Tutorial de Visual Basic Script

Autores del manual

Este manual ha sido realizado por los siguientes colaboradores de DesarrolloWeb.com:

Miguel Angel Alvarez
Director de DesarrolloWeb.com
http://www.desarrolloweb.com
(20 capítulos)

Darwin Manuel Díaz Garrampié
Br. Ing. de Sistemas - Universidad Nacional de Trujillo
(1 capítulo)

Introducción a Visual Basic Script

El lenguaje para describir páginas, HTML, queda limitado a la hora de definir cualquier tipo de interactividad. Una vez que hemos explotado su potencia, estamos en necesidad de aprender algún lenguaje nuevo para hacer pequeños efectos o interactividades.

Scripts
Son los pequeños programitas que, incrustados en las páginas, nos permiten definir aquellos efectos o interactividades.

Visual Basic Script
En este manual nos vamos a ocupar de Visual Basic Script, un lenguaje compatible con Internet Explorer y otros sistemas Microsoft, por lo que en principio es una ventaja para programadores experimentados en estos sitemas.

Otros lenguajes
Existen dos tipos principales de lenguajes de scripting, y multitud de utilidades distintas, pero cabria destacar el lenguaje **Javascript** por ser parecido en utilidad a VBScript pero compatible con los dos navegadores más habituales.

Cómo poner scripts

Para poner un script en una página web utilizamos la etiqueta de HTML **<SCRIPT>**. Todo lo que pongamos entre esa etigueta y la de cierre, **</SCRITP>**, tiene que ser código del lenguaje de scripting que estemos utilizando.
También debemos indicar el lenguaje con el que estamos programando. En nuestro caso pondremos:

```
<script language="VBScript" >
      ---Aqúi pondremos nuestros
scripts---
</script>
```

Parece una tonteria, pero fijaros que la palabra **language** en inglés se escribe con dos "G": **language**. Si os equivocais en este punto, cosa bastante probable si escribis rápido y despistados, no funcionarán vuestros scripts pues el navegador pensará que están escritos en JavaScript.

Primer Script sencillo

Para terminar este capítulo vamos a ver un primer ejemplo de script en una página web. El objetivo de este script es mostrar la fecha de la última modificación del documento

```
<html>
<head>
  <title> La &uacute;ltima modificaci&oacute;n del documento</title>
</head>
<body>
<h1>Script de la &uacute;ltima modificaci&oacute;n de un
documento</h1>
<script language="VBScript">
  document.write "Este documento fue actualizado por &uacute;ltima vez
en: "
  document.write document.lastmodified
</script>
</body>
```

```
</html>
```

La sentencia **document.write** es un procedimiento que escribe en la página web el texto que recibe por parámetro, el texto que esta después de la sentencia.
La variable **document.lastmodified** almacena la fecha y la hora de la última actualización.
Este script dará como resultado que el documento te informe de su última actualización, de una manera parecida a esta:

*Artículo por **Miguel Angel Alvarez***

Primeros pasos con el lenguaje

Los lenguajes de scripting tienen una serie de características comunes, estas suelen hacer la programación más fácil para personas inexpertas, pero a la larga pueden convertirse en una fuente de errores. Veamos cuáles son estas carcaterísticas, en concreto para VBScript.

Mayúsculas y minúsculas
En VBScript no importa si utilizamos mayúsculas o minúsculas a la hora de escribir nuestro código.

Variables
Las variables son espacios donde se almacenan los datos que utilizan los programas o scripts.
No se declaran: en VBScript las variables no se han de declarar, es decir, cuando necesitemos una variable, simplemente la utilizamos y ya está. Aín así, si deseamos declarar una variable utilizamos la palabra **DIM**
No hay tipos: las variables no están tipadas, esto quiere decir que podemos guardar en ellas igualmente números que letras que otras cosas.

Saltos de línea
Son importantes los saltos de línea. Expresan el final de una instrucción y el principio de la siguiente. No se pueden poner dos instrucciones en una misma línea.

Comentarios
En VBScript los comentarios se colocan con una comilla simple **'**. Esto sirve para que todo lo que se encuentre en esa línea después de la comilla simple sea ignorado por el explorador.

Ejemplo de todo esto

Vamos a ver a continuación un sencillo script que sirve de ejemplo para todo lo dicho anteriormente.
El ejemplo siguiente despliega unas ventanitas con mensajes (sentencia **msgbox**) siendo los mensajes el contenido de la variable **pepe**. Durante el ejemplo se cambia el valor de la variable y se vuelve a mostrar.
El ejemplo demuestra que no importan las mayúsculas y minúsculas y que es indiferente el tipo del contenido de la variable, texto o números.

```
<HTML>
<HEAD>
   <TITLE>Ejemplo2 Comentario, caja alert y
variables</TITLE>
</HEAD>
```

```
<BODY>
<script language=VBScript>
	'Esto es un comentario
	PEPE="HOLA"
	msgbox(pepe)
	pepe=3456
	'NO importan las mayusculas-minusculas
	msgbox(PEPE)
</script>
</BODY>
</HTML>
```

Pincha aquí para ver el ejemplo

*Artículo por **Miguel Angel Alvarez***

Distintas formas de ejecutar scripts

Ahora que ya sabes cómo incluir scripts en tus páginas y unos cuantos fundamentos del lenguaje, vamos a ver los dos casos en los que Internet Explorer puede ejecutar tus scripts, de paso que le damos un primer vistazo a el concepto de evento.

Las formas de ejecución de VBScript son las siguientes:

- Scripts que se ejecutan mientras que el navegador abre la página.
- Scripts que se ejecutan como respuesta a la acción de un usuario.

El primero de los casos se utiliza cuando quieres hacer algo cuando el navegador carga la página. Por ejemplo, podrías mostrar un mensaje de bienvenida que aparezca cuando el usuario entra en tu página, o que el navegador te informe de la última actualización del documento (Tal como se vió en el capitulo 1).

El segundo caso es útil cuando deseas realizar acciones como respuesta a eventos del usuario.

Los **eventos** son acciones que ocurren cuando un usuario hace alguna cosa sobre la página web, es decir, un evento podría ser que el usuario escriba algo en una caja de texto, o que se coloque con el ratón encima de un enlace, y así un montón de cosas. Casi cualquier cosa que puede realizar el usuario dentro de la página tiene un evento relacionado.

Utilizando los eventos podemos preparar algún efecto que sea solo visible cuando el usuario realice alguna acción dentro de la página web.

Ejemplo de todo esto

Veamos ahora un ejemplo para acabar de comprender las dos formas de ejecución de los scripts

Vamos hacer que el navegador nos diga su número de versión y otros datos en un cuadro de diálogo. Lo vamos a hacer de las dos maneras, según se carga la página y cuando el usuario aprete un botón.

Ejemplo de ejecución al cargar la página
Comencemos por la ejecución de scripts cuando el usuario carga la página. Esta es la forma

más sencilla, y realmente ya conoces varios ejemplos de esto que viste en los anteriores capítulos.

```
<html>
<head>
  <title>Escript de ejecución directa</title>
</head>
<body>
   Según se carga la página vamos a ver
   la versión del navegado en una caja de
diálogo.
   <script language=vbscript>
      msgbox(navigator.appVersion)
   </script>
</body>
</html>
```

Este ejemplo no tiene ningún misterio, pues es muy parecido a los ejemplos que hemos realizado. la única novedad es la variable **navigator.appVersion**. Esta almacena lo que queremos que se vea en la caja de diálogo: la versión del navegador.

Pincha aquí para ver el ejemplo

Ejemplo de ejecución como respuesta a la acción del usuario

Ahora veamos lo que hay que hacer cuando deseamos que esta caja de diálogo no aparezca hasta que el usuario pulse en un botón.

```
<html>
<head>
    <title>Script de ejecución como respuesta a un evento</title>
</head>
<body>
 Pulse el botón para ver la versión del navegador
 <input type=button value=Pulsame onclick="msgbox(navigator.appVersion)"
language="vbscript">
</body>
</html>
```

Este ejemplo tiene cosas nuevas que habría que destacar:

1. Se crea un botón con la etiqueta <INPUT>
2. Se le añade el atributo **onclick**. Este sirve para indicar (en lenguaje de script) las acciones que queremos realizar como respuesta al evento "click sobre el botón".
3. Se le añade el atributo **language** para especificar el lenguaje en el que está escrito el código script asociado al evento.

Ahora tenemos un botón que, cuando se pulse, ejecutará el código que despliega una caja de diálogo con la versión del navegador.

Pincha aquí para ver el ejemplo

*Artículo por **Miguel Angel Alvarez***

Declarar variables

Antes de pasar a temas más interesantes queda explicar la forma de declarar variables en VBScript. Hace poco se dijo que no era necesario declarar estas variables, pero puede ser una buena costumbre hacerlo y nos puede evitar errores.

Una variable se declara utilizando la palabra **DIM**, veamos cómo:

```
<script language="vbscript">
  dim mi_nueva_variable
  'Ahora ya existe la variable
  'Seguidamente voy a hacer uso de ella
  mi_nueva_variable = "Valor de la
variable"
</script>
```

Como se ha de recordar no importa que tipo de información va a contener la variable, siempre se declaran igual.

Option explicit

Se puede utilizar la clausula **Option explicit** para forzar la declaración de variables en nuestros scripts. Si deseas evitar la posible fuente de errores que supone la libertad de no declarar las variables puedes utilizar esta clausula y hará que tus scripts respondan con mensajes de error si utilizas una variable que no has declarado previamente. Veamos su uso con un ejemplo:

```
<HTML>
<HEAD>
 <TITLE>Option explicit</TITLE>
</HEAD>
<BODY>
 <script language=vbscript>
   option explicit

   dim Pepe
   pepe = 3
   tomas = 87
 </script>
</BODY>
</HTML>
```

Este script responderá con un mensaje de error cuando se ejecute, pues la variable tomas no se ha declarado antes de su uso.

*Artículo por **Miguel Angel Alvarez***

Tipos de datos

Visual Basic Script posee varios tipos de datos pero en la práctica sólo posee un tipo de variable, que va cambiando de un estado a otro según la información que introducimos dentro. Este tipo *principal* de datos es el tipo **Variant**, en él podemos introducir varios *subtipos* de datos con total libertad.

Para cambiar el subtipo de un variant, sólo tenemos que introducir un dato en la variable. La variable variant cambia automáticamente de un subtipo a otro, sin que tengamos que hacer ninguna operación adicional. Los distintos subtipos de datos que tenemos son los siguientes:

Booleano	Es un tipo de datos que contiene un si o un no. se corresponden: TRUE equivale a (-1) FALSE equivale a (0)
Byte	Numérico, entero sin signo hasta 65.000
Currency	Tipo de **moneda**, se utiliza para manipular de manera exacta valores monetarios, y en general cualquier cálculo que requiera una precisión de hasta 15 dígitos decimales

Fecha	Es un tipo de 64 bits de tamaño que almacena fechas. Se utiliza el formato americano: mes, día, hora.
Double	Coma flotante con doble precisión (64 bits)
Entero	Número entero, con signo. Desde -32.768 hasta 32.767
Entero largo	Este tipo es un valor entero con signo de doble precisión. Como los nuevos ordenadores trabajan con palabras de 32 bits, y no menos, se recomienda usar este tipo antes de el tipo entero normal.
Objeto	El subtipo de objeto es una referencia de puntero de 32 bits a una instancia de de objeto de automatización OLE. Los controles Active-X y java. Utilizan esta sintaxis: Set miobjeto = new oleObjeto
Single	Coma flotante de precisión simple
Cadena	Conjunto continuo de valores de caracteres, de longitud variable.

Como saber de qué subtipo es una variable

Para averigüar el subtipo de una variable podemos utilizar la función VarType, de esta manera:

```
v1 = 3

document.write VarType (v1)
```

En este script declaramos una variable y le metemos un número y a continuación imprimimos en la página el valor que devuelve la función VarType.

Al ejecutar este script podremos ver un "2" escrito en la página.

Según el tipo de datos que halla en la variable, VarType devolverá un valor distinto, como indica esta tabla:

Constant	Value	Description
vbEmpty	0	Empty (uninitialized)
vbNull	1	Null (no valid data)
vbInteger	2	Integer
vbLong	3	Long integer
vbSingle	4	Single-precision floating-point number
vbDouble	5	Double-precision floating-point number
vbCurrency	6	Currency
vbDate	7	Date
vbString	8	String

vbObject	9	Automation object_
vbError	10	Error
vbBoolean	11	Boolean
vbVariant	12	Variant (used only with arrays of Variants)
vbDataObject	13	A data-access object
vbByte	17	Byte
vbArray	8192	Array

*Artículo por **Miguel Angel Alvarez***

Operadores I - Aritméticos VBSscript

Visual Basic Script, como cualquier lenguaje de programación, tiene un conjunto de operadores, divididos en varias secciones:

Operadores Aritméticos

Que soportan las operaciones matemáticas más sencillas.

+	Suma
-	Resta
*	Multiplicacion
/	División en coma flotante. Es la división normal. Devuelve un numero real si es el resultado
\	División de enteros Devuelve un numero entero, resultado de la división.
^	Potencia
Mod	Resto de la división

Veamos a continuación un ejemplo de script que realiza acciones con estos operadores:

```
dim v1

dim v2

v1 = 34

v2 = 43

suma = v1 + v2

resta = v1- v2
```

```
potencia = v1 ^ v2

divisionEnteros = v1 \ v2

msgbox(divisionEnteros)

DivisionReal = v1 /v2

msgbox(divisionReal)
```

La función msgbox sirve para mostrar un valor en una ventanita de alerta típica de Windows. Podemos ver este script en funcionamiento.

*Artículo por **Miguel Angel Alvarez***

Operadores II - Comparación VBScript

Para realizar comparaciones, Visual Basic Script posee los siguientes operadores:

= <>	Igual y distinto
> <	Mayor que y menor que
>= <=	Mayor o igual que y menor o igual que

Veamos a continuación un ejemplo de script que realiza operaciones de comparación, aunque antes que verlo deberíamos puntualizar que los operadores de comparación se suelen utilizar dentro de una estructura condicional, que evalúa una expersión con estos comparadores y realiza acciones dependiendo del resultado de esas comparaciones. Por este motivo hemos incluido en el script la estructura condicional IF que veremos con profundidad más adelante.

```
precio = 20000

dineroActual = 3500

if (dineroActual = precio) then

        msgbox ("Lo tienes justo")

end if

if (dineroActual < precio) then

        msgbox ("te falta dinero")

end if
```

Podemos ver este script en funcionamiento.

*Artículo por **Miguel Angel Alvarez***

Operadores III lógicos y cadenas VBScript

Operadores lógicos

Los operadores lógicos se utilizan sobre expresiones booleanas y nos devuelven un valor booleano (verdadero o falso) resultado de esa operación. Un matiz sería que no es necesario que los operandos relacionados en el cálculo sean variables booleanas, pudienso ser de cualquier tipo.

Como operadores lógicos en Visual Basic Script tenemos:

AND	Y lógico
OR	O lógico
Xor	Xor
Not	NO lógico

Operadores de cadenas

Como operador de cadenas de caracteres en Visual Basic Script tenemos un único ejemplo: la concatenación. El operador para concatenar cadenas es el **&**. Veamos un ejemplo de utilización de este operador:

```
cadena1 = "Hola "

cadena2 = "Pepe"

concatenar = cadena1 & cadena2

msgbox (concatenar)
```

Podemos ver este script en funcionamiento.

*Artículo por **Miguel Angel Alvarez***

Estructuras de control

Las estructuras de control nos permiten realizar acciones típicas en nuestros scripts como lo pueden ser los bucles o la toma de decisiones.

VBScript tiene las estructuras de control típicas de los lenguajes de programación. Vamos a ver la sintaxis y la manera de trabajar de estas estructuras detenidamente:

Condicionales

- **IF**, condicional que decide entre si/no
- **CASE**, otro condicional con varias posibilidades

Bucles

- **FOR**, repetición un determinado número de veces
- **FOR EACH**, repetición para un conjunto de elementos
- **WHILE...WEND**, repetición mientras ocurra alguna cosa
- **DO...LOOP**, repetición un determinado número de veces

*Artículo por **Miguel Angel Alvarez***

Estructura IF

La estrutura de control IF permite decidr entre dos opciones resultantes de la evaluación de una sentencia. Si la evaluación es positiva hace una cosa, también podemos especificar accioens para realizar en caso de que la evaluación sea negativa. Veamos cómo funciona en VBscript.

```
IF (expresion) then
    Sentencias
    ....
END IF
```

Vemos que en primer lugar tenemos la sentencia IF, luego una expresión, que puede o no ir entre paréntesis, y más tarde la parabra THEN. Vemos que luego hay un salto de línea antes de colocar las sentencias asociadas a la evaluación positiva de la sentencia. En VBScript las líneas si que importan.
Después de poner las sentencias del asociadas a la evaluación positiva colocamos un END IF, para acabar la estructura del IF.

Enunciado ELSE

Opcionalmente se puede colocar una serie de sentencias asociadas a la evaluación negativa de la expresión. Estas sentencias se deben colocar después de la orden ELSE y antes del END IF.

```
IF (expresion) then
    Sentencias
    ....
ELSE
    Sentencias
    ....
END IF
```

Enunciado ELSEIF

En Visual Basic Script existe la posibilidad de utilizar un enunciado especial en el lugar donde utilizaríamos un ELSE. Sirve para encadenar sentencias IF de modo que en un resultado negativo de un IF se pueda evaluar otra expresión, que tendría a su vez otros enunciados THEN y probablemente ELSE u otro ELSEIF. Se vería en un ejemplo com más facilidad:

```
IF (expresion1) then
     Sentencias1
     ....
ELSEIF (expresion2) then
     Sentencias2
     ....
ELSEIF (expresion3) then
     Sentencias3
     ....
ELSE
     Sentencias4
     ....
END IF
```

Se evalúa la primera expresión, en caso positivo se ejecutan las sentencias1, en caso negativo se evalua la expresion 2. Si la expresión 2 es positiva se ejecutan las sentencias 2 en caso negativo evaluamos la expresión 3 con el siguiente ELSEIF. Todo acaba en un ELSE en este ejemplo, pero el ELSE final no es obligatorio.

*Artículo por **Miguel Angel Alvarez***

Estructura CASE

Con la estructura de control CASE podemos evaluar una variable y realizar acciones dependiendo del valor de esta. La diferencia con el IF consiste en que el número de posibilidades de la evaluación de esta variable no tiene por que ser si o no, pudiendo hacer cosas para un número indeterminado de valores.

La sintaxis es la siguiente:

```
SELECT CASE (variable)
   CASE (valor1):
            (acción para caso valor1)
   CASE (valor3):
            (acción para caso valor2)
   CASE (valor3):
            (acción para caso valor3)
   CASE ELSE:
            (accion en caso de que no se cumpla ningun anterior caso)
END SELECT
```

Funciona asi, primero se evalua la variable, si esa variable tiene como valor el valor1 realizamos las acciones asociadas al valor1. Si tiene el valor2, ejecutamos las acciones relacionadas con este valor3. Así con cuantos valores deseemos. Por último tenemos un ELSE para realizar acciones en caso de que no hubiesen sido ninguno de los valores anteriores. Este ELSE es opcional.

Veamos con un ejemplo esta sentencia muy sencillito. Lo primero que hace es solicitar un número y luego informa del día de la semana con el que corresponde. Si el número no es del uno al siete informa de ello también.

```
dim dia
dia = inputbox ("dime un dia de la semana")
SELECT CASE dia
CASE 1:
```

```
  msgbox("El dia es LUNES")
CASE 2:
  msgbox("El dia es MARTES")
CASE 3:
  msgbox("El dia es MIERCOLES")
CASE 4:
  msgbox("El dia es JUEVES")
CASE 5:
  msgbox("El dia es VIERNES")
CASE 6:
  msgbox("El dia es SABADO")
CASE 7:
  msgbox("El dia es DOMINGO")
CASE ELSE:
  msgbox("Tiene que ser un dia de la semana en número, del 1 al 7")
END SELECT
```

Este script se puede ver en funcionamiento.

*Artículo por **Miguel Angel Alvarez***

Bucle FOR

La sentencia FOR se utiliza para los bucles, cuando sabemos el número de veces que debemos ejecutar el bucle. Veamos su sencilla sintaxis:

```
FOR (inicializacion) TO (termino del bucle) STEP (paso)
    sentencias
    ........
NEXT
```

La sentecia realiza una repetición desde la *inicialización* hasta el *término del bucle*. Para llevar la cuenta se utiliza una variable, ya veremos en el ejemplo cómo se utiliza esta variable. Con cada ejecución del bucle se ejecutan unas sentencias. NEXT sirve para delimitar el final del bucle, cuando se encuentra con el NEXT se vuelve otra vez al principio del FOR, así hasta realizar el número de ejecuciones determinado.

Existe un valor que sirve para indicar lo grandes que se desean realizar los saltos entre ejecución y ejecución, es el valor STEP. Un STEP 2 determinaría que entre ejecución y ejecución la variable se ha de incremetar en 2 unidades. En el caso de no indicar nada se realizan pasos de 1 en 1. También podemos realizar pasos en valores negativos.

Un ejemplo de estos datos sería el siguiente:

```
for i=0 to 6 step 2
    msgbox(i)
next
```

Este ejemplo presentaría un mensaje con un numerito de la variable i, utilizada para llevar la cuenta de las ejecuciones del bucle. Se puede ver un ejemplo de este script funcionando.

*Artículo por **Miguel Angel Alvarez***

Bucle FOR EACH

La estructura de control FOR EACH sirve para moverse por los elementos de una estructra de datos, como podría ser un vector, y realizar acciones para cada una de los elementos.

Veamos con un ejemplo esta estructura de control: En el ejemplo primero creamos un vector y rellenamos con números cada una de sus casillas, con un bucle FOR normalito. Más tarde utilizamos el bucle FOR EACH para acceder a cada una de las posiciones de este vector de números y escribir en la página cada una de estos números.

```
dim tor(20)
for i=0 to 20
   tor(i) = i
next
for each i in tor
   document.write (tor(i))
next
```

Fijemonos en el segundo bucle, se indica que para cada i (i es el índice con el que podemos movernos en el bucle y en la estructura) dentro de tor (que es la estructura, en este caso un vector) haga un document.write(tor(i)). Con tor(i) accedemos a la casilla actual y document.write() sirve para escribir algo en la página web. Conbinadas lo que se escribe es lo que hay en la posición actual del vector. Se puede ver este script funcionando.

Nota: El ejemplo no esta probado para Mozilla Firefox.

*Artículo por **Miguel Angel Alvarez***

Bucle WHILE WEND

El bucle WHILE...WEND sirve para realizar un tipo de bucle muy utilizado en programación que es el bucle Mientras, que se ejecuta mientras que se cumpla una condición. A diferencia del bucle FOR, éste se utiliza cuando no conocemos el número de iteraciones que tenemos que realizar.

El bucle funciona de la siguiente manera. Cuando se va a ejecutar, evalúa una expresión y compreba que esta da resultados positivos. Si es así, ejecuta el cuerpo del bucle (las sentencias que siguen hasta el WEND), en caso contrario se sale. Podemos ver la sintaxis a continuación.

```
WHILE (condicion)
        sentencias
        ....
WEND
```

Ahora vamos a ver un ejemplito sobre este bucle, que realiza una cuenta número a número hasta llegar al 13. En cada iteración del bucle muestra en una ventanita el número actual y ofrece la posibilidad de cambiarlo, ya que la ventanita es una ventana Input, que ofrece la oportunidad de cambiar el valor y devuelve ese valor, cambiado o no. Como decíamos, si dejamos el ejemplo sin tocar nada, cuenta hasta 13, pero si introducimos un número en el inputbox continúa la cuenta por el número introducido. Si el número introducido es mayor que 13 también se sale del bucle.

```
option explicit
dim a
a = 0
WHILE (a < 13)
  a = a + 1
  a = inputbox("Dame un valor entero, please","Petición de número",a,200,100)
WEND
```

Podemos ver este ejemplo en funcionamiento.

*Artículo por **Miguel Angel Alvarez***

Bucle DO LOOP

El bucle DO...LOOP es muy versatil. Con el se pueden crear gran variedad de bucles distintos, bucles que comprueben una condición antes de ejecutar el bucle una vez, después de la primera ejecución y con conbinaciones con mientras (WHILE) que se cumple una condición o hasta (UNTIL) que esa condicion se cumpla. la sintaxis de esta estructura es la siguiente:

```
DO [WHILE | UNTIL (condicion)]
    Sentencias
    .....
LOOP [WHILE | UNTIL (condicion)]
```

Vamos a tratar de explicar esta sentencia de manera pausada para que sea más fácil de entender. Lo que siempre tendremos en estos bucles es el DO y el LOOP, entre estos dos colocaremos las sentencias que queremos ejecutar en cada iteración del bucle. Los bucles tienen que evaluar entre cada iteración si se siguen ejecutando o no, para ello evaluan una condición. Lo versatil de este bucle es que la condición se puede expresar de muchas maneras distintas.

Condición expresada al lado del DO: en este caso la condición se evalua antes de empezar a ejecutarse el bucle.
Condición expresada al lado del LOOP: en este caso la condición se evalua después de ejecutarse el bucle. Tiene como diferencia principal frenta al otro método que en este caso el bucle se ejecutará por lo menos una vez.

Además de poder expresar la condición en estos dos sitios también se puede construir la condición con un enunciado mientras (WHILE) o un enunciado hasta (UNTIL). Las diferencias semanticas de estas dos posibilidades se trasladan también a su manera de funcionar.

Vamos a ver un par de ejemplos de este bucle para comprender su funcionamiento. El ejemplo pide cunstantemente el nombre del autor de la página y no para hasta que el nombre sea "migue". También tiene el usuario la posibilidad de escribir "out", en ese caso, comprobado con un enunciado IF, se sale del bucle rompiéndolo con la sentencia EXIT DO, utilizada para romper bucles.

```
Dim entrada
entrada = ""
DO WHILE (entrada <> "migue")
  entrada = inputbox ("Dime el nombre del autor","seguridad","migue",2,3)
  if (entrada = "out") then
    msgbox "salgo por la puerta de atras"
     exit do
  end if
LOOP
```

Podemos ver este ejemplo en funcionamiento.

El siguente ejemplo realiza una cuenta y entre cuenta y cuenta se muestra el valor de la cuenta actual en una ventanita donde sale un botón de Reintentar y otro de Cancelar. Si se pulsa reintentar se sigue ejecutando el bucle y si se pulsa Cancelar se sale por la puerta de atrás, de manera similar a como se salía en el ejemplo anterior, con EXIT DO.

```
option explicit
dim cont
dim respuesta
cont = 0
DO
  cont = cont +1
  respuesta = msgbox (cont,69,"Variable del bucle, con valor 6 se sale")
  if (respuesta = 2) then
     msgbox "Cuenta Cancelada",16,"Cancelaste!"
     exit do
  end if
LOOP UNTIL (cont = 6)
```

El ejemplo es un poco rarito, pero servirá para comprender estos bucles. Se puede ver en funcionamento para entender mejor la manera de ejecutarse que tiene este bucle.

*Artículo por **Miguel Angel Alvarez***

Arrays en VBScript

Los Arrays o matrices son unas estructuras de datos muy utilizadas en cualquier lenguaje. Se tratan de variables, pero que están preparadas para guardar una cantidad mayor de elementos. Es como una variable que tiene varios compartimentos para guardar la información y a cada uno de esos compartimentos hay que acceder con un índice.

Antes de utilizar un array debemos declararlo de manera obligatoria, para ello utilizamos la palabra clave DIM, de este modo.

```
dim miArray(20)
```

Después de la palabra DIM debemos indicar el nombre del array y a continuación, entre paréntesis, se coloca el número de posición máxima del array, en este caso 20.

Los arrays en ASP comienzan desde la posición 0, es decir, el primer elemento de un array está en la posición 0. Por tanto, si el array ha sido definido con 20 casillas, como en el ejemplo, tendrá 21 elementos, primera posición será la 0 y la última posición sería la 20.

Para asignar un valor a un array se realiza igual que una variable, pero accediendo con el índice de la posición que queremos escribir.

```
miArray(0) = 234
```

Para utilizar el contenido de un array debemos hacerlo indiando el índice al que se desea acceder. Por ejemplo, si quisiesemos imprimir en la página la primera posición de nuestro Array lo haríamos de esta manera.

document.write(miArray(0))

Ahora vamos a ver un ejemplo sobre cómo utilizar los arrays, donde vamos a realizar dos recorridos, uno para escribir en él y el otro para leer la información y escribirla en la página.

```
dim matriz (10)
for i=0 to 10
  matriz(i)=100 * i
next
for i=0 to 10
  document.writeln("Posicion " & i & ": " & matriz(i) & "<br>")
next
```

Este ejemplo escribiría en la página las posiciones del array, que contienen variables numéricas que corresponden de multiplicar su índice por 100.

Si se desea, se puede ver el efecto resultante en una página web.

Arrays multidimensionales en VBScript

Se pueden construir matrices multidimensionales, es decir, que nos permitan crear matrices de varias coordenadas. Para trabajar con ellos se utiliza una coma que separa los dos índices. Por ejemplo podemos definir una matriz de 8x8 de esta manera.

dim miArray2Dimensiones (7,7)

Como el array es de 8 casillas, utilizamos un 7 y sus posiciones serán las 8 que van desde el 0 al 7. Para escribir y leer del Array podemos utilizar la coma de manera similara a como se declara. Por ejemplo, para meter datos en la posición 0,2 haríamos lo siguiente:

miArray2Dimensiones (0,2) = "texto posicion 0,2"

Redimensionar arrays

Podemos declarar también arrays que cambien el número de casillas que tienen según se necesite en tiempo de ejecución. Este tipo de arrays redimensionables se llama array dinámico. Para crear este tipo de arrays podemos utilizar la sentencia dim (como creábamos los anteriores) o la sentencia redim, con la particularidad que no le colocamos ningún valor entre paréntesis donde antes indicábamos el número de casillas del array.

```
dim mi_array()
redim mi_otro_array()
```

Cuando usamos arrays dinámicos podemos utilizar la sentencia redim para indicar el número de dimensiones y la cantidad de casillas de cada dimensión.

Con esta sentencia estamos indicando que mi_array debe tener el tamaño 10. Casillas desde la 0 hasta la 10.

redim mi_array(10)

Si indicamos la clave "preserve" estamos asegurándonos que el contenido de las casillas que había previamente en el array se mantiene.

redim preserve mi_array(20)

Por último, si en cualquier momento reducimos el número de casillas perderemos lo que pudiera haber guardado en las casillas que se han eliminado.

En el siguiente ejemplo creamos un array dinámico y lo redimensionamos inicialmente a tamaño 3. Lo rellenamos y mostramos sus distintos valores. Posteriormente lo

redimensionamos otra vez para que llegue hasta la posición 7, guardando los valores antiguos. Para acabar rellenamos las casillas que hemos creado nuevas y mostramos todos los valores del array.

```
dim frutas()
redim frutas(3)

frutas(0) = "Pera"
frutas(1) = "Uva"
frutas(2) = "Manzana"
frutas(3) = "Melón"

for each fruta in frutas
   document.write fruta & "<br>"
next

redim preserve frutas(7)

frutas(4) = "Sandía"
frutas(5) = "Naranja"
frutas(6) = "Plátano"
frutas(7) = "Mandarina"

for each fruta in frutas
   document.write fruta & "<br>"
next
```

El ejemplo se puede ver en funcionamiento en una página nueva.

> **Nota:** Los arrays de más de una dimensión también se pueden redimensionar, pero sólo se puede alterar la última dimensión.
>
> Por ejemplo, en un array de dos dimensiones miarray(2,4), se podría redimensionar la segunda dimensión redim miarray(2,8). O en un array de 3 dimensiones, donde también podríamos cambiar tan sólo la última dimensión.
>
> Un ejemplo de código que hace esto es:
>
> ```
> dim datos_prueba()
> redim datos_prueba(1,0)
>
> datos_prueba(0,0)=12
> datos_prueba(1,0)="hola!"
>
> redim preserve datos_prueba(1,1)
>
> datos_prueba(0,1)=133
> datos_prueba(1,1)="segunda casilla"
>
> for i=0 to ubound(datos_prueba)
>    document.write datos_prueba(0,i) & "-" & datos_prueba(1,i) & "<br>"
> next
> ```

Obtener el número de casillas de un array

Uno de los datos típicos que necesitamos extraer de un array es su número de posiciones, útil por ejemplo para hacer un recorrido a un array, desde la primera hasta la última casilla. Para ello utilizamos la función **uBound()** de VBScript.

uBound() recibe el array del que queremos obtener su número de posiciones y devuelve la posición más alta del array. Por ejemplo.

```
dim ciudades(5)
document.write ubound (ciudades)
```

Escribiría en la página el número de la casilla más alta del array ciudades, en este caso 5.

Además, por si algún día la necesitamos, también tenemos a nuestra disposición la función **lBound()**, que devuelve el número de la posición con índice menor del array.

document.write lbound (ciudades)

La última línea sobre nuestro array de ciudades definido anteriormente escribiría un 0 en la página web, puesto que el array comienza en la posición cero.

*Artículo por **Miguel Angel Alvarez***

Procedimientos y funciones

Los procedimientos o funciones son muy interesantes y útilies en la programación. Nos sirven para realizar una tarea concreta que probablemente se vaya a ejecutar varias veces a lo largo de la vida de la página. Esta tarea se especifica en un bloque de código de manera independiente y cuando se desean realizar las acciones del procedimiento se llama al porcedimiento o función. Una vez realizadas las acciones pertinentes se devuelve el flujo del programa al lugar desde donde se imbocó ese procedimiento o función.

Lo primero que debemos hacer al crear un procedimiento es pensar las cosas que se desean hacer dentro de la función, la información que necesitaremos (y que tendremos que recibir como parámetros) y la información que devolverá. Con estas ideas claras se pueden construir los procedimientos y funciones sin mucha dificultad, siguiendo estas estructuras.

Para un procedimiento

```
Sub nombre (parametro1, parametro2...)
   ... Código del procedimiento
end Sub
```

Para una función

```
Function nombre (parametro1, parametro2...)
   ... Código de la función
end Function
```

*Artículo por **Miguel Angel Alvarez***

Procedimientos. SUB

Decíamos que un procedimiento era una subrutina que se llamaba y realizaba acciones, pero que no devolvía ningún valor y por lo tanto, no era posible utilizarla dentro de una expresión.

Veamos algún ejemplo de procedimiento. Es una subrutina que escribe en la barra de estado un mensaje. No es muy complicada, pero tal como la presentamos aquí no se debería hacer, puesto que utilizamos un bucle vacío para que el navegador esté un poco más lento y el texto salga poco a poco. En lugar de ese bucle deberíamos utilizar una función llamada setTimeout, pero no deseamos introducirla ahora.

```
sub muestraAbajo(texto)
  dim i
  for i=0 to len(texto)
    dim actual
    actual = left(texto,i)
    window.status = actual
    dim j
    'bucle para ralentizar al navegador debería utilizarse la función setTimeOut
    for j=0 to 20000
      j = j
    next
  next
end sub
```

Este ejemplo utiliza además varias funciones de cadenas de caracteres, esperamos que no represente mucho problema para entenderlo. Básicamente es un bucle que va recorriendo toda la cadena de caracteres que recibe por parámetro. A medida que se realiza el bucle se va creando una subcadena de caracteres de la parte izquierda de la cadena original, que cada vez es más larga. Luego se imprime esa cadena en la barra de estado del navegador. Entre ejecución y ejecución del bucle se realiza un retardo, en el segundo bucle for que se debería realizarse con un setTimeout.

Podemos ver a continuación cómo se colocaría un botón en la página que llamase a este procedimiento.

```
<HTML>
<HEAD>
<TITLE>Procedimientos en VBS</TITLE>
<script language=vbscript>
option explicit
sub muestraAbajo(texto)
 dim i
 for i=0 to len(texto)
  dim actual
  actual = left(texto,i)
  window.status = actual
  dim j
  for j=0 to 20000
   j = j
  next
 next
end sub
</script>
</HEAD>
<BODY>
<h1>Procedimientos en VBS</h1>
<P>
<form>
<input type="button" name=b value=ponerAbajo!
      onclick="muestraAbajo('Saludos de Miguel')" language=vbscript>
</form>
</P>
</BODY>
</HTML>
```

Se puede ver el ejemplo en una página aparte.

*Artículo por **Miguel Angel Alvarez***

Funciones. Function

Ya vimos lo que consistía una función, que no es más que un trozo de código que opera para devolver un valor. Ahora vamos a ver con detenimiento un ejemplo de su uso.

Vamos a definir una función que realice un cálculo matemático y devuelva el resultado del mismo. Los operandos los vamos a extraer de un formulario. El ejemplo puede ser ahora mismo un poco complejo, por tratar con formularios -que no hemos visto todavía-, pero podemos ver el código de la función y hacernos una idea exacta de su uso, que al fin y al cabo es lo que nos importa.

El código de la función será el siguiente:

```
function operar (operador,op1,op2)
  select case operador
    case "+":
      operar = op1 + op2
    case "-":
      operar = op1 - op2
    case "*":
      operar = op1 * op2
    case else:
      operar = op1 / op2
  end select
end function
```

Vemos que la función recibe tres parámetros, el primero es un operador, que no es más que un texto con el signo de la operación a realizar. Los dos siguientes parámetros son los operadores que hay que tratar.

La función realiza una operación matemática dependiendo de del operador y devuelve en cada caso el resultado conveniente. Fijémonos que para devolver un valor se debe realizar una asignación del nombre de la función al valor que se desea devolver.

No creemos que revista ninguna complicación. Vamos a ver ahora el código que podríamos utilizar para hacer la llamada a la función.

```
miOperador="+"
miOperando1=221
miOperando2=32
resultado =
operar(miOperador,miOperando1,miOperando2)
```

Al final de todas estas sentencias la variable resultado tendrá como valor 253.

Veamos el ejemplo completo, que consistía en una calculadora hecha con un formulario, que usa esta función para obtener los resultados.

```
<HTML>
<HEAD>
<link rel=stylesheet type=text/css href=estiloglobal.css>
<TITLE>Funciones en VBS</TITLE>
</HEAD>
<h1>Funciones en VBS</h1>
<script language=vbscript>
function operar (operador,op1,op2)
 select case operador
  case "+":
```

```
   operar = op1 + op2
  case "-":
   operar = op1 - op2
  case "*":
   operar = op1 * op2
  case else:
   operar = op1 / op2
 end select
end function
sub opera ()
 dim res
 operador = document.forms(0).operacion.value
 operando1 = cint(document.forms(0).op1.value)
 operando2 = cint(document.forms(0).op2.value)
  res = operar (operador,operando1,operando2)
 document.forms(0).result.value = res
end sub
</script>
<BODY>
<form>
Operando 1
<input name=op1 >
<br>
Operando 2
<input name=op2 >
<br>
operacion:
<select name=operacion>
<option value="+" selected>+
<option value="-">-
<option value="*">*
<option value="/">/
</select>
<input type=button name=b value="realizar operacion"
        onclick=opera language=vbscript>
<br>
Resultado:
<input name=result >
</BODY>
</HTML>
```

Hemos tenido que utilizar un procedimiento de apoyo para hacer el ejercicio, ya que, en caso de no utilizarlo, haría un poco más compleja a la funció. Podremos enentenderlo todo ya que no reviste mucha complicación y los procedimientos los pudimos ver en el capítulo anterior. Para tener más claro todavía cómo trabaja esta página de ejemplo podemos ver el ejemplo funcionando.

Tenemos un formulario donde podemos ver campos para los operadores, una caja de selección para el operando y un último campo para el resultado. Es interesante también el botón de realizar operación, que es el que lo pone todo en marcha gracias a su manejador de evento onclick, que quiere decir que cuando se pulse sobre el botón se realice una acción. En este caso es una llamada al prodedimiento opera.

En el procedimiento opera podemos ver varias sentencias para extraer la información del formulario y también la llamada a la función que realiza los cálculos. Por último, se introduce en el campo resultado lo que devolbío la función como resultado de realizar las operaciones.

Podemos ver el ejemplo en funcionamiento.

*Artículo por **Miguel Angel Alvarez***

Más sobre procedimientos y funciones

Ahora vamos a ver algunas cosas más sobre subrutinas que nos han quedado en el tintero. Un poco en cajón de sastre.

Llamadas a subrutinas

En Visual Basic Script las funciones se utilizan como partes de expresiones y los procedimientos como si fuera una sentencia independiente.

La llamada a una función, si se utiliza como parte de una expresión se debe llamar utilizando paréntesis.

miResultado = suma(1,2)

Si no se utiliza como parte de una expresión, no tienen por que utilizarse los paréntesis, pero el resultado de la funcuión (lo que devuelve) se perderá.

suma 1,2

Call

Es una llamada a una subrutina, utilizada para transferir el flujo de la aplicación hacia una subrutina. Es necesario utilizar paréntesis cuando se utiliza. Además, si se utiliza con una función se perderá el resultado que devuelva.

call suma(1,2)

Salida de una subrutina

Podemos salirnos de un procedimiento o función en cualquier momento, independientemente de que la función haya termiando o no. El enunciado para escaparse de una función es EXIT, que se puede utilizar en cualquier lugar del procedimiento o función. La palabra exit debe ir acompañada del tipo de subrutina de la que se desea salir, así pues se deberá utilizar o bien **exit function** o **exit sub**.

*Artículo por **Miguel Angel Alvarez***

Imprimir una página sin ver el diálogo de impresión

En este artículo, eminentemente práctico, vamos a ver cómo imprimir una página web sin que se pida confirmación al usuario y sin visualizar la ventana previa de impresión. En este caso, lógicamente, la impresión se realizaría en la impresora configurada como predeterminada en el sistema cliente y con las opciones definidas por defecto para esa impresora.

Es un script en el lenguaje VBScript, que como probablemente sepamos, sólo será compatible con Internet Explorer. Si alguien conoce cómo se puede hacer esto con Javascript (si es que es posible), para que sea compatible con todos los navegadores, le ruego que incluya un comentario al artículo para compartir la información.

El código está comentado para que se pueda entender cada paso que se realiza.

```
<html>
<head>

<script language="VBScript">
SUB Print()
OLECMDID_PRINT = 6
OLECMDEXECOPT_DONTPROMPTUSER = 2
OLECMDEXECOPT_PROMPTUSER = 1
'ACA en caso de usar frames,
'enfocamos el frame a imprimir:

'window.parent.frames.main.document.body.focus()
window.document.body.focus()

'Llamamos al comando de Impresión Print

on error resume next
call IEWB.ExecWB (OLECMDID_PRINT, -1)

if err.number <> 0 then
    alert "No se pudo imprimir"
end if

END SUB
</script>

</head>
<body>

<object id="IEWB" width="0" height="0" classid="clsid:8856F961-340A-11D0-A96B-00C04FD705A2"
VIEWASTEXT></object>

Esta es una prueba de una página que se va a imprimir, pulsando el enlace de más abajo, sin pedir confirmación al
usuario.

<a href="javascript:Print ();">Imprimir</a>

</body>
</html>
```

El ejemplo se puede ver en marcha en una página aparte. Recordar que sólo funcionará en Internet Explorer.

*Artículo por **Darwin Manuel Díaz Garrampié***