OBJETOS

Aunque la mayoría de los autores así lo hacen, no estoy muy convencido de que los objetos se puedan clasificar como un tipo de dato. Sin embargo, dado que JavaScript es un lenguaje basado en objetos, considero importante tratar este tema antes de adentrarnos en el manejo de otros conceptos tales como, estructuras condicionales y de control.

Un objeto es la abstracción de un elemento que no necesariamente corresponde a un ente físico en el modelo de una aplicación. Un objeto es un encapsulador de información, que está compuesto por dos elementos básicos: los atributos y los métodos. Los primeros, son las variables del objeto, mientras que los segundos son las funciones que el objeto realiza con sus atributos.

La comunicación entre objetos se realiza mediante mensajes, que permiten enviar solicitudes a las cuales el objeto responde a través de una o varias de las funciones que tiene implementadas.

En otras palabras los objetos son componentes prestadores de servicios que tienen sus propias variables y funciones, las cuales no son susceptibles de ser cambiadas en tiempo de ejecución.

Para ilustrar un poco el concepto de objeto, piense por ejemplo en un carro:

Un carro tiene asociadas características tales como: cilindraje, tipo de combustible que usa, color, número de velocidades, modelo, número de puestos, número de puertas, etc. Las anteriores son las variables o atributos del objeto carro. Además, con un carro se pueden realizar acciones como: encender y apagar el motor, acelerar, frenar, etc. Estas últimas constituyen las funciones o métodos del objeto carro.

El concepto tradicional de programación estructurada supone la ejecución de una serie de tareas y funciones que son realizadas una tras otra sobre ciertas variables de entrada, para producir un resultado. En el paradigma de la programación orientada a objetos (OOP), éstos son responsables de la realización de tareas específicas sobre datos propios.

Una vez definido con sus atributos y métodos, un objeto puede ser reutilizado o redefinido conservando sus características iniciales.

Creación de Objetos en JavaScript

JavaScript dispone de dos formas para la creación de objetos: En la primera simplemente se asigna un nombre al objeto y se inicializan sus atributos de la misma manera como se inicializa una variable. La segunda forma consiste en crear una función constructora del objeto para luego generar copias o instancias del objeto mediante el operador new.

Mediante la inicialización

La sintaxis para la creación de objetos mediante la inicialización de los mismos es muy similar a la declaración de estructuras de datos en C:

```
NombreObjeto =
{
      atributo1:valor1,
      atributo2:valor2,
      ...
      atributoN:valorN
};
```

Como ejemplo, en las siguientes líneas se crea el objeto estudiante con los atributos nombres, apellidos, semestre y promedio.

```
estudiante =
{
    nombres:"",
    apellidos:"",
    semestre:3,
    promedio:1
};
```

El operador punto (.) permite acceder o referirse a cualquiera de los atributos de un objeto. En las siguientes líneas se asignan valores a algunos de los atributos del objeto estudiante definido anteriormente:

```
estudiante.nombres="Andrés Francisco";
estudiante.apellidos="Gaitán Guzmán";
estudiante.promedio=estudiante.promedio*4.3;
```

Ahora los atributos nombres, apellidos y promedio tienen como valores **Andrés Francisco**, **Gaitán Guzmán** y **4.3** respectivamente, mientras que el valor de semestre continúa siendo **3**. Observe que para acceder a cualquiera de los atributos de un objeto debe usar la siguiente sintaxis:

```
nombreObjeto.nombreAtributo
```

```
<HTML>
<HEAD>
    <script language="JavaScript1.2">

      estudiante =
      {
          nombres:"",
          apellidos:"",
          semestre:3,
          promedio:1
```

```
        };

        estudiante.nombres="Andrés Francisco";
        estudiante.apellidos="Gaitán Guzmán";
        estudiante.promedio=estudiante.promedio*4.3;

        document.write("El estudiante "+estudiante.nombres + " " +
        estudiante.apellidos + " de  " + estudiante.semestre +
        " semestre, obtuvo como promedio " +
        estudiante.promedio);

    </script>
<TITLE>Objeto Simple</TITLE>
        </HEAD>

<BODY>
</BODY>
</HTML>
```

El anterior ejemplo mostrará en pantalla:

```
El estudiante Andrés Francisco Gaitán Guzmán de 3 semestre, obtuvo
como promedio 4.3
```

Un objeto puede tener como atributos a otros objetos:

```
    estudiante =
    {
       nombres:"",
       apellidos:"",
       semestre:3,
       promedio:0,
       notasMaterias:
       {
           fisica:4.4,
           matematicas:4.0,
           programacion:4.2
       }
    };
```

Al objeto estudiante se le ha agregado como atributo el objeto notasMaterias, compuesto a su vez por tres variables: física, matematicas y programacion (note la falta de tildes en las palabras acentuadas). Se puede ahora calcular el promedio del estudiante con base en las notas obtenidas:

```
estudiante.promedio=(estudiante.notasMaterias.fisica+estudiante.nota
sMaterias.matematicas+estudiante.notasMaterias.programacion)/3;
```

El siguiente es el listado completo de nuestra aplicación, que mostrará; los valores de cada uno de los atributos del objeto estudiante:

```
<HTML>
<HEAD>
        <script language="JavaScript1.2">
        estudiante =
        {
            nombres:"",
            apellidos:"",
            semestre:3,
            promedio:0,
            notasMaterias:
            {
```

```
                fisica:4.4,
                matematicas:4.0,
                programacion:4.2
            }
        };

        estudiante.nombres="Andrés Francisco";
        estudiante.apellidos="Gaitán Guzmán";
        estudiante.promedio=(estudiante.notasMaterias.fisica+estudiant
        e.notasMaterias.matematicas+estudiante.notasMaterias.programac
        ion)/3;

        document.write("El estudiante "+estudiante.nombres + " " +
        estudiante.apellidos + " de  " + estudiante.semestre +
        " semestre, obtuvo como promedio " +
        estudiante.promedio);

        </script>
<TITLE>Objeto Simple</TITLE>
        </HEAD>

<BODY>
</BODY>
        </HTML>
```

Mediante el uso de una función constructora

La creación de objetos utilizando una función constructora, supone como ya se había mencionado la ejecución de dos pasos: definir el constructor y luego instanciar el objeto. La sintaxis de la función constructora es la siguiente:

```
function nombreObjeto(valor1, valor2, ... valorN)
{
    this.atributo1 = valor1,
    this.atributo2 = valor2,
    ...
    this.atributoN = valorN
}
```

Las expresiones `valor1`, `valor2`, ... `valorN`; pasados como parámetros a la función constructora, son los valores que se asignarán a los atributos `atributo1, atributo2, ... atributoN`, respectivamente.

La creación de un nuevo objeto con las características del creado mediante la función constructora, se conoce como instanciación. Su sintaxis es como sigue:

```
nombreNuevoObjeto=new nombreObjeto(valor1, valor2, ... valorN)
```

El objeto estud se puede crear mediante una función constructora de la siguiente manera:

```
function estud(nombres, apellidos, semestre, promedio)
{
    this.nombres=nombres,
    this.apellidos=apellidos,
    this.semestre=semestre,
    this.promedio=promedio
};
```

y ahora se puede crear una instancia de estud llamada estudiante:

```
estudiante = new estud("Andrés Francisco", "Gaitán Guzmán", 3, 1);
```

Al crear una instancia de un objeto, este no sólo hereda los atributos, sino que además le son asignados a estos últimos los valores pasados como parámetros. En el ejemplo anterior, el objeto estudiante tiene los mismos atributos que el constructor estud (nombres, apellidos, semestre y promedio) y "**Andrés Francisco**", "**Gaitán Guzmán**", **3** y **1** son los valores actuales de dichos atributos.

El acceso o referencia a cualquier atributo de un objeto es independiente de la forma como fue creado. En la siguiente línea, el valor del promedio del objeto estudiante que antes valía 1, tendrá como nuevo valor 4.3:

```
estudiante.promedio=estudiante.promedio*4.3;
```

Las siguientes líneas muestran el código de una página que despliega los mismos resultados obtenidos en el ejemplo anterior, sólo que aquí el objeto ha sido creado utilizando la función constructora:

```
<HTML>
<HEAD>
<script language="JavaScript1.2">
function estud(nombres, apellidos, semestre, promedio)
{
    this.nombres=nombres,
    this.apellidos=apellidos,
    this.semestre=semestre,
    this.promedio=promedio
};

estudiante = new estud("Andrés Francisco", "Gaitán Guzmán", 3, 1);
estudiante.promedio=estudiante.promedio*4.3;

document.write("El estudiante"+estudiante.nombres+" "
+estudiante.apellidos + " de  " + estudiante.semestre +
"  semestre, obtuvo como promedio " +estudiante.promedio);

</script>
<TITLE>Objeto Simple</TITLE>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Una vez instanciado un objeto, su estructura puede ser redefinida sin que se afecte el constructor. Se puede por ejemplo, agregar un atributo nuevo al objeto estudiante y esta acción no tendrá repercusiones en el objeto original estud. Para ilustrar lo anterior, al objeto estudiante arriba creado, se le agrega el atributo notaMaterias, que a su vez es un objeto. Veamos:

```
notas = {fisica:4.4, matematicas:4.0, programacion:4.2};
```

En la línea anterior se define, mediante el método de inicialización el objeto notas. A continuación se agrega este nuevo atributo al objeto estudiante:

```
estudiante.notaMaterias=notas;
```

Ahora se calcula el promedio de Andrés Francisco, con base en los valores del atributo notaMaterias:

```
estudiante.promedio=(estudiante.notaMaterias.fisica+estudiante.notaM
aterias.matematicas+estudiante.notaMaterias.programacion)/3;
```

Si se crea un nuevo objeto por ejemplo, estuadiante1, utilizando el constructor estud, estuadiante1 tendrá como atributos: nombres, apellidos, semestre y promedio.

El siguiente es el código del script cuyos resultados serán similares a los obtenidos en el ejemplo de objetos construidos mediante inicialización:

```
<script language="JavaScript1.2">
function estud(nombres, apellidos, semestre, promedio)
{
    this.nombres=nombres,
    this.apellidos=apellidos,
    this.semestre=semestre,
    this.promedio=promedio
};

estudiante = new estud("Andrés Francisco", "Gaitán Guzmán", 3, 1);

notas = {fisica:4.4, matematicas:4.0, programacion:4.2};

estudiante.notaMaterias=notas;

estudiante.promedio=(estudiante.notaMaterias.fisica+estudiante.notaM
aterias.matematicas+estudiante.notaMaterias.programacion)/3;

document.write("El estudiante "+estudiante.nombres + " "
+estudiante.apellidos + " de  " + estudiante.semestre +
"  semestre, obtuvo como promedio " +estudiante.promedio);

</script>
```

El anterior script, escrito dentro de una página HTML producirá el siguiente resultado:

```
El estudiante Andrés Francisco Gaitán Guzmán de 3 semestre, obtuvo
como promedio 4.2
```

Métodos asociados a objetos

Como ya se mencionó, un objeto consta de atributos y métodos, siendo estos últimos, funciones asociadas al objeto que se pueden "**llamar**" en cualquier momento.

Para agregar un método a un objeto se deben seguir dos pasos: definir la función y luego asociarla al objeto como si fuera un atributo más. En el siguiente ejemplo se trabaja con el objeto caja cuyos atributos son x, y, z (longitud de los lados de una caja), y la función calculoVolumen, que devuelve el producto de x*y*z:

```
<HTML>
<HEAD>
<TITLE>Untitled</TITLE>
<SCRIPT language="JavaScript">
/* Definición de la función calculoVolumen que devuelve el producto de x*y*z */
    function calculoVolumen()
    {
      return (this.x*this.y*this.z);
    }
    //Función constructora del objeto caja
    function caja (a,b,c)
    {
      this.x=a;
      this.y=b;
      this.z=c;
    /*El argumento volumen recibe el valor devuelto por la función
    calculoVolumen */
      this.volumen=calculoVolumen;
    }
    /* Instanciación del objeto cubo, cuyos lados tienen como longitud 3, 6 y 5
    respectivamente */
    cubo = new caja(3,6,5);
    /* Llamado a la función calculoVolumen, cuyo resultado se asigna a V */
    v=cubo.volumen();
    // Impresión del volumen
      document.write("El vol&uacute;men de una caja de lados
      "+cubo.x+", "+cubo.y+" y "+cubo.z+", respectivamente es
      <b>"+v+"</b>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

OBJETOS PREDEFINIDOS EN JAVASCRIPT

JavaScript es un lenguaje basado en objetos, y la implementación de cualquier aplicación necesariamente se relaciona con ellos. Netscape, en su versión 1.3 de JavaScript (con la que se trabaja en este módulo), tiene implementados alrededor de 47 objetos diferentes que se ubican en una de las dos categorías siguientes: **Objetos del lado Cliente** y **Objetos del Núcleo de JavaScript**.

Los objetos `Array`, `Boolean`, `Date`, `Function`, `Math`, `Number`, `RegExp`, y `String` son los componentes del denominado grupo Objetos del Núcleo de JavaScript.

De los 39 restantes sólo se mencionarán los denominados **Objetos del Navegador**. La lista completa de los objetos predefinidos en JavaScript se puede consultar en:

http://wp.netscape.com/eng/mozilla/3.0/handbook/javascript/

OBJETOS DEL NÚCLEO DE JavaScript

Dentro de este grupo están definidos los objetos que permiten trabajar con funciones, funciones matemáticas, expresiones regulares, y tipos de datos tales como arreglos, fechas, cadenas de caracteres y algunas otras funciones con constantes numéricas.

A continuación se hará una breve descripción de algunos de los más importantes objetos de este grupo. Nuevamente se **recomienda implementar cada uno de los ejemplos** expuestos aquí, con el fin de comprender su funcionalidad.

El objeto Array (arreglo)

Los métodos implementados en este objeto permiten entre otras cosas: crear arreglos, ordenar sus elementos, agregar y eliminar elementos, convertir los elementos de un array en una cadena de caracteres, invertir el orden de los elementos, etc.

En el capítulo dedicado a Variables y Tipos de Datos, se explicó una de las formas de crear y manipular arreglos sin utilizar la funcionalidad del objeto `Array`. Existen al menos tres formas de crear arreglos mediante el uso del operador `new` y el objeto Array:

```
nombre_Arreglo = new Array();

nombre_Arreglo = new Array(elemento_0, elemento_1, … elemento_N);

nombre_Arreglo = new Array(x);
```

De la primera forma se crea un nuevo arreglo cuyos elementos aún no han sido definidos. Un arreglo creado de esta manera, podrá ser poblado posteriormente utilizando la siguiente sintaxis:

```
nombre_Arreglo[x]="valor";
```

En el siguiente ejemplo se crea el arreglo vocales y posteriormente se definen sus elementos:

```
vocales = new Array();
vocales[0]="a";
vocales[1]="e";
vocales[2]="i";
vocales[3]="o";
vocales[4]="u";
```

En la segunda forma, además de crear el nuevo arreglo, se definen simultáneamente sus elementos en una sola línea:

```
vocales = new Array("a",  "e", "i", "o", "u");
```

Finalmente, un arreglo se puede crear definiendo el número de elementos que contendrá pero sin asignarle valor a ninguno de ellos, para posteriormente definirlos:

```
vocales = new Array(5);
vocales[0]="a";
vocales[1]="e";
vocales[2]="i";
vocales[3]="o";
vocales[4]="u";
```

En los tres casos nombre_Arreglo corresponde al nombre de la nueva instancia del Array, que debe cumplir con las normas establecidas para nombrar variables.

Métodos del objeto Array

El objeto Array de la versión 1.3 de JavaScript tiene implementados 10 métodos para manipular los elementos de un arreglo. En los ejemplos siguientes se trabajará con los arreglos **pares**, **impares** y **vocales** definidos así:

```
pares = new Array(0, 2, 4, 6, 8);
impares = new Array(1, 3, 5, 7, 9);
vocales = new Array("a", "e", "i", "o", "u");
```

El método ***concat***, permite agrupar dos o mas arreglos dentro de uno nuevo que es devuelto por el método:

```
//la siguiente línea crea el arreglo todos [0, 2, 4, 6, 8, 1, 3, 5, 7, 9]
todos = pares.concat(impares);
```

El resultado del método concat, no es más que la concatenación en un arreglo nuevo de los arreglos involucrados. En el ejemplo de arriba se han concatenado los arreglos **pares** e **impares** en un nuevo arreglo que se llama **todos**. Si por ejemplo, fuera necesario imprimir el número 7, se haría de la siguiente manera:

```
document.write("El valor del noveno elemento del arreglo todos es " +
                todos[8]);
```

El método concat puede recibir más de un argumento. La siguiente línea crea un arreglo que contiene los elementos de los arreglos pares, impares y vocales (**[0, 2, 4, 6, 8, 1, 3, 5, 7, 9, "a", "e", "i", "o", "u"]**):

```
todos = pares.concat(impares, vocales);
```

El método ***join*** agrupa los elementos de un arreglo en una variable de caracteres, separados por un carácter dado:

```
//La variable las_vocales tendrá como valor "a-e-i-o-u"
las_vocales = vocales.join("-");
```

Si se omite el carácter separador, los elementos serán separados por comas:

```
//La variable las_vocales tendrá como valor "a,e,i,o,u"
las_vocales = vocales.join();
```

El método ***pop*** elimina el último elemento de un arreglo, devolviendo el elemento eliminado, mientras que el método ***push*** inserta uno o mas elementos al final del arreglo, devolviendo el número de elementos del arreglo:

```
// La variable numero_eliminado tiene como valor 9
numero_eliminado = impares.pop();

numero_insertado = impares.push(9, 11);
/* Ahora el arreglo impares tiene como elementos [1, 3, 5, 7, 9, 11] y el valor de numero_insertado es 6 */
```

El método ***reverse*** invierte el orden de los elementos de un arreglo, el último elemento ocupará el primer lugar y el primero el último. Este método no devuelve ningún valor ni recibe ningún argumento.

```
/*las siguientes líneas imprimirán: El valor del noveno elemento del arreglo todos es 2*/
todos.reverse();
document.write("El valor del noveno elemento del arreglo todos es " + todos[8]);
```

Para eliminar y devolver el primer elemento de un arreglo se utiliza el método ***shift***. El método ***unshift*** a su vez permite insertar uno o más elementos en las primeras posiciones de un arreglo, devolviendo la longitud (número de elementos) del arreglo modificado.

El método ***slice*** extrae en un nuevo arreglo, una sección de un arreglo dado, sin alterar el contenido de éste último. La sintaxis del método slice es como sigue:

```
nuevo = unarreglo.slice(x, y);
```

el arreglo `nuevo` tendrá `y-x` elementos. El elemento `nuevo[0]` tendrá el mismo valor de `unarreglo[x]`; **nuevo[1]** valdrá lo mismo `unarreglo[x+1]` y así sucesivamente. Obviamente el número de elementos de `unarreglo` tiene que ser mayor o a lo sumo igual a `y`.

```
parte = pares.slice(1, 3);
```

En la línea de arriba, el arreglo `parte` estará compuesto por los elementos `[2, 4]`. Si se omite el segundo parámetro en el método:

```
parte = impares.slice(1);
```

los elementos del arreglo resultante serán [3, 5, 7, 9]. Es decir todos los elementos del arreglo impares excepto el primero (impares[0]).

Existe también el método ***splice***, que permite agregar y/o eliminar elementos de un arreglo simultáneamente. Veamos la sintaxis de este método:

```
splice(x, y, [elemento1, elemento2, …, elementoN]);
```

El parámetro ***x*** corresponde al índice del arreglo a partir del cual se realizarán los cambios. ***y*** corresponde al número de elementos que se quiere eliminar. Es claro que este último parámetro debe ser un entero positivo. Si ***y*** es igual a cero, entonces se debe especificar al menos un elemento a ser insertado. En el siguiente ejemplo, se agregarán al arreglo vocales las consonantes ***b***, ***c*** y ***d***, y, simultáneamente se eliminarán las vocales ***e***, ***i*** y ***o*** (línea 2), posteriormente en la línea 3 se removerá la última vocal (**u**), y por último se insertarán los caracteres ***e***, **f** y ***g*** en la línea 4:

```
vocales = new Array("a", "e", "i", "o", "u");
vocales.splice(1, 3, "b", "c", "d");
vocales.splice(4, 1);
vocales.splice(4, 0, "e", "f", "g");
```

El resultado final del arreglo vocales será vocales[“a”, “b”, “c”, “d”, “e”, “f”, “g”]

El objeto Date

Este objeto no tiene atributos (variables), pero cuenta con un gran número de funciones que permite trabajar y manipular variables de tiempo y fechas.

El valor de un objeto fecha en JavaScript corresponde al número de milisegundos transcurridos desde el 1 de enero de 1970 a las cero horas, hasta la fecha definida. Para instanciar un nuevo objeto Date, se utiliza la siguiente sintaxis:

```
fecha = new Date(parámetros);
```

Los parámetros pueden ser cualquiera de los siguientes:

- Una cadena de caracteres con el siguiente formato:

```
“Mes día, año hora:minutos:segundos”
 //Ejemplo
mi_Cumpleanos = new Date(“April 6, 2004 23:25:10”);
```

Si se omite hora, minutos o segundos, su valor se asumirá como cero.

- Un conjunto de valores numéricos enteros con el siguiente formato:

```
Año, mes, día, hora, minutos, segundos
```

```
//Ejemplo
mi_Cumpleanos = new Date(2004, 3, 6, 23, 25, 10);
```

Se debe tener cuidado cuando se usa este formato, ya que Enero (January) corresponde al mes cero (0) y Diciembre al mes 11.

- Si no se definen parámetros, el objeto tendrá como valor la fecha y hora actual del sistema:

```
Ahora = new Date();
```

Métodos del objeto Date

Aunque Date tiene gran cantidad de métodos para manipular fechas, solo se mencionarán algunos que permiten establecer y obtener los valores de sus componentes.

```
//Creación del objeto Fecha
Fecha = new Date("August 7, 1819");
/* El siguiente método obtiene el valor numérico del día (entre 1 y
31), en este caso 7 */
dia = Fecha.getDate();
/* El siguiente método obtiene el valor numérico del día, entre 0
(para Domingo - Sunday)  y 6 (para Sábado - Saturday), en este caso
devolverá 6. Sabían que la batalla de Boyacá fue un Sábado?*/
dia = Fecha.getDay();
/* En las siguientes líneas las variables hora, minutos y segundos,
tendrán como valor cero */
hora = Fecha.getHours();
minutos = Fecha.getMinutes();
segundos = Fecha.getSeconds();
//La variable annio, tendrá como valor 1819
annio = Fecha.getYear();
/* Mientras que la variable mes, tendrá como valor 7. Recuerde que
Enero corresponde en JavaScript al mes 0, Febrero al mes 1 y así
sucesivamente. */
mes = Fecha.getMonth();
```

Date tiene también métodos para establecer los valores anteriores:

```
//Creación del objeto Fecha, con la fecha actual del sistema
Fecha = new Date();
/* El siguiente método establece el valor del día en 7. El valor de
éste parámetro puede estar entre 1 y 31 */
Fecha.setDate(7);
/* En las siguientes líneas, la hora, minutos y segundos del objeto
Fecha, tendrán como valor 10. El rango para hora está entre 0 y 23,
mientras que para minutos y segundos está entre 0 y 59 */
Fecha.setHours(10);
Fecha.setMinutes(10);
Fecha.setSeconds(10);
//El año de fecha, tendrá como valor 1819
Fecha.setYear(1819);
/* Mientras que mes, tendrá como valor August (7).*/
Fecha.setMonth(7);
```

El objeto Math

El objeto Math es a mi modo de ver el más útil y el más fácil de manejar dentro de los objetos predefinidos de JavaScript. Cuenta con métodos y propiedades para el trabajo con funciones matemáticas básicas y constantes numéricas.

Atributos del objeto Math

La siguiente tabla resume la sintaxis de los principales atributos (variables), de este objeto. A diferencia de otros, para trabajar con el objeto Math, no es necesario crear una nueva instancia.

Atributo	Descripción	Valor que toma X
X = Math.E	Valor de la constante de Euler	2.718281828459045
X = Math.LN10	Logaritmo natural de 10	2.302585092994046
X = Math.LN2	Logaritmo natural de 2	0.6931471805599453
X = Math.LOG10E	Logaritmo en base 10 de E	0.4342944819032518
X = Math.LOG2E	Logaritmo en base 2 de E	1.4426950408889633
X = Math.PI	Valor de la constante PI	3.141592653589793
X = Math.SQRT1_2	Raíz cuadrada de 1/2	0.7071067811865476
X = Math.SQRT2	Raíz cuadrada de 2	1.4142135623730951

Métodos del objeto Math

La versión 1.3 de JavaScript tiene implementados aproximadamente 18 métodos dentro del objeto Math.

Sintaxis	Descripción	Ejemplo
Z = Math.abs(x)	Valor absoluto de x	Z = Math.abs(-10) Z = 10
Z = Math.acos(x)	Arco (en radianes) cuyo coseno es x	Z = Math.acos(0) Z = 1.5707963267948965
Z = Math.asin(x)	Arco (en radianes) cuyo seno es x	Z = Math.asin(0) Z = 0
Z = Math.atan(x)	Arco (en radianes) cuya tangente es x	Z = Math.atan(0) Z = 0
Z = Math.atan2(x,y)	Arco (en radianes) cuya tangente es x/y	Z = Math.atan2(0,100) Z = 0
Z = Math.ceil(x)	Devuelve el menor entero,	Z = Math.ceil(-30.5)

	mayor o igual que x	Z = -30
Z = Math.cos(x)	Coseno de x (en radianes)	Z = Math.cos(0) Z = 1
Z = Math.exp(x)	Devuelve el valor de E elevado a la x	Z = Math.exp(0) Z = 1
Z = Math.floor(x)	Devuelve el mayor entero, menor o igual que x	Z = Math.floor(-30.5) Z = -31
Z = Math.log(x)	Logaritmo natural de x	Z = Math.log(10) Z = 2.302585092994046
Z = Math.max(x,y)	Devuelve el mayor de los valores entre x e y	Z = Math.max(10.1, 10) Z = 10.1
Z = Math.min(x,y)	Devuelve el menor de los valores entre x e y	Z = Math.min(10.1, 10) Z = 10
Z = Math.pow(x,y)	Devuelve x elevado a la potencia y	Z = Math.pow(10, 2) Z = 100
Z = Math.random()	Devuelve un número aleatorio entre 0 y 1	Z = Math.random() Z = 0.7883010359801177
Z = Math.round(x)	Devuelve el valor de x redondeado al entero mas próximo	Z = Math.round(-30.5) Z = -30
Z = Math.sin(x)	Seno de x (en radianes)	Z = Math.sin(0) Z = 0
Z = Math.sqrt(x)	Raíz cuadrada de x	Z = Math.sqrt(100) Z = 10

El objeto String

Otro objeto muy importante para los propósitos del presente módulo es el objeto String. Permite a través de sus métodos trabajar con cadenas de caracteres aunque estas no estén rigurosamente declaradas como objetos.

El atributo más importante del objeto `String` es aquel que devuelve la longitud de una cadena de caracteres. En muchas ocasiones cuando se diseñan formularios, es necesario comprobar que ciertos campos de texto no excedan un límite determinado. El método `length` del objeto `String` devuelve el número de caracteres de una cadena dada:

```
cadena="Cuántos caracteres tendré?";
y=cadena.length;
document.write("La variable cadena tiene " + y + " caracteres,
incluidos los espacios.");
```

String tiene implementados alrededor de 26 métodos, entre los cuales se destacan:

El método ***anchor***, que permite crear un enlace con nombre, cuyo comportamiento es similar a escribir la etiqueta de HTML `<A NAME="nombre"> Nombre</A>`

```
cadena="nombre"
document.write(cadena.anchor("Nombre"));
```

Los métodos ***big, blink, bold, fixed, italics, small, strike, sub*** y ***sup*** muestran la cadena formateada con ciertas características. Por ejemplo:

```
cadena="Texto"
document.write(cadena.bold());
```

mostrará la palabra "**Texto**" en negrillas. De forma similar trabajan los otros métodos. Es importante señalar que el método `blink`, funciona únicamente con el navegador de Netscape.

El método ***cadena.charAt(x)***, devuelve el carácter ubicado en la posición `x` de la variable cadena (de izquierda a derecha). Es importante recordar que el primer carácter de un string tiene como índice cero. En el siguiente ejemplo, la variable `y` tendrá como valor la letra "`M`" y la variable `z` será igual a "`a`".

```
nombre="María Teresa";
y=nombre.charAt(0);
z=nombre.charAt(nombre.length-1);
```

cadena.charCodeAt(x), devuelve el código ASCII del carácter ubicado en la posición `x`.

```
nombre="María Teresa";
y=nombre.charCodeAt(1);
document.write("El código ASCII de a es "+y);
```

El método `concat`, concatena dos o más cadenas dentro de una nueva, que es devuelta por el método:

```
nombre1="Andres";
nombre2=" Francisco";
apellidos=" Gaitán Guzmán";
completo=nombre1.concat(nombre2, apellidos);
document.write("Nombres completos "+completo);
```

El contenido de la cadena `completo` en el anterior script es **Andrés Francisco Gaitán Guzmán**.

Para convertir todos los caracteres de una cadena dada a minúsculas, se utiliza el método ***toLowerCase()***, mientras que ***toUpperCase()*** convierte todos los caracteres de un string a mayúsculas:

```
nombre="Andrés";
min=nombre.toLowerCase();
may=min.toUpperCase();
document.write("En minúsculas "+min);
document.write(" en mayúsculas "+may);
```

Los métodos ***match, search*** y ***replace*** serán tratados en el capítulo correspondiente a Expresiones Regulares.