VISUAL BASIC SCRIPT

INTRODUCCIÓN

El Visual Basic Script (en adelante VBScript) es un lenguaje de script, directamente derivado de Visual Basic. Los lenguajes de script son versiones recortadas de otros lenguajes. Estas versiones se usan para su integración en páginas web. Un código escrito en un lenguaje de script se incorpora directamente dentro de un código HTML y se ejecuta interpretado, no compilado. Este temario trata el lenguaje VBScript desde el principio; sin embargo presuponemos por parte del lector los necesarios conocimientos de HTML y del entorno web. Para incorporar un fragmento de código script en una página HTML se introduce el script entre los tags **<SCRIPT>** y **<SCRIPT>** . Dos son los lenguajes de script que hay en la actualidad: el VBScript (derivado de Visual Basic) y el JavaScript (derivado de Java). En este temario nos ocuparemos del primero de ellos, siendo el segundo objeto de estudio en su correspondiente temario. Para insertar código VBScript en una página HTML añadiremos al tag **<SCRIPT>** el parámetro LANGUAGE="VBScript" , que determina cual de los lenguajes de script utilizamos.

Decimos que los lenguajes de script se ejecutan interpretados, no compilados. Esto significa que un código escrito en un lenguaje de script no sufre ninguna transformación previa a su ejecución. Cada línea de código es traducida a lenguaje máquina justo antes de su ejecución. Después es ejecutada y la traducción no se conserva en ningún sistema de almacenamiento (como discos, cintas, etc). Si es necesaria otra ejecución, el intérprete se verá abocado a realizar una nueva traducción de cada línea de código. Sin embargo el lenguaje Visual Basic, del cual deriva el VBScript, es un leguaje compilado. Esto significa que un código en Visual Basic sufre un proceso global de traducción a lenguaje máquina. Todo el código es traducido de una sola vez y el resultado de esa traducción se almacena en el disco con lea extensión **.EXE**. Cuando llega el momento de la ejecución, se ejecuta el código compilado, no el código original del programa (llamado código nativo o código fuente). Cada sistema tiene sus ventajas e inconvenientes. Veámoslos:

INTERPRETACIÓN

VENTAJAS	INCONVENIENTES
• El código es cómodo para depurar, ya que no es necesario volver a compilar tras un cambio. • No es necesario disponer de un compilador, ya que el intérprete (que forma parte del navegador) ejecuta el script. • El mantenimiento es fácil y rápido, por parte del autor o de otro programador.	• La ejecución se ralentiza, al ser necesaria la interpretación línea a línea cada vez. • El código es visible y puede ser objeto de plagio por parte de otras personas. • El usuario tiene acceso al código y puede modificarlo, estropeando alguna operación.

COMPILACIÓN

VENTAJAS	INCONVENIENTES
• El código compilado se ejecuta muy rápido, al no ser necesaria una traducción cada vez. • El código compilado no puede ser "abierto" por otras personas. No es necesario transmitir el código fuente. • El código compilado puede estar, íntegramente, incluido en un solo fichero.	• Es necesario disponer de un compilador-linkador para el proceso de la compilación. • El código compilado suele ocupar bastante en disco, ya que incorpora en el propio código algunas librerías del sistema. • Depurar un programa implica volver a compilar tras los cambios.

El código en VBScript puede, además, estar diseñado para su ejecución en el lado del cliente o en el del servidor. La diferencia es que un código que se ejecuta en el lado del servidor no es visible en el lado del cliente. Este recibe los resultados, pero no el código. El código que se debe de ejecutar en el lado del servidor estará incluido en la página web correspondiente entre los tags **<% y %>** . Además habrá que renombrar la página para aplicarle la extensión **.asp** (Active Server Page -página activa en servidor-). El funcionamiento intrínseco de la ASP's está fuera de los objetivos de éste temario, aunque lo comentaremos por encima al final. Este texto se refiere, por tanto, al lenguaje VBScript en el lado del cliente.

El lenguaje VBScript solo funciona correctamente con el navegador Internet Explorer 4.0 y superiores, por lo que será necesario disponer del mismo. Los usuarios de otros navegadores no podrán disfrutar de la potencia y versatilidad del VBScript.

CAPITULO 1: La orientación a objetos

La tendencia actual en lenguajes de programación de alto nivel es que sean lenguajes orientados a objetos. La Programación Orientada a Objetos (POO) es una filosofía que se basa en considerar cada elemento que se usa en un programa como un objeto individual. Así, por ejemplo, el documento que se está ejecutando es un objeto; cada texto que contiene es un objeto; cada imagen, cada sonido, cada vídeo son objetos. También lo es la zona de trabajo o ventana donde se ejecuta un documento o programa. Los objetos tienen una estructura y se organizan de una determinada manera, como vamos a ver a continuación.

PROPIEDADES

Los objetos tienen una serie de características (se llaman **propiedades**) que los definen. Por ejemplo. Un texto es un objeto; el color del texto es una propiedad. Cada objeto tiene un conjunto de propiedades que le son inherentes y que constituyen, junto con los **métodos**, la **clase** del objeto. Los objetos se hallan clasificados según una **jerarquía**, en la que hay objetos de mayor nivel y otros de menor nivel. La jerarquía se refiere a que objetos están contenidos en otros (**son propiedad de** otros). Así pues un texto es un objeto propiedad del documento que lo contiene; una celda de una tabla es un objeto propiedad de la tabla a la que pertenece. Un objeto que contiene a otro es **el objeto padre** de aquel al que contiene; el objeto contenido es **un objeto derivado** de aquel que le contiene. Un objeto puede tener varios objetos derivados, pero cada objeto sólo tiene un objeto padre. Un objeto tiene, por defecto, las mismas propiedades (la misma clase) que su objeto padre. Es lo que se conoce como herencia. Sin embargo algunas o todas esas propiedades pueden ser alteradas en algún momento por el propio programa o por las acciones del usuario. Las clases no son inmutables.

METODOS

Los objetos tienen además, unas funciones inherentes que pueden ejecutar para obtener determinados resultados. Estas funciones propias de los objetos se llaman **métodos**, para evitar confusiones con las **funciones de usuario** (aquellas que el programador escribe e incorpora a su código). Por ejemplo. El objeto que corresponde al documento activo tiene un método que permite escribir texto en la pantalla, un objeto de tipo fecha contiene un método que permite obtener la hora del sistema, etc.

OTROS ASPECTOS DE LA POO

Existen dos conceptos importantes en POO. Son las **instancias** y las **implementaciones**. Una instancia es una referencia a un objeto o a una propiedad del mismo. Cuando manejamos las propiedades de un objeto refiriéndonos a él por una instancia, estamos afectando directamente al objeto. No a una copia del mismo. Esto es importante porque si modificamos una copia de un objeto, el objeto original no resulta modificado. Pero si actuamos sobre una instancia a un objeto, el objeto original resultará afectado. Una implementación es la programación de una función que se asigna a una clase determinada para que los objetos de esa clase dispongan de esa función como un método propio.

POO EN VBSCRIPT

El VBScript, como el Visual Basic del que deriva, incorpora la POO, aunque en menor grado que otros lenguajes actuales. En realidad la estructura de este lenguaje es, hasta cierto punto, un poco anárquica, vestigio de las antiguas versiones de Basic, que eran totalmente procedimentales. Otros lenguajes actuales, como Java o C++ son mucho mas rígidos en su concepción. Por ejemplo. Visual Basic no es Case Sensitive, esto es, no distingue entre mayúsculas o minúsculas en el código. Esta es sólo una de las características menores del lenguaje. Veremos algunas más en el siguiente capítulo. En el Apéndice A tenemos un listado de la jerarquía de los objetos, así como de las propiedades y métodos que acepta cada uno de ellos.

CAPITULO 2: Empezando con VBScript

Como hemos dicho anteriormente, los lenguajes de Script se insertan directamente en el listado de una página HTML y realiza ciertas operaciones que el lenguaje HTML, por si sólo, no es capaz de llevar a cabo. Un código VBScript en una página web tiene el siguiente el siguiente aspecto general:

```
<SCRIPT LANGUAGE = "VBScript">
   código Visual Basic Script
</SCRIPT>
```

Este formato es el que se usa, insisto una vez más, para escribir código VBScript ejecutable en el lado del cliente, no del servidor. El código VBScript se puede teclear dentro de la cabecera o del cuerpo de la página web. Lo normal es incluir en la cabecera el código que debe estar en memoria antes de la ejecución de la página y en el cuerpo el que debe ejecutarse con la página. A lo largo de este temario veremos varios ejemplos de ambos casos que nos disiparán las dudas que podamos tener al respecto. Por supuesto puede haber código VBScript dentro de la cabecera y dentro del cuerpo, simultáneamente, si el diseño de nuestra página lo requiere. Y basta ya de cháchara. Empecemos a ver algunas de las cosas que podemos hacer con nuestro flamante VBScript.

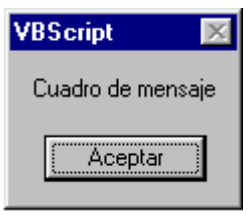
NUESTRO PRIMER EJEMPLO

En los siguientes ejemplos de código verás la forma en que se escribe este lenguaje: En cada línea física, una instrucción con sus correspondientes parámetros, si los tiene.

Veamos un ejemplo de una sencillísima página web que incluye código VBScript:

```
<HTML>
<HEAD>
<TITLE>Cuadro de mensaje</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   MSGBOX ("Ejemplo de mensaje")
</SCRIPT>
</BODY>
</HTML>
```

Este código incluye la instrucción MSGBOX ("Cuadro de mensaje") , que genera en pantalla lo siguiente:



En este ejemplo vemos varias cosas. En primer lugar la estructura de la instrucción en sí. Tal como hemos dicho, tecleamos en una sola línea la instrucción **MSGBOX** y los parámetros que recibe. En este

caso solo el mensaje a mostrar, entre paréntesis y comillas **("Cuadro de mensaje")** . En la línea de código no hay nada más. No debe haber nunca mas de una instrucción en la misma línea, salvo un caso puntual del que hablaremos mas adelante.

En otro orden de cosas vemos que la instrucción MSGBOX genera una caja de aviso en pantalla con un mensaje y un botón **Aceptar**. Llegado este punto, la ejecución se detiene hasta que el usuario pincha el botón. Así nos aseguramos de que verá el mensaje. El mensaje se pone entre paréntesis y, si se trata de una cadena literal, entre comillas. También se puede poner el nombre de una variable sin utilizar comillas. En este caso la caja de aviso mostrará el contenido de la variable. Hablaremos de variables y de tipos de datos un poco mas adelante.

Una última reflexión respecto a este ejemplo. Hemos dicho anteriormente que VBScript no es sensible al uso de mayúsculas o minúsculas. Esta instrucción hubiera funcionado exactamente igual si la hubiéramos escrito **msgbox** o **MsGbOx**, por ejemplo. El utilizar en nuestro código letras mayúsculas para las palabras reservadas del lenguaje y minúsculas para lo demás (excepto iniciales, nombres propios, etc.) obedece a una razón práctica: facilitar la legibilidad del código. El lector podrá usar este sistema o, si lo prefiere, un editor de textos dotado de Chroma Code. El Chroma Code es un sistema que hace aparecer las palabras reservadas en un color, los nombres de variables en otro, las cadenas en otro, etc. Como es lógico, el editor dotado de Chroma Code debe ser específico para Visual Basic, o no funcionará adecuadamente.

Dada la importancia del hecho de que VBScript no es Case Sensitive (de hecho es uno de los pocos lenguajes actuales que presenta esta característica), vamos a insistir una vez más en ello, ilustrándolo con un ejemplo que lo demuestra.

El código es el siguiente:

```
<HTML>
<HEAD>
<TITLE>Prueba de la no distinción entre MAYÚSCULAS y minúsculas</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   variable = 1
   VARIABLE = 2
   MSGBOX (variable)
   MSGBOX (VARIABLE)
</SCRIPT>
</BODY>
</HTML>
```

Este código asigna a la variable llamada **variable el valor 1**. Después asigna **a la misma variable el valor 2**. En las dos cajas de mensaje se obtiene el mismo valor. Así:

al aceptar aparece de nuevo

lo que demuestra que ambas variables son la misma.

COMENTARIOS

En VBScript, como en cualquier lenguaje de programación, se pueden insertar comentarios en el código con el fin de facilitar la legibilidad y el mantenimiento del mismo. El intérprete ejecuta el código ignorando los comentarios. Al contrario de lo que muchos programadores novatos piensan, los comentarios no afectan a la velocidad ni a ningún otro aspecto de la ejecución, por lo que podemos usarlos libremente, con toda la profusión necesaria para que nuestro código sea fácil de comprender. Para insertar un comentario, tecleamos la palabra clave REM o bien una comilla simple. Todo lo que haya en la línea de ahí en adelante será considerado por el intérprete como un comentario.

Por ejemplo:

```
<HTML>
<HEAD>
<TITLE>Prueba de la no distinción entre MAYÚSCULAS y minúsculas</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   variable = 1  ' Esto es un comentario.
   VARIABLE = 2
   MSGBOX (variable)  REM Esto también lo es.
   MSGBOX (VARIABLE) ' Aunque este formato es mas habitual.
</SCRIPT>
</BODY>
</HTML>
```

El resultado de este código será, exactamente, el mismo que en el caso anterior. Así pues, en nuestros programas en VBScript deberemos insertar comentarios siempre que lo consideremos oportuno, sin excedernos cuando no resulten necesarios, ya que a lo que si afectan los comentarios, lógicamente, es al tamaño de nuestro archivo.

El argumento de MSGBOX (lo que va entre paréntesis) puede ser simple o compuesto. Hasta ahora hemos visto ejemplos que mostraban un argumento simple. Vamos a ver ahora un ejemplo que muestra un argumento compuesto de una cadena alfanumérica y una variable (lo que se llama una **concatenación**):

Ejemplo de concatenación

```
<HTML>
<HEAD>
<TITLE>Muestra de argumento compuesto</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   variable = "amigo"
   MSGBOX ("Hola, " + variable) ' Ejemplo de concatenación.
</SCRIPT>
</BODY>
</HTML>
```

Este ejemplo dará como resultado lo siguiente:

En este caso hemos concatenado una cadena alfanumérica con una variable cuyo contenido es también alfanumérico, y hemos usado para ello el **signo + de la suma**. Si queremos concatenar una cadena alfanumérica con una variable cuyo contenido es un valor numérico emplearemos el signo **& (Ampersand)** , como en el siguiente ejemplo:

Ejemplo del uso de &

```
<HTML>
<HEAD>
```

```
<TITLE>Muestra de argumento compuesto</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   variable = 1
   MSGBOX ("Eres el Nº " & variable) ' Ejemplo de concatenación.
</SCRIPT>
</BODY>
</HTML>
```

Este será el resultado:

El signo Ampersand puede emplearse también en el caso de concatenación de cadenas alfanuméricas con otras cadenas y/o variables de cadena. Por supuesto las concatenaciones pueden ser múltiples, es decir se pueden concatenar varias variables y/o cadenas alfanuméricas.

CAPITULO 3: Datos y Variables en VBScript

Al contrario que en otros lenguajes de programación, en VB solo existe un tipo general de datos que se conoce con el nombre de **Variant**. En otros lenguajes existen datos de tipo String (Cadena) para almacenar contenidos alfanuméricos, distintos tipos de datos numéricos enteros y en coma flotante, datos booleanos, etc. Esta característica es muy útil, ya que permite reasignar un valor de un tipo a una variable de otro tipo. En la actualidad es el único lenguaje de alto nivel que implementa esta característica Los datos se clasifican en subtipos en función del contenido en un momento dado. Así se logra toda la funcionalidad de gestión de datos en lenguajes de alto nivel, pero con una mayor flexibilidad. Para cambiar una variable de un subtipo a otro, es suficiente con asignarle un dato de diferente tipo. Por ejemplo. El siguiente fragmento de código daría un error en lenguajes como C++ o Java; sin embargo, en VB es absolutamente correcto:

```
Variable = 1
' más código
' más código
' más código
' más código
Variable = "cadena"
```

Mas adelante veremos ejemplos operativos que ilustrarán esta cualidad. En el Apéndice B tienes un listado completo de los subtipos de datos aceptados por VB (y, por tanto, por VBScript). A lo largo de este temario veremos ejemplos de uso de los distintos subtipos, que ilustrarán su funcionamiento. Es necesario recalcar un hecho. Al pertenecer todos los datos a un tipo único, cuando le damos un valor a un dato se constituye, automáticamente, del subtipo adecuado para ese valor. Este proceso es totalmente transparente al programador quien, de esta forma, no necesita preocuparse de declarar un dato como de uno u otro tipo.

VARIABLES

Los datos que se manejan en un lenguaje de programación y cuyo valor puede cambiar durante la ejecución de un programa se conocen con el nombre genérico de variables. Podemos concebir una variable como una caja cuyo contenido varía a lo largo de la ejecución. La caja tiene un nombre que sirve para identificarla y poder utilizar el contenido o valor. Los nombres de variables deben empezar con una

letra y pueden tener letras, números o el signo de subrayado (único signo de puntuación que se admite en el nombre de una variable). Las letras que formen parte del nombre de una variable deberán ser de la alfabetización internacional, no de la española o específica de algún idioma en particular. Así pues, no deberá haber en un nombre de variable letras como la ñ, letras acentuadas, la ç, etc. Un nombre de variable no deberá contener caracteres especiales (como p.e. $,%,&,^, etc.). Tampoco deberá contener espacios en blanco, puntos, comas, ni ningún otro signo de puntuación. Por supuesto, el contenido de una variable alfanumérica SI podrá contener cualquier cosa que deseemos. Además deberemos tener cuidado de no emplear como nombres de variable las palabras clave del lenguaje. Los siguientes ejemplos de variables serán correctos en VBScript y en VB:

```
Variable = "Esto está en una cadena."
cosa = "*&%$*/87*/&)ñ"
Edad2 = 74
Casado_si_o_no = true
```

Sin embargo, los siguientes ejemplos ilustran declaraciones de variables que el lenguaje no aceptará:

```
72Edad = 28 ' No lo admite por empezar con un número
MsgBox = false ' No lo admite por ser una palabra reservada
Año = 1977 ' No lo admite por contener el nombre una ñ
Pitón& = "Esto no vale nada" ' No vale por tener una letra acentuada y un &
```

En otro orden de cosas, para usar una variable es necesario dar dos pasos: declararla e inicializarla. La declaración es la forma de decirle al lenguaje que se va a usar una variable y se hace con la palabra reservada DIM, seguida del nombre de la variable. Así:

```
DIM variable
```

Esto reserva espacio en memoria para la variable. Sin embargo aún no le hemos asignado ningún contenido. En realidad tiene un contenido de subtipo **Null (nulo)** . La inicialización de la variable será la que le asigne su primer contenido aunque, como ya sabemos, éste podrá cambiar a lo largo de la ejecución. La inicialización es, simplemente una asignación. Por ejemplo:

```
variable = "Cacharro"
```

La declaración de la variable es opcional. Es decir, si no se hace la declaración de una variable, ésta se llevará a cabo, de forma automática al realizar la inicialización. Sin embargo es conveniente realizar las declaraciones de forma manual (escribiendo la instrucción **DIM**), a fin de incrementar el nivel de estructuración de nuestros programas y facilitar la legibilidad de los mismos. La declaración debe ir SIEMPRE antes de la inicialización. Lo correcto es realizar la declaración de todas las variables al principio de nuestro código. Existe una forma de asegurarnos de que tengamos que hacer las oportunas declaraciones. Es incluyendo la instrucción **OPTION EXPLICIT** en nuestro código. Si incluimos esa línea, el programa no podrá usar ninguna variable que no haya sido declarada. Nosotros no la habíamos incluido en los ejemplos anteriores, ya que aún no habíamos hablado de ella, pero la incluiremos a partir de ahora en todos nuestros ejemplos, ya que constituye, sobre todo, una buena práctica de programación.

MATRICES

Hasta ahora hemos hablado de variables simples, que constituyen lo que se llama un **par nombre-valor**, es decir son un nombre de dato, con un valor asignado. Sin embargo no hay ninguna razón por la que no podamos formar y utilizar estructuras de datos mas complejas. Por ejemplo. Supongamos que necesitamos gestionar un conjunto de variables que contengan las edades de los niños de un colegio. Estas variables tienen todas algo en común, así que parece lógico tratar de establecer una relación de similitud entre ellas. Lo que hacemos es crear una **tabla o matriz** de variables. Una matriz es un conjunto de variables que reciben todas el mismo nombre. En el ejemplo que nos ocupa, este nombre podría ser, por ejemplo, edades. Ahora bien. Si todas las variables o elementos de la matriz reciben el mismo nombre ¿cómo los identificamos de forma inequívoca? Para ello usamos un **índice**. Un índice es un número de orden que identifica a cada **elemento** de la matriz. Así pues ya no nos referimos a cada variable con su nombre, si no con el nombre de la matriz en la que está la variable, seguido del número de orden, o lugar que ocupa, esa variable en la matriz. Asumimos que los distintos elementos está colocados uno detrás de otro en fila. **El primer elemento** se identifica con el número 0, el segundo con el 1, el tercero con el 2, y así sucesivamente. Bien esta es la teoría. Veamos como utilizamos matrices. En primer lugar vamos a

declarar una matriz de cinco elementos. Para seguir con el ejemplo anterior, será una matriz donde introduciremos las edades de cinco niños. A la matriz la llamaremos edades:

```
DIM edades (4)
```

El lector pelín avispado ya se habrá dado cuenta de que hemos declarado la matriz con cuatro elementos, cuando habíamos dicho que la íbamos a declarar con cinco. Ocurre que, como el primer elemento se le conoce con el número 0, en realidad la instrucción anterior crea sitio para cinco elementos, del 0 al 4 (0, 1, 2, 3 y 4). Es importante reseñar que una matriz, una vez declarada, no podrá redimensionarse. Es decir, si hemos creado esta matriz con cinco elementos no podremos meter mas elementos en la misma. Esto no supone ningún problema si necesitamos menos de cinco elementos ya que, los que no necesitemos, podemos dejarlos vacíos (Null), pero no podremos usar NUNCA mas de cinco elementos en esa matriz. Así pues, una matriz debe ser declarada con el número MÁXIMO de elementos que deberá contener. Si no sabemos cuantos elementos podrá contener una matriz como máximo, no podremos usar matrices.

Bien. Ahora veamos como asignar contenidos a una matriz. El nombre de un elemento de una matriz está compuesto por el nombre de la matriz y el número de índice del elemento, este último entre paréntesis. Por lo tanto, la asignación de un valor a un elemento se realizará de la siguiente manera:

```
Edades (0) = 14
```

Las matrices, también llamadas **arrays**, son una herramienta muy potente y versátil. Puedes, por ejemplo, asignarle un contenido alfanumérico a un elemento de la matriz y un contenido numérico a otro elemento de la misma matriz. Sin embargo esta cualidad no es de uso muy frecuente, por la propia naturaleza. Recordemos que una matriz se emplea para almacenar un conjunto de datos relacionados entre sí. Mira el siguiente ejemplo:

Uso de matrices

```
<HTML>
<HEAD>
<TITLE>Prueba de Matrices de varios tipos</TITLE>
<SCRIPT LANGUAGE = "VBScript">
   DIM matriz (1)
   matriz (0) = "A ver que pasa"
   matriz (1) = 6 * 3 ' El asterisco se usa como signo de multiplicación.
   MSGBOX (matriz (0))
   MSGBOX (matriz (1))
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Este código da como resultado lo siguiente:

Al pulsar el botón **Aceptar** aparecerá lo siguiente:

Lo que nos demuestra que se respeta el contenido de cada elemento de la matriz.

Otra cosa respecto de las matrices. Hasta ahora hemos visto matrices que son, por así decirlo, una fila de elementos. Pero puede haber matrices de varias filas, de forma que constituyan un cuadro de elementos. Son las llamadas matrices bidimensionales. De la misma forma que un elemento de una matriz unidimensional se identifica por un índice que se refiere a su posición en la fila, un elemento de una matriz bidimensional se identifica por dos índices. Un indica la columna y otro la fila que ocupa el elemento en la tabla. La forma de declarar una matriz bidimensional es la siguiente:

```
DIM matriz_bidimensional (10,20)
```

Ten en cuenta que todo lo que hemos dicho de matrices unidimensionales es válido también para matrices bidimensionales, sólo que en este caso entre los paréntesis aparecen dos índices, separados por una coma.

Para asignar un contenido a un elemento de una matriz bidimensional se emplea el siguiente formato:

```
matriz_bidimensional (4,7) = "Esto es un elemento de una matriz"
```

Por supuesto se pueden usar matrices de mas de dos dimensiones (hasta un máximo de 60 dimensiones) indicando entre paréntesis un índice por cada dimensión. En la práctica se usan, como mucho, matrices de cuatro dimensiones.

Por lo demás, recuerda que con los elementos de una matriz se pueden efectuar las mismas operaciones que con variables simples. Hasta ahora solo hemos visto unas pocas posibilidades. Veremos más en próximos capítulos.

CAPITULO 4: Entrada y salida de datos

Uno de los aspectos mas importantes y llamativos de un lenguaje de programación es la comunicación con el usuario. Esta comunicación se establece en dos direcciones fundamentales: la salida de datos por pantalla y la introducción de datos por teclado. En este capítulo vamos a ver los aspectos mas básicos de ambas, de forma que ya podremos establecer unos canales de comunicación con el usuario. Mas adelante en éste mismo temario veremos otras técnicas que dotarán a nuestra código de dinamismo e interactividad.

SALIDA POR PANTALLA

Hemos visto hasta ahora una manera de obtener en pantalla algunos resultados. La instrucción MSGBOX () Nos permite sacar una cadena alfanumérica, el contenido de una variable o combinaciones de ambas en un cuadro don el aspecto típico de Windows. Este cuadro está dotado de un botón Aceptar que el usuario debe pulsar para que prosiga la ejecución. Además, como sin duda has podido comprobar, al pulsar el botón el cuadro desaparece. Bien. Supongamos que lo que queremos ahora es obtener un texto en la pantalla, sin necesidad de ningún botón, ni cuadro, ni detención de la ejecución, y además que no desaparezca. Hemos dicho que VBScript es un lenguaje orientado a objetos. Y el documento activo (programa en ejecución) es un objeto llamado, precisamente, **document**. Este objeto tiene un método de escritura, llamado write (), que nos permite escribir en la pantalla. Si esto te resulta un poco arcano, revisa la primera parte de este temario que versa sobre programación orientada a objetos y mira el Apéndice A. Bien; veamos un ejemplo de lo que acabamos de comentar.

Uso de Document.Write

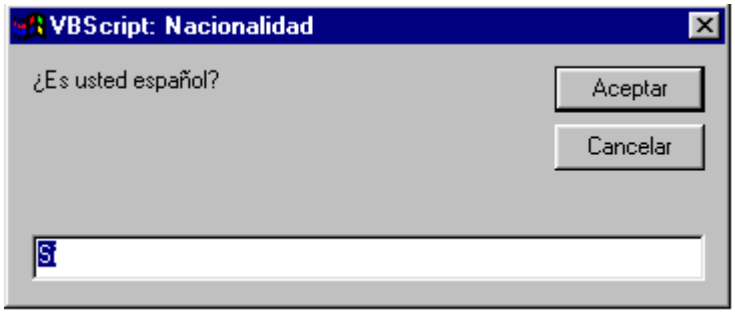
```
<HTML>
<HEAD>
<TITLE>Prueba de document.write ()</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   DOCUMENT.WRITE ("Hola desde VBScript")
```

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

El resultado de este código sería ver en pantalla la frase **Hola desde VBScript**. Como en el caso de MSGBOX, la impresión puede ser una combinación de cadena /s alfanumérica /s y/o variable /s, y las concatenaciones se rigen por las mismas normas que ya conocemos.

ENTRADA POR TECLADO

Una de las formas mas atractivas de introducir datos simples por teclado es la instrucción INPUTBOX (). Esta instrucción abre una caja de diálogo, con el siguiente aspecto:



En esta caja distinguimos lo siguiente:

1. La pregunta que se le hace al usuario. Cuando se espera que el usuario introduzca algo por teclado, suele ser la respuesta a alguna pregunta. En este caso la pregunta es "¿Es usted español?".
2. Una línea en blanco (normalmente llamada "Caja de texto" donde el usuario introduce su respuesta. Esta caja de texto puede aparecer vacía o con una posible respuesta por defecto. En este último caso la respuesta por defecto aparece seleccionada (texto blanco sobre fondo azul).
3. Un botón de Aceptar. Una vez introducida la respuesta el usuario pulsa la tecla INTRO del teclado o el botón Aceptar para validarla. Llegados a este punto conviene aclarar que, de forma similar a como ocurre con MSGBOX, la ejecución del programa queda detenida hasta que el usuario responde a la pregunta.
4. Un botón Cancelar. Este es nuevo. No lo habíamos visto hasta ahora. Se usa el botón si el usuario quiere dejar la pregunta sin responder y continuar adelante con la ejecución del programa.
5. En la parte superior de la ventana, y a todo lo ancho de la misma, aparece una banda azul con texto blanco. Esta banda es la barra de título.

Bien. Conozcamos la sintaxis de INPUTBOX (). Es la siguiente:

```
INPUTBOX (Pregunta, Título, Respuesta, pos x, pos y)
```

Como vemos esta instrucción puede recibir varios parámetros separados por comas. Son los siguientes:

1. Pregunta. Es la pregunta que se le formula al usuario y a la que deberá responder.
2. Título. Es un literal que aparecerá en la barra de título.
3. Respuesta. Es la respuesta por defecto que queremos ofrecerle al usuario.
4. pos x - pos y. Son las coordenadas donde queremos que se sitúe la esquina superior izquierda de la caja de diálogo. Estas coordenada se expresan en twips, que es una unidad de medida propia de Windows. Para coger soltura con esta medida lo mejor es que practiques con distintos valores.

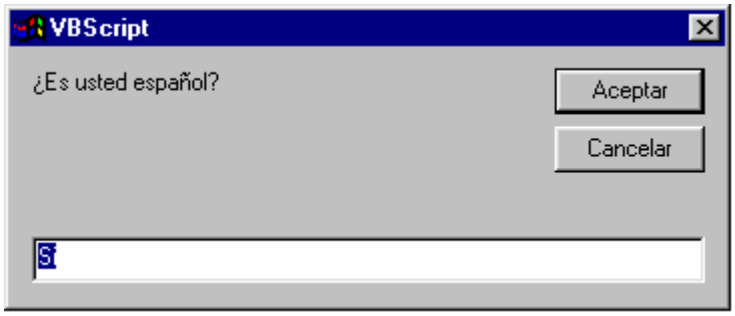
En el caso de la caja de diálogo que hemos visto al principio de esta página, la instrucción que la genera es:

```
resp = INPUTBOX ("¿Es usted español?", "Nacionalidad", "Si", 1500,1500)
```

En esta instrucción vemos que la respuesta del usuario se almacena en una variable llamada **resp** para su posterior uso. El único parámetro obligatorio es el primero, es decir, el enunciado de la pregunta. Los demás son opcionales. Supongamos que no queremos, por ejemplo, que aparezca el titulo **Nacionalidad**. En ese caso teclearemos la instrucción de la siguiente manera:

```
resp = INPUTBOX ("¿Es usted español?", , "Si", 1500, 1500)
```

El resultado será:



Si el usuario pincha el botón cancelar, la variable a la que se asigna la repuesta quedará con el valor Null.

En principio, todas las respuestas que el usuario introduzca por este sistema son procesadas como cadenas alfanuméricas. Mas adelante veremos que ocurre si se espera del usuario una respuesta numérica, de fecha, etc.

CAPITULO 5: Condicionales

Cuando se introduce una respuesta a una pregunta, o cuando se obtiene un valor como resultado de una operación, o en otras muchas situaciones, es necesario que el código "decida" si va a hacer una cosa u otra (o ninguna) con lo que se ha obtenido. Por ejemplo. Si a la pregunta de si el usuario es español éste respondió afirmativamente, habrá que preguntarle cual es su numero del DNI. En caso contrario habrá que preguntarle por su número de pasaporte. Hay varias posibilidades de establecer condicionales durante la ejecución de un programa. Veámoslas:

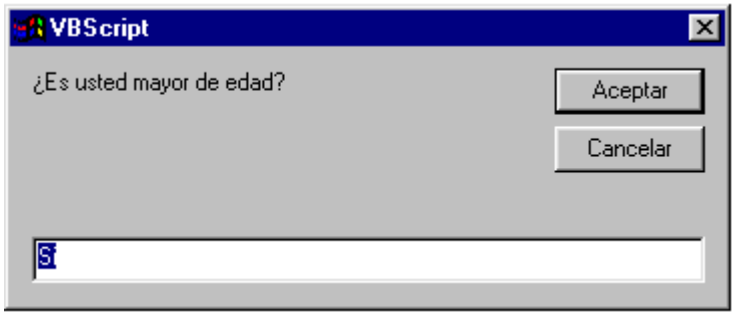
SENTENCIAS IF... THEN... ELSE

La sentencia **IF** significa, en castellano **si (condicional, no afirmativo)** . La palabra **THEN** significa **entonces o como consecuencia**. La palabra **ELSE** se podría traducir por **en caso contrario**. Cuando queremos que el programa realice una o mas operaciones si se cumple una condición, podemos usar una estructura básica como la del siguiente ejemplo:

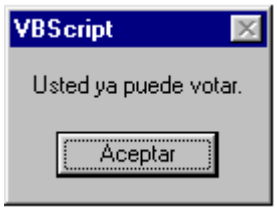
Condicionales

```
<HTML>
<HEAD>
<TITLE>Prueba básica de condicional</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   edad = INPUTBOX ("¿Es usted mayor de edad?",,"Si")
   IF (edad = "Si") THEN
      MSGBOX ("Usted ya puede votar")
   END IF
</SCRIPT>
</BODY>
</HTML>
```

Este código muestra la siguiente ventana en la pantalla:



Si el usuario responde Si (Aceptando la respuesta por defecto), el programa muestra el siguiente resultado:



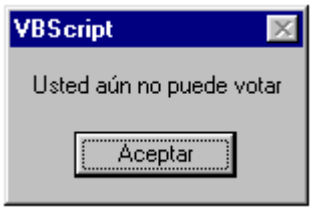
En caso de que el usuario responda otra cosa o pinche en cancelar el programa no muestra ningún resultado. Como hemos visto, la condición va entre paréntesis. Esto no es obligatorio en VBScript, pero facilita la legibilidad del código. Nosotros lo haremos siempre así, por sistema.

Vamos a sofisticar un poco mas nuestro programa:

Condicionales

```
<HTML>
<HEAD>
<TITLE>Prueba básica de condicional</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   edad = INPUTBOX ("¿Es usted mayor de edad?",,"Si")
   IF (edad = "Si") THEN
      MSGBOX ("Usted ya puede votar")
   ELSE
      MSGBOX ("Usted aún no puede votar")
   END IF
</SCRIPT>
</BODY>
</HTML>
```

En caso de una respuesta diferente a Si por parte del usuario, ahora veremos en pantalla lo siguiente:



Como vemos, un condicional va encerrado entre las instrucciones IF y END IF. A continuación de la palabra clave IF, en la misma línea de código aparece la condición que determina la ejecución de unas instrucciones o de otras. Después de la palabra clave THEN aparecen las instrucciones que han de ejecutarse si se cumple la condición. A continuación de la palabra clave ELSE aparecen las instrucciones

que han de ejecutarse si la condición no se cumple. Todo el bloque del condicional termina con las palabras clave END IF. Por lo tanto, la estructura general de este tipo de condicionales es la siguiente:

```
IF (condición) THEN
   BLOQUE DE INSTRUCCIONES 1
ELSE
   BLOQUE DE INSTRUCCIONES 2
END IF
```

CONDICIONALES ANIDADOS

El ejemplo anterior evalúa una condición. Si se cumple la condición hace una cosa y si no se cumple hace otra. Ahora supongamos la siguiente estructura:

```
IF (condición_1) THEN
   BLOQUE DE INSTRUCCIONES 1
ELSE
   IF (condición_2) THEN
      BLOQUE DE INSTRUCCIONES 2
   ELSE
      BLOQUE DE INSTRUCCIONES 3
   END IF
END IF
```

Esta estructura es un poco mas compleja. Si se cumple la condición_1 ejecutará el BLOQUE DE INSTRUCCIONES 1. Si no se cumple evalúa la condición_2. En caso de cumplirse ésta, ejecuta el BLOQUE DE INSTRUCCIONES 2. Solo en caso de que las dos condiciones sean falsas se ejecutará el BLOQUE DE INSTRUCCIONES 3. Veamos un ejemplo:

Condicionales anidados

```
<HTML>
<HEAD>
<TITLE>Prueba básica de condicional</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   edad = INPUTBOX ("¿Cuál es su edad?",,"0")
   IF (edad < 18) THEN
      MSGBOX ("Eres un jovenzuelo")
   ELSE
      IF (edad > 65) THEN
         MSGBOX ("Usted ya está jubilado")
      ELSE
         MSGBOX ("Usted es adulto")
      END IF
   END IF
</SCRIPT>
</BODY>
</HTML>
```

En primer lugar una observación. En el capítulo anterior decíamos que INPUTBOX () recibía un valor alfanumérico, aunque aquí recibe un valor numérico (la edad) y lo procesa sin problemas como un número. Esto es posible gracias a la propia estructura de datos del lenguaje (ver el capítulo 3: Datos y variables en VBScript).

Bien. Veamos lo que ocurre al ejecutar el código anterior. En primer lugar se le pide al usuario que introduzca su edad. La respuesta por defecto es 0. Una vez que la ha introducido y ha pulsado en Aceptar, se comprueba si la edad **es menor que** 18. Para ello se usa el operador **<** (mira el Apéndice C: Operadores en VBScript). En caso de que sea así, se muestra el siguiente mensaje:

En caso de no cumplirse la condición, se comprueba si la edad es mayor que 65. Si se cumple esta condición se muestra el siguiente mensaje:

Solo en caso de no cumplirse ninguna de las dos condiciones anteriores se muestra el último mensaje:

Hay que resaltar que las condiciones son excluyentes entre sí. Es decir. En el momento que alguna de ellas resulte ser cierta, se ejecuta el bloque de instrucciones correspondiente y se abandona el condicional. Por ejemplo. Si la primera condición es cierta, ya no se evaluará la segunda, como es lógico.

CONDICIONES COMPUESTAS

Supongamos que solo queremos evaluar si la edad del usuario está entre 18 y 65 años. Solo nos interesa saber si está o no en ese rango de edad. Veamos un ejemplo:

Condiciones compuestas

```
<HTML>
<HEAD>
<TITLE>Prueba básica de condicional</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   edad = INPUTBOX ("¿Cuál es su edad?",,"0")
   IF (edad > 18 AND edad < 65) THEN
      MSGBOX ("Usted es un adulto en edad laboral")
   ELSE
      MSGBOX ("Usted no está en edad laboral")
   END IF
</SCRIPT>
</BODY>
</HTML>
```

Aquí vemos que la condición son, en realidad dos condiciones unidas por el operador lógico AND (Mira el Apéndice C). Esta línea se podría leer como "Si la edad es menor que 18 y la edad es menor que 65 entonces...". De esta forma se evalúan condiciones múltiples.

MAS SOBRE CONDICIONES MULTIPLES

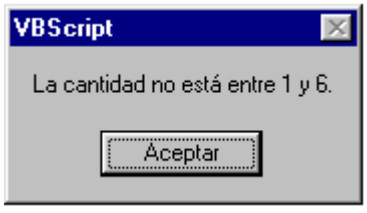
Cuando hay que evaluar muchas condiciones no se deben utilizar condicionales anidados: resulta antiestético, por no decir chapucero. Para ello se utilizan las sentencias SELECT CASE y END SELECT. Veamos un ejemplo:

Condiciones múltiples

```
<HTML>
<HEAD>
<TITLE>Prueba de Select</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
OPTION EXPLICIT
DIM cantidad
cantidad = 3

SELECT CASE cantidad
   CASE 1:
      MSGBOX ("La cantidad vale 1")
   CASE 2:
      MSGBOX ("La cantidad vale 2")
   CASE 3:
      MSGBOX ("La cantidad vale 3")
   CASE 4:
      MSGBOX ("La cantidad vale 4")
   CASE 5:
      MSGBOX ("La cantidad vale 5")
   CASE 6:
      MSGBOX ("La cantidad vale 6")
   CASE ELSE:
      MSGBOX ("La cantidad no está entre 1 y 6")
END SELECT
</SCRIPT>
</BODY>
</HTML>
```

El resultado será el siguiente:

Supongamos que la línea cantidad = 3 la sustituimos por cantidad = 7. El resultado sería el siguiente.



CAPITULO 6: Bucles 1ª parte

En cualquier lenguaje de programación se nos plantea, en muchas ocasiones, la necesidad de realizar una operación mas de una vez, en un proceso reiterativo. Esa operación puede realizarse un número determinado o indeterminado de veces. En VBScript tenemos varias estructuras que nos lo permiten. Vamos a conocerlas todas y así sabremos como elegir una u otra según las circunstancias.

BUCLES FOR... NEXT

Esta estructura se emplea cuando es necesario repetir un bloque de operaciones un número determinado de veces. Un bucle FOR ... Next emplea para ello un variable de control que actúa como contador de las veces que se ha procesado el bucle. La variable que actúa como contador parte de un valor_inicial. Cada vez que se ejecuta el BLOQUE DE SENTENCIAS incluido en el bucle la variable de contador se incrementa en una unidad. Opcionalmente el incremento puede ser diferente de la unidad, incluyendo el parámetro STEP seguido del índice de incremento que deseemos. Cuando el contador alcanza el valor_final se deja de ejecutar el bucle y se sigue ejecutando el programa a partir de la instrucción que va detrás de NEXT. La estructura general es la siguiente:

```
FOR contador = valor_inicial TO valor_final STEP incremento
   BLOQUE DE SENTENCIAS
NEXT
```

Veamos unos ejemplos de uso.

Bucle For...Next

```
<HTML>
<HEAD>
<TITLE>Prueba de bucle FOR...NEXT</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   OPTION EXPLICIT
   DIM contador
   FOR contador = 0 TO 5
      DOCUMENT.WRITE (contador & "<BR>")
   NEXT
   DOCUMENT.WRITE ("Se acabó la cuenta")
</SCRIPT>
</BODY>
</HTML>
```

El resultado de la ejecución de éste código será el siguiente:

```
0
1
2
3
4
5
Se acabó la cuenta
```

Bien. Ahora supongamos el siguiente ejemplo de código:

Bucle For...Next

```
<HTML>
<HEAD>
<TITLE>Prueba de bucle FOR...NEXT</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   OPTION EXPLICIT
   DIM contador
   FOR contador = 0 TO 10 STEP 2
      DOCUMENT.WRITE (contador & "<BR>")
   NEXT
   DOCUMENT.WRITE ("Se acabó la cuenta")
</SCRIPT>
</BODY>
</HTML>
```

El resultado será:

```
0
```

```
2
4
6
8
10
Se acabó la cuenta
```

Como vemos en el último ejemplo, la cuenta se ha realizado de dos en dos por la coletilla STEP 2 que hemos añadido a la instrucción FOR.

BUCLES DO WHILE...LOOP

Este tipo de bucles se emplean para efectuar (DO) una operación un número indeterminado de veces mientras (WHILE) se cumpla una condición. Su estructura general es la siguiente:

```
DO WHILE (condición)
   BLOQUE DE INSTRUCCIONES
LOOP
```

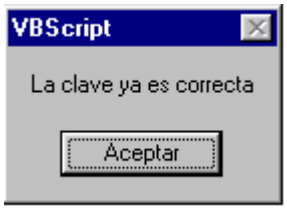
El BLOQUE DE INSTRUCCIONES, contenido entre las líneas DO y LOOP se ejecutará mientras se cumpla la condición. Supongamos que ha llegado la hora de pedirle al usuario una palabra clave para continuar ejecutando un programa. Esta palabra será, por ejemplo, AUTORIZADO. Veamos como lo haríamos:

Bucle Do...While

```
<HTML>
<HEAD>
<TITLE>Prueba de DO WHILE ... LOOP</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   OPTION EXPLICIT
   DIM clave
   clave = ""
   DO WHILE (clave <> "AUTORIZADO")
      ' Ejecuta mientras la clave no es igual a  <> AUTORIZADO
      clave = INPUTBOX ("Teclea la clave:")
   LOOP
   MSGBOX ("La clave ya es correcta")
</SCRIPT>
</BODY>
</HTML>
```

Este ejemplo mostrará el siguiente cuadro de diálogo y no seguirá adelante hasta que se introduzca la palabra AUTORIZADO. Si se introduce mal la clave, seguirá mostrando la pregunta indefinidamente.

Una vez introducida la clave correctamente el programa continuará su ejecución mostrando lo siguiente:



Bien. Veamos más posibilidades.

CAPITULO 7: Funciones y Procedimientos

En VBScript existen tres formas básicas de ejecución. La primera de ellas se conoce con el nombre de **ejecución inmediata**. Es un código VBScript insertado dentro de un código HTML y que se ejecuta allí donde está insertado. Es la forma de ejecución de todos los ejemplos que hemos visto hasta ahora en este temario. Hay otras dos formas de ejecución en las que el código se carga en memoria en segundo plano, sin ejecutarse hasta que es invocado en algún punto concreto de la página. Estas dos formas de ejecución se conocen con el nombre de **funciones y procedimientos**. Dada su propia naturaleza, el código de las funciones y los procedimientos suele ir al principio de la página, normalmente dentro de la cabecera. Ahí se cargan en memoria pero no se ejecutarán hasta que sean invocados desde algún punto de la página. Esto se hace así porque es importante haberlas definido (cargado en memoria) antes de invocarlas. Vamos a ver como funcionan ambos sistemas de ejecución.

FUNCIONES

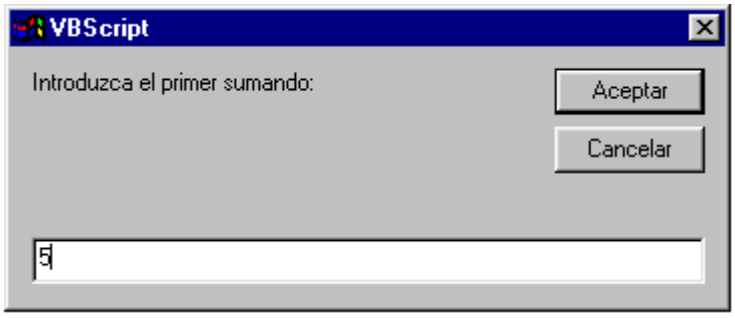
Una función es un fragmento de código que recoge unos **parámetros** (valores que se le pasan para su ejecución) y devuelve un **resultado**. El código de la función se incluye entre las palabras clave **FUNCTION y END FUNCTION**. Todo lo que haya entre estas dos líneas será considerado como parte de la función. Además, a la función se le asigna un nombre, por el que se la invocará desde el código y será el nombre del resultado que devuelva. Así pues, en VBScript una función sólo puede devolver un resultado. Veamos un ejemplo que lo dejará todo mas claro. Vamos a escribir un programa que pida dos números por teclado y los sume entre sí:

Funciones

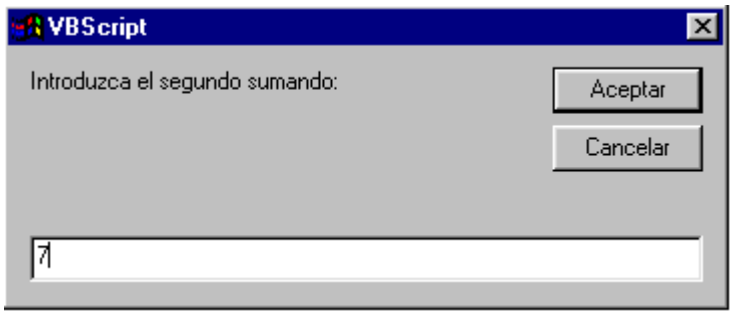
```
<HTML>
<HEAD>
<TITLE>Ejemplo del uso de funciones</TITLE>
<SCRIPT LANGUAGE = "VBScript">
   ' En la siguiente línea empieza la función.
   FUNCTION sumar (sumando_1, sumando_2)
      sumar = (CLNG(sumando_1) + CLNG(sumando_2))
   END FUNCTION
   ' Ya se ha acabado la función.
</SCRIPT>

</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
  OPTION EXPLICIT
  ' A continuación se definen las tres variables que se usarán.
  ' La variable resultado almacenará el resultado que devuelva la función.
  ' Las variables sum1 y sum2 almacenan los valores que se pasarán
  ' como parámetros a la función.
  DIM sum1
  DIM sum2
  DIM resultado

  sum1 = 0
  sum2 = 0
  resultado = 0
```

```
' El siguiente bloque pide por teclado el primer sumando, comprobando que
' se trate de un numero positivo.
DO UNTIL (sum1  > 0)
  sum1 = INPUTBOX ("Introduzca el primer sumando:")
LOOP
' El siguiente bloque pide por teclado el segundo sumando, comprobando que
' sea también un número positivo.
DO UNTIL (sum2  > 0)
  sum2 = INPUTBOX ("Introduzca el segundo sumando:")
LOOP
resultado = sumar (sum1, sum2)
MSGBOX ("El resultado es " & resultado)
</SCRIPT>
</BODY>
</HTML>
```

El resultado de este código es que pide al usuario que introduzca que introduzca un número que será el primer sumando de la suma, como ya conocemos:



En este caso hemos introducido un 5. A continuación se nos pide de igual modo el segundo sumando:



Como ves hemos introducido un 7. El programa invoca entonces a la función de la suma a la que hemos llamado **sumar**. Esta función devuelve un resultado que es mostrado en la siguiente forma:

Vamos a entrar en detalle en el funcionamiento de este proceso. En primer lugar deberemos reparar en que el nombre de las variables que se pasan a la función como parámetros (también llamados **argumentos de la función**) no tiene por que coincidir con los nombres que se le dieron a los parámetros en el momento de definirla. Si te fijas verás que, en la definición de la función se han incluido como parámetros **sumando_1 y sumando_2**, tal como vemos en la correspondiente línea de código:

```
FUNCTION sumar (sumando_1, sumando_2)
```

Sin embargo en la invocación se usan las variables **sum1 y sum2**:

```
resultado = sumar (sum1, sum2)
```

Esto es así porque el sistema coge la primera variable de la invocación y la asigna al primer argumento de la función; la segunda variable de la invocación la asigna al segundo argumento de la función, y así sucesivamente si hubiera que pasar mas parámetros. De esta forma, para invocar una función nos basta con saber el número de parámetros que es necesario pasarle. Los parámetros deben estar encerrados entre paréntesis y separados por comas, tanto los de la definición como los de la invocación. Dentro de la definición de la función vemos también la siguiente línea:

```
sumar = (CLNG(sumando_1) + CLNG(sumando_2))
```

En ella apreciamos que el resultado que devolverá la función, sale de ésta con el mismo nombre que la función **(sumar)** , aunque luego se alojará en la variable **resultado**, que es la que invoca a la función.

Además observamos la palabra clave CLNG (). Tiene todo el aspecto de ser una función que se aplica a **sumando_1 y a sumando_2**. Sin embargo en nuestro código no aparece definida en ninguna parte. Esto es así porque CLNG ()no es una **función de usuario**, sino una **función del lenguaje** y, por lo tanto, su comportamiento está definido en el intérprete. Hablaremos de las funciones del lenguaje en el apartado del mismo nombre, dentro de éste capítulo. De momento nos basta saber que si no aplicamos esta función, el resultado no es el que esperamos (la suma aritmética de los parámetros), si no la concatenación de ambos como si fueran cadenas alfanuméricas. Una última consideración acerca de las funciones. Es posible definir y utilizar funciones que no reciban ningún argumento, pero siempre devuelven un resultado.

CAPITULO 7: Procedimientos

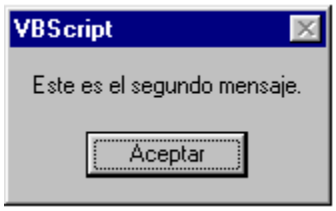
Un procedimiento se parece muchísimo en su concepción y uso a una función, con la diferencia fundamental de que no devuelve ningún resultado y algunas otras diferencias menores, que veremos en este apartado. En primer lugar los procedimientos se incluyen entre las palabras clave **SUB y END SUB**. Para invocar al procedimiento, simplemente teclearemos su nombre en una línea de comando, como si fuera una instrucción. Veamos un ejemplo:

Procedimientos

```
<HTML>
<HEAD>
<TITLE>Ejemplo del uso de procedimientos</TITLE>
<SCRIPT LANGUAGE = "VBScript">
SUB doble_mensaje()
   MSGBOX ("Este es el primer mensaje.")
   MSGBOX ("Este es el segundo mensaje.")
END SUB
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   doble_mensaje
</SCRIPT>
</BODY>
</HTML>
```

El resultado sería, como cabe esperar, que primero visualizaremos el mensaje:

Y a continuación, tras pulsar el botón ACEPTAR, veremos:



Tras pulsar el botón aceptar, finaliza la ejecución. Como vemos, el procedimiento no devuelve ningún resultado. Simplemente ejecuta una acción. En este caso, además hemos visto que se trata de un procedimiento que no recibe ningún parámetro. Por eso, en la definición aparece el nombre del procedimiento con dos paréntesis sin nada en medio y en la invocación aparece, simplemente, el nombre del procedimiento. Si hubiera que pasar parámetros al procedimiento, existen dos formas de realizar la invocación. La primera es la mas sencilla. El nombre del procedimiento seguido de los parámetros necesarios, separados por comas y sin paréntesis. Así:

```
procedimiento parám1, parám2, parám3, ... , parámN
```

La segunda forma es mas elegante y mas adecuada al concepto de programación estructurada. Consiste en anteponer al nombre de procedimiento la palabra reservada CALL y encerrar la lista de parámetros, separados por comas, entre paréntesis. Así:

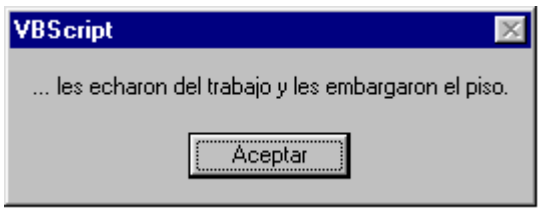
```
CALL procedimiento (parám1, parám2, parám3, ... , parámN)
```

Modos de llamar a un procedimiento

```
<HTML>
<HEAD>
<TITLE>Ejemplo de llamadas a procedimiento</TITLE>
<SCRIPT LANGUAGE = "VBScript">
SUB cuento (final)
   IF (final = "Bonito") THEN
      MSGBOX ("...y fueron felices y comieron perdices.")
   END IF
      IF (final = "Feo") THEN
      MSGBOX ("... les echaron del trabajo y les embargaron el piso.")
   END IF
END SUB
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   cuento "Bonito"
   CALL cuento ("Feo")
</SCRIPT>
</BODY>
</HTML>
```

El resultado de la ejecución de este programa es el siguiente:

Y, al pulsar Aceptar veremos lo siguiente:



Con lo que termina la ejecución del programa. Con esto hemos visto como operan las funciones de usuario y los procedimiento.

RUPTURAS

Al igual que sucedía con los bucles, es posible interrumpir la ejecución de una función de usuario o de un procedimiento y devolver el control de la ejecución al código llamante. Para ello se utiliza, una vez más, la instrucción EXIT. Si se trata de una función de usuario emplearemos EXIT FUNCTION. Si se trata de un procedimiento, emplearemos EXIT SUB. De todas formas no es bueno acostumbrarse a usar estas rupturas forzadas, porque se acaba abusando de ellas y escribiendo códigos chapuceros. En realidad se puede resolver casi cualquier problema sin necesidad de rupturas. Solo deben incluirse como solución de emergencia si se genera algún error indeseado, por ejemplo. Aclarado esto, es el momento adecuado para hablar de funciones del lenguaje.

CAPITULO 7: Funciones del lenguaje

En VBScript existe una librería de funciones predefinidas que el usuario puede emplear si lo considera conveniente. Al estar predefinidas no es necesario hacer nada para cargarlas en memoria. Se cargan, automáticamente, con el intérprete. Solo es necesario invocarlas donde se necesiten. Estas funciones se clasifican en grupos en base al tipo de datos con el que se emplean. Veamos cuales son y que hacen:

ABS (número)

Recibe como argumento un número (o variable que lo contiene) y devuelve el mismo número sin signo. No produce ningún efecto con números positivos y elimina el signo en los negativos.

ARRAY (Elemento1, Elemento2, ... , Elemento N)

Devuelve una matriz con los elementos que recibe, separados por comas, como argumento. Si no se le pasan argumentos, devuelve una matriz de cero elementos.

ASC (carácter)

Devuelve el valor ASCII del carácter que recibe como argumento. Si recibe una cadena, devuelve el código ASCII del primer carácter de la misma. Si recibe un dato Null, devuelve un valor Null.

ATN (número)

Recibe un número (o variable que lo contiene) que representa a un ángulo en radianes y devuelve el arco tangente de ese ángulo.

CBOOL (número)

Recibe como argumento un número (o variable que lo contiene y devuelve un valor lógico. Falso si el número es 0 y verdadero en otro caso.

CBYTE (número)

Recibe como argumento un número (o variable que lo contiene) y lo convierte a un número en formato Byte.

CCUR (número)

Recibe como argumento un número (o variable que lo contiene) y lo convierte a un número en formato Currency.

CDATE (cadena)

Recibe como argumento una cadena alfanumérica representando una fecha y la convierte en un dato se subtipo DATE.

CDBL (número)

Recibe como argumento un número (o variable que lo contiene) y lo transforma a un número en formato Double.

CHR (número)

Devuelve el carácter asociado al código ASCII pasado como argumento. El número deberá estar comprendido entre 0 y 255.

CINT (número)

Recibe como argumento un número (o variable que lo contiene) y lo transforma en un dato con formato Int.

CLNG (número)

Recibe como argumento un número (o variable que lo contiene) y lo transforma en un dato con formato Long.

COS (número)

Recibe un número (o variable que lo contiene) que representa a un ángulo en radianes y devuelve el coseno de ese ángulo.

CSNG (número)

Recibe como argumento un número (o variable que lo contiene) y lo transforma en un dato con formato Single.

CSTR (argumento)

Recibe como argumento un dato (o variable que lo contiene) que no sea una cadena alfanumérica y lo devuelve transformado en una cadena.

DATE ()

No recibe ningún argumento y devuelve la fecha del sistema.

DATEADD (intervalo, cantidad, fecha)

Esta función devuelve el resultado de sumarle un periodo de tiempo a una fecha determinada. Recibe tres parámetros obligatorios:

El **intervalo** es una cadena de texto que indica la unidad de tiempo que queremos añadir a la fecha (horas, días, etc). Los posibles valores están indicados a continuación.

Cadena	Significado
yyyy	año completo
q	trimestre
m	mes
d	día
w	semana
h	hora
m	minuto
s	segundo

La cantidad es el número de unidades del intervalo especificado que se le sumará a la fecha indicada. Este valor puede ser positivo, para referirse a fechas futuras o negativo para referirse a fechas pasadas.

La fecha es aquella a la que se le suman los intervalos especificados para que la función devuelva una nueva fecha como resultado de la operación.

Por ejemplo. Si tecleamos la siguiente línea de código:

```
nueva = DATEADD ("yyyy",1,10-02-1996) resultado será 10-02-1997.
```

Esta función tiene en cuenta incluso los años bisiestos.

DATEDIFF (intervalo, fecha 1, fecha2)

Calcula la diferencia entre las dos fechas especificadas y la devuelve expresada en los periodos indicados en **intervalo**. Por lo tanto devuelve un resultado numérico. Los posibles intervalos a especificar son los mismos que en el caso anterior.

DAY (fecha)

Esta función recibe como argumento una fecha y devuelve un número que corresponde al día del mes de la fecha indicada.

EXP (número)

Recibe como argumento un número (o variable que lo contiene) y devuelve el número e elevado a la potencia indicada.

FIX (número)

Recibe un número (o variable que lo contiene). Devuelve la parte entera de un número, truncando los decimales. Si el argumento es un número negativo, esta función devuelve el primer negativo igual o mayor que encuentre.

FORMATCURRENCY (número, dígitos_decimales, cero_decimal, negativos_paréntesis, agrupar_dígitos)

Esta función se usa para representar números en formato de moneda, es decir, con la apariencia de valores económicos. Recibe cuatro argumentos. El primero de ellos es el número (o variable que lo contiene) que hay que representar como cifra económica. Los otros tres son opcionales.

El parámetro **dígitos_decimales** especifica cuantos dígitos se quiere que aparezcan a la derecha de la coma digital.

El parámetro **cero_decimal** indica si se quiere completar el número con ceros a la derecha de los decimales. P.e. Si se especificaron dos decimales en el parámetro anterior y un número tiene un solo decimal ¿Queremos un cero a la derecha de ese decimal? Si lo queremos, este parámetro será un -1. Si no lo queremos, el parámetro será 0.

El parámetro **negativos_paréntesis** especifica si queremos que los números negativos aparezcan entre paréntesis. Si lo queremos, pondremos -1. Si no, un 0.

El parámetro **agrupar_dígitos** indica si queremos que los dígitos aparezcan agrupados de tres en tres, en millares, millones, etc. Si lo queremos pondremos un -1. Si no lo deseamos así, pondremos un 0.

Estos tres últimos parámetros pueden recibir como valor, en lugar de -1 o 0, un -2. En éste último caso se toman las características establecidas en la configuración regional del sistema, en el panel de control de Windows.

FORMATNUMBER (número, dígitos_decimales, cero_decimal, negativos_paréntesis, agrupar_dígitos)

Esta función opera igual que la anterior, solo que se refiere a un formato genérico, sin asociar a ninguna moneda.

HEX (número)

Recibe como argumento un número decimal (o variable que lo contiene) y lo convierte en hexadecimal.

HOUR (hora)

Esta función recibe como argumento una variable que contiene una hora en formato hh:mm:ss y devuelve la hora como un número de 0 a 23.

INSTR (comienzo, cadena 1, cadena 2)

Busca la primera aparición de la cadena 2 dentro de la cadena 1. Los parámetros comienzo y comparación son opcionales. El parámetro **comienzo** indica a partir de que carácter de la cadena 1 se empieza a buscar la cadena 2. Si se omite, la búsqueda se inicia desde el primer carácter.

INSTRREV (cadena 1, cadena 2, comienzo)

Esta función es similar a la anterior, solo que empieza la búsqueda por el final de la cadena.

INT (número)

Recibe un número (o variable que lo contiene). Devuelve la parte entera de un número, truncando los decimales. Si el argumento es un número negativo, esta función devuelve el primer negativo igual o menor que encuentre.

ISARRAY (variable)

Esta función recibe como argumento un nombre de una variable y comprueba si es una matriz. Devuelve un valor booleano.

ISDATE (variable)

Esta función recibe como argumento el nombre de una variable y comprueba si es una fecha. Devuelve un valor booleano.

ISEMPTY (variable)

Esta función recibe como argumento el nombre de una variable. Devuelve un valor booleano. Es verdadero si la variable no está inicializada y falso si lo está

ISNULL (variable)

Esta función recibe como argumento un nombre de variable y devuelve un valor lógico. Verdadero si la variable contiene un Null. Falso si no lo contiene.

ISNUMERIC (variable)

Recibe una variable, supuestamente numérica. Devuelve un valor lógico verdadero si el parámetro que ha recibido es una variable que contiene un número y un valor lógico falso en caso contrario.

LCASE (cadena)

Recibe como argumento una cadena (o una variable que contiene una cadena) y la convierte a minúsculas.

LEFT (cadena, longitud)

Devuelve una sub-cadena compuesta por los caracteres que hay a la izquierda de una cadena. Los argumentos que recibe son una cadena (o variable que la contiene) y la longitud (cantidad de caracteres) de la sub-cadena.

LEN (cadena)

Recibe como argumento una cadena (o variable que la contiene) y devuelve la cantidad de caracteres que componen dicha cadena.

LOG (número)

Esta función recibe como argumento un número (o una variable que lo contenga) que deberá ser mayor de 0 y devuelve el logaritmo en base e de dicho número.

LTRIM (cadena)

Recibe como argumento una cadena o variable que la contiene. Devuelve la cadena sin los espacios en blanco que hay a la izquierda de la misma.

MID (cadena, inicio, longitud)

Recibe como argumentos una **cadena** (o variable que la contiene), una posición de **inicio** (o variable numérica que la contiene) y una **longitud** (o variable **numérica** que la contiene). Devuelve una sub-cadena extraída de una cadena original. La sub cadena tiene los caracteres expresados en longitud, contados a partir del carácter **inicio**. Por lo tanto, el primer argumento es alfanumérico y los otros dos son numéricos.

MINUTE (hora)

Esta función recibe como argumento una variable que contiene una hora en formato hh:mm:ss y devuelve los minutos como un número de 0 a 59.

MONTH (fecha)

Esta función recibe como argumento una fecha o una variable de fecha y devuelve un número del 1 al 12 que indica el mes de la fecha.

MONTHNAME (mes, abreviado)

Esta función recibe como argumento un número del 1 al 12 (o una variable que lo contiene) y devuelve el nombre del mes correspondiente. El otro parámetro que recibe es un valor lógico para indicar si el nombre debe aparecer abreviado o no.

OCT (número)

Recibe como argumento un número decimal (o variable que lo contiene) y lo convierte en octal.

REPLACE (cadena 1, cadena 2, cambia_por, comienzo, veces)

Esta función encuentra la **cadena 2** dentro de la **cadena 1** y la sustituye por **cambia_por**. Comienza a buscar a partir del carácter cuyo número de orden es el indicado en **comienzo** y, si la **cadena_2** aparece mas de una vez en la **cadena 1** la cambia el número de **veces** indicado.

RIGHT (cadena, longitud)

Devuelve una sub-cadena compuesta por los caracteres que hay a la derecha de una cadena. Los argumentos que recibe son una cadena (o variable que la contiene) y la longitud (cantidad de caracteres) de la sub-cadena.

RND()

Recibe un argumento vacío y devuelve un número aleatorio. Para que funcione correctamente, es necesario incluir en el código VBScript una línea con la instrucción RANDOMIZE.

ROUND (número, decimales)

Esta función redondea el **número** (o variable que lo contiene) y lo devuelve con el número de decimales expresado en **decimales**.

RTRIM (cadena)

Recibe como argumento una cadena o variable que la contiene. Devuelve la cadena sin los espacios en blanco que hay a la derecha de la misma.

SCRIPTENGINEBUILDVERSION ()

Esta función no recibe ningún argumento y devuelve el número de versión del motor de Script que se está utilizando.

SECOND (hora)

Esta función recibe una expresión de hora en formato hh:mm:ss y devuelve el número de segundos correspondiente.

SGN (numero)

Recibe un argumento numérico y devuelve un 1 si el número es positivo, un -1 si es negativo y un 0 si es 0.

SIN (número)

Recibe un número (o variable que lo contiene) que representa a un ángulo en radianes y devuelve el seno de ese ángulo.

SPACE (numero)

Recibe como argumento un número (o variable que lo contiene) y devuelve una cadena formada por espacios en blanco; tantos como especifica el número.

SQR (número)

Devuelve la raíz cuadrada del número (o variable numérica) que recibe como argumento.

STRCOMP (cadena 1, cadena 2)

Recibe como argumentos dos cadenas alfanuméricas (o variables que las contienen) separadas por comas. Si ambas cadenas son iguales devuelve un 0. Si la primera es mayor que la segunda, devuelve un 1. Si la segunda es mayor que la primera devuelve un -1. Si alguna cadena tiene un valor Null, devuelve Null. En este sentido debemos recordar que un carácter es mayor o menor que otro en función de sus códigos ASCII. Así, p.e., la a es mayor que la A, porque el código ASCII de la A es 65 y el de la a es 97.

STRING (número, carácter)

Recibe como argumentos un número (o variable que lo contiene) y un carácter (o variable que lo contiene). Devuelve una cadena compuesta por el carácter especificado, repetido las veces que indica el número.

STRREVERSE (cadena)

Recibe como argumento una cadena (o variable que la contiene) y devuelve la cadena tras invertir el orden de todos los caracteres que la componen. Así pues, el primero de la cadena original será el último de la cadena resultante.

TAN (número)

Recibe un número (o variable que lo contiene) que representa a un ángulo en radianes y devuelve la tangente de ese ángulo.

TIME ()

Esta función no recibe ningún argumento y devuelve una expresión que representa la hora del sistema en formato hh:mm:ss.

TIMESERIAL (número 1, número 2, número 3)

Esta función recibe tres parámetros numéricos y los convierte a una hora en formato hh:mm:ss. El número 1 debe estar comprendido entre 0 y 23; el número 2 debe estar comprendido entre 0 y 59 y el número 3 también.

TIMEVALUE (fecha)

Esta función recibe una variable de fecha y extrae y devuelve la parte de la hora.

TRIM (cadena)

Recibe como argumento una cadena o variable que la contiene. Devuelve la cadena sin los espacios en blanco que hay a la izquierda y a la derecha de la misma.

TYPENAME (variable)

Esta función recibe un nombre de variable y devuelve el nombre de subtipo del dato que contiene.

UCASE (cadena)

Recibe como argumento una cadena (o una variable que contiene una cadena) y la convierte a mayúsculas.

WEEKDAY (fecha, primer_día)

Esta función recibe dos parámetros. Es primero es una fecha o variable que la contiene; el segundo es una constante que indica el que es el primer día de la semana en nuestro país. Devuelve un número correspondiente al día de la semana de la fecha introducida. Las constantes para identificar que día de la semana es el primero en nuestro país son:

Constante	Corresponde a
VBSUNDAY	Domingo
VBMONDAY	Lunes
VBTUESDAY	Martes
VBWEDNESDAY	Miércoles
VBTHURSDAY	Jueves
VBFRIDAY	Viernes
VBSATURDAY	Sábado

WEEKDAYNAME (día_semana, abreviado, primer_dia)

Esta función recibe un número de día de la semana, un valor lógico y una constante que indica cual es el primer día de la semana en nuestro país. Devuelve como resultado el nombre del día de la semana que corresponde al número introducido. Si el valor lógico es verdadero, el nombre que devuelve aparece en abreviatura. Las constantes que identifican el día de la semana son las mismas que en el caso anterior.

CAPITULO 7: Ejemplos

Vamos a ver algunos ejemplos.

Uso de la función CBOOL

```
<HTML>
<HEAD>
<TITLE>
Prueba de funciones de conversión.
</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   OPTION EXPLICIT
   DIM variable
   variable = 1
   MSGBOX (CBOOL (variable))
</SCRIPT>
</BODY>
</HTML>
```

El resultado de este código será el siguiente:

La función CBOOL en particular devuelve un valor falso si el argumento es 0 y un valor verdadero si el argumento es cualquier otro número.

Una observación acerca de las conversiones entre subtipos numéricos. Si un número que convertimos, p.e., a Byte es muy grande para caber en este subtipo, se truncará. Supongamos la siguiente sentencia: **numero = CBYTE (567.384.896.456,29)** . Esto es claramente una irregularidad, ya que el número es demasiado grande para ser di subtipo Byte.

Veamos mas ejemplos:

Uso de Funciones de cadena: Instr y Mid

```
<HTML>
<HEAD>
<TITLE>Prueba de funciones de cadena.</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   OPTION EXPLICIT
   DIM variable
   DIM parte_central

   variable = CHR (65)
   DOCUMENT.WRITE ("Valor de CHR (65): " & variable & "<BR>")

   variable = "Alfa"
   parte_central = INSTR (variable,"l")
   DOCUMENT.WRITE (parte_central & "<BR>")

   Variable = "Manzana"
   Parte_central = MID (variable, 3, 2)
   DOCUMENT.WRITE (parte_central)
</SCRIPT>
</BODY>
</HTML>
```

Como resultado nos aparece en pantalla lo siguiente:

```
Valor de CHR (65): A
2
nz
```

Fíjate en la segunda línea. Es el resultado de la función INSTR. Si analizamos la correspondiente línea de código veremos que esta función recibe dos argumentos. El primero es la cadena (o variable que la contiene) donde hay que buscar algo. El segundo es la cadena (o variable que la contiene) que hay que buscar dentro. El número que devuelve como resultado (en este caso el 2) es la posición que ocupa la segunda cadena dentro de la primera.

Ahora observa la última línea, con el texto nz. Es el resultado de aplicar la función MID. Observa la correspondiente línea de código. Ves que esta función recibe tres argumentos. El primero es una cadena (o la variable que la contiene) de la que se extraerá una sub-cadena. El segundo es un número (o una variable que contiene un número), que indica que lugar de la cadena iniciará la sub-cadena. El tercero es un número (o una variable que contiene un número) que indica cuantos caracteres se cogerán para la sub-cadena. Hay que puntualizar que la cadena original no sufre amputación ni transformación de ningún tipo.

Otro ejemplo:

Funciones matemáticas

```
<HTML>
<HEAD>
<TITLE>Prueba de funciones matemáticas.</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
  OPTION EXPLICIT
  RANDOMIZE

  DIM numero
  numero = 0

  DO UNTIL numero > 0
        numero = INPUTBOX ("Introduzca un número")
        IF (NOT ISNUMERIC (numero)) THEN
                numero = 0
        END IF
  LOOP
  DOCUMENT.WRITE ("El número es: " & numero & "<BR>")
  numero = INT (numero)
  DOCUMENT.WRITE ("La parte entera es: " & numero & "<BR>")
  DOCUMENT.WRITE ("El valor hexa es: " & HEX (numero) & "<BR>")
  DOCUMENT.WRITE ("Un número aleatorio es: " & RND() & "<BR>")
</SCRIPT>
</BODY>
</HTML>
```

Este programa nos pide un número por teclado como ya conocemos. Supongamos que introducimos 45,7. Los resultados serán los siguientes:

```
El número es: 45,7
La parte entera es: 45
El valor hexa es: 2D
Un número aleatorio es: 0,1896936
```

La primera línea nos muestra el número que hemos introducido.

La segunda nos muestra el resultado de obtener la parte entera con la función INT.

La tercera línea nos muestra el valor hexa de 45.

La cuarta línea es un número aleatorio generado con RND(). Esta función siempre genera un número aleatorio que está entre 0 y 1. Observa que el código incluye la instrucción RANDOMIZE.

Observa también que se ha usado la función ISNUMERIC para anular la entrada por teclado si lo que se introdujo no es un número. En este caso se ha precedido la función del operador lógico NOT. Consulta el Apéndice C para obtener mas información respecto a los operadores.

Otro ejemplo mas:

Funciones de fecha

```
<HTML>
<HEAD>
<TITLE>Prueba de una función de fecha</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = "VBScript">
   OPTION EXPLICIT
   DIM fecha
   fecha = DATE ()
   IF (WEEKDAY (fecha, VBMONDAY) = 7) THEN
      DOCUMENT.WRITE ("Hoy es domingo.")
   ELSE
```

```
        DOCUMENT.WRITE ("Hoy se trabaja.")
    END IF
</SCRIPT>
</BODY>
</HTML>
```

Este código recoge, en primer lugar, la fecha del sistema. Si la fecha corresponde a un domingo, muestra:

```
Hoy es domingo.
```

En caso contrario, muestra:

```
Hoy se trabaja.
```

Para hacer esto, utiliza la función WEEKDAY, indicando, como primer día de la semana, el Lunes.

GENERALIDADES FINALES

Recuerda. Las tres diferencias básicas entre funciones de usuario y procedimientos son:

1. La función de usuario se define entre las instrucciones FUNCTION y END FUNCTION. El procedimiento se define entre SUB y END SUB.
2. Las funciones (de usuario y del lenguaje) devuelven un resultado. Los procedimientos, no.
3. La forma de invocar una función es con su nombre seguido de paréntesis. Si la función recibe argumentos, éstos van entre los paréntesis. Para invocar un procedimiento hay dos formas. La primera es el nombre del procedimiento seguido de los argumentos, si los hay, separados por comas. La segunda es la palabra clave CALL, seguida del nombre del procedimiento, seguido éste, a su vez de paréntesis. Si hay argumentos, éstos van entre los paréntesis, seguidos por comas.

CAPITULO 8: Uso avanzado de MSGBOX ()

Hemos visto el uso genérico de MSGBOX. En realidad esta es una función del lenguaje aunque, por su particular relevancia le vamos a dedicar un capítulo aparte. Esta función tiene muchas mas posibilidades de las que hemos visto hasta ahora. De hecho sirve para mostrar distintos tipos de cuadros de confirmación, mensajes o preguntas al usuario. Algunos de esos cuadros tienen varias posibles actuaciones por parte del usuario y son capaces de identificar cual ha sido la respuesta elegida y actuar en consecuencia. Como particularidad debemos reseñar que MSGBOX, en todos los usos que se le den, es una función, ya que devuelve un resultado. Si nos acordamos, existe una modalidad, la mas simple, que se llamaba como una función:

```
MSGBOX ("Esto es un resultado")
```

Esta instrucción no produce mas que un cuadro de mensaje en pantalla y queda esperando a que el usuario pulse el botón Aceptar. Como ves tiene algo de excepcional. Es una función y no devuelve ningún resultado. Ahora Veamos una línea un tanto especial:

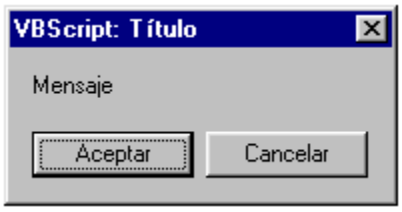
```
CALL MSGBOX ("Mensaje", VBOKONLY, "Título")
```

Fíjate que se está usando la función llamándola como si fuera un procedimiento. Esta línea produce un resultado idéntico a la anterior.

Ahora vamos a usar la función en toda su plenitud. Supongamos la siguiente línea de código:

```
confirmacion = MSGBOX ("Mensaje", VBOKCANCEL, "Título")
```

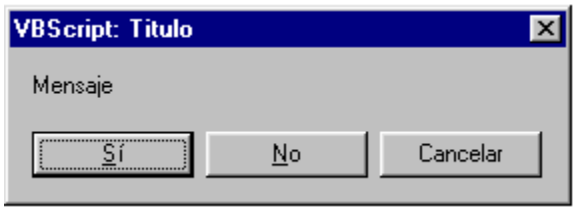
Esto generará el siguiente cuadro:



A continuación esperará a que el usuario pulse uno de los dos botones. Y ¿qué se guardará en la variable **confirmacion?** Pues un número que será distinto según el botón que haya pulsado el usuario. Si pulsa en Aceptar se guardará un 1. Si pulsa en cancelar se guardará un 2. De esta forma, podremos usar, por ejemplo, un condicional para que nuestro programa ejecute una operación u otra, según la decisión del usuario. Veamos mas posibilidades. Al final de este capítulo tienes una lista completa de los posibles botones que se muestran a continuación y los valores numéricos que generan para su posterior identificación.

```
resultado = MSGBOX ("Mensaje", VBYESNOCANCEL, "Titulo")
```

Genera el siguiente cuadro:




```
resultado = MSGBOX ("Mensaje", VBABORTRETRYIGNORE, "Título")
```

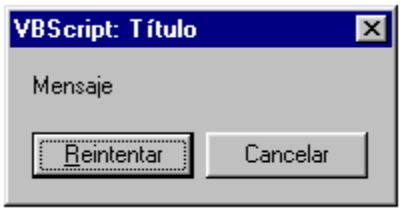
Genera el siguiente cuadro:


```
resultado = MSGBOX ("Mensaje", VBYESNO, "Título")
```

Genera el siguiente cuadro:


```
resultado = MSGBOX ("Mensaje", VBRETRYCANCEL, "Título")
```

Genera el siguiente cuadro:




```
resultado = MSGBOX ("Mensaje", VBCRITICAL, "Título")
```

Genera el siguiente cuadro:


```
resultado = MSGBOX ("Mensaje", VBQUESTION, "Título")
```

Genera el siguiente cuadro:


```
resultado = MSGBOX ("Mensaje", VBEXCLAMATION, "Titulo")
```

Genera el siguiente cuadro:


```
resultado = MSGBOX ("Mensaje", VBINFORMATION, "Título")
```

Genera el siguiente cuadro:

Los diferentes valores que recibiría la variable resultado, según el botón que pulse en cada caso el usuario, son:

Botón	Valor
Aceptar	1
Cancelar	2
Anular	3
Reintentar	4
Ignorar	5
Sí	6
No	7

CAPITULO 9: Programación ASP

Hemos visto todo lo que necesitamos saber acerca de VBScript en el lado del cliente. Pero ¿Qué pasa con el lado del servidor? En muchas ocasiones es necesario que nuestra página realice ciertas operaciones en el lado del servidor. Por ejemplo, si queremos encriptar datos o código. O cuando queremos manejar bases de datos que no tienen por qué bajar al cliente. En esos casos se ejecuta un programa en el lado del servidor y al cliente solo le bajan los resultados de la ejecución. Además de todo, esto tiene una ventaja adicional. Cuando hay que actualizar el programa, basta con hacerlo en el servidor y, automáticamente, todos los clientes tienen la versión actualizada. Además, se dispone de unos recursos dinámicos que no hay manera de obtener con HTML en el lado del cliente.

El sistema tradicional de ejecutar documentos en el servidor es la tecnología CGI (Common Gateway Interface). Esta tecnología se sigue utilizando en aquellas páginas que la incorporan, pero ha sido superada y cada vez se ve menos. Hay un tipo nuevo en la ciudad. Se llama ASP (Active Server Pages, Páginas Activas en el Servidor). La tecnología ASP es el objeto de estudio de este capítulo. La tecnología ASP se basa en una serie de instrucciones de un lenguaje determinado (por ejemplo VBScript) que, al ser llamada la página desde el cliente, no se descargan con el resto de la página, sino que se ejecutan en el servidor y solo se descarga el resultado de esa ejecución. Lo primero que debemos tener en cuenta es que la página que contenga código de éste tipo no deberá tener la extensión **.htm, ni .html**, sino la extensión **.asp** . Además, deberá ejecutarse en un sistema operativo de servidor tal como Windows NT Server o Unix. No funcionará en una plataforma de cliente, como Windows 95, Windows NT WorkStation, etc. Por último hay que tener en cuenta que el directorio en el que se halle alojada la página deberá tener activados los permisos de escritura. Esto es una cuestión de la que hablaremos en el temario de Windows NT - IIS. Bien. Supongamos que ya tenemos todo esto preparado. ¿Cómo introducimos código ASP en nuestra página? Veamos un ejemplo y lo comentamos:

Ejemplo de página ASP

```
<HTML>
<HEAD>
<TITLE>Ejemplo de ASP</TITLE>
</HEAD>
<BODY>
<HR>
   Esta página es una prueba de ASP
   <%
```

```
    Mi_Email = "goldenchip@informaticos.com"
    Response.Write (Mi_Email)
    %>
<HR>
</BODY>
</HTML>
```

En este código vemos dos líneas con una sintaxis sospechosamente parecida a la de VBScript:

```
    Mi_Email = "goldenchip@informaticos.com"
    Response.Write (Mi_Email)
```

Y, en efecto. Se trata de este lenguaje. Pero es VBScript de servidor. El código ya no aparece entre los tags **<SCRIPT>y </SCRIPT>** sino entre **<% y %>** . Estos últimos tags no existen en HTML, por lo que no pueden dar lugar a ningún tipo de confusión. El objeto Response es un objeto de servidor equivalente al objeto document de cliente. La ejecución de ésta página, si todo está correctamente configurado, mostrará en la pantalla lo siguiente:

goldenchip@informaticos.com

Sin embargo, la ventaja es que el usuario no podrá ver el código nativo de nuestra página.

Bien. Sirva esta pequeña introducción a la tecnología ASP para abrir boca. En nuestro temario de Windows NT Server aprenderemos todo lo necesario para usar ASP incluyendo, entre otros aspectos interesantes, la gestión de bases de datos.

APÉNDICE A: Jerarquía de objetos en POO.

En éste apéndice vamos a listar los objetos de que disponemos en VBScript, así como sus distintas propiedades, métodos y eventos. Este apéndice está dedicado a los objetos en el lado del Cliente. En el lado del servidor, la lista se amplía bastante. Como hemos comentado, este manual está orientado a VBScript en el lado del Cliente. Sin embargo, he incluido una breve introducción a la tecnología ASP en el mismo. Bien. Empecemos asegurando posiciones. Vamos a puntualizar cuatro conceptos cuya comprensión es vital para entender el funcionamiento de la POO y la jerarquía de objetos.

OBJETO: Es cada una de los elementos que se gestionan en una página web o en cualquier aplicación informática que tienen una identidad propia. Un objeto podría ser la ventana de navegación, o el documento activo, o un campo de un formulario, o una tabla, etc. La lista de los objetos que podemos manejar en VBScript de Cliente se incluyen en este Apéndice.

PROPIEDAD: Es cada una de las características de un objeto. Una propiedad sería el color de fondo de una tabla, la barra de estado de la ventana de navegación, etc. También llamamos propiedad a un objeto derivado de otro. Se dice que el objeto derivado es propiedad del objeto padre.

METODO: Es una operación que se puede realizar dentro de un objeto. Por ejemplo. Cerrar una ventana sería un método del objeto ventana. Poner el foco en un campo de formulario sería un método del objeto que representa a ese campo en concreto.

EVENTO: Un evento es la previsión de que el usuario realice una determinada acción. En realidad el usuario puede realizar la acción o no realizarla, pero se deja prevista en la programación la posibilidad de que la realice. De esta forma, si la realiza sucederá algo (lo que hayamos programado) como respuesta a esa acción. Un evento sería pasar el ratón sobre una imagen determinada, hacer clic en una parte de la pantalla, pulsar una tecla, etc. También existe la posibilidad de prever eventos del sistema. Un evento de sistema es, por ejemplo, la carga de una página, o un error en un proceso. Resumiendo: un evento se dispara cuando ocurre la acción prevista por el mismo. Supongamos el siguiente código. Utiliza el evento ONMOUSEOVER, asociado al hiperenlace. Cuando el usuario va a pulsar el enlace y coloca el puntero del ratón sobre el mismo, se dispara el evento, que llama a la función mensaje(). Esta genera un cuadro de aviso en la pantalla y obliga al usuario a pulsar el botón Aceptar. Con lo que nunca podrá usar el enlace.

Evento onMouseOver

```
<HTML>
<HEAD>
<TITLE>Prueba de evento ONMOUSEOVER</TITLE>
<SCRIPT LANGUAGE = "VBScript">
  FUNCTION mensaje()
    MSGBOX ("Nunca podrás pulsarlo")
  END FUNCTION
</SCRIPT>
</HEAD>
<BODY>
<H1>
Intenta pulsar el enlace... <BR>
si puedes.
</H1>
<A HREF = "http://www.gratisweb.com/infomegacine" ONMOUSEOVER = "mensaje()">
Acceder a la página de Latinmail
</A>
</BODY>
</HTML>
```

Para referirse a una propiedad o un método de un objeto usamos una notación de puntos. Así:

```
OBJETO.PROPIEDAD
```

O bien:

```
OBJETO.METODO()
```

Observa que el método lleva unos paréntesis y la propiedad no. Esto es así porque, en definitiva, los métodos son casos particulares de funciones, asociadas a objetos. Y, como cabría esperar, algunos métodos reciben argumentos y otros no. Bien. Veamos la lista de objetos de VBScript de Cliente.

Veamos otro ejemplo, para afianzar conceptos:

VBScript en el navegador

```
<HTML>
<HEAD>
<TITLE>Otra prueba de POO</TITLE>
</HEAD>
<BODY>
<A HREF = "http://www.gratisweb.com/infomegacine" ONMOUSEOVER
 = "window.status = 'El mejor site de cine';return true">
Pulsa aquí para ver infomegacine
</A>
</BODY>
</HTML>
```

Aquí podemos ver varias cosas que van mas allá de todo lo estudiado hasta ahora. En primer lugar vamos a aclarar lo que hace el código. Muestra un hiperenlace y, cuando el usuario pasa el ratón por encima, muestra un mensaje en la barra de estado, que ya permanece ahí. Bien. Analicemos el código para ver como funciona. Realmente toda la gracia está en esta línea:

```
<A HREF = "http://www.gratisweb.com/infomegacine" ONMOUSEOVER
="window.status = 'El mejor site de cine';return true">
```

La primera parte es un hiperenlace normal de HTML sin mas. Lo primero que nos interesa ahora es la palabra ONMOUSEOVER, que indica al navegador que detecte la presencia del cursor del ratón sobre el texto que constituye el hiperenlace. ONMOUSEOVER es el **evento** que queremos detectar. A la derecha el signo igual aparece la acción que queremos desencadenar si se produce el evento especificado. Esta acción aparece entre comillas dobles. En concreto se trata de establecer un valor para la propiedad status del objeto window; es decir, el contenido de la barra de estado de la ventana de navegación. En este caso hemos programado que aparezca en la barra de estado una cadena literal (el mejor site de cine). Fíjate

que la cadena de texto aparece encerrada entre comillas simples. A continuación aparece un signo de punto y coma y la instrucción return true. Esta última sirve, en este caso, para que el cambio de la barra de estado se produzca de forma inmediata. De no usarla, al poner el ratón sobre el enlace veremos la dirección de la página especificada y, solo al quitar el ratón, veremos el mensaje que habíamos programado. Evidentemente ésto último no es lo que perseguíamos. Bien. Casi tenemos completo el estudio de esta línea, salvo por un pequeño detalle. Palabras reservadas del lenguaje, como son window, status, return y true aparecen en minúsculas en este ejemplo, cuando hemos dicho que en este temario las palabras reservadas las poníamos en mayúsculas. Esto es así porque en este caso las instrucciones no han sido programadas dentro de VBScript, sino directamente en HTML. Existen algunas instrucciones de los lenguajes de Script que pueden programarse de esta manera (de hecho, algunas veces DEBEN programarse de esta manera). En estos casos, el navegador no interpreta las instrucciones como de VBScript, sino como de JavaScript. Y JavaScript (ya lo veremos en el correspondiente temario) es, al contrario que VBScript, muy exigente con el tema de las mayúsculas y minúsculas. Cada palabra debe escribirse de un modo exacto, y no de otro, o no funcionará. En la lista que incluyo en éste Apéndice de objetos, propiedades, métodos y eventos he seguido este último criterio, a fin de familiarizar al lector con la sintaxis mas adecuada.

La lista de los principales objetos de VBScript es:

- Objeto window
- Objeto location
- Objeto document
- Objeto navigator
- Objeto frame
- Objeto history
- Objeto link
- Objeto anchor
- Objeto form

APÉNDICE A: Propiedades y métodos de los objetos

Como ya hemos mencionado en este temario, los objetos siguen una jerarquía. El de mas alto nivel es el objeto WINDOW, que representa a la ventana activa. Todos los demás, son objetos derivados de éste. Veamos cuales son las propiedades y métodos que acepta cada uno, así como los eventos que les afectan.

Objeto window

Representa la ventana activa del navegador y es el mas alto de la jerarquía.

Propiedades:

defaultStatus	se refiere al mensaje que aparecerá por defecto en la barra de estado.
document	representa al documento HTML en ejecución en ese momento.
frames []	es una matriz que contiene los frames de la ventana.
history	representa un registro histórico de las páginas visitadas en la actual sesión de uso de Internet.
length	contiene el número total de frames de la ventana.
location	representa a la dirección (URL) actual de Internet.
name	contiene el nombre de la ventana activa.
navigator	representa al navegador que estamos utilizando.
self	se refiere a la propia ventana. Es el mismo objeto window.
status	es el mensaje que aparece en la barra de estado en un momento determinado.

window	representa a la ventana activa o a otra ventana o sub-ventana de navegación.

Métodos:

close ()	permite cerrar la ventana activa. Su sintaxis es self.close()
open ()	permite abrir una nueva ventana, como sub ventana de la actual. Su sintaxis es: nueva_ventana = window.open ("URL", "Target", "Opciones")

En la sintaxis expresada, nueva_ventana es el nombre que queremos darle a la sub-ventana. URL es la dirección de la página que queremos que se cargue en la sub-ventana. Target es la dirección del frame donde queremos que se abra la nueva ventana (caso de existir frames) como hacemos en HTML con los hiperenlaces. Opciones son las propiedades de la nueva ventana, de acuerdo a la siguiente tabla.

propiedad	Tipo de dato	Explicación
toolbar	booleano	Ventana con barra de herramientas.
location	booleano	Ventana con barra de direcciones.
directories	booleano	Ventana con directorios.
Status	booleano	Ventana con barra de estado
menubar	booleano	Ventana con barra de menús.
scrollbars	booleano	Ventana con barras de desplazamiento.
resizable	booleano	Ventana de tamaño redefinible por el usuario.
width	píxeles	Anchura de la ventana
height	píxeles	Altura de la ventana.
top	píxeles	Posición Y de la ventana
left	píxeles	Posición x de la ventana

Eventos:

onLoad	Se ejecuta cuando se carga la página.
onUnload	Se ejecuta cuando se descarga (se cierra) la página.

Objeto document

Representa el documento activo.

Propiedades:

alinkColor	Representa el color de los enlaces activos.
bgColor	Representa el color de fondo del documento.
fgColor	Representa el color del texto
lastModified	Representa la fecha de la última modificación.
linkColor	Representa el color de los enlaces.
location	Representa la URL del documento.
title	Representa el título del documento.
vlinkColor	Representa el color de los enlaces visitados.

Métodos:

Write ()	Escribe un texto.

Writeln()	Escribe una línea de texto.

Eventos:

Ninguno

Objeto form

Este objeto se refiere a un formulario empleado en el documento. Se accede a cada formulario mediante un índice: document.form [índice]

El número de índice se corresponde con el orden de creación de formulario en el documento.

Propiedades:

action	Representa la URL donde está el programa encargado de procesar un formulario (al que se llama al activar el botón Submit).
length	Es el número de elementos del formulario.
method	Es el método de envío (GET o POST).

Métodos:

submit	Se usa para forzar el envío.

Eventos:

onSubmit	Se produce cuando se pulsa el botón Submit del formulario.

Objeto location

Contiene la URL de la página actual

Propiedades:

href	Representa la propia URL.
pathname	Representa la ruta del disco del servidor donde se aloja la página.

Métodos:

Ninguno.

Eventos:

Ninguno.

Objeto navigator

Representa al navegador actual.

Propiedades:

appName	Es el nombre del navegador.
appVersion	Se refiere a la versión del navegador.

Métodos:

Ninguno.

Eventos:

Ninguno.

Objeto history

Representa el historial de las páginas visitadas en la sesión actual de uso de Internet

Propiedades:

length	Representa la cantidad total de páginas visitadas.

Métodos:

back ()	navega a la página anterior.
forward()	navega a la página siguiente.
go (n)	navega n páginas hacia delante (o hacia atrás, si n es negativo).

Eventos:

Ninguno.

Tipos de eventos

Aquí se listan los principales eventos que se pueden asociar a una imagen, hipervínculo, cadena de texto, etc. A continuación aparecen agrupados según donde se originen (ratón teclado, etc). Estos eventos son los mas usados. Existen otros, pero no siempre funcionan bien con los dos navegadores, y además son bastantes exóticos, académicos y de uso poco práctico. No los veremos aquí.

Eventos de ratón

ONCLICK	Se activa con un botón del ratón.
ONDBLCLICK	Se activa si se hace un doble click.
ONMOUSEDOWN	Se activa si se pulsa el botón izquierdo del mouse.
ONMOUSEMOVE	Se activa si se mueve el mouse.
ONMOUSEOVER	Se activa cuando el puntero se sitúa sobre el objeto que incluye al evento.
ONMOUSEOUT	Se activa cuando el puntero sale del objeto que incluye al evento.
ONMOUSEUP	Se activa si se suelta un botón pulsado en el mouse (es contrario a ONCLICK).
ONDRAGSTART	Se activa cuando se inicia un arrastre.
ONSELECTSTART	Se activa cuando se inicia una selección con el ratón.
ONSELECT	Se activa cuando se ha realizado una selección con el ratón.

Eventos de teclado

ONKEYDOWN	Se activa si se pulsa una tecla cualquiera.
ONKEYPRESS	Se activa si se pulsa y suelta una tecla.
ONKEYUP	Se activa cuando se suelta una tecla pulsada.
ONHELP	Se activa si se pulsa la tecla de ayuda (normalmente F1).

Eventos de enfoque

ONFOCUS	Se activa cuando se entra en el ámbito de un elemento al que está asociado el evento.
ONBLUR	Se activa al abandonar el ámbito del elemento al que está asociado.

Eventos de formulario

ONRESET	Se activa al pulsar un botón de reset de un formulario.
ONSUBMIT	Se activa al enviar un formulario.

Eventos de carga de página

ONABORT	Se activa cuando se aborta la carga de la página.
ONERROR	Se activa cuando se produce un error inesperado durante la carga de la página.
ONLOAD	Se activa cuando se carga la página.
ONUNLOAD	Se activa cuando el usuario descarga la página (es decir, carga otra o pretende salir del navegador).
ONAFTERUPDATE	Se activa si se actualiza o recarga la página.

APÉNDICE B: Subtipos de datos.

A continuación aparecen listados los subtipos de datos aceptados por VB. Son los siguientes:

String	Datos de tipo cadena (también llamado alfanuméricos).
Byte	Números enteros del 0 al 255.
Integer	Números enteros del -32.768 al 32.767.
Long	Números enteros del -2.147.483.648 al 2.147.483.647.
Single	Números en coma flotante de simple precisión.
Double	Números en coma flotante de doble precisión.
Currency	Números en coma flotante del -922.337.203.685.477,5808 al 922.337.203.685.477,5808.
Boolean	Datos lógicos verdadero o falso (true o false).
Null	Un dato Variant sin definir contenido de ningún subtipo.
Date	Un valor de Fecha / Hora.
Object	Contiene la representación de un objeto.

Error	Identifica los errores mediante un número.

Una consideración importante respecto a los valores numéricos. Si la configuración regional de nuestro PC tiene fijado como separador de decimales la coma y el navegador (y por tanto el intérprete de VBScript) está en castellano, los números decimales deberán introducirse con una coma. Si se teclea un punto no funcionará bien nuestro código. Consulta el panel de control de tu PC y tu manual de Windows si tienes dudas al respecto.

APÉNDICE C: Operadores en VBScript.

Aquí vamos a conocer cuales son los operadores que maneja VBScript, clasificados según el tipo de operación que realizan. Los operadores son unos signos que se usan para realizar operaciones matemáticas, comparativas, concatenatorias o lógicas.

Operadores

Operador Operación
+ Suma aritmética. Concatenación de cadenas.
- Resta aritmética
* Multiplicación aritmética.
/ División (muestra el resultado en coma flotante).
\ División (muestra la parte entera del resultado).
^ Potenciación
Mod Obtiene el módulo de una división
= Igual que
<> No igual que
> Mayor que
< Menor que
>= Mayor o igual que
<= Menor o igual que
& Concatenación de datos de diferente subtipo.
And Devuelve true si las dos expresiones a ambos lados del operador son verdaderas.
Or Devuelve true si alguna de las expresiones es verdadera
Xor Devuelve true si sólo una de las dos expresiones es verdadera.
Not Cambia una expresión verdadera en una falsa y viceversa. Devuelve true si la expresión es falsa.