

# **Curso de páginas web avanzadas con ASP**

Organizado por:



Profesor responsable: Gabriel Plana Gavalda

Julio del 2002

# ÍNDICE

<b>1.- Introducción a ASP .....</b>	<b>3</b>
Programación en el lado servidor .....	4
Breve historia .....	7
ASP vs CGI.....	7
Herramientas que necesitamos.....	9
 <b>2.- El lenguaje VBScript.....</b>	 <b>10</b>
Variables .....	10
Funciones de conversión.....	12
OPERADORES .....	15
Funciones de VBScript .....	16
Sentencias de control (If, select case, while, for, for each). .....	19
Procedimientos y Funciones .....	24
 <b>Objetos implícitos ASP y Componentes ASP .....</b>	 <b>26</b>
 <b>3.- Objetos Application y Session .....</b>	 <b>27</b>
 <b>4.- Global.asa .....</b>	 <b>31</b>
 <b>5.- Response y Request .....</b>	 <b>34</b>
Caché y caducidad de la página.....	34
Recibir información de formularios. Métodos GET y POST .....	36
Trabajar con Cookies .....	38
 <b>6.- Objeto Server .....</b>	 <b>40</b>
HTMLEncode .....	40
Acceso a bases de datos .....	42
ADODB.connection.....	43
ADODB.Recordset .....	45
Lenguaje SQL .....	49
 <b>7.- Seguridad: Proteger nuestra web de Hackers. ....</b>	 <b>54</b>
Caso de un foro. ....	55
Caso de búsquedas con formularios.....	57
 <b>Apéndice A: Gestión de nuestro site .....</b>	 <b>59</b>
 <b>Bibliografía .....</b>	 <b>61</b>

## 1.- Introducción a ASP

ASP, abreviatura de Active Server Pages, es una plataforma creada por Microsoft, destinada a servir como soporte para la creación de páginas web de contenido dinámico.

Sus principales características son las siguientes:

- **Se ejecuta en el servidor:** No es el navegador quien ejecuta código ASP, sino el servidor Web, quien mandará el resultado de la ejecución al navegador.
- **Se programa en lenguaje script:** No es un lenguaje de programación propiamente dicho, ya que no sirve para crear programas independientes. En su lugar, los lenguajes script se entremezclan dentro de un documento (por ejemplo una página HTML) para dar funcionalidad en este documento e interactuar con él.

Esto implica una serie de ventajas respecto el HTML con JavaScript. Son las siguientes:

- **Mayor seguridad:** Al ejecutarse en el servidor, el código fuente nunca es enviado al navegador, con lo que al internauta le es imposible obtener el código fuente de nuestras páginas.
- **Mayor funcionalidad:** Al ejecutarse en el servidor, podemos realizar cosas imposibles de realizar en el cliente, como por ejemplo, guardar datos en una base de datos, compartir datos entre distintos usuarios (por ejemplo, para realizar un contador de visitas), etc...
- **Mayor compatibilidad con los navegadores:** Al poder realizar toda la programación en el servidor, es posible generar páginas que contengan exclusivamente HTML, con lo cual no estamos forzando a que el navegador deba soportar JavaScript. Existen, además, muchas incompatibilidades entre los modelos de objetos de JavaScript de distintos navegadores (como por ejemplo entre Netscape Navigator y Microsoft Internet Explorer).
- **Lenguaje más fácil:** ASP se suele programar en Visual Basic Script (VBScript), el cual es un lenguaje casi idéntico a Visual Basic, lenguaje muy popular por su facilidad y velocidad de aprendizaje.
- **Multi-lenguaje:** ASP no es, en realidad, un lenguaje de programación, sino una plataforma de soporte para la programación de distintos lenguajes script en el servidor. Ello implica que, aunque el lenguaje utilizado suele ser VBScript, también podemos usar otros lenguajes como JavaScript, PerlScript, Rexx, Python, y muchos otros. Esto hace que si un programador conoce muy bien otro lenguaje script, pueda utilizarlo en lugar de VBScript, aprovechando sus conocimientos y experiencia ya adquiridos, es decir, no está obligado a aprender un lenguaje nuevo.

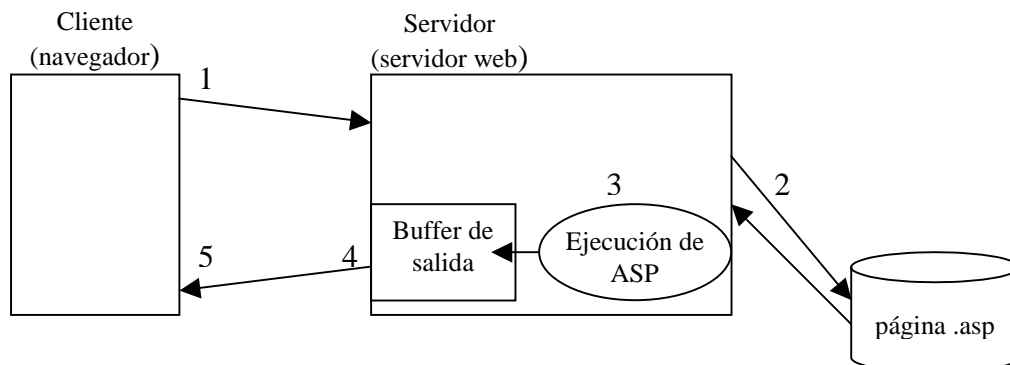
## Programación en el lado servidor

Como se ha comentado, ASP no se ejecuta en el navegador, sino en el servidor web. Cuando un internauta hace una petición de una página ASP se producen los siguientes acontecimientos:

- 1.- El navegador localiza al servidor web y le lanza una petición http donde se le solicita una página asp.
- 2.- El servidor busca esta página asp en su disco (o en su caché) y si no la encuentra genera un error. Si la encuentra la carga en el servidor.
- 3.- El servidor interpreta la página ASP, la cual contiene mezclados código HTML y código script de servidor. Cuando encuentra código HTML lo envía “tal cual” a un buffer de salida. Cuando encuentra código script de servidor, lo ejecuta, y el resultado generado es enviado al mismo buffer de salida.
- 4.- El servidor envia el buffer de salida al navegador, el cual contendrá únicamente código HTML<sup>1</sup>.
- 5.- El navegador recibe el documento html (aunque la extensión que aparece en la URL sea .asp).

Como se puede apreciar, el paso más importante es el paso 3, donde en caso que fuera una página HTML, simplemente se copiara el fichero del disco al buffer de salida.

El siguiente esquema resume estos pasos:



En ASP, para añadir código script (script de servidor), puede emplearse también la marca HTML `<SCRIPT>`, tal y como se hace con JavaScript (script de cliente). Sin embargo, hay que añadirle un parámetro para indicarle al servidor web que este código script es de servidor, en vez de cliente. El parámetro a añadir se llama RUNAT, y los valores posibles son “Client” (para indicar que el script se ejecute en el cliente, este es el valor por defecto), y “Server” (para indicar que el script se ejecute en el servidor, este será el valor que usaremos nosotros).

La marca SCRIPT, acepta otro parámetro para indicar el lenguaje en el cual está escrito el script. Este parámetro es el parámetro LANGUAGE, y sus valores posibles son, entre otros, “VBScript”, “JavaScript”, “PerlScript”. Nosotros usaremos el lenguaje VBScript por ser el lenguaje más usado, más fácil y que mejor se integra con ASP.

---

<sup>1</sup> En realidad, el buffer de salida puede ser cualquier documento HTML válido, es decir, no tiene porqué contener sólo marcas HTML, pudiendo contener código JavaScript mezclado en el documento HTML, por ejemplo. Sin embargo, si tiene código script, será siempre código de cliente, nunca código ASP.

Por lo tanto, para añadir nuestro código ASP en una página HTML, deberemos escribir:

```
<SCRIPT LANGUAGE="VBSCRIPT" RUNAT="Server">  
    aquí irá nuestro código de servidor, escrito en VBScript  
</SCRIPT>
```

Estas marcas `<SCRIPT>` i `</SCRIPT>`, con todos sus parámetros, se tendrían que añadir cada vez que quisiéramos insertar un código script. Esto hace que el programador tenga que escribir bastante, añadiendo un overhead de código innecesario.

Por este motivo, los ingenieros de Microsoft crearon una alternativa para añadir código script de servidor, usando las marcas `<%` y `%>` (equivalentes a `<SCRIPT LANGUAGE="VBScript" RUNAT="Server">`, y `</SCRIPT>`, respectivamente).

Por lo tanto, el anterior fragmento de código puede también ser escrito de la siguiente manera:

```
<%  
    aquí irá nuestro código de servidor, escrito en VBScript  
%>
```

siendo esta notación la más usada por su simplicidad.

Pero si nos paramos a pensar un poco con los conocimientos que hemos adquirido hasta ahora, nos daremos cuenta de que algo no cuadra. Por un lado, ASP se puede programar tanto en VBScript como con otros lenguajes script, y por otro lado con esta nueva notación parece que no estamos indicando el lenguaje que usamos. Entonces, como sabe el intérprete de ASP que el código que está entre las marcas `<%` y `%>` es código escrito en VBScript? La respuesta es simple: VBScript es el lenguaje predeterminado por ASP. Sin embargo, si quisiéramos usar JavaScript en lugar de VBScript, tenemos que indicarle de algún modo al intérprete de ASP que el código script está escrito en JavaScript. Para esto se usa una directiva de compilador llamada `LANGUAGE`, cuyos valores posibles pueden ser "VBScript" o "JavaScript", entre otros.

Las directivas de compilación deben añadirse al comienzo de una página asp, y se ponen entre las marcas `<% @` y `%>`.

Por ejemplo, para indicar que el lenguaje script usado en nuestra página ASP va a ser VBScript, deberíamos escribir al comienzo de la página:

```
<% @ LANGUAGE="VBScript"%>
```

Igualmente, si quisiéramos usar el lenguaje JavaScript en lugar de VBScript, escribiríamos, en lugar del código anterior, el siguiente código (también al comienzo de la página):

```
<% @ LANGUAGE="JavaScript" %>
```

Otras marcas específicas de la plataforma ASP son las marcas `<%=` y `%>`. Estas marcas sirven para introducir expresiones (en lugar de instrucciones, como pasaba con `<%` y `%>`) el resultado de las cuales aparecerá en la página HTML resultante. Por ejemplo, si

dentro de una página ASP escribo `<%= 2+2 %>`, al enviar el código HTML al navegador, este trozo será substituido por un 4, tal y como ilustra el siguiente ejemplo:

Página ASP:

```
<HTML>
  <HEAD>
    <TITLE>Prueba de ASP</TITLE>
  </HEAD>
  <BODY>
    2 + 2 son <%= 2+2 %>
  </BODY>
</HTML>
```

Tras ejecutar esta página en el servidor, el código HTML que será enviado al navegador será:

```
<HTML>
  <HEAD>
    <TITLE>Prueba de ASP</TITLE>
  </HEAD>
  <BODY>
    2 + 2 son 4
  </BODY>
</HTML>
```

## Breve historia

La programación en el lado del servidor existe desde los comienzos de internet, aunque entonces se usaban tecnologías más rudimentarias. Concretamente, se usaba la tecnología CGI (Common Gateway Interface, pasarela de interfície común), que básicamente son programas independientes, escritos en cualquier lenguaje de programación (como C, Pascal, Fortran, etc...) los cuales ejecutan mandatos para generar una salida HTML. Esta tecnología todavía existe, pero cada vez se encuentra más en desuso por sus inconvenientes, nombrados en el siguiente apartado.

Posteriormente, con la aparición de lenguajes script, apareció el concepto de script de servidor, es decir, lenguajes script (que se mezclan con el documento HTML) que se ejecutan en el servidor. Esto significó un salto importante porque ahora ya no haría falta construir todo un programa entero en un lenguaje “duro” de programación, sino que en su lugar, solo se tendría que escribir las 3 o 4 líneas de código en el sitio adecuado dentro de nuestro documento HTML para que el servidor ejecutara alguna acción específica.

La primera versión que apareció de ASP, en diciembre de 1996, fue la versión 1.0 de ASP, la cual se ejecutaba sobre IIS 3.0 (el servidor web profesional de Microsoft, como se verá más adelante).. Más tarde surgió una nueva versión de ASP, la versión 2.0, que se ejecuta sobre IIS 4.0, el cual se incluye en el option Pack para Windows NT 4.0). Este es probablemente el sistema más usado actualmente, aunque existen 2 nuevas versiones, que todavía no son muy usadas por ser bastante recientes.

Recientemente ha surgido la versión 3.0 de ASP, que se ejecuta sobre IIS 5.0, el cual se incluye junto con el sistema operativo Microsoft Windows 2000 Server.

Finalmente, existe una nueva versión llamada ASP .Net, que forma parte de una nueva filosofía en las herramientas de desarrollo de Microsoft.

## ASP vs CGI

Como se ha comentado en el apartado anterior, CGI es una tecnología más antigua que ASP, la cual está cada vez en más desuso debido a la facilidad y eficiencia de ASP.

Un programa CGI, a pesar de ser un programa independiente (esto es, un programa compilado escrito en un lenguaje de programación que puede ser mucho más optimizado que ASP) suele ofrecer un rendimiento muy inferior a las páginas ASP. El motivo es simple: al ser un programa independiente, debe comunicarse con el servidor web usando comunicación entre procesos. Esta comunicación entre el programa CGI y el servidor web es muy lenta, y además suele ocupar bastantes recursos.

Sin embargo, esto no ocurre con las páginas ASP, las cuales utilizan otra tecnología llamada ISAPI (Internet Server Application Programing Interface, Interface de Programación de Aplicaciones de Servidores de Internet) , que hace que el código ASP sea ejecutado por un módulo que se encuentra dentro del propio Servidor Web.

Además, en ciertas circunstancias, el servidor web puede compilar las páginas en lugar de interpretarlas, con lo que la eficiencia aumenta todavía más.

## Alternativas: JSP, PHP, ColdFusion, ...

ASP no es la única alternativa a los programas CGI. Existen otras plataformas como JSP (Java Server Pages, de Sun Microsystems), ColdFusion (de Allaire) o PHP (de libre distribución bajo Open System).

Algunas de las ventajas que ofrece ASP respecto a estas otras alternativas son:

- Su bajo coste (frente a JSP o ColdFusion), pues el servidor web viene incluido con los sistemas operativos de Microsoft.
- La facilidad del lenguaje, pues ASP se suele programar en VBScript, lenguaje mucho más simple que Java o PHP (el cual es similar al C).
- Es, probablemente, la tecnología más usada (sobretudo si lo comparamos con JSP o ColdFusion). Esto hace que sea muy fácil obtener información sobre el tema totalmente gratis, pues hay muchas páginas en internet con manuales, tutoriales, artículos, o código fuente de páginas ASP<sup>2</sup>.
- Existen servidores gratuitos para hospedar nuestras páginas ASP, y los que son de pago (los cuales suelen ofrecer una mayor funcionalidad) no suelen ser caros, pudiendo costar unas 15.000 pts. al año.
- ASP utiliza componentes Active-X, los cuales amplían las funcionalidades de la plataforma ASP. Se puede encontrar en internet una cantidad innumerable de componentes Active-X que pueden ser usados en nuestras páginas ASP.

---

<sup>2</sup> En la bibliografía, al final de esta documentación, se nombran algunas de estas webs.



## Herramientas que necesitamos.

### El servidor web

Lo primero que necesitamos para ejecutar una página ASP es tener instalado en el servidor un servidor web que soporte ASP. Dado que ASP es una tecnología propietaria de Microsoft, existen básicamente 2 servidores web, los 2 de Microsoft, los cuales soportan ASP: IIS y PWS.

IIS (Internet Information Server) es la solución profesional. Es decir, es un servidor web que soporta muchos usuarios conectados a la vez, y ofrece un rendimiento muy bueno. IIS viene incluido en los sistemas operativos basados en tecnología NT Server, tal como Windows-NT Server 4.0 o Windows-2000 Server.

PWS (Personal Web Server) es la solución para el desarrollo. Es decir, es un servidor web con la misma funcionalidad que IIS, pero que soporta un limitado número de usuarios conectados a la vez. Concretamente, suele estar limitado a 10 usuarios a la vez, con lo que sirve a la perfección para desarrollar nuestro site, hacer pruebas, etc..., pero no sirve, en la mayoría de los casos, para hacer de servidor real de nuestra web, a no ser que ésta tenga pocas visitas.

La ventaja de PWS es que éste se incluye en los sistemas operativos personales de Microsoft, como Microsoft Windows-95 y Microsoft Windows-98. Concretamente, el PWS se encuentra en un directorio del CD de instalación llamado Add-ons. Por ejemplo, si nuestro sistema operativo es Windows-98 y nuestra unidad de CD es la D:, para instalar el PWS en nuestra máquina deberemos ejecutar D:\Add-ons\PWS\Instalar.exe. Éste será el servidor web que usaremos en el curso, ya que tiene la misma funcionalidad que la versión profesional.

### El navegador

Con el fin de poder probar nuestras páginas ASP, necesitaremos también de un navegador, que bien puede ser Internet Explorer, Netscape Navigator, o cualquier otro que nos permita visualizar páginas HTML.

### Herramientas para el desarrollo

Una página ASP, no es más que una página HTML que contiene código ASP dentro. Es decir, una página ASP no es más que un fichero de texto llano. Por este motivo, podemos utilizar nuestro editor de texto preferido, como por ejemplo el bloc de notas del windows, el editor del MS-DOS, o cualquier editor de HTML como por ejemplo Dreamweaver.

Existen también herramientas específicas para la ayuda al desarrollo de las páginas ASP, tal como Microsoft Visual InterDev, incluido en la Suite Microsoft Visual Studio. Sin embargo, estas herramientas quedan fuera del alcance de este curso debido a que aquí nos limitaremos a dar unas pinceladas sobre los conceptos básicos de esta plataforma, sin afán de entrar en temas más profundos como la gestión y administración de proyectos en internet.

## 2.- El lenguaje VBScript

### Variables

Al igual que ocurre con JavaScript, VBScript es un lenguaje débilmente tipificado. Esto significa que cuando declaramos una variable, no declaramos de qué tipo es, pues las variables pueden cambiar su tipo en función del valor que les sea asignado. Esto tiene como ventaja que podemos reciclar variables, pudiendo usar la misma variable para 2 (o más) cosas distintas a lo largo de nuestro programa, incluso aunque cada una de estas cosas sea de un tipo diferente, pudiendo obtener, pues, un ahorro importante de memoria.

Para declarar una variable en VBScript se usa la palabra clave Dim (igual que ocurre en Visual Basic). Así pues, si queremos declarar una variable llamada nombre\_cliente, escribiremos:

```
Dim nombre_cliente
```

En VBScript no es obligatorio, por defecto, declarar las variables. Sin embargo, es una buena práctica obligar a declarar las variables para evitar posibles errores a la hora de escribir su nombre. Por ejemplo, si queremos ejecutar la siguiente instrucción:

```
numero_clientes = numero_clientes + 1
```

que incrementa el valor contenido en la variable numero\_clientes, pero nos equivocamos y escribimos:

```
numero_clientes = numeroclientes + 1
```

VBScript no nos avisará de que se ha producido un error, sino que en su lugar creará una nueva variable numeroclientes, inicializada con un valor predeterminado (probablemente 0). El resultado, pues, de esta instrucción, será asignar un 1 a la variable numero\_clientes.

Para evitar esto, debemos añadir la instrucción Option Explicit al inicio de nuestro código VBScript. Esta instrucción nos obligará a declarar todas las variables que usemos y, por lo tanto, hará que el intérprete de ASP detecte el error anterior.

Aunque en la declaración de una variable no declaremos su tipo, el valor que contiene una variable en un instante determinado es siempre de un cierto tipo. En realidad, se dice que todas las variables de VBScript son de tipo Variant (un tipo que puede ir cambiando durante la ejecución del script), y que el tipo Variant tiene distintos subtipos, equivalentes a cada uno de los tipos de valores que puede contener la variable.

La tabla siguiente muestra varios subtipos de datos que puede contener un tipo **Variant**.

Subtipo	Descripción
<b>Empty</b>	<b>Variant</b> está sin inicializar. El valor es 0 para variables numéricas o una cadena de longitud cero ("" ) para variables de cadena.
<b>Null</b>	<b>Variant</b> contiene intencionadamente datos no válidos.
<b>Boolean</b>	Contiene <b>True</b> o <b>False</b> .
<b>Byte</b>	Contiene un entero entre 0 y 255.
<b>Integer</b>	Contiene un entero -32.768 y 32.767.
<b>Currency</b>	Número de -922.337.203.685.477,5808 a 922.337.203.685.477,5807.
<b>Long</b>	Contiene un entero -2.147.483.648 y 2.147.483.647.
<b>Single</b>	Contiene un número real de coma flotante de precisión simple entre -3,402823E38 y -1,401298E-45 para valores negativos, y entre 1,401298E-45 y 3.402823E38 para valores positivos.
<b>Double</b>	Contiene un número de coma flotante de precisión doble en el intervalo de -1,79769313486232E308 y -4,94065645841247E-324 para valores negativos, y entre 4,94065645841247E-324 y 1,79769313486232E308 para valores positivos.
<b>Date (Time)</b>	Contiene un número que representa una fecha entre el 1 de enero de 100 y el 31 de diciembre de 9999.
<b>String</b>	Contiene una cadena de longitud variable que puede contener hasta 2 mil millones de caracteres de longitud.
<b>Object</b>	Contiene un objeto.
<b>Error</b>	Contiene un número de error.

Se pueden usar las funciones de conversión para convertir datos de un subtipo a otro. Además la función **VarType** devuelve información acerca de cómo se almacenan los datos en un tipo **Variant**.

## Funciones de conversión

**Asc:** Devuelve el código ANSI del primer carácter del String pasado por parámetro.

**Cbool:** Devuelve el valor booleano resultado de evaluar la expresión booleana pasada por parámetro. Si se pasa un entero, devuelve falso si el entero es 0.

**CByte:** Convierte la expresión pasada a un entero pequeño (que cabe en un sólo byte). Si hay decimales éstos se redondean. Si el valor no cabe en un byte ocurrirá un error.

**CCur:** Convierte la expresión pasada por parámetro a tipo Currency (moneda).

**CDate:** Convierte la expresión pasada por parámetro a tipo Date (Fecha).

**Cdbl:** Convierte la expresión pasada a tipo Double (número real de doble precisión).

**Chr:** Convierte un código de carácter ANSI a su respectivo carácter, devolviendo un string con este carácter.

**CInt:** Convierte la expresión pasada por parámetro a número entero (entero de 2 bytes). Es decir, convierte al subtipo Integer.

**CLng:** Convierte la expresión pasada por parámetro a número entero largo (entero de 4 bytes). Es decir, convierte al subtipo Long.

**CSng:** Convierte la expresión pasada por parámetro a un Single (número real de simple precisión).

**CStr:** Convierte la expresión pasada por parámetro a String.

**Hex:** Devuelve un string que representa el valor hexadecimal del número pasado por parámetro.

**Oct:** Devuelve un string que representa el valor octal del número pasado por parámetro.

## Restricciones de nombre de las variables

Los nombres de variables siguen las normas estándar de denominación en VBScript. Los nombres de las variables deben cumplir los siguientes requisitos:

- Debe comenzar con un carácter alfabético.
- No puede contener caracteres reservados del lenguaje (como puntos, operadores aritméticos, etc...), siendo los caracteres válidos cualquier carácter alfabético o numérico, así como el carácter \_
- No debe superar los 255 caracteres de longitud
- Debe ser único en el alcance donde se declara.

## Alcance y vida de las variables

El alcance de una variable se determina cuando se declara. Cuando declara una variable dentro de un procedimiento, sólo el código dentro de ese procedimiento puede tener acceso o cambiar el valor de esa variable. Tiene scope local y se llama variable de **nivel de procedimiento**. Si declara una variable fuera de un procedimiento, la hace reconocible en todos los procedimientos de la secuencia de comandos. Este tipo de variable es de **nivel de secuencia de comandos** y tiene alcance de nivel de secuencia de comandos.

El tiempo que una variable existe es su vida. La vida de una variable de nivel de secuencia de comandos se extiende desde el momento en que se declaró hasta el momento en que finaliza la ejecución de la secuencia de comandos. A nivel de procedimiento, una variable existe sólo cuando se encuentra en el procedimiento. Cuando sale del procedimiento, la variable se destruye. Las variables locales son adecuadas como espacio de almacenamiento temporal cuando se ejecuta un procedimiento. Puede tener variables locales del mismo nombre en varios procedimientos diferentes porque cada una sólo se reconoce en el procedimiento en que se declaró.

## Variables escalares y variables de matrices

Muchas veces, sólo desea asignar un único valor a una variable que ha declarado. Una variable que contiene un único valor es una variable escalar. Otras veces, es útil asignar más de un valor relacionado a una única variable. Entonces puede crear una variable que pueda contener una serie de valores. Esto se llama una variable de matriz, también conocido como un array.

Las variables matriz se declaran de la misma forma que las variables escalares. La diferencia es que una declaración de una variable de matriz utiliza paréntesis ( ) a continuación del nombre de la variable. En el siguiente ejemplo se declara una matriz de una dimensión que contiene 11 elementos:

```
Dim A(10)
```

Aunque el número que se muestra entre paréntesis es 10, todas las **matrices** en VBScript son de base cero, por lo que la matriz realmente contiene 11 elementos (desde el elemento 0 hasta el elemento 10). En una matriz de base cero, el número de elementos de la misma siempre es el número mostrado entre paréntesis más uno. Este tipo de matriz se llama una matriz de tamaño fijo.

Asigne datos a cada elemento de la matriz utilizando un índice dentro de la matriz. Comenzando en cero y terminando en 10, es posible asignar datos a los elementos de una matriz del siguiente modo:

```
A(0) = 256  
A(1) = 324  
A(2) = 100  
...  
A(10) = 55
```

Del mismo modo, se pueden recuperar los datos de cualquier elemento utilizando el índice que desee dentro del elemento de la matriz deseado. Por ejemplo:

```
...  
AlgunaVariable = A(8)  
...
```

Las matrices no están limitadas a una única dimensión. Puede tener hasta 60 dimensiones aunque la mayoría de las personas no pueden comprender más de tres o cuatro. Las dimensiones múltiples se declaran separando con comas los números de tamaño de la matriz dentro del paréntesis. En el siguiente ejemplo, la variable MiTabla es una matriz bidimensional que consta de 6 filas y 11 columnas:

```
Dim MiTabla(5, 10)
```

En una matriz bidimensional, el primer número siempre es el número de filas y el segundo el número de columnas.

### Declaración de constantes

Una constante es una variable cuyo valor se define en su declaración y no puede ser cambiado (su valor no es variable, sino constante).

Para crear una constante en VBScript se usa la palabra clave **Const** en lugar de Dim. Tras poner el nombre de la constante, se escribe el valor que contendrá dicha constante, separando el nombre de la constante y su valor con una asignación (que en VBScript es el símbolo = ).

Por ejemplo:

```
Const MiCadena = "Esta es mi cadena."  
Const MiEdad = 49
```

Los literales de tipo cadena de caracteres (string) se escriben siempre entre comillas dobles ( “ ” ), sin ser posible que estas comillas sean simples, tal y como ocurre con JavaScript, el cual permite tanto el empleo de comillas simples como dobles. Esto es debido a que en VBScript la comilla simple se emplea para realizar comentarios de línea. Dicho de otro modo, una comilla simple ( ' ) en VBScript equivale a dos barras seguidas ( // ) en C++.

Los literales de fecha, se escriben siempre entre sostenidos o almohadillas (#)

Const FechaCorte = #6-1-97#

## OPERADORES

Los operadores son símbolos que se insertan entre variables o constantes para formar expresiones. Existen operadores aritméticos (tal como la suma o la resta), operadores de comparación (tal como el igual o el distinto), operadores de concatenación (para juntar dos strings en uno) y operadores lógicos (como la y-lógica y la o-lógica)

### Prioridad de los operadores

Cuando se producen varias operaciones en una expresión, cada parte se evalúa y se resuelve en un orden predeterminado. Este orden se conoce como prioridad de los operadores. Puede utilizar paréntesis para invalidar el orden de prioridad y forzar que se evalúen algunas partes de una expresión antes que otras. Las operaciones entre paréntesis siempre se ejecutan antes que las de fuera. Sin embargo, dentro de los paréntesis se mantiene la prioridad de los operadores.

Cuando las expresiones contienen operadores de más de una categoría, se evalúan primero los operadores aritméticos, a continuación los operadores de comparación y por último los lógicos. Todos los operadores de comparación tienen la misma prioridad; esto quiere decir que se evalúan en el orden en que aparecen, de izquierda a derecha. Los operadores aritméticos y lógicos se evalúan en el siguiente orden de prioridad:

Aritméticos		De comparación		Lógicos	
Descripción	Símbolo	Descripción	Símbolo	Descripción	Símbolo
Exponenciación	^	Igualdad	=	Negación lógica	Not
Cambio de signo	-	Desigualdad	<>	Conjunción lógica	And
Multiplicación	*	Menor que	<	Disyunción lógica	Or
División	/	Mayor que	>	Exclusión lógica	Xor
División entera	\	Menor o igual que	<=	Equivalencia lógica	Eqv
Módulo aritmético (resto de la división)	Mod	Mayor o igual que	>=	Implicación lógica	Imp
Suma	+	Equivalencia de objeto	Is		
Resta	-				
Concatenación de cadenas	&				

Cuando una multiplicación y una división se producen juntas en una expresión, cada operación se evalúa como ocurre de izquierda a derecha. Del mismo modo, cuando una suma y una resta se producen juntas en una expresión, cada operación se evalúa según el orden de aparición, de izquierda a derecha.

El operador de concatenación de cadenas (&) no es un operador aritmético, pero en la prioridad se ejecuta después de todos los operadores aritméticos y antes de todos los operadores de comparación. El operador **Is** es un operador de comparación de referencia de objeto. No compara objetos o sus valores; sólo se comprueba para determinar si dos referencias a objetos se refieren al mismo objeto

## Funciones de VBScript

VBScript incorpora funciones propias para realizar tareas de distintos tipos. Cada una de estas funciones están explicadas detalladamente en las MSDN de Microsoft (ver bibliografía).

A continuación se nombran algunas de estas funciones clasificadas por categorías, junto con una breve descripción:

### **Funciones de tratamiento de fechas:**

#### Obtención de la fecha y hora actual

Date: Obtiene la fecha actual

Time: Obtiene la hora actual

Now: Obtiene el instante (fecha + hora) actual

#### Funciones para operar con fechas

DateAdd: Suma un número de días a una fecha

DateDiff: Obtiene el número de días entre 2 fechas

#### Obtención de una parte de la fecha:

Year: Obtiene el año de una fecha

MonthName: Obtiene el nombre del mes de una fecha

Month: Obtiene el número de mes de una fecha

WeekDayName: Obtiene el nombre del día de la semana.

WeekDay: Obtiene el número de día de la semana

Day: Obtiene el día (dentro del mes) de una fecha

Hour: Obtiene la hora (nº entre 0 y 23) de una hora (obtenida con Time).

Minute: Obtiene los minutos (entre 0 y 59) de una hora concreta.

Second: Obtiene los segundos (entre 0 y 59) de una hora concreta



## **Funciones matemáticas:**

### Funciones trigonométricas:

- Sin: Obtiene el seno de un ángulo en radianes.
- Cos: Obtiene el coseno de un ángulo en radianes.
- Tan: Obtiene la tangente de un ángulo en radianes.
- Atn: Obtiene el arco-tangente de un ángulo en radianes.

### Funciones de signo:

- Abs: Obtiene el valor absoluto de un número
- Sgn: Obtiene el signo de un número

### Funciones de conversión y redondeo:

- Fix: Obtiene la parte entera de un número real
- Int: Obtiene la parte entera de un número real<sup>3</sup>
- Hex: Devuelve un string con el valor hexadecimal de un número
- Oct: Devuelve un string con el valor octal de un número
- Round: Redondea un número real a un número concreto de decimales.

### Funciones de exponenciación y logarítmicas

- Exp: Realiza la operación de exponenciación
- Log: Obtiene el logaritmo natural
- Sqr: Obtiene la raíz cuadrada de un número

### Otras:

- Eval: Evalúa una expresión devolviendo su resultado
- Rnd: Obtiene un número aleatorio

## **Funciones de tratamiento de strings**

### Espacios:

- LTrim: Elimina los espacios iniciales de un string
- RTrim: Elimina los espacios finales de un string
- Trim: Elimina espacios de un string
- Space: Crea un string conteniendo un número determinado de espacios.

### Obtención de una subcadena:

- Left: Obtiene un prefijo de un string
- Right: Obtiene un sufijo de un string
- Mid: Obtiene una parte central de un string

### Mayúsculas/Minúsculas:

- Ucase: Pasa un string a mayúsculas.
- Lcase: Pasa un string a minúsculas.

### Otras:

- Len: Obtiene el número de caracteres de un string
- InStr: Obtiene la posición de una subcadena dentro de otro string.
- InStrRev: Hace lo mismo que InStr pero empezando por el final.
- Replace: Sustituye dentro de un texto una cadena por otra.
- StrReverse: Invierte el orden de caracteres de un string.
- Split: Parte un string en 2
- StrComp: Comparación de 2 strings

---

<sup>3</sup> Fix i Int son equivalentes cuando se usan con números positivos. Sin embargo, no hacen lo mismo cuando el número es negativo. Ver la ayuda de las MSDN para más información.

### **Funciones de tipos de datos**

#### Averiguar si una variable es de un cierto tipo:

IsArray: Devuelve true si la variable es un array

IsDate: Devuelve true si la variable es de tipo Date

IsEmpty: Devuelve true si la variable está vacía.

IsNull: Devuelve true si la variable es Null

IsNumeric: Devuelve True si la variable es de tipo numérico.

IsObject: Devuelve True si la variable es un objeto.

#### Obtención del tipo de una variable:

VarType: Devuelve una constante indicando el tipo de variable.

TypeName: Devuelve un string con el tipo de la variable

### **Otras:**

RGB: Devuelve un único valor numérico que representa un color RGB.

## Sentencias de control (If, select case, while, for, for each).

Las sentencias de control se usan para controlar el flujo de ejecución de nuestro programa, permitiendo ejecutar distintas alternativas en función de una condición, o añadir un bucle para ejecutar un fragmento de código repetidamente mientras se cumpla una condición.

### Sentencia IF

Una de las sentencias de control que tiene todo lenguaje de programación, es la sentencia if, que nos permite realizar alternativas dentro del código en función de una condición.

La sintaxis del if más simple es:

```
If condición_booleana Then
    Instrucciones a ejecutar
End If
```

Como se puede apreciar, la sintaxis es parecida a la instrucción if de C o de JavaScript, pero poniendo la palabra clave Then en lugar de { y la palabra End If al finalizar el If (en lugar de }). Otra diferencia entre VBScript y JavaScript (o también C) es que VBScript no distingue entre mayúsculas y minúsculas, siendo equivalente, pues, escribir IF, if, If o incluso iF.

La sintaxis más simple del IF es la siguiente:

```
If edad >= 18 Then
    ' Instrucciones a ejecutar para los mayores de edad
End if
```

Sin embargo se puede añadir la cláusula else para hacer que se ejecute un código alternativo en caso que no se cumpla la condición del if. Un ejemplo de esto sería:

```
If edad >= 18 Then
    'Instrucciones a ejecutar para los mayores de edad
Else
    'Instrucciones a ejecutar para los menores de edad
End If
```

Finalmente, la palabra clave ElseIf, nos permite realizar más de una comparación, permitiendo realizar más de 2 alternativas. Un ejemplo de esta sintaxis es la siguiente:

```
If hora > 7 and hora < 14 Then
    'Instrucciones para dar los Buenos Días
Elseif hora >=14 and hora < 21:00 Then
    'Instrucciones para dar las Buenas Tardes
Else
    'Instrucciones para dar la Buenas Noches.
End If
```

En el ejemplo anterior, sólo se entra en el else si la variable hora contiene un número  $\leq 7$  o si contiene un número  $\geq 21$ .

### Sentencia SELECT CASE

Otra instrucción para realizar alternativas es la instrucción **Select Case**. Una instrucción Select Case proporciona una funcionalidad similar a la instrucción **If...Then...Else**, pero muchos lenguajes la incluyen debido a que hace el código más eficiente y legible.

En C o en JavaScript existe una instrucción equivalente llamada switch.

Una estructura **Select Case** trabaja con una expresión de comprobación sencilla que se calcula una vez, al comienzo de la estructura. Después el resultado de la expresión se compara con los valores para cada **Case**. Si existe una coincidencia, se ejecuta el bloque de instrucciones asociado con ese **Case**:

```
Select Case dia_semana
    Case 1
        'Instrucciones para el Lunes
    Case 2
        'Instrucciones para el Martes
    Case 3
        'Instrucciones para el Miércoles
    Case 4
        'Instrucciones para el Jueves
    Case 5
        'Instrucciones para el Viernes
    Case 6
        'Instrucciones para el Sábado
    Case 7
        'Instrucciones para el Domingo
End Select

If hora > 7 and hora < 14 Then
    'Instrucciones para dar los Buenos Días
Elseif hora >=14 and hora < 21:00 Then
    'Instrucciones para dar las Buenas Tardes
Else
    'Instrucciones para dar la Buenas Noches.
End If
```

Observe que la estructura **Select Case** calcula una expresión una vez al comienzo de la estructura. Por el contrario, la estructura **If...Then...ElseIf** puede evaluar una expresión diferente en cada instrucción **ElseIf**. Sólo puede reemplazar una estructura **If...Then...ElseIf** con una estructura **Select Case** si cada instrucción **ElseIf** calcula la misma expresión y si cada condición contiene el operador de comparación  $=$ .

Se puede añadir como último caso el **Case Else**, el cual ejecutará instrucciones si no la expresión del Select case no es igual a ningún valor de ninguno de los Case. Es decir, la

alternativa Case Else se ejecutará sólo cuando no se ejecute ninguna otra alternativa. El Select Case equivale al default de C o JavaScript.

## BUCLES

La mayoría de lenguajes disponen también de mecanismos para hacer bucles o iteraciones, que consiste en la capacidad de poder repetir un bloque de instrucciones mientras se cumpla una cierta condición, o durante un número de veces concreto.

En VBScript están disponibles las siguientes instrucciones de iteración:

- **While...Wend:** ejecuta el bucle mientras una condición sea cierta (**true**)
- **Do...Loop:** ejecuta el bucle mientras o hasta que una condición sea cierta (**true**)
- **For...Next:** utiliza un contador para ejecutar instrucciones un número de veces específico
- **For Each...Next:** Repite un grupo de instrucciones para cada elemento de una colección o para cada elemento de una matriz.

### Uso de bucles Do

Se pueden utilizar las instrucciones Do...Loop para ejecutar un bloque de instrucciones un número de veces indefinido. Las instrucciones se repiten mientras una condición es **True** o hasta que una condición pasa a ser **cierta**.

### Repetición de instrucciones mientras una condición es cierta

Usaremos la palabra clave **While** para comprobar una condición en una instrucción Do...Loop. Se puede comprobar la condición antes de entrar en el bucle (como se muestra en el ejemplo de a continuación) o puede comprobarla después que el bucle se haya ejecutado al menos una vez (como se muestra en el ejemplo 2). En el primer ejemplo, si n se establece a 9 en lugar de 20, las instrucciones que hay dentro del bucle nunca se ejecutarían. En el ejemplo 2, las instrucciones que hay dentro del bucle sólo se ejecutan una vez porque la condición ya es **False**.

El siguiente código se ejecuta mientras el valor de la variable n sea mayor que 10.

```
Dim n
n = 20
Do While n > 10
    n = n - 1
Loop
```

Ejemplo 2:

```
Dim n
n = 9
Do
    n=n-1
Loop While n > 10
```

### Repetición de una instrucción hasta que una condición pasa a ser True

Puede utilizar la palabra clave **Until** de dos formas para comprobar una condición en una instrucción **Do...Loop**. Puede comprobar la condición antes de entrar en el bucle (como se muestra en el siguiente ejemplo) o puede comprobarla después de que el bucle se haya ejecutado al menos una vez (como se muestra en el ejemplo 2). Mientras la condición es **False** se ejecuta el bucle.

El siguiente código se ejecuta hasta que la variable n sea menor o igual que 10

```
Dim n
n = 9
Do Until n<=10
    n = n - 1
Loop
```

Ejemplo 2:

```
Dim n
n = 9
Do
    n = n - 1
Loop Until n<=10
```

### Uso de While ... Wend

La instrucción **While ... Wend** se proporciona en VBScript para las personas que están familiarizadas con su uso. Sin embargo, se recomienda que utilice **Do ... Loop** debido a la escasa flexibilidad de la instrucción **While ... Wend**. La instrucción **While ... Wend** es equivalente a **Do While ... Loop**.

### Uso de For ... Next

Puede utilizar las instrucciones **For...Next** para ejecutar un bloque de instrucciones un número de veces específico. Utilice una variable de tipo contador cuyo valor aumente o disminuya con cada repetición del bucle.

Por ejemplo, el siguiente código ejecuta 50 veces un procedimiento llamado MiProc. La instrucción **For** especifica la variable de tipo contador x y los valores inicial y final de la misma. La instrucción **Next** aumenta la variable de tipo contador de 1 en 1.

```
Sub HazMiProc50Veces()
    Dim x
    For x = 1 To 50
        MiProc
    Next
End Sub
```

Utilizando la palabra clave **Step** puede aumentar o disminuir la variable tipo contador en el valor que especifique. En el siguiente ejemplo, la variable tipo contador aumenta

de 2 en 2 cada vez que se repite el bucle. Cuando el bucle finaliza, total es la suma de 2, 4, 6, 8 y 10.

```
Sub TotalesdePares()
    Dim j, total
    For j = 2 To 10 Step 2
        total = total + j
    Next
    MsgBox "El total es " & total
End Sub
```

Para disminuir la variable tipo contador, utilice un valor **Step** negativo. Cuando lo haga, debe especificar un valor final que sea menor que el valor inicial. En el siguiente ejemplo, la variable tipo contador miNum disminuye de 2 en 2 cada vez que se repite el bucle. Cuando el bucle termina, total es la suma de 16, 14, 12, 10, 8, 6, 4 y 2.

```
Sub NuevoTotal()
    Dim miNum, total
    For miNum = 16 To 2 Step -2
        total = total + miNum
    Next
    MsgBox "El total es " & total
End Sub
```

Puede salir de cualquier instrucción **For...Next** antes de que el contador alcance su valor final utilizando la instrucción **Exit For**. Como normalmente desea salir sólo en ciertas situaciones, como cuando se produce un error, debe utilizar la instrucción **Exit For** en el bloque de instrucciones **True** de una instrucción **If...Then...Else**. Si la condición es **False**, el bucle se ejecuta normalmente.

Uso de For Each...Next

Un bucle **For Each...Next** es parecido a un bucle **For...Next**. En lugar de repetir las instrucciones un número determinado de veces, un bucle **For Each...Next** repite un grupo de instrucciones para cada elemento de una colección de objetos para cada elemento de una matriz. Esto es especialmente útil si no desea conocer cuántos elementos hay en una colección.

En el ejemplo de código HTML completo siguiente, el contenido de un objeto Dictionary se utiliza para colocar texto en varios cuadros de texto:

```
<HTML>
<HEAD><TITLE>Formularios y elementos</TITLE></HEAD>
<SCRIPT LANGUAGE="VBScript">
<!--
Sub cmdCambiar_OnClick
    Dim d
    'Crea una variable
    Set d = CreateObject("Scripting.Dictionary")
    d.Add "0", "Atenas" 'Agrega algunas claves y elementos
    d.Add "1", "Belgrado"
    d.Add "2", "El Cairo"

    For Each I in d
```

```

        Document.frmForm.Elements(I).Value = D.Item(I)
    Next
End Sub
-->
</SCRIPT>
<BODY>
<CENTER>
<FORM NAME="frmForm"

    <Input Type = "Text"><p>
    <Input Type = "Text"><p>
    <Input Type = "Text"><p>
    <Input Type = "Text"><p>
    <Input Type = "Button" NAME="cmdCambiar" VALUE="Haga clic
aquí"><p>
</FORM>
</CENTER>
</BODY>
</HTML>

```

## Procedimientos y Funciones

Para crear procedimientos y funciones en VBScript, se usan las palabras clave **Sub** y **Function** al principio del procedimiento o función, y **End Sub** y **End Function** al final del procedimiento o función, respectivamente

La sintaxis completa para **crear un procedimiento** es:

```

Sub nombre_procedimiento(parámetros)
    Cuerpo_del_procedimiento
End Sub

```

Mientras que la sintaxis completa para **crear una función** es:

```

Function nombre_función(parámetros)
    Cuerpo de la función
End Function

```

**Parámetros:** Dado que VBScript es un lenguaje débilmente tipificado, tal y como ocurre con JavaScript, la lista de parámetros es simplemente una lista con los nombres de cada parámetro separados por comas.

Para retornar el valor que debe retornar la función, simplemente se asigna al nombre de la función el valor que deseamos retornar, como si el nombre de la función fuera una variable.

El siguiente ejemplo es una función que incrementa un número:

```

Function incrementa(n)
    Incrementa = n+1
End function

```



El siguiente ejemplo es una función que retorna el número mayor de entre 2 números pasados por parámetro:

```
Function max(x,y)
    If x > y Then
        max = x
    Else
        max = y
    End If
End Function
```

Para **llamar a un procedimiento** se usa la palabra clave **call** seguido del nombre del procedimiento al que queremos llamar. Si éste tiene parámetros, éstos se ponen después del nombre del procedimiento, entre paréntesis y separados por comas.

Para **llamar a una función**, se escribe el nombre de la función, seguido de sus parámetros entre paréntesis y separado por comas. Hay que tener en cuenta que la llamada a una función devuelve siempre un valor, por lo que no se puede usar una función como instrucción, sino como expresión (por ejemplo, se puede poner una llamada a una función a la derecha de una asignación o en la condición de un if).

Por ejemplo, para llamar a la función max pasándole como parámetros las variables a y b y guardando el resultado en la variable c, escribiríamos:

```
c = max(a,b)
```

## Objetos implícitos ASP y Componentes ASP

ASP es una plataforma basada en objetos. Esto significa que ofrece un conjunto de objetos para ampliar la funcionalidad del lenguaje que estemos usando (por ejemplo, de VBScript).

ASP ofrece 5 objetos, cada uno de los cuales contiene una serie de métodos, propiedades y eventos. Estos 5 objetos son: **Response**, **Request**, **Application**, **Session**, y **Server**.

El objeto **Request** representa los datos que el navegador web envía al servidor web, es decir, representa la petición del navegador hacia el servidor web

El objeto **Response** representa los datos que el servidor web envía al navegador, es decir, representa la respuesta que el servidor web da al navegador.

El objeto **Application** representa nuestra aplicación web, es decir, el conjunto de páginas web que forman nuestro site.

El objeto **Session** representa la sesión de usuario actual, es decir, el cliente que está usando nuestra aplicación.

Finalmente, el objeto **Server** representa al servidor web.

Estos 5 objetos son los llamados objetos implícitos de ASP, los cuales son fijos (siempre son estos 5) y son la base de toda esta plataforma.

Existen, sin embargo, otro tipo de objetos llamados componentes ASP, destinados a ampliar aún más la funcionalidad de nuestras páginas ASP. Estos componentes son componentes realizados con la tecnología Active-X de Microsoft, lo que implica que no estamos hablando de una lista cerrada, sino de una lista que es ampliable, pudiendo incluso crear nuestros propios componentes Active-X desde otras herramientas de Microsoft como por ejemplo Visual Basic o Visual C++.

Sin embargo, la creación de componentes Active-X queda totalmente fuera del alcance de este curso, por lo que nosotros nos limitaremos a usar algunos de los componentes que se instalan por defecto con ASP. Concretamente usaremos los componentes ADODB.Connection y ADODB.Recordset, los cuales nos posibilitan el que nuestras páginas puedan acceder a datos almacenados en una bases de datos.

### 3.- Objetos Application y Session

Ya hemos visto que con la palabra reservada Dim, podemos crear variables en VBScript, Sin embargo, estas variables son visibles tan solo dentro de nuestra página (o solamente dentro de nuestro procedimiento o función, si la variable está definida dentro de este procedimiento o función).

Sin embargo, en muchas ocasiones nos interesará crear variables globales que sean visibles por todo el mundo. Éstas son las llamadas **variables de aplicación**.

Imaginaros, por ejemplo, que queremos que en nuestra web aparezca un contador de visitas, esto es, un número que inicialmente sea 0 y que vaya incrementándose cada vez que alguien vea nuestra página. Esto sería imposible de conseguir con variables locales.

Para acceder a una variable de aplicación, simplemente ponemos entre comillas y paréntesis el nombre de la variable que queremos crear, siguiendo al objeto Application. Por ejemplo, si queremos acceder a una variable de aplicación llamada n\_visitas, escribiremos:

```
Application("n_visitas")
```

Luego, si queremos escribir una instrucción para incrementar el número de visitas, escribiremos:

```
Application("n_visitas") = Application("n_visitas") + 1
```

utilizando Application("n\_visitas") como si fuera una variable "normal", sin embargo, el valor que posee esta variable es compartido por todos los usuarios que están accediendo a la página, así como por todas las páginas de nuestro site (en el caso que nuestro site tenga más de una página).

Así pues, para crear un contador de visitas para una página, escribiríamos algo así:

```
<HTML>
  <HEAD><TITLE>Mi web</TITLE></HEAD>
  <BODY>
    Bienvenido a mi página web.<BR>
    <% Application("n_visitas") = Application("n_visitas") + 1
    Eres el visitante número: <%= Application("n_visitas") %>
  </BODY>
</HTML>
```

Como se ha dicho antes, el valor que contiene Application("n\_visitas") es accesible por todas nuestras páginas. Por lo tanto, ahora podríamos escribir una segunda página de esta forma:

```
<HTML>
  <HEAD><TITLE> Segunda pagina de mi web</TITLE>
  <BODY>
    Ahora ya han accedido <%= Application("n_visitas") %> visitantes
  </BODY>
</HTML>
```

Existe un peligro en las variables de aplicación, y es el hecho del paralelismo de acceso a datos.

En el ejemplo anterior, imaginad qué pasaría si 2 usuarios ejecutan la instrucción `Application("n_visitas") = Application("n_visitas") + 1` exactamente en el mismo instante:

Instante inicial: `Application("n_visitas")` contiene el valor 4

2 usuarios ejecutan: `Application("n_visitas") = Application("n_visitas") + 1` a la vez

Instante final: `Application("n_visitas")` contiene el valor 5

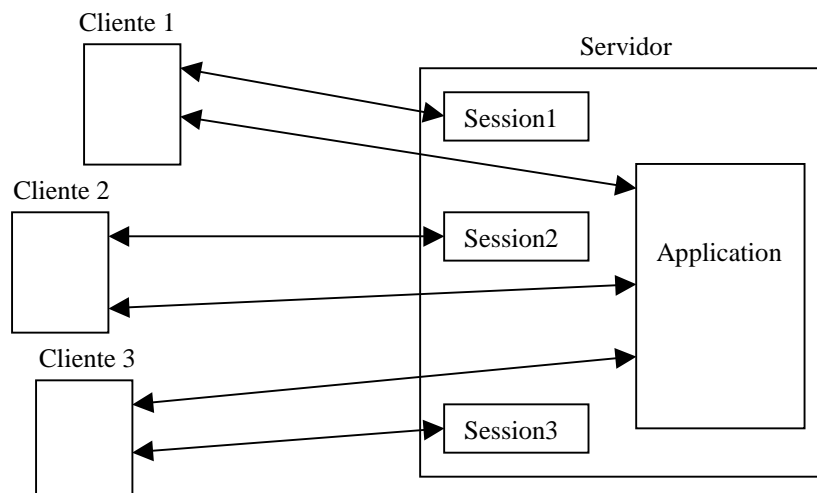
En el ejemplo anterior, `Application("n_visitas")` debería valer 6, sin embargo, debido a que los dos usuarios han ejecutado la instrucción de incremento en el mismo instante, cada uno de ellos ha leído el valor 4, ha sumado 1, y ha guardado el resultado en la variable, con lo que la variable contiene erróneamente el valor 5 en lugar de 6.

Para evitar este problema, debemos asegurarnos que cuando estamos accediendo a la variable `Application("n_visitas")` lo estamos haciendo exclusivamente, es decir, debemos bloquear la variable antes de ejecutar la instrucción, y liberarla justo después.

Existen 2 métodos del objeto `Application` para bloquear y liberar, respectivamente: son los métodos **Lock** y **Unlock**. Si queremos que no se pierdan visitas, deberemos escribir:

```
<%  
Application.Lock  
Application("n_visitas") = Application("n_visitas") + 1  
Application.Unlock  
%>
```

El objeto **Session** sirve también para crear variables cuyo valor sea compartido por distintas páginas, sin embargo, el valor de las variables de sesión no se comparte entre las distintas sesiones (es decir, entre los distintos clientes o usuarios).



En la figura se puede observar esquemáticamente como cada cliente tiene acceso sólo al objeto `Session` que le corresponde. Sin embargo todos pueden acceder al objeto `Application`, el cual es un único objeto compartido por todos los clientes (o compartido por todas las sesiones).

Este tipo de variables se usa para almacenar valores personalizados como pueden ser datos personales del navegante, opciones que ha escogido, etc...

Por ejemplo, podríamos disponer de una variable de sesión para guardar el idioma en el que el navegante desea ver el site. Éste valor puede ser distinto para cada cliente, por lo que no podemos guardarlo en una variable de aplicación porque queremos un valor para cada cliente en lugar de un único valor para todos los clientes (tal y como sucedía en el ejemplo anterior del contador).

En este caso podríamos escribir una página de bienvenida de la siguiente forma:

```
<HTML>
  <HEAD><TITLE>Hola!</TITLE></HEAD>
  <BODY>
    <% If Session("Idioma") = "Castellano" Then %>
      Bienvenido a mi sitio web!
    <% ElseIf Session("Idioma") = "Català" Then %>
      Benvingut al meu site d'internet!
    <% Else %>
      Welcome to my internet site!
    <% End If %>
  </BODY>
</HTML>
```

El objeto session tiene un método llamado **abandon** el cual al ser llamado fuerza el cierre de la sesión, aunque todavía no haya expirado el timeout.

Cada sesión se identifica internamente por un identificador global (GUID). Aunque no es habitual, este identificador se puede obtener mediante la propiedad **SessionID** del objeto Session. De esta forma podríamos comprobar, por ejemplo, *qué pasa en el caso que un usuario abra dos navegadores: son dos sesiones distintas o comparten la sesion?*

El objeto Session no requiere los métodos Lock y Unlock, ya que cada usuario tiene acceso a un objeto Session distinto (SU objeto Session), con lo que no hay conflictos de acceso.

Una característica importante de las variables Globales (tanto de aplicación como de sesión) es que éstas no se declaran, creándose la variable en el primer momento en que ésta sea utilizada.

*Ejercicio 1: hacer que aparezca en la página la hora en que alguien ha entrado en esta página por última vez.*

*Ejercicio 2: hacer que aparezca en la página la hora en la que yo he actualizado esta página por última vez.*



## 4.- Global.asa

En el ejemplo del contador de visitas, existe un pequeño percance: no hemos inicializado el número de visitas. Pero la pregunta es: en qué momento deberíamos inicializarlo? La respuesta sería: en el momento en que arrancamos nuestro servidor web.

Existe un fichero especial en todo site ASP, llamado Global.asa, el cual contiene 4 eventos que podemos programar. Son los eventos de inicio de aplicación (Application\_OnStart), inicio de sesión (Session\_OnStart), fin de aplicación (Application\_OnEnd) y fin de sesión (Session\_OnEnd). Este fichero debe estar en el directorio home de nuestro site.

Este fichero se programa también en VBScript (o en cualquier otro lenguaje script soportado por ASP), pero en lugar de usar las marcas <% y %>, estamos obligados a usar la sintaxis de las marcas SCRIPT. Es decir, al principio de este fichero deberemos escribir:

```
<SCRIPT LANGUAGE="VBScript" RUNAT="Server">
```

y al final del fichero deberemos escribir:

```
</SCRIPT>
```

escribiendo entre estas marcas aquellos procedimientos de evento que nos interese reprogramar (de entre los 4 procedimientos posibles)

### Application\_OnStart

El procedimiento de evento Application\_OnStart se ejecuta en el momento de arrancar nuestro servidor web. Éste es el procedimiento que se utiliza para hacer las inicializaciones que necesitemos a nivel de aplicación. Por ejemplo, en el caso del contador de visitas, usaríamos este procedimiento para inicializar la variable Application("n\_visitas") a 0.

Así pues, para hacer que el contador de visitas funcione, escribiríamos en el fichero Global.asa:

```
<SCRIPT LANGUAGE="VBScript" RUNAT="Server">  
Sub Application_OnStart()  
    Application("n_visitas") = 0  
End Sub  
</SCRIPT>
```

## Application\_OnEnd

Este procedimiento de evento se ejecuta en el momento en que nuestro servidor web se para. Se puede programar, por ejemplo, para que guarde todas las variables Application en una base de datos con el fin de poder recuperar posteriormente los valores actuales.

## Session\_OnStart

Este procedimiento se ejecuta en el momento en que se inicia una nueva sesión. Esto es, en el momento en que entra un nuevo usuario.

En el ejemplo del número de visitas, estamos incrementando las visitas cada vez que alguien ve nuestra página. Esto tiene 2 graves problemas:

- 1.- Si un usuario refresca la página 10 veces, se contará 10 veces más como número de visitas.
- 2.- Si un usuario entra en nuestro site a través de otra página, no se incrementará el número de visitas.

En realidad, deberíamos incrementar el número de visitantes cada vez que entra un nuevo visitante, y no cada vez que un visitante ve esta página. Por eso, lo correcto sería programar el incremento de visitas en el procedimiento Session\_OnStart. Deberíamos, por lo tanto, reescribir el archivo Global.asa de la siguiente forma:

```
<SCRIPT LANGUAGE="VBScript" RUNAT="Server">  
  Sub Application_OnStart()  
    Application("n_visitas") = 0  
  End Sub  
  Sub Session_OnStart()  
    Application("n_visitas") = Application("n_visitas") + 1  
  End Sub  
</SCRIPT>
```

## Session\_OnEnd

Finalmente, existe un evento que se activa cuando un usuario abandona una sesión. Sin embargo, un navegador web no envía ninguna señal al servidor web cuando el usuario cierra el navegador. Por este motivo, la forma de detectar este evento es el siguiente:

Cada vez que un cliente hace una petición, se inicia un contador de tiempo, el cual es reiniciado cada vez que se detecta cualquier actividad de ese cliente (o sesión). Cuando transcurrido un timeout un cliente no ha tenido ninguna actividad, es decir, cuando el contador de tiempo alcanza este timeout, se considera que el usuario ha abandonado la sesión.

El timeout por defecto es de **20 minutos**, sin embargo, este timeout es configurable por la propiedad **Timeout** del objeto Session. Por ejemplo, si queremos modificar el timeout de la sesión actual a 15 minutos, escribiríamos:

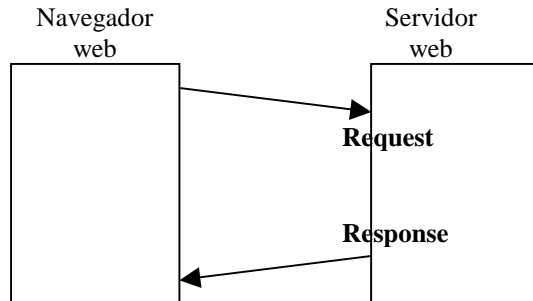
```
Session.Timeout = 15
```



*Ejercicio: Hacer un contador de visitas activas: que me aparezca en cada página cuantos usuarios están actualmente conectados en cualquiera de las páginas de mi site.*

## 5.- Response y Request

Los objetos Response y Request representan, respectivamente, la respuesta que el servidor web envía al navegador, y los datos que el navegador envía al cliente.



Existe, por ejemplo, un método del objeto Response que se encarga de escribir en la salida HTML que se enviará al navegador. Es el método write, que tiene como único parámetro el código HTML que queremos escribir.

Así pues, si escribimos:

```
<% Response.write("Hola <B>" & nombre_usuario & " </B>") %>
```

se generará como salida HTML:

Hola **Gabriel**

(suponiendo que la variable nombre\_usuario contiene el texto "Gabriel").

En realidad, el método write no es muy usado porque se suele usar la notación <%= en lugar de este método. El ejemplo anterior se podría escribir también como:

```
<%= "Hola <B>" & nombre_usuario & " </B>" %>
```

o también así:

```
Hola <B> <%= nombre_usuario %> </B>
```

Sin embargo, el método Response posee otros métodos y propiedades que sí se utilizan.

### Caché y caducidad de la página

Una de las grandes funcionalidades que poseen prácticamente todos los navegadores es el poder guardar las páginas usadas más recientemente en una zona temporal llamada **caché**. Esto, aunque disminuye el tráfico de red y aumenta la velocidad de navegación, es una fuente de problemas, ya que las páginas almacenadas en la caché no ejecutan código en el servidor.

Por ejemplo, en el caso del contador de visitas, si un cliente visita nuestra página y al día siguiente vuelve a visitarla, puede que esta segunda visita no sea contabilizada por culpa de la caché.

Si queremos controlar este problema, debemos indicarle al navegador que no guarde esta página en la caché. Uno de los parámetros que el servidor web le da al navegador cuando éste recibe una página HTML es la **fecha de caducidad** de esta página. La fecha de caducidad indica el momento a partir del cual la página está caducada y por lo tanto el navegador está obligado a solicitar una nueva versión de la página al servidor web, aunque tenga esta página en caché. Es decir, un navegador guarda una página en la caché hasta que esta página “caduca”.

Esta fecha de caducidad está contenida en la cabecera http, y es uno de los parámetros que podemos controlar mediante el objeto Response. Concretamente, el objeto Response tiene una propiedad llamada **expires** en la cual le indicamos el número de minutos dentro de los cuales queremos que la página caduque.

Si queremos que nuestra página nunca sea guardada en la caché del navegador para que éste pida siempre la página a nuestro servidor web, obligando así a que el código ASP sea ejecutado cada vez que la página sea visionada, deberemos escribir:

```
<% Response.Expires = 0 %>
```

con lo cual estamos indicando que la página caduque inmediatamente (dentro de 0 minutos).

Debido a que el parámetro Expires viaja en la cabecera http, esto es, antes que el cuerpo del documento HTML sea enviado, deberemos escribir esta instrucción antes de enviar ningún carácter HTML al navegador (esto es, antes de la marca <HTML>).

Existen otros métodos y propiedades del objeto Response. Quizás uno de los métodos más utilizados (a parte de la propiedad Expires) es el método **Redirect**, que sirve para redireccionar la página actual a una nueva página. El método Redirect tiene un solo parámetro que es la dirección URL donde queremos que nuestra página sea redireccionada. Esta URL puede ser tanto absoluta como relativa.

Por ejemplo, si queremos que la página actual sea redireccionada a una página llamada error\_en\_el\_site.asp la cual se encuentra en el mismo directorio que el de la página actual, escribiremos:

```
<% Response.Redirect(“error_en_el_site.asp”) %>
```

## Recibir información de formularios. Métodos GET y POST

Hemos hablado bastante sobre como escribir scripts de servidor que generen la página HTML, incluso que modifiquen propiedades de ésta como es el caso de su caducidad. Sin embargo, una de las cosas sobre la que no se ha hablado todavía y que es de vital importancia es de la obtención de datos introducidos por el usuario.

El principal mecanismo que tiene HTML para enviar información introducida por un usuario son los formularios. Los formularios son estructuras lógicas de un documento HTML que pueden contener distintos controles donde el usuario puede introducir información. Estos controles pueden ser cuadros de texto, listas, checkboxes, radio buttons, etc..., que es donde el usuario introduce o selecciona de entre un conjunto de opciones la información que se le solicita. Finalmente, los formularios suelen tener un botón llamado botón de Submit, el cual, al ser pulsado, toda la información del formulario es enviada al servidor.

Para hacer formularios en HTML se utiliza la marca <FORM>. Los parámetros importantes de esta marca son:

**ACTION:** indica la URL donde hay que enviar estos datos. En nuestro caso, indica la página ASP que recibirá estos datos.

**METHOD:** indica el método que se desea usar para enviar la información del formulario. Existen dos métodos posibles: el método **get** y el método **post**.

Ejemplo de formulario HTML<sup>4</sup>:

```
<FORM ACTION="mostrar_datos.asp" METHOD="GET">  
  <INPUT TYPE="Text" NAME="Apellido">  
  <INPUT TYPE="Text" NAME="Nombre">  
  <INPUT TYPE="Submit" VALUE="Aceptar">  
</FORM>
```

Éste formulario presentaría dos cuadros de texto y un botón de aceptar que al ser pulsado enviaría los datos del formulario (esto es, el apellido y el nombre que el usuario haya tecleado en los 2 cuadros de texto) a la página mostrar\_datos.asp.

Para recoger estos datos en nuestra página mostrar\_datos.asp, deberemos usar el objeto Request, el cual representa el conjunto de datos que el navegador envía a nuestro servidor web.

Cuando un formulario es enviado usando el método get, sus datos son enviados como parámetros de la URL. Esto es, la URL que se enviará al servidor será:

Mostrar\_datos.asp?Apellido=Plana&Nombre=Gabriel

---

<sup>4</sup> Se puede obtener una descripción más detallada sobre la creación de formularios HTML en la documentación del curso de internet básico organizado por BJT, o en manuales de HTML.

Para obtener los parámetros de una URL, esto es, para obtener los datos de un formulario que ha sido enviado usando el método GET, se usa la propiedad `QueryString` del objeto `Request`. Esta propiedad nos devuelve todo lo que hay después del interrogante (que es la marca de inicio de parámetros de una URL):

?Apellido=Plana&Nombre=Gabriel

Ahora que sabemos como obtener un string con los parámetros de la URL, podríamos usar las funciones de tratamiento de strings de VBScript para obtener cada uno de los valores de cada parámetro, los cuales están separados por símbolos & y estan en formato `nombre_parámetro = valor`.

Sin embargo, existe una forma mucho más cómoda de obtener cada uno de los parámetros. La propiedad `QueryString` es, en realidad, lo que se llama una **colección**, esto es, una lista de valores (algo parecido a un Array, pero donde cada elemento tiene un nombre que lo identifica). Podemos acceder a un elemento concreto de una colección añadiendo detrás de la colección el nombre del elemento cuyo valor deseamos obtener entre comillas y paréntesis. En el ejemplo anterior, para hacer que la página `mostrar_datos.asp` muestre los datos introducidos por el usuario, escribiríamos:

Nombre introducido: <%= Request.**QueryString**("Nombre") %>

Apellido introducido: <%= Request.**QueryString**("Apellido") %>

De hecho, ya habíamos usado colecciones sin saberlo cuando hemos usado las variables de aplicación y de sesión, ya que los objetos `Application` y `Session` son, en realidad, colecciones de variables.

El método `QueryString`, aunque es muy útil para recoger parámetros que nos pasan por URL, no es siempre la forma más indicada para transportar los datos de un formulario, ya que posee 2 inconvenientes:

- 1.- Una dirección URL suele estar limitada en cuanto al número de caracteres que ésta puede llegar a tener, siendo este número de 1024 la mayoría de veces. Por este motivo, puede que si nuestro formulario contiene muchos datos, éstos sean cortados (imaginad el caso que pidamos al usuario no sólo su nombre y apellido, sino un comentario sobre nuestra web).
- 2.- Cuando se envían los datos a través de una URL, ésta viaja en la cabecera del mensaje de protocolo http, en lugar de viajar en el cuerpo del mensaje. Esto implica que las URLs son enviadas siempre como texto llano, siendo pues estos datos, muy vulnerables a ser leídos por terceras personas, aunque se usen protocolos de seguridad como SSL (el cual encripta los datos, pero sólo los del cuerpo del mensaje http).

Por este motivo, existe un método mucho más fiable para enviar los datos de un formulario. Es el llamado método POST. Cuando un formulario se envía usando el método post, sus datos viajan dentro del cuerpo del mensaje de protocolo http, siendo posible codificar dicho contenido con certificados de seguridad como SSL. Además, no existe límite teórico en cuanto a bytes para un mensaje http. Por este motivo, este es el protocolo más usado para el envío de formularios.

Para recoger los datos de un formulario enviado mediante el método POST, se usa la propiedad Form del objeto Request, siendo la sintaxis la misma que la de la propiedad QueryString (es decir, Form es también una colección, cuyos elementos son identificados por el nombre de cada uno de los controles del formulario, y en cada uno de estos elementos se encuentra su valor).

Visto lo anterior, para recoger los datos del siguiente formulario:

```
<FORM ACTION="mostrar_datos.asp" METHOD="POST">  
  <INPUT TYPE="Text" NAME="Apellido">  
  <INPUT TYPE="Text" NAME="Nombre">  
  <INPUT TYPE="Submit" VALUE="Aceptar">  
</FORM>
```

Escribiríamos en la página mostrar\_datos.asp:

```
Nombre introducido: <%= Request.Form("Nombre") %>  
Apellido introducido: <%= Request.Form("Apellido") %>
```

## Trabajar con Cookies

Otra de las posibilidades que nos ofrecen los objetos Request y Response es el de poder trabajar con cookies.

Las cookies son pequeñas informaciones que el navegador puede guardar en una zona especial del disco duro del cliente (por ejemplo, Netscape guarda las cookies en un fichero llamado cookies.txt dentro del directorio de aplicación de Netscape, mientras que Internet Explorer guarda las cookies en un directorio llamado cookies dentro del directorio de windows.

Cuando el navegador hace una petición http al servidor, éste envía junto con la petición, sus cookies. Es posible, pues, obtener estas cookies mediante el objeto Request.

El objeto Request ofrece una propiedad llamada Cookies la cual contiene una colección con cada una de las cookies enviada por el cliente. Toda cookie posee un nombre que la identifica, un valor, y una fecha de caducidad a partir de la cual la cookie desaparece.

Podemos, por ejemplo, obtener el valor de una cookie llamada "Idioma predeterminado" con Request.cookies("Idioma predeterminado").

Sin embargo, si lo que deseamos es asignar un valor a una cookie, no podemos usar el objeto Request, sino que debemos usar el objeto Response, ya que lo que queremos es que nuestro servidor web envíe al navegador un valor de una cookie para que éste la guarde en el disco duro del cliente. El objeto Response ofrece también una propiedad llamada cookies que es también una colección de cookies. La diferencia entre la propiedad cookies del objeto Response y la propiedad cookies del objeto Request es que la del objeto Response es de sólo escritura y la del objeto Request es de sólo lectura, ya que Request nos permite consultar los valores de las cookies recibidas del cliente,

mientras que `response` nos permite enviar al navegador nuevos valores para las cookies (sean nuevas o no).

Por ejemplo, para asignar el valor “Castellano” a la cookie llamada “Idioma predeterminado”, deberíamos escribir:

```
<% Response.cookies("Idioma predeterminado") = "Castellano" %>
```

Sin embargo, el resultado de la línea anterior no será exactamente el resultado esperado. Esto es debido a que toda cookie tiene una fecha de caducidad que indica cuando esta cookie deja de ser válida (y por lo tanto su valor ya no será enviado al servidor), y al no haberla asignado, se cogerá el valor de caducidad por defecto.

Por defecto, una cookie caduca al caducar la sesión. Por este motivo, cuando cerremos el navegador, la cookie desaparecerá, en realidad sin que ésta haya llegado a ser guardada en disco.

Por eso deberíamos establecer una fecha de caducidad para la cookie. Para establecer una fecha de caducidad para la cookie deberemos utilizar la propiedad **expires** que tiene toda cookie. Es decir, escribiríamos la siguiente instrucción:

```
<% Response.cookies("Idioma predeterminado").expires = #2/23/2003# %>
```

que asignaría la fecha del 23 de febrero del 2003 para la caducidad de la cookie. Nótese que la fecha hay que escribirla en formato inglés, esto es, `#mes/día/año#`.

Podemos también asignar una fecha de caducidad relativa a la fecha actual escribiendo:

```
<% Response.cookies("Idioma predeterminado").expires = Now() + 365 %>
```

Lo que haría que la cookie caduque dentro de un año (concretamente, 365 días). Como puede observarse, cuando sumamos un número a una fecha, VBScript interpreta el número como número de días.

## 6.- Objeto Server

El objeto Server representa al Servidor Web, y aunque tiene otros métodos, los más importantes son: HTMLEncode y CreateObject.

### HTMLEncode

El método HTMLEncode del objeto server se utiliza para codificar un texto en HTML. Este método tiene un solo parámetro que es el texto a codificar.

Como es sabido, si queremos escribir texto en una página HTML debemos seguir unas ciertas reglas, codificando ciertos caracteres que en HTML están prohibidos. Por ejemplo, si queremos escribir el texto:

Añádalo aquí

deberemos escribir:

A&ntilde;&aacute;dalo aqu&iacute;

ya que en caso contrario, el texto no se mostraría bien si el juego de caracteres del navegador es distinto del juego de caracteres del sistema desde donde hemos generado nuestra página (por ejemplo, si escribimos nuestra página desde el editor del MS-DOS, y después intentamos visualizarla mediante Internet Explorer para Windows, ya que los códigos para los caracteres acentuados en MS-DOS y en Windows son distintos).

Para evitar tener que hacer este tedioso trabajo, el objeto implícito Server de ASP nos ofrece el método HTMLEncode que realiza esta codificación por nosotros. Así, para obtener el texto codificado en HTML anterior, escribiríamos:

```
<%= Server.HTMLEncode("Añádalo aquí") %>
```

La existencia de este método es vital, pues debemos tener en cuenta que no siempre conoceremos los textos que debemos mostrar. Si por ejemplo tenemos el siguiente código ASP:

```
<%= Request.Form("Nombre") %>
```

no tenemos forma de saber a priori si el nombre que recibiremos tendrá caracteres especiales, tal como ocurre con "Iñaki", "María", ...

El hecho de no codificar los textos a formato HTML puede implicar no sólo que ciertos caracteres aparezcan incorrectamente, sino que nuestra página no sea interpretada correctamente por el navegador.

Pongamos el caso anterior, en el que en lugar de pedirle el nombre al usuario, le pedimos que escriba una fórmula matemática:

```
<%= Request.Form("Formula") %>
```



Qué ocurre si el texto recibido es “x<y o y>z” ?

En este caso peculiar, si enviamos este texto al navegador, éste interpretará el carácter < como inicio de marca HTML, leyendo hasta que encuentre el carácter >, y interpretando lo contenido entre estos caracteres (es decir, el texto “y o y”) como una marca HTML que el navegador no conoce y que por lo tanto ignorará. El resultado: se mostrará el texto “xz” en lugar de “x<y o y>z”.

Como explicaré más adelante, en ciertos casos esto puede implicar un agujero de seguridad que puede ser explotado por hackers.

## Acceso a bases de datos

### Conceptos previos

Una **base de datos** es un conjunto de información guardada de forma persistente (los datos no se borran hasta que nosotros los borremos), estructurada (cada una de las entidades que deseamos guardar se guarda en estructuras lógicas separadas, llamadas tablas) y flexible (podemos añadir un nuevo atributo (llamado campo en el lenguaje de las bases de datos) a una tabla de manera fácil.

Las tablas se estructuran en filas y columnas. Cada columna es un **campo**, mientras que cada fila representa cada ocurrencia que deseamos guardar. Estas filas se llaman **registros**. Por ejemplo, si tenemos una tabla de coches, cada uno de los coches es un registro de esta tabla, mientras que los distintos atributos que puede tener un coche (fabricante, modelo, color, nº de matrícula, etc...) son los campos.

El **sistema gestor de la base de datos** es el programa encargado de gestionar estos datos para que sean guardados en disco de manera eficiente. Ejemplos de sistemas gestores de bases de datos son Microsoft Access, SQL-Server y Oracle.

Todo sistema gestor de base de datos incluye un mecanismo llamado **clave primaria** para evitar la duplicidad de los datos. Una clave primaria es un campo<sup>5</sup> cuyo valor no puede ser repetido en más de un registro de la tabla. En el ejemplo anterior, el campo matrícula podría ser la clave primaria de la tabla coches, ya que no pueden existir dos coches con la misma matrícula.

El mundo del diseño de las bases de datos es todo un mundo a parte, y no es objetivo de este curso enseñar a diseñar correctamente una base de datos. BJT organiza cursos de bases de datos (concretamente de Oracle, que es el sistema gestor de bases de datos más potente del mercado, o Access que es el más económico y que permite crear bases de datos y aplicaciones con mayor rapidez, ideal para la pequeña y mediana empresa). En estos cursos de bases de datos, se esclarecen estos conceptos y otros más avanzados como clave foránea, creación de índices, sentencias SQL avanzadas, PL-SQL, y un largo etc..., y donde se explica como estructurar bien la información para que los datos sean guardados de forma coherente y óptima.

### Acceso a bases de datos desde ASP

Sin duda alguna, uno de los mayores atractivos de las páginas ASP (y de los lenguajes script de servidor en general), es su capacidad de interactuar con bases de datos, pudiendo generar el contenido de la página HTML en función del contenido de una base de datos.

Sin embargo, VBScript no tiene instrucciones para el acceso a bases de datos, así como tampoco los tiene directamente ninguno de los 5 objetos implícitos de ASP. Es por este motivo que para acceder a bases de datos hay que utilizar componentes Active-X.

---

<sup>5</sup> En realidad, una clave primaria es un conjunto de campos cuya combinación de valores no puede ser repetido, aunque lo más usual es que este conjunto sea un único campo.

Los componentes Active-X que se instalan por defecto al instalar la plataforma ASP (ya sea instalando IIS o instalando PWS) son el componente ADODB.Connection para crear una **conexión** a una base de datos, y el componente ADODB.Recordset, el cual se usa para obtener un conjunto de registros (**recordset** en inglés) de una conexión realizada a una base de datos.

El objeto Server nos ofrece el método **CreateObject** para crear una instancia de un componente Active-X, esto es, para crear una conexión (una instancia de ADODB.Connection) o un recordset (una instancia del componente ADODB.Recordset). El método CreateObject tiene un único parámetro que es el nombre del tipo de componente Active-X del cual queremos crear una instancia. Por ejemplo, para crear una instancia de un componente ADODB.Connection escribiremos:

```
Server.CreateObject("ADODB.Connection")
```

y para crear una instancia de un componente ADODB.Recordset escribiremos:

```
Server.CreateObject("ADODB.Recordset")
```

Las instancias de los objetos se declaran también con Dim, sin embargo, para asignar un objeto debemos usar la palabra clave Set.

Así pues, por ejemplo, para crear una instancia de un componente ADODB.Connection y asignarla a una variable (un objeto) llamada cn, escribiremos:

```
Dim cn  
Set cn = Server.CreateObject("ADODB.Connection")
```

## ADODB.connection

El objeto ADODB.connection sirve para crear una conexión a una base de datos mediante los protocolos estándares de comunicación con las bases de datos ODBC (Open DataBase Connectivity) y OLE-DB (Object Linked and Embedded – DataBase). Tanto ODBC como OLE-DB son protocolos creados por Microsoft. La diferencia básica entre los dos es que OLE-DB es de posterior creación, siendo más eficiente, funcional y flexible. Es por esta razón que en este curso nos centraremos en este tipo de conexiones.

## El método open

Lo primero que hay que hacer siempre antes de acceder a una base de datos, es abrir una conexión a ella, y esto es justamente lo que nos permite el método **open** del componente OLEDB.Connection.

El método open consta de 3 parámetros: cadena de conexión, usuario y contraseña. Sin embargo, los 2 últimos parámetros son opcionales porque la cadena de conexión puede incluir ya el usuario y contraseña con los que acceder a la base de datos.

La cadena de conexión es un string donde se especifica el proveedor OLE-DB, la base de datos concreta que se quiere utilizar, el usuario y la contraseña.

El proveedor OLE-DB se especifica mediante la palabra clave **Provider**, y sirve para escoger el controlador de la base de datos. Por ejemplo, en el caso que el controlador de la base de datos sea el de Microsoft Access 2000 o XP, el proveedor de la base de datos será Microsoft.Jet.OLEDB.4.0. No podemos mostrar la lista de todos los proveedores de datos OLE-DB existentes en el mercado dado su número y dado que cada día surge alguno nuevo. Sin embargo, ofreceré una mini-lista de los proveedores OLE-DB de bases de datos Microsoft más usados:

Nombre proveedor OLE-DB	Descripción
Microsoft.Jet.OLEDB.3.51	Proveedor de datos OLE-DB para bases de datos Access-97
Microsoft.Jet.OLEDB.3.6	Proveedor de datos OLE-DB mejorado para bases de datos Access-97
Microsoft.Jet.OLEDB.4.0	Proveedor de datos OLE-DB para bases de datos Access-2000 y Access-XP.
SQLOLEDB	Proveedor de datos OLE-DB para bases de datos SQL-Server

De la anterior tabla podemos deducir, por ejemplo, que el proveedor OLE-DB para SQL-Server se llama SQLOLEDB.

La base de datos, en el caso de Microsoft Access, se refiere al archivo .mdb que deseamos abrir. Se especifica con la palabra clave **Data Source**.

El usuario con el que queremos acceder a la base de datos se puede especificar mediante el parámetro **User ID**, mientras que la contraseña se especificará con la palabra clave **Password**.

Así pues, para acceder a una base de datos hecha en Microsoft Access 2000, ubicada en el fichero dat.mdb en el mismo directorio que nuestra página web, con el usuario Gabriel y contraseña Plana, deberíamos escribir:

```
cn.open "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=dat.mdb; User ID=Gabriel; Password=Plana"
```

## El método Execute

El método `execute` del componente `ADODB.connection` se usa para enviar una sentencia SQL al proveedor de base de datos al que estamos conectados. SQL (Structured Query Language) es un lenguaje universal para el acceso a las bases de datos que nos permite, por ejemplo, insertar, modificar o eliminar registros en una tabla de la base de datos. Se verán unas nociones muy básicas en apartados posteriores dentro de este capítulo.

La sintaxis del método `execute` es:

```
cn.execute instrucción_SQL
```

donde `cn` es nuestra conexión a la base de datos (es decir, una instancia de `ADODB.connection`) y `instrucción_SQL` es un string que contiene la sentencia SQL que queremos ejecutar en la base de datos.

## El método close

Otro método importante del componente `connection` es el método `close`, el cual cierra la conexión, liberando todos los recursos que ésta necesitaba. Es importante utilizar el método `close` cuando hemos terminado de usar la conexión. Igualmente, también es importante liberar la memoria usada por la instancia creada. Esto se consigue asignando a la variable que contiene la instancia el valor `Nothing`.

Resumiendo, cuando no usemos más la conexión, deberemos escribir:

```
cn.close  
Set cn = Nothing
```

## ADODB.Recordset

El componente `ADODB.Recordset` representa un conjunto de registros obtenidos de nuestra base de datos. Se usa principalmente para mostrar en nuestra página web datos obtenidos de nuestra base de datos.

Al igual que ocurría con el componente `ADODB.Connection`, lo primero que hay que hacer es declarar una variable y asignarle una instancia de este componente. Esto se consigue escribiendo:

```
Dim rs  
Set rs = Server.CreateObject("ADODB.Recordset")
```

Seguidamente, y tal como ocurre con la conexión, hay que usar el método **open** para abrir el recordset recientemente creado. La diferencia es que un recordset siempre hay que abrirlo utilizando una conexión a la base de datos. Por eso, uno de los parámetros

(concretamente el segundo) del método `open` del recordset es la conexión. El otro parámetro es una sentencia SQL para la obtención de los datos (Sentencia `SELECT`)<sup>6</sup>.

Para abrir un recordset que contenga todos los datos (todos los registros con todos los campos) de la tabla `coches`, escribiríamos:

```
rs.open "Select * from coches", cn
```

Como se puede ver, la sentencia SQL para obtener todos los datos (filas y columnas) de una tabla es: `SELECT * FROM nombre_tabla`. En el siguiente apartado se darán unas nociones muy básicas sobre el lenguaje SQL, las cuales pueden ser ampliadas con un curso sobre bases de datos.

Un recordset es un conjunto de registros con un cursor que apunta a el **registro actual**. Cuando se abre un recordset, el registro actual es el primero de los registros obtenidos (siempre que haya como mínimo un registro). Si no hay ningún registro (en nuestro caso, si la tabla de coches no contiene información de ningún coche), el registro actual apunta a un registro no válido.

El recordset posee dos propiedades para detectar si el registro actual es válido o no. Son las propiedades booleanas **BOF** y **EOF**. Si estas 2 propiedades son `True`, significa que el recordset está vacío, es decir, no hay ningún registro, y por lo tanto, el registro actual no es válido.

### Los métodos **Move**

La gracia de los recordsets es poder recorrer todos los registros que éste contiene. Es lógico, pues, pensar que el recordset contendrá unos métodos para poder desplazar el registro activo dentro del conjunto de registros. Existen 4 métodos de desplazamiento en un recordset: son los métodos **MoveFirst**, **MoveNext**, **MovePrevious** y **MoveLast**, los cuales desplazan el cursor actual al primer registro del recordset (`MoveFirst`), al siguiente registro (`MoveNext`), al registro anterior (`MovePrevious`) y al último registro (`MoveLast`).

Así pues, para movernos al siguiente registro, escribiríamos:

```
rs.MoveNext
```

Dado que tras abrir el recordset el registro activo se posiciona al principio, podemos realizar un recorrido secuencial de todo el recordset simplemente utilizando el método `MoveNext` repetidas veces hasta llegar al final del recordset, pero necesitamos poder detectar si hemos llegado al final del recordset, lo cual es indicado por la propiedad **EOF** (Abreviación de End Of File).

Esta forma no es la única manera de proceder, pudiendo también realizar un recorrido inverso. Para realizar un recorrido inverso, nos desplazaríamos al último registro tras

---

<sup>6</sup> En realidad, el método `open` dispone de otros parámetros opcionales que nos permiten especificar, entre otras cosas, el tipo de recordset que deseamos. En este curso sólo usaremos recordsets de sólo lectura, que son los que se asignan por defecto, y son los que ocupan menos recursos, aumentando así la eficiencia, aunque también son menos potentes.

abrir el recordset (usando MoveLast) y después iríamos ejecutando repetidamente el método MovePrevious para ir retrocediendo de registro, hasta que llegáramos al principio. Cuando llegamos al principio (concretamente, cuando estamos al primer registro y pedimos el registro anterior, es decir, cuando nos salimos del recordset por su principio), se activa la propiedad **BOF** con valor True (abreviatura de Begin Of File).

### Obtener el valor de un campo en el registro actual

Un recordset, es también una colección, siendo esta colección el conjunto de campos que posee el recordset. Esto supone que podemos acceder al valor de cualquier campo (del registro actual) escribiendo el nombre del campo entre comillas y paréntesis seguido de nuestra variable que contiene el recordset.

Por ejemplo, en el caso de los coches, podríamos acceder al campo matricula escribiendo:

```
rs("matricula")
```

Visto lo anterior, el código entero para abrir un recordset con todos los datos de los coches y hacer un recorrido secuencial por estos datos para mostrar en la página HTML la lista de matrículas, sería el siguiente:

```
<%  
    Dim cn, rs  
    Set cn=Server.CreateObject("ADODB.Connection")  
    cn.open "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=dat.mdb"  
    Set rs=Server.CreateObject("ADODB.Recordset")  
    rs.open "Select * from coches", cn  
    While Not EOF  
        Response.write("Matrícula: " & rs("matricula") & "<BR>")  
        rs.MoveNext  
    Wend  
    rs.close  
    Set rs=Nothing  
    cn.close  
    Set cn=Nothing  
%>
```

## **Otras posibilidades de los recordsets**

Los recordsets ofrecen otras funcionalidades que, por cuestiones de tiempo, no pueden verse en este curso. Estas posibilidades son:

Buscar un registro dentro del recordset que cumpla un cierto criterio: Esto se consigue usando métodos Find (FindFirst, FindNext, FindPrevious y FindLast) que tienen como único parámetro el criterio de búsqueda. Un grave error frecuente de eficiencia es utilizar este método en lugar de filtrar los datos mediante SQL.

Paginación: Ciertos tipos de recordsets de OLE-DB permiten paginar el conjunto de registros. Esto se emplea mucho en las páginas web cuando se desea mostrar un conjunto de datos grande, los cuales son partidos en distintas páginas. Este concepto es un poco más avanzado y se deja para un posible segundo curso de ASP.

Actualización de datos desde un recordset: En ciertos tipos de recordsets es posible actualizar los datos del recordset y hacer que estos cambios sean reflejados directamente en la base de datos. En este curso no se utilizarán ya que se emplean recordsets de sólo lectura.



## Lenguaje SQL

El lenguaje SQL (Structured Query Language) es el lenguaje universal de las bases de datos, soportado por todos los sistemas gestores de bases de datos importantes como Oracle, SQL-Server e incluso Access.

El éxito de este lenguaje se debe a su compatibilidad con todos los gestores de bases de datos relacionales, a su clara sintaxis (siendo ésta muy parecida al lenguaje natural inglés) y a su alto nivel (en una instrucción SQL simplemente se indica lo que se desea hacer, sin tener que escribir la forma de proceder para conseguirlo, siendo esto último tarea del sistema gestor de la base de datos).

Las sentencias del lenguaje SQL se dividen en dos grandes categorías:

- **Sentencias DDL:** Las sentencias DDL (Data Definition Language) son las sentencias SQL que nos permiten definir las estructuras de los datos. Aquí están sentencias para crear tablas, índices usuarios, y un largo etcétera. Este tipo de sentencias no son las que nos interesan a nosotros, pues nosotros supondremos que la base de datos ya está creada. Además, algunos sistemas gestores de bases de datos (como por ejemplo Microsoft Access) incorporan asistentes para crear las tablas de forma visual.
- **Sentencias DML:** Las sentencias DML (Data Manipulation Language) son las sentencias SQL que nos permiten insertar, modificar, borrar o consultar los datos almacenados en una tabla. Existen 4 sentencias básicas, que se corresponden a las 4 operaciones anteriormente mencionadas: INSERT, UPDATE, DELETE y SELECT. Estas sentencias son usadas por todo programa que acceda a bases de datos, sin importar la plataforma de programación (ASP, Visual Basic, Delphi, Visual C++, JSP, PHP, etc...).

### La sentencia SELECT

La sentencia SELECT se usa para obtener un conjunto de registros de la base de datos. La sintaxis más básica de una sentencia SELECT es:

**Select** campos **From** tabla

Donde tabla es el nombre de la tabla que contiene los datos a consultar, y campos es una lista de los nombres de los campos que deseamos obtener separados por comas, o bien un asterisco (\*) indicando que se desean todos los campos.

Así pues, para obtener los campos modelo y matricula de la tabla coche, escribiríamos:  
Select modelo, matricula From coche

Para obtener únicamente el campo matrícula de los coches, escribiríamos:  
Select matricula from coche

Y finalmente, para obtener todos los campos de los coches, escribiríamos:  
Select \* from coche

Esto nos permite elegir exactamente qué campos queremos obtener, sin embargo, siempre obtenemos todos los registros (es decir, todos los coches). Esto se puede evitar usando la cláusula **where**.

La cláusula **where** es una palabra clave que se puede añadir en una sentencia **SELECT** después de la cláusula **from**. El **where** nos permite definir una condición que deben cumplir los registros para que sean incluidos como resultado, es decir, el **where** nos permite obtener sólo aquellos registros que cumplan una cierta condición. Esta condición es una expresión booleana que puede contener campos de la tabla especificada en la cláusula **from**.

Así pues, si queremos obtener todos los campos del coche cuya matrícula es B-9919-GP, escribiremos:

```
Select * From coches where matricula='B-9919-GP'
```

Si queremos obtener las matrículas de los coches cuyo número de multas es mayor que 3 (suponiendo que en la tabla hay un campo llamado **n\_multas** que contiene el número de multas puestas en cada coche), escribiremos:

```
Select matricula from coches where n_multas > 3
```

Existen más cláusulas posibles de la sentencia **SELECT**, como son las cláusulas **GROUP BY**, **HAVING** y **ORDER BY**, sin embargo estos conceptos son más avanzados y son explicados en cursos de bases de datos. También es posible obtener, mediante una sola **SELECT**, datos combinados de 2 o más tablas, haciendo una operación llamada **JOIN**. Este concepto se explica también en cursos de bases de datos.

## Sentencia **INSERT**

La sentencia **insert** sirve para añadir nuevos registros a una tabla. Su sintaxis es:

**Insert into** tabla (campo1, campo2, ... campoN) **values** (valor1, valor2, ... valorN)

Donde **campo1, campo2, ... campoN** es una lista de los nombres de los campos (con la misma sintaxis que la lista de campos de la sentencia **select**) para los cuales deseamos insertar un valor. Los campos de la tabla que no aparezcan en la lista, quedarán vacíos o con el valor que en el momento de crear la tabla se haya definido por defecto.

**valor1, valor2, .... valorN** representa una lista de los valores para cada uno de los campos (el **valor1** corresponde al valor del **campo1**, **valor2** corresponde al valor del **campo 2**, y así sucesivamente).

Hay que destacar que todo campo obligatorio debe ser incluido en la lista de campos del **insert**. Por ejemplo, todo campo que es clave primaria es obligatorio por definición, lo que implica que hay que incluir siempre la clave primaria en una sentencia **insert**.

En realidad, la lista con los nombres de los campos no es obligatoria. Siendo posible la notación:

```
Insert into nombre_tabla values (valor1, valor2, ... valorN)
```

Sin embargo, usando esta notación se sobreentiende que la lista de valores corresponde a todos los campos de la tabla, siendo pues obligatorio la inclusión de cada valor para todos los campos, debiendo usar, además, el mismo orden de campos que el que hay en la definición de la tabla.

Esta notación, además, no es recomendable ya que provoca posibles futuros problemas de mantenimiento de la aplicación, tal como se explica en el curso de bases de datos.

Visto todo lo anterior, si queremos insertar un coche cuya matrícula es B-0000-MD, deberíamos escribir:

```
Insert into coches (matricula) values ('B-0000-MD')
```

En el ejemplo anterior, se insertaría un coche con matrícula B-0000-MD y todos los otros campos vacíos (o con los valores que se haya predeterminada para cada campo en el momento de creación de la tabla).

Si queremos insertar un coche del cual conocemos su matrícula, el modelo, el fabricante y el número de multas, escribiremos:

```
Insert into coches (matricula, modelo, fabricante, n_multas)  
Values ('B-6969-XX', 'Fiesta', 'Ford', 8)
```

En el ejemplo anterior, el orden de los campos carece de importancia, siempre que se corresponda con el orden de los valores. Así pues, la sentencia anterior se podría escribir también como:

```
Insert into coches(n_multas, matricula, fabricante, modelo)  
Values (8, 'B-6969-XX', 'Ford', 'Fiesta')
```

## Sentencia UPDATE

La sentencia UPDATE se usa para modificar el valor de algun campo de uno o más registros existentes.

Su sintaxis más simple es:

**Update** tabla **SET** campo = valor

Donde tabla es el nombre de la tabla que contiene los registros que deseamos modificar, campo es el nombre del campo que deseamos modificar y valor es una expresión (del mismo tipo que el campo) con el nuevo valor que debe adoptar el campo.

Esta notación actualiza todos los registros de la tabla.

Por ejemplo, si escribimos:

```
Update coches set n_multas = 0
```

Estaríamos cambiando el valor del campo `n_multas` de todos los coches, asignándole el nuevo valor: 0 (es decir, hemos quitado todas las multas a todos los coches).

También podemos usar nombres de campo dentro del valor. Por ejemplo, si escribimos:

```
Update coches set n_multas = n_multas + 1
```

Incrementaríamos el número de multas de todos los coches, haciendo que aquellos coches que tenían 0 multas pasen a tener 1 multa, que los coches que tenían 1 multa pasen a tener 2, y así sucesivamente.

La sentencia `update` también permite actualizar más de un campo. Esto se consigue añadiendo más parejas `campo=valor` separándolas por comas.

Por ejemplo, la siguiente sentencia:

```
Update coches set n_multas = 0, propietario = 'Gabriel'
```

Quitaría las multas de todos los coches y asignaría el campo `propietario` a 'Gabriel' (suponiendo que existe el campo `propietario` dentro de la tabla `coches`).

Finalmente, tal y como ocurría con la sentencia `select`, se puede usar la cláusula **where** para filtrar los registros. Es decir, podemos evitar que todos los registros sean modificados, modificando sólo aquellos que cumplan una determinada condición.

Por ejemplo, si escribimos:

```
Update coches set n_multas = 0 where matricula = 'B-9919-GP'
```

estaríamos quitando las multas del coche cuya matrícula es 'B-9919-GP'.

## Sentencia DELETE

La sentencia `Delete` sirve para eliminar registros de una tabla. Su sintaxis es:

```
Delete from tabla where condición
```

Donde `tabla` es el nombre de la tabla de la cual queremos borrar registros, y `condición` es la condición que deben cumplir los registros a eliminar (los registros que no cumplan la condición no se verán afectados por la sentencia `delete`, es decir, no serán borrados).

La cláusula `where` (junto con su condición) no es, en realidad, obligatoria. Sin embargo, si no la incluimos, se borrarán todos los registros. Esto es, se borrará toda la tabla (aunque la tabla conservará su estructura, pero ésta quedará totalmente vacía, es decir, sin ningún registro).

Algunos ejemplos de la sentencia delete son:

Delete from coches where matricula = 'B-6969-XX'

Que borraría de la tabla coches el coche cuya matrícula es 'B-6969-XX'.

La sentencia:

Delete from coches where fabricante='Renault'

Borraría todos los coches cuyo fabricante es Renault.

## 7.- Seguridad: Proteger nuestra web de Hackers.

Una de las características de las aplicaciones por internet, es que éstas son accesibles a todo el mundo y muy a menudo de una forma anónima.

Es por esto que cuando desarrollemos este tipo de aplicaciones (ya sea con ASP o usando otras tecnologías) hay que prestar una atención muy especial al tema de la seguridad.

En general, la seguridad tiene distintos niveles. Por ejemplo, podemos hablar de que la plataforma ASP es segura, en cuanto a que como plataforma de internet te ofrece unas garantías, esto es, una base sobre la que partir la cual se considera segura<sup>7</sup>.

Sin embargo, es un error muy grave pensar que si la plataforma es segura, nuestra aplicación será segura. Nuestra aplicación será segura sólo si la plataforma es segura y si seguimos ciertas reglas, sobretodo al tratar elementos que el usuario introduce.

Una de las cosas que hay que evitar es que nuestra página pueda producir errores. Es decir, que nuestra página esté totalmente exenta de errores. Al decir errores, no me estoy refiriendo a errores de compilación, sino al hecho que, por ejemplo, el usuario pueda introducir valores que causen errores en nuestras páginas.

Por ejemplo, imaginemos que estamos haciendo una calculadora por internet donde el usuario introduce dos números y tiene botones para sumar, restar, multiplicar y dividir. Las operaciones que ASP nos ofrece para recoger los valores del usuario (Request.Form), para realizar la operación que haya pulsado (operadores +, -, \* o /), y para mostrar el resultado (Response.write) están (en principio) exentas de errores. Sin embargo, podría darse el caso que el usuario ponga como segundo número el 0 y pulse el botón de dividir. Al realizar el cálculo de división la aplicación daría un error, haciendo que ASP genere como salida HTML una página indicando dicho error.

Igualmente, tampoco podemos estar seguros que el usuario teclee sólo números en las casillas. Podemos pensar el caso de controlar esto con un script en el cliente, sin embargo, algunos navegadores no ejecutan JavaScript, y otros permiten desactivarlo, por lo que esta forma de proceder no es una opción nada segura (en general, cualquier sistema de protección que implementemos usando JavaScript, no es nada seguro, siendo JavaScript útil únicamente para evitar tráfico excesivo por la red).

El hecho de controlar muy bien todos los errores que puedan suceder es muy importante no sólo por cuestiones de imagen (menuda será nuestra web si falla cada dos por tres), sino por cuestiones de seguridad, pues la mayoría de veces un error o un caso no tenido en cuenta por el programador, puede ser explotado por un hacker hábil.

A continuación se presentan 2 casos a modo de ejemplo para ilustrar lo anteriormente descrito. Estos casos no pretenden fomentar la práctica del hacking sino, al contrario, conocerlas bien con el fin de poder proteger nuestras aplicaciones de estos ataques.

---

<sup>7</sup> En realidad, ninguna plataforma es segura al 100%, y conviene estar inscrito en grupos de noticias sobre seguridad para saber en todo momento los agujeros de seguridad que se descubran en la plataforma que usamos, y podernos bajar la correspondiente actualización que solucione el problema. Por ejemplo, en las primeras versión 3.0 de Internet Information Server, éste tenía un agujero de seguridad que permitía ver el código ASP desde el navegador.

## Caso de un foro.

El caso que explicaré en este apartado es una explotación de un error que se suele hacer cuando debería usarse el método HTMLEncode del objeto Server y éste no es usado por desconocimiento o descuido del programador.

Imaginaros un site que contiene el típico foro, esto es, una página donde aparece un conjunto de opiniones que distintas personas han ido añadiendo, con un posterior formulario que nos permite añadir nuestras opiniones propias. Estas opiniones podrían ser guardadas en una base de datos, para ser extraídas después desde ASP.

Así pues, tras las marcas HTML HEAD y BODY iniciales, y tras haber abierto un recordset sobre la variable rs para obtener los datos de esta tabla de opiniones donde habría el campo autor y el campo comentario, la página tendría un aspecto como el que sigue:

```
<% while not rs.EOF %>
    Comentario de: <%= rs("autor") %>: <BR>
    <%= rs("comentario") %>
    <% rs.MoveNext %>
<% Wend %>
<FORM ACTION="guardar.asp" METHOD="POST">
    <INPUT TYPE="Text" NAME="Autor">
    <INPUT TYPE="Text" NAME="Comentario">
    <INPUT TYPE="Submit" VALUE="Enviar">
</FORM>
</BODY></HTML>
```

Cuando un usuario, tras introducir su nombre y su comentario, pulsara el botón Enviar, los datos del formulario serían enviados a la página guardar.asp para que ésta guardara los datos introducidos en la base de datos. Ésta página tendría el siguiente aspecto:

```
<%
    Dim cn
    Dim consulta_SQL
    Set cn = Server.CreateObject("ADODB.Connection")
    cn.open "Provider=Microsoft.Jet.ADODB.4.0;Data Source=dats.mdb"
    consulta_SQL = "INSERT INTO comentarios (Autor, Comentario) VALUES ("
    consulta_SQL = consulta_SQL & Request.Form("Autor") & ","
    consulta_SQL = consulta_SQL & Request.Form("Comentario") & ")"
    cn.execute consulta_SQL
    Response.Redirect("pagina_inicial_foro.asp")
%>
```

Esta forma de proceder provocará que en ciertas ocasiones nuestra página falle. Imaginemos el caso que el tema tratado en el foro sea sobre unas fórmulas matemáticas. Qué ocurre si alguien escribe: *En lugar de  $x < y$  debería poner  $x \leq y$  ?* Al guardar este texto tal cual en la base de datos y después incluirlo tal cual en el documento HTML generado, el navegador interpretará el carácter < como inicio de marca, y no mostrará los caracteres que sigan hasta que encuentre el carácter >, con lo que puede que parte de

nuestro comentario desaparezca o, peor aún, puede no mostrarse parte del comentario posterior, o incluso varios comentarios posteriores, hasta que se encuentre un carácter >.

Éste error, que simplemente se debe al descuido del programador de no haber usado el método HTMLEncode antes de sacar por la salida HTML un valor introducido por el usuario, puede ser explotado por alguien para cerrar este foro para siempre, consiguiendo que nadie más pueda introducir sus opiniones.

La forma de proceder es realmente sencilla, y simplemente hay que tener en cuenta el hecho de que el navegador está interpretando el texto escrito por el usuario como código HTML. Se deja al lector la deducción sobre como alguien podría cerrar un foro de este tipo.

La solución a este problema, tal y como se ha insinuado en líneas anteriores, pasaría por usar el método HTMLEncode en el momento de guardar en la base de datos los valores obtenidos del formulario, o bien en el momento de sacar por la salida HTML el valor obtenido de la base de datos.

Es decir, lo podemos solucionar modificando las siguientes dos líneas

```
consulta_SQL = consulta_SQL & Request.Form("Autor") & ","  
consulta_SQL = consulta_SQL & Request.Form("Comentario") & ")"
```

por estas otras:

```
consulta_SQL = consulta_SQL & Server.HTMLEncode(Request.Form("Autor")) & ","  
consulta_SQL = consulta_SQL & Server.HTMLEncode(Request.Form("Comentario")) & ")"
```

Con esto evitamos el error del usuario que escribía el comentario *En lugar de x<y debería poner x<=y*, ya que éste será escrito en HTML como:

“En lugar de x<y deberí;a poner x<=y”

y por lo tanto este error ya no puede ser explotado por hackers.

También se puede no hacer este cambio y usar el método HTMLEncode en

```
<%= rs("autor") %>  
y <%= rs("comentario") %>
```

pero no en los dos sitios, pues si no estaríamos codificando en HTML código que ya ha sido codificado.



## Caso de búsquedas con formularios

Otro caso típico donde podemos hacer un error fácilmente explotable por hackers es cuando realizamos formularios HTML para hacer búsquedas en bases de datos.

Por ejemplo, imaginaos el siguiente formulario de ejemplo:

```
<FORM ACTION="buscar.asp" METHOD="Post">  
  Nombre a buscar: <INPUT TYPE="Text" NAME="Nombre"> <BR>  
  <INPUT TYPE="Submit" VALUE="Buscar">  
</FORM>
```

y la siguiente página buscar.asp:

```
<%  
  Dim cn, rs, consulta_SQL  
  Set cn = Server.CreateObject("ADODB.Connection")  
  cn.open "Provider=Microsoft.Jet.ADODB.4.0;Data Source=bd.mdb"  
  Set rs = Server.CreateObject("ADODB.Recordset")  
  consulta_SQL = "SELECT tel FROM personas WHERE nombre = ""  
  consulta_SQL = consulta_SQL & Request.Form("Nombre") & ""  
  rs.open consulta_SQL, cn  
  Response.write("Tu tel. es:" & rs("tel") )  
  rs.close  
%>
```

La aplicación anterior pretende pedir un nombre al usuario para después buscar en la base de datos el teléfono de la persona cuyo nombre es el nombre introducido, para mostrar este teléfono en el documento HTML resultante.

Por ejemplo, si escribimos en el formulario *Gabriel*, tras pulsar el botón de Buscar se ejecutará la página buscar.asp que lanzará la siguiente consulta al gestor de bases de datos (en nuestro caso, a Access):

```
SELECT tel FROM personas WHERE nombre='Gabriel'
```

En general la aplicación anterior funcionará bien, sin embargo, no funciona en todos los casos. Imaginaos el caso del señor Brian O'Connor. En este caso se intentará enviar la siguiente sentencia SQL a la base de datos:

```
SELECT tel FROM personas WHERE nombre='Brian O'Connor'
```

Sin embargo, si analizamos bien esta sentencia, nos daremos cuenta que el apóstrofe (o comilla simple) causará problemas al gestor de la base de datos, quien provocará un error de sentencia SQL.

Este error se puede explotar para obtener cualquier información de nuestra base de datos (por ejemplo, toda lista de nuestros clientes, con sus direcciones de e-mail, sus números de VISA, ...) o incluso otras informaciones que podamos guardar en la base de datos (como por ejemplo, contraseñas de acceso a partes restringidas de nuestra aplicación,

etc...). La forma de proceder es aprovechándonos del hecho que cuando escribimos un apóstrofe estamos cerrando el string del valor buscado en el where, dejando pues al usuario la opción de que pueda completar la sentencia SQL. Por eso, un hacker con algunos conocimientos SQL podría obtener cualquier información de nuestra base de datos, haciendo una select con una UNION<sup>8</sup>.

Una forma de evitar este error es simplemente sustituyendo los apóstrofes por otro carácter (por ejemplo, un acento abierto sobre un espacio), o bien escribiendo la codificación que use el SGBD para poner una comilla entre comillas. Por ejemplo, en el caso de Access, para poner una comilla entre comillas se deben poner dos comillas juntas. Es decir, el texto anterior deberíamos introducirlo como:

WHERE nombre = 'Brian O''Neil'

Es decir, deberemos sustituir una comilla simple por dos comillas simples. Para este cometido, se suele usar la función Replace.

El tema de la seguridad es un tema sobre el que se puede hablar mucho, y no se pretende que este curso sea un curso de seguridad. Simplemente se desea dar una visión al lector de lo importante que es que controlemos todos los casos posibles cuando programamos, sin dar nada por supuesto, pues de lo contrario siempre nos podemos encontrar con usuarios que por error o intencionadamente introduzcan datos que provoquen casos que no hemos previsto.

Quisiera hacer nuevamente hincapié en que el código JavaScript que podamos añadir a nuestra página HTML no supondrá ninguna mejora en el control de errores ni en la seguridad, ya que este se ejecutará en el cliente (si es que se ejecuta), el cual está totalmente fuera de nuestro control.

Finalmente, hay otras muchas reglas que se deberían seguir, como criptografiar los datos vulnerables (como contraseñas o números de VISA), no mostrar nunca los contenidos de ciertos campos por pantalla, etc... Sin embargo esto se describiría con más detalle en algún curso sobre seguridad.

---

<sup>8</sup> Este tipo de SELECTs es explicada en cursos de bases de datos

## Apéndice A: Gestión de nuestro site

El siguiente apéndice pretende dar unas nociones básicas de lo que es la gestión de nuestro site usando PWS. La gestión de IIS se hace de una forma prácticamente idéntica, por lo que no se analizará en este curso.

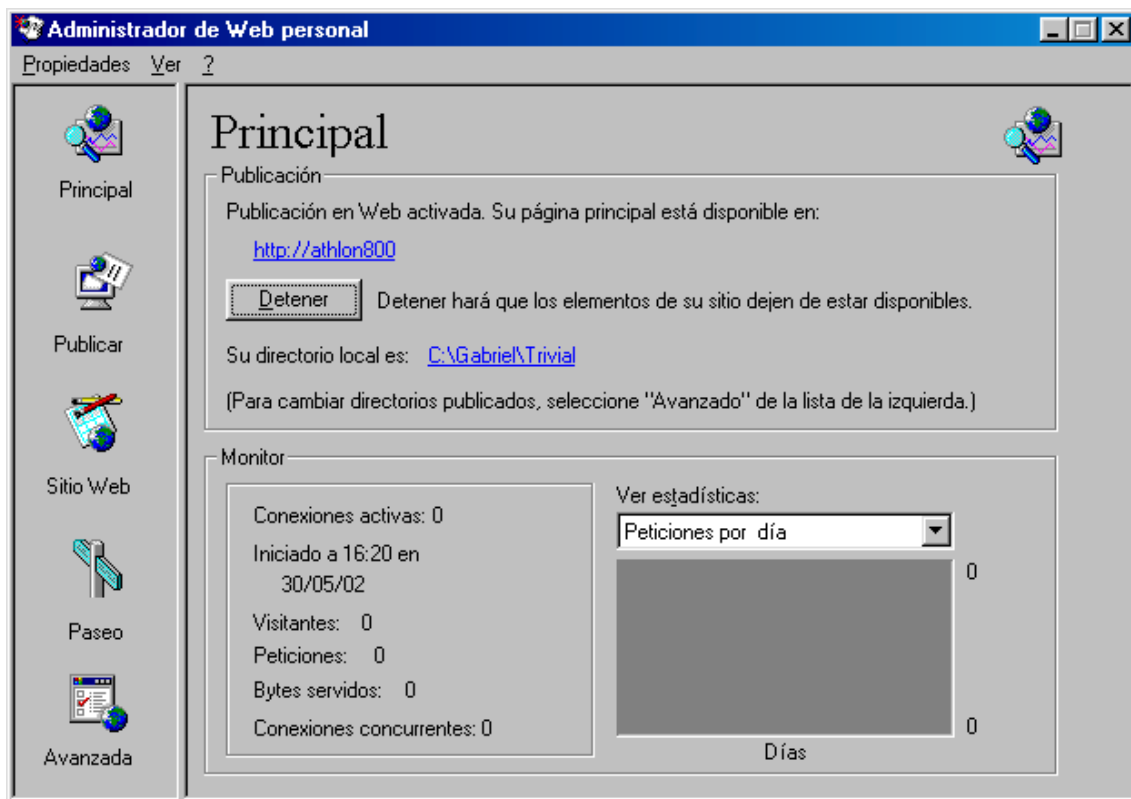
Si hemos instalado PWS en nuestra máquina, podremos observar que en la parte de la derecha de la barra de tareas de Windows aparece el icono de PWS:



NOTA: Este icono representa al Personal Web Server cuando éste está detenido. Si PWS está arrancado, aparecerá este otro icono:



Si hacemos doble-click sobre él, aparecerá la siguiente ventana:

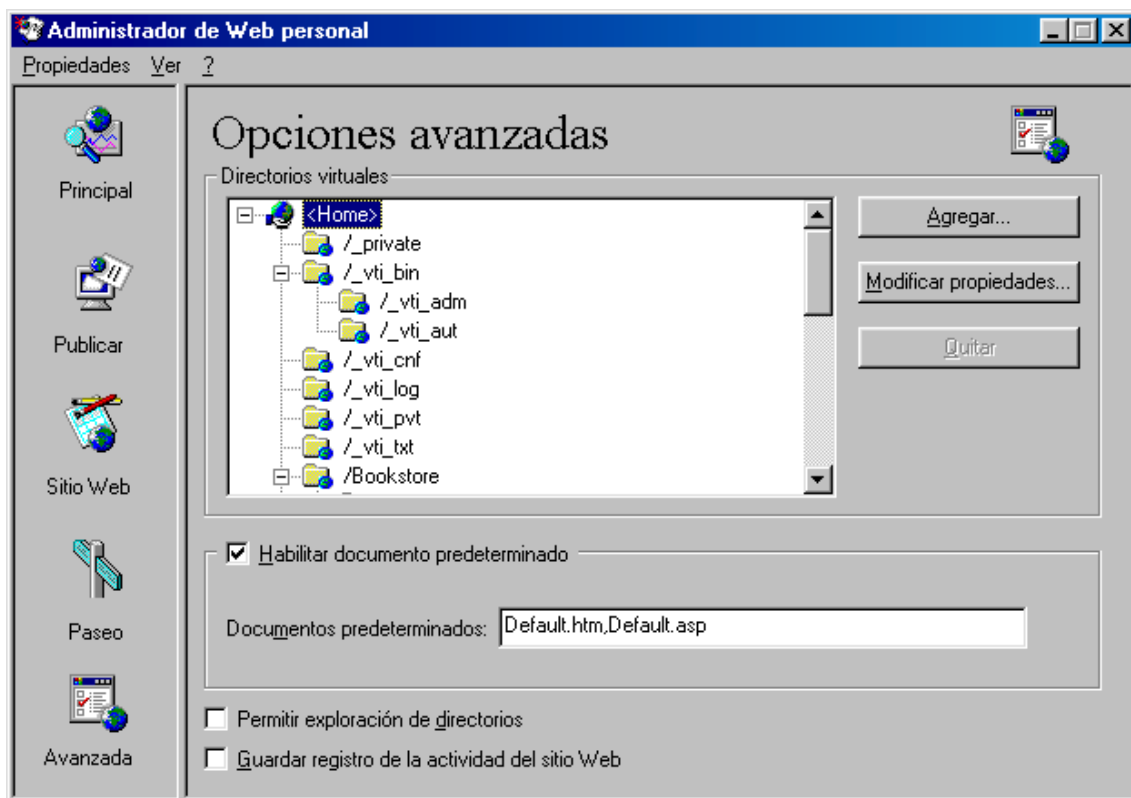


Esta ventana Nos ofrece la opción de detener el servidor web (pulsando el botón de detener), o bien la opción de arrancarlo (pulsando el mismo botón, cuyo texto es modificado por “Iniciar” si PWS se encuentra detenido).

En la parte inferior de esta pantalla, se nos muestran las estadísticas de nuestro site, esto es, las peticiónes o visitas por día o por hora (según el valor que seleccionemos en el cuadro combinado).

Encima del botón de Detener/Iniciar aparece el nombre de nuestro site. Pulsando sobre este link podemos acceder a nuestra página.

Justo debajo del botón de Detener/Iniciar, aparece el directorio local. Este directorio es el directorio de nuestro disco duro que representa el directorio home de nuestra web (en este ejemplo, el directorio C:\Gabriel\Trivial), y puede ser modificado en la pantalla de opciones avanzadas, a la cual accedemos si pulsamos el botón “Avanzada” que aparece en la zona izquierda de la ventana.



Pulsando el botón “modificar propiedades” podremos modificar el directorio real del disco duro sobre el que se mapea el directorio lógico de nuestro site.

En esta misma pantalla, también podremos definir la página por defecto, esto es, la página que se mostrará en caso que el navegador nos pida como dirección la URL de nuestro site, sin especificar el documento concreto (ya sea htm o lasp) a abrir.

Como se puede deducir de todo lo anterior, la estructura lógica del site no tiene porqué coincidir con la estructura física.

En el directorio home es donde debemos tener la página por defecto de nuestro site (normalmente, default.asp), así como el fichero global.asa.

## Bibliografía

### Webs con artículos, tutoriales o manuales sobre ASP:

Debido a que ASP es una plataforma muy popular, potente y fácil de usar, hay muchas webs con artículos, manuales y tutoriales sobre ASP. Algunas de ellas son:

<http://asp.com-e.net>  
<http://www.toptutoriales.com>  
<http://asp.programacion.net>  
<http://members.es.tripod.de/smaug/asp/>

### Webs de alojamiento ASP

Existen también otras webs destinadas al alojamiento de páginas ASP, algunas de ellas, como websamba, totalmente gratis:

[www.websamba.com](http://www.websamba.com)  
[www.digival.es](http://www.digival.es)  
[www.arsys.es](http://www.arsys.es)

### Libros:

#### **Programación de Active Server Pages**

Scot Hillier, Daniel Mezick  
Ed. McGrawHill, Microsoft Press  
ISBN: 84-481-1466-3

#### **Programación Avanzada con Microsoft Visual Basic 6.0.**

Francisco Balena  
Ed. McGrawHill  
ISBN: 84-481-2681-5  
Capítulo 20: Aplicaciones de Internet Information Server

#### **Microsoft Visual Basic 6.0, Manual del Programador**

Ed. McGrawHill, Microsoft Press  
ISBN: 84-481-2062-0

### Otras documentaciones:

MSDN (Microsoft Developer Network): ayuda que Microsoft proporciona a los programadores, disponible en CD y también en internet en <http://msdn.microsoft.com>.

Encontraréis ayuda del lenguaje Visual Basic Script en:

MSDN Library → Platform SDK → Scripting → VBScript

Encontraréis ayuda de la plataforma ASP en:

MSDN Library → Tools and Technologies → Active Server Pages