. Tiene el mismo comportamiento que cuando se crean funciones con el comando **`function`**.

5.3.2.1. eval(string)

Esta función intenta evaluar su argumento de tipo string como una expresión y devolver su valor. Esta función es muy potente porque evalúa cualquier expresión de *JavaScript*.

Ejemplo 16:
```
w = (x * 14) - (x / z) + 11
z = eval ("(x * 14) - (x / z) + 11")
```

Si x es una variable con el valor 10 las siguientes expresiones asignan 146 a `w` y `z`.

5.3.2.2. parseFloat(string)

Esta función intenta convertir su argumento de tipo string como un número decimal (de coma flotante).

Ejemplo17:
```
parseFloat("3.14pepecomeperas345") = 3.14
```

5.3.2.3. parseInt(string, base)

Esta función se comporta de forma muy similar a la anterior. Intenta convertir su argumento de tipo string como un entero en la base elegida.

Ejemplo 18:
```
ParseInt(10111, 2)= 23
```

Se convierte el número 10111 en base binaria.

5.3.3. Clases del browser

El modelo de clases de *JavaScript* y su conjunto de clases, métodos y funciones predefinidas dan un moderno lenguaje de programación. Puesto que *JavaScript* se designó para trabajar con y en el World Wide Web tuvo que haber un nexo entre *JavaScript* y el contenido de las páginas HTML. Este nexo viene dado por un conjunto de clases del browser y del HTML.

Las clases del browser son una extracción del entorno del browser e incluye clases para usar en la página actual, en la lista de documentos visitados (history list) y en el URL actual. Existen métodos para abrir nuevas ventanas, mostrar cajas de diálogos y escribir directamente HTML (en algunos ejemplos se ha usado el método ***write*** de la clase ***document***.

Las clases del browser (o navegador) son el nivel más alto de la jerarquía de objetos de *JavaScript*. Representan información y acciones que no necesariamente hay que asociar con una página Web. Dentro de una página Web cada elemento HTML tiene su objeto correspondiente, un objeto HTML dentro de la jerarquía de objetos. Cada formulario HTML y cada elemento dentro de un formulario HTML tiene su correspondiente objeto.

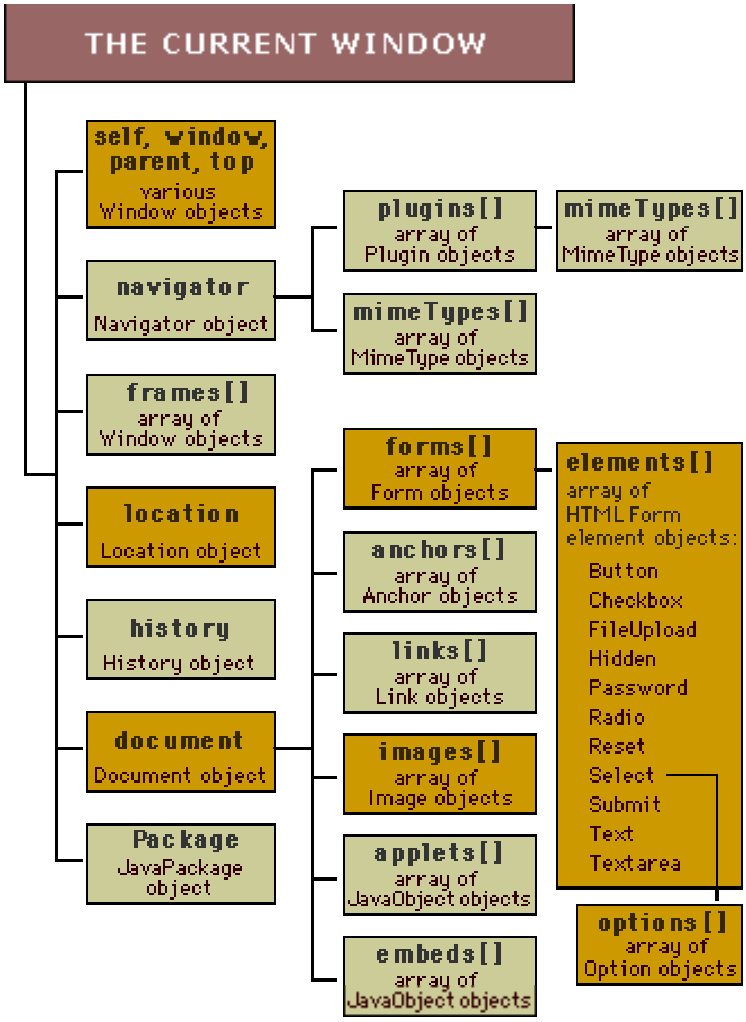
La siguiente figura muestra la jerarquía de los objetos del browser y del HTML referidos a todos los elementos de una página Web.

5.3.3.1. Clase Window

Es el nivel más alto de la jerarquía de objetos de *JavaScript*. Cada ventana de un browser que está abierta tiene su correspondiente objeto window. Todo el resto de objetos desciende del objeto window. Normalmente, cada ventana se asocia a una página Web y la estructura HTML de esa página se refleja en el objeto *document* de la ventana. Cada ventana se corresponde con algún URL que se refleja en el objeto location. Cada ventana tiene una lista de documentos visitados que se han mostrado en esa ventana (history list), las cuales se representan por varias propiedades del objeto history.

Los métodos de un objeto window son:

- *alert*(string_mensaje)
- *confirm*(string_mensaje)
- *open*(URL_string, nombre_ventana)
- *close*( )
- *prompt*(string_mensaje)



Ejemplo 19:

```
alert("No clicar el botón izquierdo")
confirm("¿Quieres continuar?")
window.open("fichero.htm", ventana_1)
window.close()  // cierra una ventana
prompt("Rellena el cuestionario")
```

Todos estos métodos se usan para manipular el estado de la ventana del browser. Los métodos alert y confirm se usan para mostrar su argumento string_mensaje en una caja de diálogo. El método alert se usa para avisar al usuario sobre algo que no debe hacer. La caja de diálogo de alert contiene el botón OK,

mientras que la de confirm muestra el mensaje con un botón OK y otro Cancel. Devuelve, como valor de retorno, **`true`** si se clica OK o **`false`** si se clica Cancel.

El método prompt se usa para solicitar al usuario una entrada de datos, en forma de cadena de caracteres. Muestra una caja de diálogo con el string_mensaje y un campo de texto editable. Este método acepta un segundo argumento opcional que se usa para fijar un valor por defecto en el campo de texto. Devuelve lo que escribe el usuario en el campo de texto.

El método open se usa para abrir una ventana nueva en el browser. El argumento URL_string representa el URL que será cargado en la ventana donde el otro argumento nombre_ventana da nombre a la ventana. Este método devuelve una instancia del objeto window que representa la nueva ventana creada. Este método también acepta un tercer argumento opcional que se usa para especificar los modos de mostrar la nueva ventana. Cuando el método *close* se llama desde un ejemplar del objeto window, esa ventana se cierra y el URL se descarga.

5.3.3.2. Clase Document

Cada ventana se asocia con un objeto document. El objeto document contiene propiedades para cada ancla, link, y formulario en la página. También contiene propiedades para su título, color de fondo, colores de los links y otros atributos de la página. El objeto document tiene los siguientes métodos:

- clear()
- close()
- open()
- write(string)
- writeln(string)

El método clear se usa para borrar completamente un documento. Tiene mucho uso si se está construyendo una página Web sólo con *JavaScript*, y se quiere asegurar que está vacía antes de empezar. Los métodos open y close se usan para empezar y parar la salida de datos a memoria. Si se llama al método open, se ejecutan series de write y/o writeln, y se llama al método close, el resultado de las operaciones que se han escrito se muestran en la página.

El método write se usa para escribir cualquier cadena de caracteres, incluyendo programación HTML, al documento actual. Este método puede usar un número variable de argumentos. El método writeln es idéntico al método write, excepto que en la salida imprime un salto de línea al acabar de escribir sus argumentos. Hay que notar que el salto de línea será ignorado por el browser, el cual no incluye espacios en blanco, a menos que el writeln esté dentro de texto preformateado.

Ejemplo 20:

```
document.clear()
document.close()
document.open()
document.write("Juan come peras") -> Juan come peras
document.writeln("Juan come peras") -> Juan come peras
```

5.3.3.3. Clase Location

El objeto location describe el URL del documento. Este tiene propiedades representando varios componentes del URL, incluyendo su parte de protocolo, de hostname, de pathname, de número de puerto, entre otras propiedades. También tiene el método toString el cual se usa para convertir el URL a una cadena de caracteres. Para mostrar el URL actual podemos usar el siguiente código:

Ejemplo 21:

```
var lugar = document.location
document.write("<BR>El URL actual es " + lugar.toString())
document.write("<BR>")
```

5.3.3.4. Clase History

El objeto history se usa para referirse a la lista de URLs visitados (history list) anteriormente. Tiene una propiedad conocida como length, la cual indica cuántos URLs están presentes en la history list actualmente. Tiene los siguientes métodos:

- *back*()

- *forward*()
- *go*(donde)

El método go se usa para navegar en la history list. El argumento donde puede ser un número o un string. Si el argumento donde es un número indica el número de orden del lugar donde se desea ir en la history list. Un número positivo significa que avance tantos documentos como indique el número, y un número negativo significa que se atrase tantos documentos como indique el número. Si *donde* es una cadena de caracteres que representa un URL, entonces pasa a ser como el documento actual.

Ejemplo 22:

```
history.back()
history.forward()
history.go(-2)
history.go(+3)
```

5.3.4. Clases del documento HTML (anchors, forms, links)

Para entender cómo trabajan los objetos HTML en *JavaScript*, hay que considerar ciertas piezas de HTML que crean un ancla, un formulario y un link a este ancla. Para aclarar estos conceptos nos basamos en el ejemplo 23.

Este código crea una página HTML con un ancla al principio de la página y un link al ancla al final. Entre ambas hay un simple formulario que permite al usuario poner su nombre. Hay un submit button (botón de envío) por si se quiere enviar y un botón de reset por si no se quiere enviar. Si el usuario envía con éxito el contenido del formulario vía post al e-mail ficticio nobody@dev.null.

Ejemplo 23:

```
<HTML>
<HEAD><TITLE>Ejemplo sencillo de página HTML</TITLE></HEAD>
<BODY>
<A NAME="principio">Este es el principio de la página</A> <HR>
<FORM METHOD="post">
<P> Introduzca su nombre: <INPUT TYPE="text" NAME="me" SIZE="70"></P>
<INPUT TYPE= "reset" VALUE="Borrar Datos">
<INPUT TYPE= "submit" VALUE="OK">
</FORM><HR>
Clica aquí para ir al
<A HREF="#principio">principio</A> de la página // link
<BODY/>
</HTML>
```

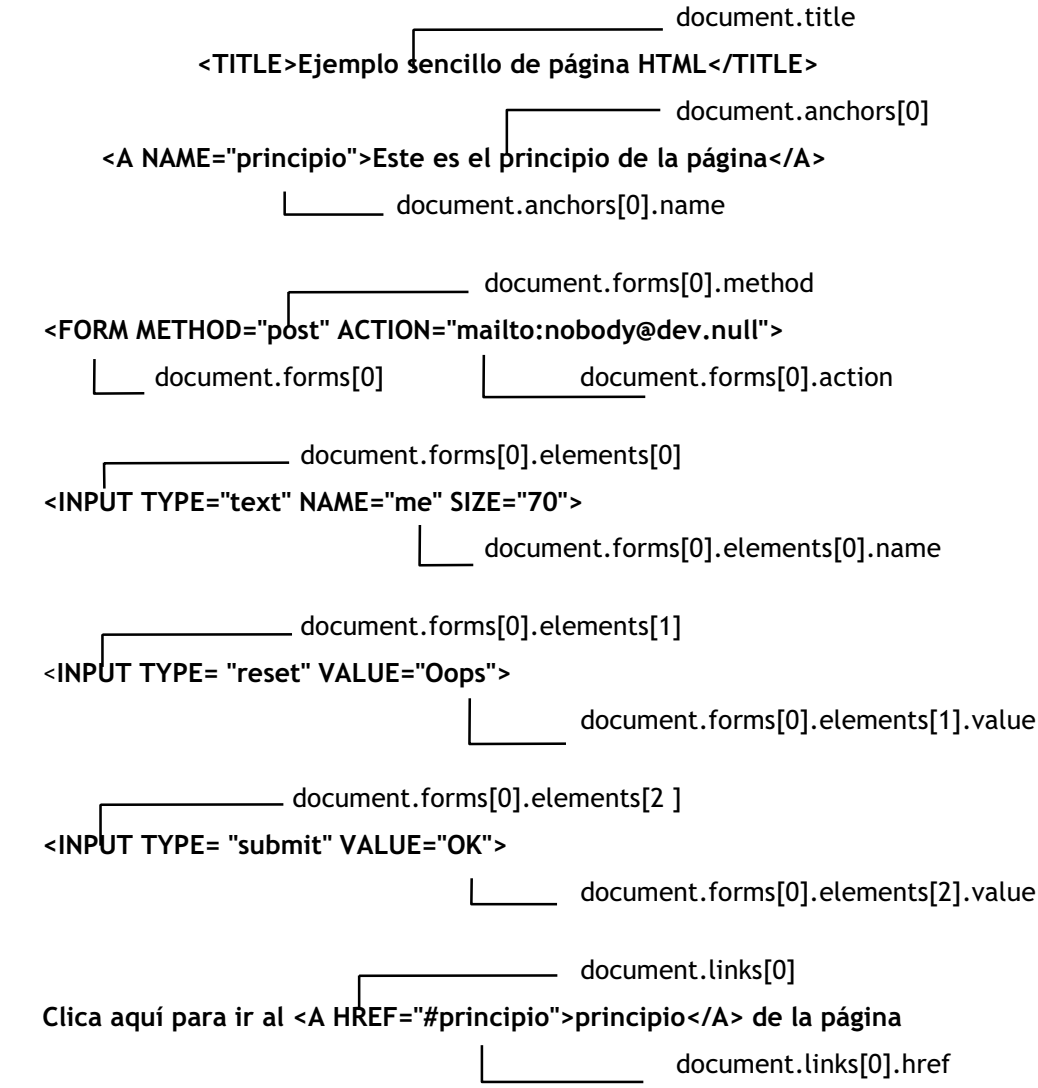
El aspecto más importante de este ejemplo es el hecho de que los elementos HTML se reflejan en la jerarquía de objetos de *JavaScript*. Se puede acceder al título del documento a través de la propiedad title del objeto documento. Se puede acceder a otros elementos HTML de este documento usando las siguientes propiedades:

- *anchors*(anclas)
- *forms*(formularios)
- *links*

Estas propiedades del objeto document son arrays que representan cada elemento HTML como un ancla, formulario o link de una página. En el ejemplo, el ancla en el principio de la página se referiría como document.anchors[0], el link al final de la página como document.links[0], y el formulario en medio de la página como document.forms[0]. Estos son el nivel más alto de los objetos representados por este documento. Cada uno de estos elementos tiene propiedades y métodos que se usan para describir y manipularlos.

El objeto form correspondiente a forms[0] tiene sub-objetos para cada uno de los tres elementos (el botón de reset, el botón de envío y el campo de texto) y propiedades para el método submit. forms[0].elements[0] corresponde a la entrada del campo de texto. forms[0].elements[0].name es el nombre de este campo, como el especificado por el atributo NAME, el cual en este caso es "me". La

siguiente figura representa el código HTML del ejemplo y muestra cómo cada elemento en la página se asocia con el objeto HTML.



5.4. Clases y Funciones definidas por el usuario.

Para entender este tipo de programación vamos a ver un ejemplo donde creamos un objeto llamado casa que tiene las siguientes propiedades: nº de habitaciones, año de construcción, ¿tiene garaje?, estilo arquitectónico. Para definir un objeto para guardar esta información (las propiedades) hay que hacer una función que las muestre en un listado. Nota: Esta función usa el comando this, que hace referencia al objeto activo. En este caso hace referencia al objeto activo que estamos creando.

Esta función es una especie de constructor que se usa en *C++* para definir un objeto. Esto es porque en este sentido los objetos de *JavaScript* son similares a las estructuras de *C* y a las clases de *C++*. En los 3 casos a los miembros, funciones miembro o propiedades del objeto/clase/estructura se accede con el operador punto (.):

```
estructura.miembro
clase.función_miembro()
objeto.propiedad.
```

Hay varias cosas a ver en este ejemplo. El nombre de la función es el nombre del objeto: casa (en *C++* el constructor tiene el nombre de la clase). La función no tiene valor de retorno.

El ejemplo muestra cómo se define un objeto casa, pero no crea ningún objeto específico del tipo casa.

Ejemplo 24:
```
function casa( habs, estil, fecha, garage){
    this.habitaciones = habs
    this.estilo = estil
    this.fecha_construcción = fecha
    this.tiene_garage = garage
}
```

Un objeto específico de casa tendrá las 4 propiedades con sus valores. Las instancias se crean usando la sentencia ***new*** con una función de llamada. Se crearía un objeto de casa, llamado micasa del siguiente modo:

Ejemplo 25:
```
var micasa = new casa(10,"Colonial", 1989, verdadero)
```

Ahora el objeto micasa es otra variable. Tiene que declararse usando var. Ahora que micasa ha sido creada podemos referirnos a sus propiedades con el operador punto (.):

Ejemplo 26:
```
micasa.habitaciones = 10 (entero, int)
micasa.estilo = "colonial" (cadena de caracteres, String)
micasa.fecha_construcción = 1989 (entero ,int)
micasa.tiene_garage = true (booleano)
```

No hay nada que evite poner "yes" en la propiedad tiene_garage en vez de un valor booleano, por esto hay que tener cuidado con este tipo de confusión. Si se pone "yes", tiene_garage no será booleana sino una cadena de caracteres.

5.4.1. Funciones (métodos)

Uno de los aspectos más potentes de la programación orientada a objetos de *JavaScript* es la posibilidad de crear clases con funciones miembro, llamadas métodos. Esto tiene varias ventajas como la organización y la asociación de funciones con clases. Los métodos, aunque programando se trabaje como si fueran propiedades no se tienen en cuenta a la hora de contarlos como tales.

Por ejemplo, tenemos una función que muestra las propiedades de los objetos de la clase casa llamada muestra_props( ). Para añadirla como propiedad (se llamará muestra) a un objeto o a su clase se debería escribir:

Ejemplo 27:
```
this.muestra = muestra_props  dentro de la definición de la clase casa.
Micasa.muestra = muestra_props   como una sentencia normal.
```

y para usarlas, con los objetos de la clase casa:

```
micasa.muestra( ) o bien  muestra_props( micasa )
```

5.4.2. Objetos como Arrays (Vectores)

Algunos lenguajes de programación soportan datos de tipo array (*C*, *C++*, Visual Basic, Java, etc.). Un array es una colección de items con índices los cuales son del mismo tipo. En *C* por ejemplo para declarar un array de 10 datos de tipo entero, se hace de la forma: int nombre[10]; y estos enteros son definidos desde el nombre[0] al nombre[9]. Es más común que la base del primer índice sea un **0** (zero-based indexing) que un 1 (one-based indexing).

JavaScript usa 0 como base del primer índice. En *JavaScript*, quizá, arrays y objetos son 2 puntos de vista del mismo concepto. Cada objeto es un array de los valores de sus propiedades, y cada array es también un objeto. Volviendo al ejemplo anterior, el objeto micasa es un array con los siguientes cuatro elementos:

Ejemplo 28:
```
micasa[0]=10 (habitación)
micasa[1]="colonial" (estilo)
micasa[2]=1989 (fecha_construcción)
micasa[4]=True (tiene garaje)
```

No parece haber muchas ventajas al referirse a objetos de este modo más numérico y menos informático. Quizá, esta forma alternativa permite acceder a las propiedades secuencialmente (p. ej con un bucle) lo que es muy usado.

Es aconsejable definir todos las clases con una propiedad que dé el número de propiedades en el objeto y haciéndola la primera propiedad. Ahora el objeto micasa es un array de 5 elementos. La nueva propiedad se denomina length (longitud).

Ejemplo 29:

```
micasa.lenght = 5
micasa[0]=10 (habitación)
micasa[1]="colonial" (estilo)
micasa.habitaciones = micasa[2]=1989 (fecha_construcción)
micasa.estilo=micasa[3] = True (tiene garaje)
micasa.tiene_garage = micasa[4] = True (tiene garaje)
```

Aun hay otro modo de dar valor a las propiedades:

Ejemplo 30:

```
micasa["length"] = 5
micasa["habitaciones"] = 10
micasa["estilo"] = "colonial"
micasa["fecha_construccion"] = 1989
micasa["tiene_garaje"] = true
```

5.4.3. Extender Objetos

Qué pasa si queremos más propiedades en un objeto: nada. Es posible extender dinámicamente un objeto simplemente tomando una nueva propiedad. Con el Ejemplo 31 se verá mejor:

Ejemplo 31:

```
tucasa = new casa(26, "restaurante",1993, false)
tucasa.paredes = 6
tucasa.tiene_terraza = true
```

Estas dos sentencias añaden dos propiedades al final del array tucasa. Las extensiones dinámicas se aplican sólo a objetos específicos. El objeto micasa no se ve afectado ni la clase casa se ve afectada ni el objeto casa cambia de ningún modo.

Esta característica que se puede aplicar a los objetos simplifica mucho la programación. Para asegurarse de que no se producen errores al intentar mostrar las propiedades de un objeto es importante cambiar la propiedad que almacena el número de propiedades.

Ejemplo 32:

```
tucasa.length += 2
```

Un caso común donde la extensión dinámica es muy usada es en arrays de número variable.

5.4.4. Funciones con un número variable de argumentos.

Todas las funciones de *JavaScript* tienen las siguientes 2 propiedades: **caller** y **arguments**.

La propiedad **caller** es el nombre de cada uno que llama a la función. La propiedad **arguments** es un array de todos los argumentos que no están en la lista de argumentos de la función. La propiedad **caller** permite a una función identificar y responder al entorno desde donde se llama. La propiedades **arguments** permite escribir funciones que toman un número de argumento variable. Los argumentos de la lista de argumentos de una función son obligatorios mientras que los que están en la propiedad **arguments** son opcionales.

El siguiente ejemplo muestra los argumentos obligatorios y opcionales de una función.

Ejemplo 33:

```
function anadir( string){
  var nargumentos = anadir.arguments.length
  var totalstring
  var parcialstring =
  for (var i = 1; i < nargumentos; i++ ){
    parcialstring += " " + anadir.arguments[i] ","
  }
  totalstring = "El argumento obligatorio es " + string + ". El número de
argumentos opcionales es " + nargumentos + " y los argumentos opcionales
son " + parcialstring
  return (totalstring)
}
```

5.5. Expresiones y operadores de JavaScript.

5.5.1. Expresiones

Una expresión es cualquier conjunto de letras, variables y operadores que evalúa un valor simple. El valor puede ser un número, una cadena, o un valor lógico. Hay dos tipos de expresiones:

- aquellas que asignan un valor a una variable, x =7
- aquellas que simplemente tienen un valor, 3 + 4
- *JavaScript* tiene los siguientes tipos de expresiones:
- Aritméticas: evalúan un número.
- De cadena: evalúan un string de carácter, por ejemplo "Fred" or "234".
- Lógicas: evalúan si son verdadero o falso.

5.5.1.1. Expresiones Condicionales

Una expresión condicional puede tomar uno de dos posibles valores según una condición. La sintaxis es *(condición) ? valor1 : valor2*.

Si la condición es verdadera (*true*) toma el valor valor1, y si es falsa (*false*) toma el valor2. Se puede usar una expresión condicional en cualquier parte.

Por ejemplo: estado = (edad >= 18) ? "adulto" : "menor de edad". Si edad es mayor o igual que 18 a la variable estado se le asigna la cadena "adulto", si no se le asigna el valor "menor de edad".

*5.5.2. Operadores de asignación (=, +=, -=, *=, /=)*

Un operador de asignación da un valor a la izquierda de su operando basado en la parte derecha de su operando. El operador básico de asignación es el igual (=), el cual asigna el valor de la derecha de su operando al de la izquierda de su operando. *x = y*, asigna el valor de *y* a *x*.

Los otros operadores de asignación para operaciones aritméticas son los siguientes:

- x = y
- x += y significa x = x + y
- x -= y significa x = x - y
- x *= y significa x = x * y
- x /= y significa x = x / y
- x %= y significa x = x % y

Estas sentencias primero operan a la derecha del operador y después devuelven el valor obtenido a la variable situada a la izquierda del operador.

5.5.3. Operadores aritméticos

Los operadores toman valores numéricos (sean letras o variables) y devuelven un único valor numérico.

- Los operadores estándar son la suma (+), la resta (-), la multiplicación (*), y la división (/). Estos operadores trabajan de la forma estándar *operando1 operador operando2*.

Además hay otros operadores no tan conocidos como:

- Resto (%) El operador resto devuelve el resto entero al dividir el primer operando entre el segundo operando. Sintaxis: var1 % var2 .

Ejemplo 34:
```
12 % 5 returns 2.
```

- Incremento (++) El operador incremento se usa de la siguiente manera: var++ o ++var. Este operador suma uno a su operando y devuelve el valor.

  1. var++, primero devuelve el valor de la variable y después le suma uno.
  2. ++var, primero suma uno y después devuelve el valor de la variable.

Por ejemplo si x es 3, la sentencia *y* = *x*++, incrementa *x* a 4 y asigna 3 a *y*, pero si la sentencia es *y* = ++*x*, incrementa *x* a 4 y asigna 4 a *y*.

- Decremento (--) El operador decremento se usa de la siguiente manera: var-- o --var. Este operador resta uno a su operando y devuelve el valor.

  1. var-- primero devuelve el valor de la variable y después le resta uno.
  2. --var, primero suma uno y después devuelve el valor de la variable.

Por ejemplo si x es 3, la sentencia *y* = *x*--, decrementa *x* a 2 y asigna 3 a *y*, pero si la sentencia es *y* = --*x*, decrementa *x* a 2 y asigna 2 a *y*.

- Negación (-) El operador negación precede a su operando. Devuelve su operando negado. Por ejemplo, *x* = -*x*, hace negativo el valor de *x*, que si fuera 3, sería -3.

5.5.4. Operadores lógicos

Los operadores lógicos toman valores lógicos (booleanos) como operandos. Devuelven un valor lógico. Los valores lógicos son *true* (verdadero) y *false* (falso). Se suelen usar en las sentencias de control.

- And (&&) Uso: expr1 && expr2. Este operador devuelve *true* si ambas expresiones lógicas son verdaderas, o *false* si alguna no es *true*.
- Or (||)Uso: expr1 || expr2. Este operador devuelve *true* si una de las dos expresiones lógicas, o ambas, son verdaderas, o *false* si ambas son falsas.
- Not (!) Uso: !expr. Este operador niega su expresión. Devuelve *true* si es *false* y *false* si es *true*.

Ejemplo 35:
```
if ((edad_pepe>=18)&&(edad_juan==18)){
    document.write("Juan y pepe son adultos")
} else {
    document.write("Uno de los dos no es adulto")
}
```

5.5.5. Operadores de Comparación (= =, >, >=, <, <=, !=)

Un operador de comparación compara sus operandos y devuelve un valor lógico según sea la comparación verdadera o no. Los operandos pueden ser números o cadenas de caracteres. Cuando se usan cadenas de caracteres, las comparaciones se basan en el orden alfabético. Al igual que los operadores lógicos, se suelen usar en sentencias de control.

Los operadores son:

- Igual (==), devuelve *true* si los operandos son iguales.
- Desigual (!=), devuelve *true* si los operandos son diferentes.
- Mayor que (>), devuelve *true* si su operando izquierdo es mayor que el derecho.
- Mayor o igual que (>=), devuelve *true* si su operando izquierdo es mayor o igual que el derecho.
- Menor que (>), devuelve *true* si su operando izquierdo es menor que el derecho.
- Menor o igual que (>=), devuelve *true* si su operando izquierdo es menor o igual que el derecho.

5.5.6. Operadores de String

Además de los operadores de comparación, que pueden usarse con cadenas de caracteres, existe el operador concatenación (+) que une dos cadenas, devolviendo otra cadena que es la unión de las dos anteriores.

Ejemplo 36:
```
"mi " + "casa" devuelve la cadena "mi casa".
```

El operador de asignación += se puede usar para concatenar cadenas. Por ejemplo, si la variable letra es una cadena con el valor "alfa", entonces la expresión letra += "beto" evalúa a "alfabeto" y asigna este valor a letra.

5.5.7. Prioridad de los operadores

La prioridad de los operadores determina el orden con el cual se aplican cuando se evalúan. Esta prioridad se rompe cuando se usan paréntesis.

La prioridad de operadores, de menor a mayor es la que sigue:

- coma ,
- asignación = += -= *= /= %=
- condicional ?:
- lógico-or ||
- logical-and &&
- igualdad == !=
- relación < <= > >=
- adición/sustracción + -
- multiplicación / división / resto * / %
- negación/incremento ! ~ - ++ --
- paréntesis, corchetes () [] .

5.6. Sentencias de control de JavaScript.

JavaScript soporta un conjunto de sentencias que se pueden usar para hacer interactivas las páginas Web.

5.6.1. La sentencia if

Una sentencia if es como un interruptor. Si la condición especificada es cierta, se ejecutan ciertas sentencias. Si la condición es falsa, se pueden ejecutar otras. Un sentencia if es :

```
if (condición) {
sentencias 1 }
[else {
sentencias 2}]
```

La parte else es opcional.

Ejemplo 37:
```
if ((edad_pepe>=18)&&(edad_juan==18)){
    document.write("Juan y pepe son adultos")
} else {
    docuemnt.write("Uno de los dos no es adulto")
}
```

5.6.2. Bucles

Un bucle es un conjunto de comandos que se ejecutan repetidamente hasta que una condición especificada se encuentra. *JavaScript* proporciona dos tipos de bucles: for y while.

5.6.2.1. Bucle for

Una sentencia ***for*** repite un bucle hasta que una condición se evalúe como ***false***. Este bucle es similar al tradicional bucle for en Java, *C* y *C++*.

Un bucle ***for*** es:

```
for ([expresión_inicial]; [condición] ;[expresión_incremento]) {
  sentencias
}
```

Ejemplo 38:

```
for (var contador = 0; contador <= 5; contador++) {
    document.write("Número "+ contador + "<br>")
}
```

Y la salida por pantalla del ejemplo es:

Número 0

Número 1

Número 2

Número 3

Número 4

Número 5

Cuando se encuentra un bucle ***for***, se ejecuta la expresión inicial. Las sentencias se ejecutan mientras la condición sea ***true***. La expresión_incremento se ejecuta cada vez que vuelve a realizarse una vuelta o paso en el bucle.

5.6.2.2. Bucle while

Una sentencia ***while*** repite un bucle mientras la condición evaluada sea ***true***. Un bucle ***while*** es:

```
while (condición)
{
  sentencias
}
```

Ejemplo 39:

```
var contador = 0
while (contador <= 5){
    document.write("Número "+ contador + "<br>")
    contador++
}
```

Es el mismo ejemplo que el 38 pero con el bucle while.

Si la condición llega a ser false, las sentencias dentro del bucle dejan de ejecutarse y el control pasa a la siguiente sentencia después del bucle.

La condición se evalúa cuando las sentencias en el bucle han sido ejecutadas y el bucle está a punto de ser repetido. Dentro del bucle debe haber una sentencia que en algún momento haga parar la ejecución del bucle.

La comprobación de la condición tiene lugar únicamente cuando las sentencias del bucle se han ejecutado y el bucle está a punto de volverse a ejecutar. Esto es, la comprobación de la condición no es continuada, sino que tiene lugar por primera vez al principio del bucle y de nuevo a continuación de la última sentencia del bucle, cada vez que el bucle llega a este punto.

Ejemplo 40:

```
n = 0; x = 0
while( n < 3 ) {
  n++; x += n;
}
```

5.7. Comentarios

Los comentarios son anotaciones del autor para explicar qué hace cada sentencia. Son ignorados por el navegador. *JavaScript* soporta el estilo de comentarios de *C*, *C++* y Java.

- Los comentarios de una sola línea son precedidos por una doble barra normal (//).
- Los comentarios de más de una línea van escritos entre /* y */.

Ejemplo 41:

```
// Esto es un comentario de una sola línea.


/* Esto es un comentario de más de una línea. Puede
tener la extensión que se quiera. */
```