

***CURSO DE FORMACIÓN CONTINUA***

***ACTIVE SERVER PAGES***

Por: Ing. Carlos JEREZ  
jerezc@ucbca.edu.bo

## INDICE DE CONTENIDOS

<b>1 PREÁMBULO.....</b>	<b>2</b>
1.1 <i>Introducción</i> .....	2
1.2 <i>Aplicaciones de las páginas ASP</i> .....	3
1.3 <i>Requisitos</i> .....	4
<b>2 CONCEPTOS INICIALES.....</b>	<b>4</b>
2.1 <i>Declaración del lenguaje</i> .....	4
2.2 <i>Bloques de código y comentarios</i> .....	4
2.3 <i>Forma de una página ASP</i> .....	5
<b>3 ENTRADA Y SALIDA.....</b>	<b>6</b>
3.1 <i>Response</i> .....	6
3.1.1 <i>Response.Write</i> .....	6
3.1.2 <i>Response.Redirect</i> .....	8
3.2 <i>Request</i> .....	8
3.1.1 <i>Método GET</i> .....	10
3.1.2 <i>Método POST</i> .....	10
<b>4 VARIABLES, OPERADORES Y SENTENCIAS DEL LENGUAJE.....</b>	<b>12</b>
4.1 <i>Variables</i> .....	12
4.2 <i>Operadores Aritméticos</i> .....	13
4.3 <i>Operadores de Comparación</i> .....	14
4.4 <i>Operadores Lógicos</i> .....	14
4.5 <i>Sentencias Condicionales</i> .....	15
4.5.1 <i>Sentencia IF ... ELSE</i> .....	15
4.5.2 <i>Sentencia SELECT</i> .....	16
4.6 <i>Bucles</i> .....	16
4.6.1 <i>Sentencia WHILE</i> .....	17
4.6.2 <i>Sentencia FOR</i> .....	17
<b>5 PROCEDIMIENTOS Y FUNCIONES .....</b>	<b>19</b>
5.1 <i>Procedimientos</i> .....	19
5.2 <i>Funciones</i> .....	19
5.3 <i>Librerías</i> .....	20
<b>6 Bases de datos.....</b>	<b>22</b>
6.1. <i>Declarar el driver de base de datos</i> .....	22
6.2. <i>Realizar operaciones con la base de datos</i> .....	23
6.4. <i>Casos de Estudio</i> .....	30

## INTRODUCCIÓN A ACTIVE SERVER PAGES

# 1 PREÁMBULO

### 1.1 *Introducción*

### 1.2 *Aplicaciones de las páginas ASP*

### 1.3 *Requisitos*

#### 1.1 Introducción

Active Server Pages (ASP) o Páginas de Servidor Activas, es una tecnología creada por Microsoft. Se trata básicamente de un lenguaje de tratamiento de guiones (scripts), basado en Basic, que se denomina VBScript (Visual Basic Script).

Los *scripts* ASP<sup>1</sup> se ejecutan en el servidor antes de enviar la respuesta al navegador y puede utilizarse conjuntamente con HTML<sup>2</sup> y JavaScript<sup>3</sup> para realizar tareas interactivas y en tiempo real con el cliente.

Con ASP se pueden realizar fácilmente páginas de consulta de bases de datos, funciones sencillas como obtener la fecha y la hora actual del servidor, cálculos matemáticos, etc.

ASP es un lenguaje que se ejecuta en el servidor, por esto no es necesario que el cliente o navegador soporte el lenguaje, el proceso básico de tareas realizadas cuando se solicita una página ASP es mostrado en la figura 1.



Fig. 1 Tareas realizadas para ejecutar una página ASP

---

<sup>1</sup> ASP está agrupado en la categoría de lenguajes script o lenguajes de guión.

<sup>2</sup> HTML Hyper Text Markup Language

<sup>3</sup> JavaScript Lenguaje de guión de JAVA

## 1.2 Aplicaciones de las páginas ASP

La facilidad para conectar con una Base de datos y extraer datos de la misma dinámicamente visualizándolos en el navegador es la utilidad más practicada de las páginas ASP.

ASP puede conectarse a gestores de Base de datos SQL<sup>4</sup>, Access, Oracle, o cualquier otro motor que disponga de driver ODBC.

Comercio electrónico, portales, agendas y todas aquellas aplicaciones en las que el protagonista es la información dinámica.

## 1.3 Requisitos

Para procesar una página ASP no existe ninguna restricción especial para el cliente, por lo que es indiferente utilizar *Internet Explorer* o *NetScape Communicator* sin embargo, en el servidor, es necesario un servidor Web que pueda interpretar el código, siendo el servidor más extendido Internet Information Server (más conocido como IIS).

Los servidores que soportan ASP para plataformas *Microsoft* son:

- ? Internet Information Server 3.0 o superior (para sistema operativo NT o Windows 2000)
- ? Personal Web Server (para Windows 95, Windows 98 y Windows 2000)

Para plataformas *Unix* además de el servidor de páginas es necesario añadir un software que actúe como intérprete siendo algunos de los más conocidos:

- ? Chilisoft (<http://www.chilisoft.com/>)
- ? Instant ASP (<http://developer.novell.com/ndk/halcyon.htm>)

---

<sup>4</sup> SQL Structured Query Language

## 2 CONCEPTOS INICIALES

- 2.1 *Declaración del lenguaje*
- 2.2 *Bloques de código y comentarios*
- 2.3 *Forma de una página ASP*

### 2.1 Declaración del lenguaje

Como ocurre en otros lenguajes de programación, en ASP existe una sentencia de declaración *opcional* del lenguaje.

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

Esta declaración se escribe al principio del archivo, antes de cualquier otra expresión.

### 2.2 Bloques de código y comentarios

Para introducir bloques de *sentencias* hay que escribir los símbolos reservados:

```
<% {sentencias} %>
```

Donde *sentencias* pueden ser una o varias expresiones del lenguaje, como se muestra en el siguiente ejemplo:

```
...  
<%  
Request("parametro")  
Response.Write(Time)  
while not condicional do  
    rem do nothing  
loop  
>  
...
```

Las sentencias en VBScript *no se separan por punto y coma (;)*.

Los comentarios de código VBScript se especifican mediante la palabra reservada *rem* o con el carácter comilla simple (') y tienen el ámbito de una línea. Por ejemplo:

```
<%  
rem Esto es un comentario  
' que ocupa varias  
rem líneas  
>
```

Y este es un comentario *mal construido*:

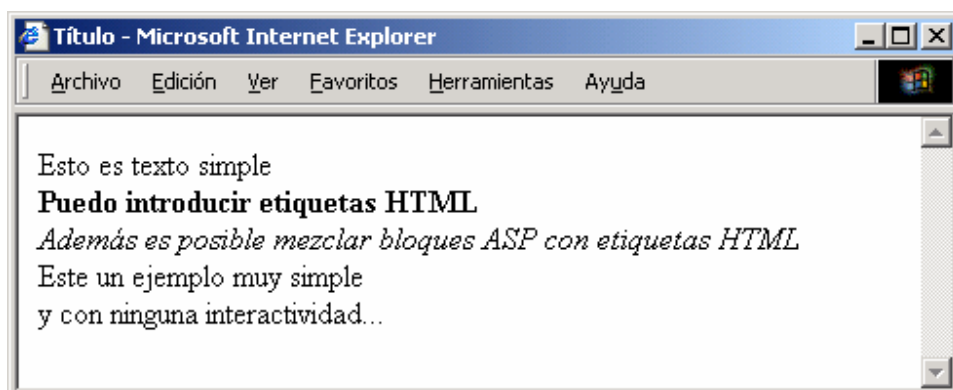
```
<%  
rem Esto es un comentario  
pero es ya no lo es así que el procesador de ASP  
lo interpretará como código, y dará error  
%>
```

## 2.3 Forma de una página ASP

A continuación veremos el formato de un archivo de *texto* que contiene código ASP y que genera como salida, un documento HTML, es decir, una página Web.

```
<%@ LANGUAGE="VBScript" %>  
<%  
rem Declaración de variables y funciones a  
rem realizar antes de visualizar el documento  
rem como por ejemplo, inicializar drivers de  
rem bases de datos, o redireccionar a  
rem otros documentos  
%>  
<HTML>  
<HEAD>  
<TITLE>Título</TITLE>  
</HEAD>  
<BODY>  
<%  
'Este texto se ve en el documento cuando lovisualizamos  
Response.Write("Esto es texto simple <BR>")  
Response.Write("<B>Puedo introducir etiquetas HTML</B><BR>")  
%>  
<I>Además es posible mezclar bloques ASP con etiquetas HTML</I><BR>  
<%  
Response.Write("Este un ejemplo muy simple <BR>")  
Response.Write("y con ninguna interactividad...")  
%>  
</BODY>  
</HTML>
```

En un navegadoreste código se vería de la siguiente manera:



## 3 ENTRADA Y SALIDA

### 3.1 *Response*

#### 3.1.1 *Response.Write*

#### 3.1.2 *Response.Redirect*

### 3.2 *Request*

#### 3.1.1 *Método GET*

#### 3.1.2 *Método POST*

### 3.1 Response

La única manera que tenemos en ASP para producir una salida es usando el objeto *Response* que envía respuestas al documento HTML que se visualizará en el navegador.

#### 3.1.1 *Response.Write*

El objeto *Response* tiene varios métodos y concretamente el método *Write(cadena de texto)* nos permite producir una salida que ira al navegador, la sintaxis es:

```
<%  
Response.Write({cadena})  
%>
```

Una cadena es cualquier combinación de caracteres ASCII, quitando la *comilla doble*. Si queremos que aparezca este símbolo debemos introducirlo dos veces (""). Veamos algunos ejemplos:

```
<%@ LANGUAGE="VBScript" %>  
<%  
Response.Write("<HTML>")  
Response.Write("<HEAD>")  
Response.Write("<TITLE></TITLE>")  
Response.Write("</HEAD>")  
Response.Write("<BODY>")  
Response.Write("Esta página genera todas las etiquetas de un  
documento<BR>")  
Response.Write("HTML normal y corriente...")  
Response.Write("</BODY>")  
Response.Write("</HTML>")  
%>
```

```
<%@ LANGUAGE="VBScript" %>  
<HTML>  
<HEAD>  
<TITLE></TITLE>  
</HEAD>  
<BODY>  
<%  
Response.Write("Esta página genera todas las etiquetas de un  
documento<BR>")  
Response.Write("HTML normal y corriente...")
```

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

Los dos ejemplos anteriores son equivalentes. Si además queremos escribir el valor de alguna variable:

```
<%@ LANGUAGE="VBScript" %>
<%
ho y = Date()
%>
Response.Write("Hoy es:" & hoy)
%>
</BODY>
<HTML>
```

Es de suponer que existe una manera más ágil y abreviada para realizar lo anterior, con menos caracteres:

```
<%@ LANGUAGE="VBScript" %>
<HTML>
<BODY>
<%
Response.Write("Hoy es:" & Date())
%>
</BODY>
<HTML>
```

Existe también otra manera más corta de imprimir el valor de una variable, usando la forma reducida

```
<%=variable%>
```

Que equivaldría a la instrucción

```
<% Response.Write(variable) %>
```

A Continuación mostramos un ejemplo mas completo de esta función.

```
<html>
<head>
  <title>Ejemplo de ASP</title>
</head>
<body>
<%
  Dim tex,num,fecha
  tex="variable de texto"
  num=45345
  fecha=date
  Response.Write("Texto simple<br>")
  Response.Write(tex & "<br>")
  Response.Write("Un numero:" & num & "<br>")
  Response.Write("Una fecha:" & fecha & "<br>")
%>
<%=num%>
</body>
</html>
```



### 3.1.2 *Response.Redirect*

Es útil tener una página que tras un determinado tratamiento de algún dato obtenido del cliente *llame* a otra página, o simplemente, se utilice como método de actualización de enlaces antiguos. En cualquiera de estos casos se utiliza la sentencia *Response.Redirect*:

```
<%@ LANGUAGE="VBScript" %>
<%
rem Este enlace ha quedado obsoleto, redireccionar a...
Response.Redirect("http://www.ucbcba.edu.bo/2001/")
rem Todo lo que hay por debajo de este punto: etiquetas HTML,
rem código ASP no llega a ser interpretado por el procesador de
rem ASP en ningún caso
%>
```

La utilidad de este código queda demostrada si tenemos en cuenta que, con la dinamicidad de la red, frecuentemente se dan modificaciones en las localizaciones de los recursos. Veamos ahora ejemplo de redireccionamiento para tratamiento de datos.

```
<%@ LANGUAGE="VBScript" %>
<%
opcion = 1
Select Case opcion
Case 1: Response.Redirect("pag1.html")
Case 2: Response.Redirect("pag2.html")
Case 3: Response.Redirect("pag3.html")
End Select
%>
```

### 3.2 Request

La sentencia *Request* tiene como misión obtener valores de parámetros que las páginas ASP pueden recibir. La forma de pasar pares atributo-valor (parámetro-valor) es la siguiente:

```
ejemplo.asp?nombre=carlos&apellidos=Agreda%20Vargas&ci=3986887
```

Con los datos introducidos por el cliente (en un formulario, o por otros medios), llamamos a una página de tratamiento de esos datos. Los parámetros que se pasan son:

```
nombre=carlos, apellidos=Agreda Vargas, C.I.=3986887
```

Los caracteres *%20* que muestra el ejemplo entre los apellidos hacen referencia al *carácter espacio* en codificación UTP.

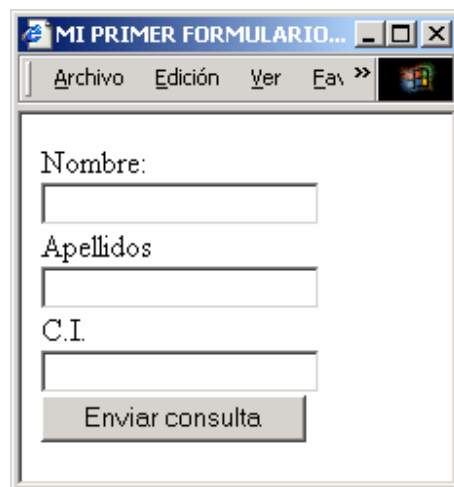
También es posible pasar parámetros a otra página a partir de *campos de un formulario*. En este caso, los nombres de los parámetros vienen dados por los

nombres asignados a dichos campos (fields), y la página que trata o recoge los datos se especifica con el atributo ACTION de la etiqueta FORM.

A continuación mostramos un ejemplo de un formulario que recoge los datos mencionados anteriormente.

```
<%@ LANGUAGE="VBScript" %>
<HTML>
<HEAD>
<TITLE>MI PRIMER FORMULARIO</TITLE>
</HEAD>
<BODY>
<FORM METHOD="GET" ACTION="guardar.asp">
Nombre:
<br><INPUT TYPE="TEXT" NAME="nombre">
<br>Apellidos
<br><INPUT TYPE="TEXT" NAME="apellidos">
<br>C.I.
<br><INPUT TYPE="TEXT" NAME="ci">
<br><INPUT TYPE="SUBMIT">
</FORM>
</BODY>
</HTML>
```

Que da como resultado:

A screenshot of a web browser window. The title bar reads "MI PRIMER FORMULARIO...". The menu bar includes "Archivo", "Edición", "Ver", "Fav", and a right-pointing arrow. The main content area displays a form with three text input fields. The first field is labeled "Nombre:", the second "Apellidos", and the third "C.I.". Below the fields is a button labeled "Enviar consulta".

Para recoger los valores que han sido pasados a través de un formulario, tenemos que usar el objeto del servidor *Request*. Si hemos utilizado el método *GET* usaremos *Request.QueryString("Nombre variable")* y si hemos utilizado el método *POST* usamos *Request.Form("Nombre variable")*.

```
<html>
<head>
  <title>Ejemplo de ASP</title>
</head>
<body>
  <H1>Ejemplo de procesamiento de formularios</H1>
```

```
El nombre que ha introducido es: <%=Request.QueryString("nombre")%>
<br>
</FORM>
</body>
</html>
```

### 3.2.1 Método GET

Los datos de un formulario se envía mediante el método indicado en el atributo *METHOD* de la etiqueta *FORM*, los dos métodos posibles son *GET* y *POST*.

Si usamos el método *GET* los datos son enviados mediante el URL y pueden ser vistos en la barra de direcciones. Para recogerlos deberemos usar *Request.QueryString("Nombre variable")*

```
<html>
<head>
  <title>Ejemplo de ASP</title>
</head>
<body>
<H1>Ejemplo de procesamiento de formularios</H1>

  <FORM ACTION="procesaGET.asp" METHOD="GET">
  Introduzca su nombre:<INPUT TYPE="text" NAME="nombre"><BR>
  Introduzca sus apellidos:<INPUT TYPE="text" NAME="apellidos"><BR>
  <INPUT TYPE="submit" VALUE="Enviar">
  </FORM>
</body>
</html>
```

#### procesaGET.asp

```
<html>
<head>
  <title>Ejemplo de ASP</title>
</head>
<body>
<H1>Ejemplo de procesamiento de formularios</H1>
  El nombre que ha introducido es: <%=Request.QueryString("nombre") &
  " " & Request.QueryString("apellidos") %>
  <br>

</body>
</html>
```

### 3.2.2 Método POST

Ahora usaremos el método *POST*, a diferencia del método *GET* los datos enviados no se ven en la barra del navegador. Para recogerlos debemos usar *Request.Form("Nombre variable")*

```
<html>
<head>
  <title>Ejemplo de ASP</title>
```

```
</head>
<body>
<H1>Ejemplo de procesamiento de formularios</H1>
<FORM ACTION="procesaPOST.asp" METHOD="POST">
Introduzca su nombre:<INPUT TYPE="text" NAME="nombre"><BR>
Introduzca sus apellidos:<INPUT TYPE="text" NAME="apellidos"><BR>
<INPUT TYPE="submit" VALUE="Enviar">
</FORM>
</body>
</html>
```

### procesaPOST.asp

```
<html>
<head>
  <title>Ejemplo de ASP</title>
</head>
<body>
<H1>Ejemplo de procesamiento de formularios</H1>
El nombre que ha introducido es: <%=Request.Form("nombre") & " " &
Request.Form("apellidos") %>
<br>

</body>
</html>
```

## 4 VARIABLES, OPERADORES Y SENTENCIAS DEL LENGUAJE

- 4.1 *Variables*
- 4.2 *Operadores Aritméticos*
- 4.3 *Operadores de Comparación*
- 4.4 *Operadores Lógicos*
- 4.5 *Sentencias Condicionales*
  - 4.5.1 *Sentencia IF ... ELSE*
  - 4.5.2 *Sentencia SELECT*
- 4.6 *Bucles*
  - 4.6.1 *Sentencia WHILE*
  - 4.6.2 *Sentencia FOR*

### 4.1 Variables

Una variable es un contenedor de información, donde podemos almacenar números enteros, números decimales, caracteres, etc. El contenido de las variables puede ser leído y modificado durante la ejecución de una página ASP.

En ASP no es necesario *definir* las variables antes de usarlas. Tampoco se tiene la noción de *tipos*, es decir que una misma variable puede contener un número y luego puede contener caracteres.

```
<html>
<head>
  <title>Ejemplo de VARIABLES</title>
</head>
<body>
<%
  Dim a,b,c `Declaración opcional

  a = 1
  b = 3.34
  c = "Hola Mundo"
  Response.Write(a & "<br>" & b & "<br>" & c)
%>
</body>
</html>
```

En este ejemplo definimos tres variables, a, b y c. Con la instrucción *Response.Write* imprimimos los valores que contienen, insertando un salto de línea entre ellas.

Existen *dos clases* de variables clasificadas por el ámbito de uso, las *variables locales* que sólo pueden ser usadas dentro de *funciones* y las *variables globales* que tienen su ámbito de uso en toda la página ASP.

## 4.2 Operadores Aritméticos

Los operadores de *VBScript* son muy parecidos a los de *Visual Basic* porque el primero procede de este último. Si se conoce *Visual Basic* estos conceptos resultarán familiares y fáciles de reconocer.

Estos son los operadores que se pueden aplicar a las variables y constantes numéricas.

Operador	Nombre	Ejemplo	Descripción
+	Suma	5 + 6	Suma dos números
-	Resta	7 - 9	Resta dos números
*	Multiplicación	6 * 3	Multiplica dos números
/	División	4 / 8	Divide dos números
%	Módulo	7 mod 2	Devuelve el resto de dividir ambos números, en este ejemplo el resultado es 1
^	Exponente	8 ^ 4	Eleva 8 a 4.

A continuación mostramos el uso de los *operadores aritméticos* en una página ASP.

```
<html>
<head>
  <title>Ejemplo de OPERADORES ARITMETICOS</title>
</head>
<body>
<%
  Dim a,b
  a = 8
  b = 3
  Response.Write( a + b & "<br>")
  Response.Write( a - b & "<br>")
  Response.Write( a * b & "<br>")
  Response.Write( a / b & "<br>")
  Response.Write( a mod b & "<br>")
  Response.Write( a ^ b & "<br>")
%>
</body>
</html>
```

### 4.3 Operadores de Comparación

Los operadores de comparación son usados para comparar valores y así tomar decisiones.

Operador	Nombre	Ejemplo	Retorna VERDAD cuando:
=	Igual	a = b	a es igual b
<>	Distinto	a <> b	a es distinto b
<	Menor que	a < b	a es menor que b
>	Mayor que	a > b	a es mayor que b
<=	Menor o igual	a <= b	a es menor o igual que b
>=	Mayor o igual	a >= b	a es mayor o igual que b

A continuación mostramos es uso de los *operadores de comparación* en una página ASP.

```
<html>
<head>
  <title>Ejemplo de OPERADORES DE COMPARACION</title>
</head>
<body>
<%
  Dim a,b
  a = 8
  b = 3
  c = 3
  Response.Write( (a = b) & "<br>" )
  Response.Write( (a <> b) & "<br>" )
  Response.Write( (a < b) & "<br>" )
  Response.Write( (a > b) & "<br>" )
  Response.Write( (a >= c) & "<br>" )
  Response.Write( (b <= c) & "<br>" )
%>
</body>
</html>
```

### 4.4 Operadores Lógicos

Los operadores lógicos son usados para evaluar varias comparaciones.

Operador	Nombre	Ejemplo	Retorna VERDAD cuando:
And	Y	(7 > 2) and (2 < 4)	Devuelve verdadero cuando ambas condiciones son

			verdaderas.
Or	O	$(7 > 2)$ or $(2 < 4)$	Devuelve verdadero cuando al menos una de las dos es verdadera.
Xor	XOR	$(7 > 2)$ xor $(2 < 4)$	Devuelve verdadero cuando sólo una de las dos es verdadera.
Not	No	not $(7 > 2)$	Niega el valor de la expresión.

A continuación mostramos es uso de los *operadores lógicos* en una página ASP.

```
<html>
<head>
  <title>Ejemplo de OPERADORES LOGICOS</title>
</head>
<body>
<%
  Dim a,b,c
  a = 8
  b = 3
  c = 3
  Response.Write( ((a = b) and (c > b)) & "<br>")
  Response.Write( ((a = b) or (b = c)) & "<br>")
  Response.Write( (not (b <= c)) & "<br>")
%>
</body>
</html>
```

## 4.5 Sentencias Condicionales

Las sentencias condicionales nos permiten ejecutar o no ciertas instrucciones dependiendo del resultado de una condición.

### 4.5.1 Sentencia: IF ... ELSE

```
<%
  if condición then
    Sentencias a ejecutar cuando la
    condición es cierta.
  else
    Sentecias a ejecutar cuando la
    condición es falsa.
  end if
%>
```

La sentencia *if* ejecuta una serie de instrucciones dependiendo de la condición impuesta. A continuación un ejemplo.

```
<html>
<head>
```



```
<title>Ejemplo de la sentencia IF ELSE</title>
</head>
<body>
<%
  Dim a,b
  a = 8
  b = 3
  if a < b then
    Response.Write("a es menor que b")
  else
    Response.Write("a no es menor que b")
  end if
%>
</body>
</html>
```

#### 4.5.1 Sentencia: SELECT

```
<%
Select Case opcion
Case 1: Sentencias a ejecutar cuando la opción es 1.
Case 2: Sentencias a ejecutar cuando la opción es 2.
Case 3: Sentencias a ejecutar cuando la opción es 3.
End Select
%>
```

La sentencia *select* ejecuta una serie de instrucciones dependiendo de la opción escogida. A continuación un ejemplo.

```
<html>
<head>
  <title>Ejemplo de la sentencia SELECT</title>
</head>
<body>
<%
  Dim opcion
  opcion = 3
  Select Case opcion
  Case 1: response.write("esta es la opción 1")
  Case 2: response.write("esta es la opción 2")
  Case 3: response.write("esta es la opción 3")
  End Select
%>

</body>
</html>
```

#### 4.6 Bucles

Los bucles nos permiten iterar conjuntos de instrucciones, es decir repetir la ejecución de un conjunto de instrucciones mientras se cumpla una condición.

#### 4.6.1 Sentencia: WHILE

```
<%  
    while condición  
        intrucciones a ejecutar.  
    wend  
>%
```

Mientras la condición sea cierta se reiterará la ejecución de las instrucciones que están dentro de *while*.

En el siguiente ejemplo, el valor de *i* comienza en 0, durante la ejecución del bucle, se suma 1 al valor de *i* de manera que cuando *i* vale 10 ya no se cumple la condición y se termina la ejecución del bucle.

```
<html>  
<head>  
    <title>Ejemplo de sentencia WHILE</title>  
</head>  
<body>  
Inicio<BR>  
<%  
    Dim i  
    i=0  
    while i<10  
        Response.Write("El valor de i es " & i & "<br>")  
        i=i+1  
    wend  
>%  
Fin<BR>  
</body>  
</html>
```

#### 4.6.2 Sentencia: FOR

```
<%  
    for variable=inicial to final  
        intrucciones a ejecutar.  
    next  
>%
```

La instrucción *for* permite indicar un rango de valores válidos para la *variable de iteración*, desde el valor indicado al principio hasta el valor indicado al final, en el siguiente ejemplo *i* varía de 0 a 9.

```
<html>  
<head>  
    <title>Ejemplo de sentencia FOR</title>  
</head>  
<body>  
Inicio<BR>  
<%  
    Dim i  
    for i=0 to 9
```

```
        Response.Write("El valor de i es " & i & "<br>")
    next
%>
Final<BR>
</body>
</html>
```

# 5 PROCEDIMIENTOS Y FUNCIONES

## 5.1 *Procedimientos*

## 5.2 *Funciones*

## 5.3 *Librerías*

El uso de procedimientos y funciones nos brinda la capacidad de agrupar varias instrucciones bajo un solo nombre y poder llamar a estos conjuntos varias veces desde diferentes sitios, ahorrándonos la necesidad de escribir el código nuevamente.

### 5.1 Procedimientos

```
<%  
    sub Nombre(parametro1, parametro2,...)  
        instrucción1;  
        instrucción2;  
        instrucción3;  
        instrucción4;  
    end sub  
%>
```

Para llamar a un *procedimiento* tenemos dos sintaxis distintas:

#### *Sin paréntesis*

```
Nombre parametro1, parametro2...
```

#### *Con paréntesis*

```
call Nombre(parametro1, parametro2)
```

### 5.2 Funciones

Las funciones son iguales que los procedimientos pero nos permiten devolver un resultado.

```
<%  
    function Nombre(parametro1, parametro2,...)  
        instrucción1;  
        instrucción2;  
        instrucción3;  
        instrucción4;  
  
        Nombre = Valor de retorno  
    end function  
%>
```

Para llamar a una función se utiliza la siguiente sintaxis:

```
Nombre(parametro1, parametro2...)
```

A continuación un ejemplo.

```
<html>
<head>
  <title>Ejemplo de FUNCIONES</title>
</head>
<body>
<%

  function media_aritmetica(a, b)
    Dim media
    media=(a + b)/2
    media_aritmetica = media
  end function

  Response.Write(media_aritmetica(4,6) & "<br>")
  Response.Write(media_aritmetica(3242,524543) & "<br>")

%>
</body>
</html>
```

### 5.3 Librerías

El uso de librerías es tremendamente útil, nos permiten agrupar procedimientos, funciones y variables en un mismo fichero, para luego incluir ésta librería en distintas páginas y disponer del código fácilmente.

A continuación definimos la librería *protomartir.asp*

```
<%
  sub CabeceraPagina
%>
  <FONT SIZE=1>Esta cabecera estará en todas sus
  páginas.</FONT><BR>
  <hr>
<%
  end sub

  sub PiePagina
%>
  <hr>
  <FONT SIZE="1"> Autor: Pedro Domingo Murillo
  <BR>Fecha: 16 de Julio de 1809.
  </FONT><BR>

<%
  end sub
%>
```

Ahora vamos a crear 2 páginas que usan la librería definida anteriormente para conseguir que las dos páginas tengan la misma cabecera y pie de página.

La instrucción para incluir una librería en nuestra página es:

```
<!-- #include file="nombre de librería" -->
```

Definición de *pagina1.asp*

```
<html>
<head>
  <title> PAGINA 1 - ejemplo de LIBRERIAS</title>
</head>
<body>
<!-- #include file="protomartir.asp" -->
<% call CabeceraPagina %>

Página 1
<BR><BR><BR><BR><BR>

Hoy es un día fatídico para la nación que recién<BR><BR>
empieza a gestarse...<BR><BR>

fin<BR><BR>

<% call PiePagina %>
</body>
</html>
```

Definición de *pagina2.asp*

```
<html>
<head>
  <title>PAGINA 2 - ejemplo de LIBRERIAS </title>
</head>
<body>
<!-- #include file="protomartir.asp" -->
<% call CabeceraPagina %>

Felicidades La Paz<BR><BR>
por cumplir tu... <BR><BR>

Esta es otra página pero comparte el pie y la cabecera con la
anterior.<BR><BR>

<% call PiePagina %>
</body>
</html>
```

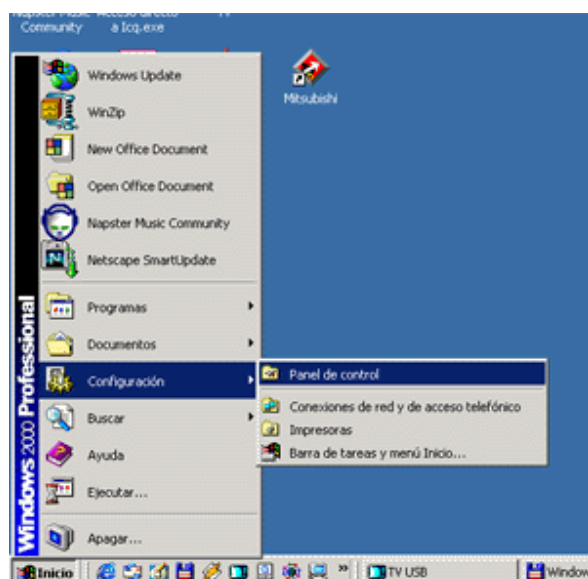
## 6 Bases de datos

- 6.1. Declarar el driver de base de datos
- 6.2. Realizar operaciones con la base de datos
- 6.3. Utilizando SQL
  - 6.3.1 Crear la base de datos
  - 6.3.2 Conexión a la base de datos
  - 6.3.3 Consultas a la base de datos
  - 6.3.4 Inserción de registros
  - 6.3.5 Borrado de registros

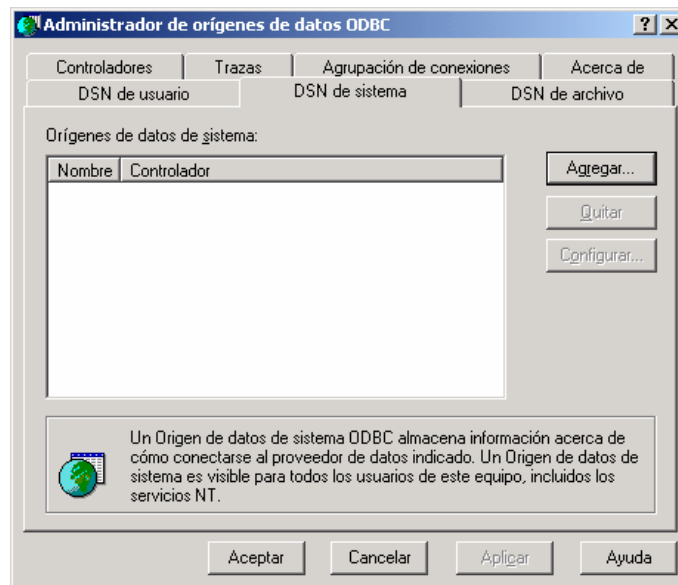
### 6.1. Declarar el *driver* de base de datos

Sin duda alguna, lo más importante que hay que saber respecto al manejo de bases de datos en ASP es la inicialización del *driver* de base de datos y unos ligeros conocimientos en SQL y Access. Existen dos maneras de hacerlo:

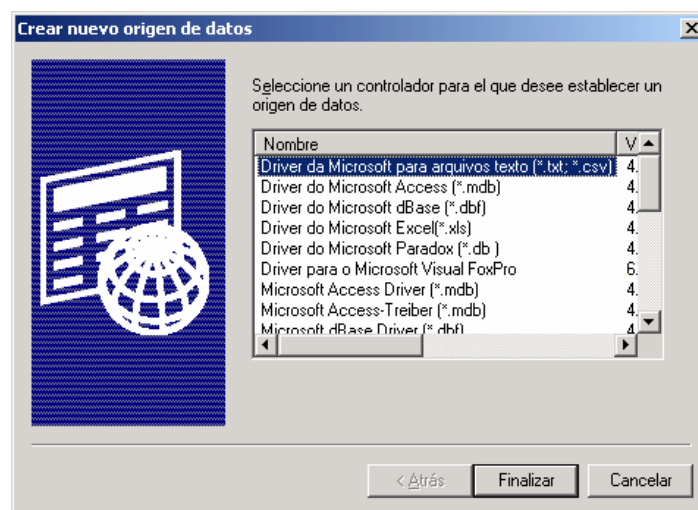
**La primera** consiste en declarar un **DSN de Sistema** a través de **ODBC**. Para ello iremos al botón Inicio -> Configuración -> Panel de Control. Como se ve en la figura.



En la ventana que aparece debemos dirigirnos a ODBC (o a ODBC de 32 bits, según el sistema), y abriremos una nueva ventana con una serie de solapas. Escogemos DSN de sistema.



En este punto vamos a añadir nuestro nuevo DSN para la base de datos que queremos tratar. Para ello seleccionamos en botón *Agregar*. Se abre una ventana que lleva por título *Crear un nuevo origen de datos* en el que se nos muestran los *drivers* de base de datos disponibles en nuestro sistema. Seleccionamos el deseado, en nuestro caso *Microsoft Access Driver* y pulsamos *Finalizar*.



Hecho esto se abre una nueva ventana, de nombre *ODBC Microsoft Access Setup*. En el campo *Nombre de Origen de Datos (Data Source Name)* debemos escribir el *identificador* que emplearemos para la base de datos (si por ejemplo se trata de una base de datos de libros cuyo archivo se llama *biblio.mdb*, podríamos llamarla *libros*). Luego presionamos el botón *Seccecionar (Select)* para seleccionar el archivo de base de datos dentro de la jerarquía de directorios del sistema, y tras esto pulsamos *Ok*. Y ya podremos hacer referencia a ese origen de datos



desde nuestras páginas. Esta primera opción es muy rápida de configurara, sin embargo, es muy frecuente desarrollar las páginas en una máquina y ponerlas en otra (un servidor propiamente dicho), por lo que resulta lioso tener un DSN para cada base de datos del sistema.

La segunda es un poco más pesada, por el hecho de que hay que incluir una serie de **líneas de código** en cada página que haga uso de la base de datos, pero es mucho más flexible, puesto que si cambiamos de sistema, no debemos crear un nuevo DSN.

La declaración del driver debe hacerse antes de que se escriba algo en el documento HTML de salida, y es tan simple como esto:

```
<%@ LANGUAGE="VBScript" %>
<%
  ` Declaramos el objeto de conexión a la base de datos
  Set ConexionBD
  Server.CreateObject("ADODB.Connection")
  ` Abrimos el objeto con el driver específico
  ` ConexionBD.Open "libros"
  ConexionBD.Open "DRIVER={Microsoft Access Driver
  (*.mdb)}; " & "DBQ=" & Server.MapPath("/ruta/bd.mdb")
  %>
<HTML>
...

```

En la sentencia *ConexionBD.Open*, tenemos *Server.MapPath()*, que es una variable que devuelve la ruta local del directorio raíz del servidor Web, y el parámetro que le pasamos hace referencia a la situación de la base de datos dentro de la jerarquía del servidor. Veamos el siguiente ejemplo.

Si tenemos nuestro servidor Web (*http://127.0.0.1* ó *localhost*) en un directorio del sistema denominado *C:\inetpub*, y nuestra base de datos estará en *C:\inetpub\biblioteca\libros.mdb*, en *Server.MapPath* deberemos indicar lo siguiente:

```
...
ConexionBD.Open "DRIVER={Microsoft Access Driver
 (*.mdb)}; " & "DBQ=" &
Server.MapPath("/biblioteca/libros.mdb")
...

```

Atención, cuando nos referimos al sistema de directorios local utilizamos la barra (\) para separar los directorios, pero cuando hacemos referencia al servidor, se separan con el otro tipo de barra (/).

Con esto hemos cumplido con el primer paso, definir el driver para utilizar la base de datos, pero todavía no podemos realizar ninguna consulta ni modificación.

Para ello tenemos que definir el *RecordSet*, que no es más que una agrupación lógica de registros.

## 6.2. Realizar operaciones con la base de datos

Para ver qué es un *RecordSet* y para que sirve volvamos otra vez al ejemplo:

```
<%@ LANGUAGE="VBScript" %>
<%
  ` Declaramos el objeto de conexión a la base de datos
  ` ConexionBD.Open "libros"
Set ConexionBD =
Server.CreateObject("ADODB.Connection")
  ` Abrimos el objeto con el driver específico
ConexionBD.Open "DRIVER={Microsoft Access Driver
(*.mdb)}; " & "DBQ=" &
Server.MapPath("/biblioteca/libros.mdb")
Set RS = ConexionBD.Execute("select * from libros")
%>
<HTML>
...
```

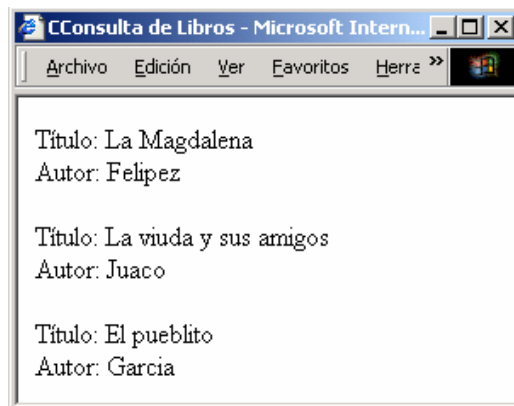
Con esto conseguimos que el objeto RS (RecordSet) esté *enlazado* con el resultado de una consulta de la tabla libros de la base de datos *libros.mdb*.

Pero todavía no tenemos resultados visibles, si ponemos el código de arriba con algunos aditamentos podremos observar los datos de esa base de datos, en el navegador y en tiempo real.

```
<%@ LANGUAGE="VBScript" %>
<%
  ` Declaramos el objeto de conexión a la base de datos
  ` ConexionBD.Open "libros"
Set ConexionBD =
Server.CreateObject("ADODB.Connection")
  ` Abrimos el objeto con el driver específico
ConexionBD.Open "DRIVER={Microsoft Access Driver
(*.mdb)}; " & "DBQ=" &
Server.MapPath("/biblioteca/libros.mdb")
Set RS = ConexionBD.Execute("select * from libros")
%>
<HTML>
<HEAD>
<TITLE>Consulta de Libros</TITLE>
</HEAD>
<BODY>
<%
  ` Como la bd no está vacía hacemos un tratamiento
  hasta que no queden registros...
Do while not RS.EOF
  ` Escribimos en la salida los datos que nos interesa
Response.Write("<P>Título: " & RS("titulo") &
"<BR>")
```

```
Response.Write("Autor: " & RS("autor") & "</P>")
` nos movemos al siguiente registro
RS.MoveNext
Loop
%>
</BODY>
</HTML>
```

Este Código nos da el siguiente resultado:



## 6.3 Utilizando SQL

### 6.3.1 Crear la base de datos

Para crear las tablas en la base de datos, la descripción de las tablas contienen la estructura de la información que almacenaremos en ellas. Para lo cual usaremos en lenguaje de consultas *SQL* común para todas las bases de datos relacionales.

Para estos sencillos ejemplos crearemos una base de datos que hemos llamado datos.mdb, y dentro de esta base de datos una tabla que hemos llamado prueba que tiene la siguiente estructura.

```
prueba.sql
CREATE TABLE prueba
(id_Prueba COUNTER,
Nombre varchar(100),
Apellidos varchar(100))
```

En este ejemplo creamos una tabla llamada prueba con 3 campos: un campo identificador, que nos servirá para identificar unívocamente una fila con el valor de dicho campo, otro campo con el nombre de una persona y por último un campo con el apellido de la persona.

### 6.3.2 Conexión a la base de datos

Una vez que tenemos creada la base de datos en nuestro servidor, el siguiente paso es conectarnos a la misma desde una página ASP.

Para acceder a la base de datos usaremos **ADO** (ActiveX Data Objects), ADO son un conjunto de objetos que nos permiten acceder a la base de datos independientemente del motor de base de datos que usemos, así pues estos ejemplos usan MS Access pero funcionarían igual si el motor de Base de Datos fuese MS SQL Server. Tan solo habría que cambiar el driver.

```
<html>
<head>
  <title>Ejemplo de ASP</title>
</head>
<body>
<%
Dim Conn

Set Conn = Server.CreateObject("ADODB.Connection")

Conn.Open("DRIVER={Microsoft Access Driver (*.mdb)}; DBQ=" &
Server.MapPath("\db\datos.mdb"))

Response.Write("Conexión con la base de datos conseguida.<br>")

Conn.Close
set Conn = nothing
%>
</body>
</html>
```

Para conectarnos a la Base de datos creamos un objeto de *ADO* de tipo conexión para ello usamos *Server.CreateObject*, una vez que tenemos el objeto, le indicamos el *driver* que tiene que usar, en este caso el de *Access* y en donde se encuentra la base de datos.

Para cerrar la conexión con la base de datos usaremos el método *Close* y seguidamente como ya no usaremos el objeto lo destruimos asignándole *nothing*.

### 6.3.3 Consultas a la base de datos

Una vez que nos hemos conectado con el servidor de bases de datos, ya podemos realizar consultas a las tablas de la base de datos.

```
<html>
<head>
  <title>Ejemplo de ASP</title>
</head>
<body>
```

```
<H1>Ejemplo de uso de bases de datos con ASP y ADO</H1>
<%
    Dim Conn, strSQL, RS

    Set Conn = Server.CreateObject("ADODB.Connection")

    Conn.Open("DRIVER={Microsoft Access Driver (*.mdb)}; DBQ=" &
Server.MapPath("\db\datos.mdb"))

    strSQL = "SELECT Nombre, Apellidos FROM prueba"
    Set RS = Conn.Execute(strSQL)

%>
<TABLE BORDER=1 CELLSPACING=1 CELLPADDING=1>
    <TR><TD>Nombre</TD><TD>Apellidos</TD></TR>
<%

    while (not RS.EOF)
        Response.Write("<tr><td>" & RS("Nombre") & "</td><td>" &
RS("Apellidos") & "</td></tr>")
        RS.MoveNext
    wend

    Conn.Close
    set RS = nothing
    set Conn = nothing

%>
</table>
</body>
</html>
```

En este ejemplo hemos ejecutado una consulta *SQL* a la base de datos con el método *Execute* del objeto conexión, esto nos devuelve un objeto de tipo *RecordSet* del cual podemos obtener los datos de la tabla.

El método *EOF* nos permite saber si hemos llegado al final del *RecordSet*, y el método *MoveNext* nos permite avanzar hacia adelante en el *RecordSet*, de esta manera recorreremos todo el *RecordSet* mostrando los datos que este contiene. Y finalmente cerramos la conexión con la base de datos y destruimos el *RecordSet* y el objeto *Connection*.

### 6.3.4 Inserción de registros

Hasta ahora nos hemos conectado a una base de datos y hemos hecho consultas a la misma, ahora presentaremos como introducir nuevo registros en la base de datos.

Para ello usaremos un formulario y en el *ACTION* del *FORM* `<FORM ACTION="programaASP">` indicaremos que debe ser procesado una página ASP, esta página introducirá los datos del formulario en la base de datos.

## DatosInsertar.asp

```
<html>
<head>
  <title>Ejemplo de ASP</title>
</head>
<body>
<H1>Ejemplo de uso de bases de datos con ASP y ADO</H1>
<form action="insertar.asp" method="post">
<TABLE>
<TR>
  <TD>Nombre:</TD>
  <TD><INPUT TYPE="text" NAME="nombre" SIZE="20"
MAXLENGTH="30"></TD>
</TR>
<TR>
  <TD>Apellidos:</TD>
  <TD><INPUT TYPE="text" NAME="apellidos" SIZE="20"
MAXLENGTH="30"></TD>
</TR>
</TABLE>
<INPUT TYPE="submit" NAME="accion" VALUE="Grabar">
</FORM>
</body>
</html>
```

## insertar.asp

```
<%
  Dim Conn,strSQL

  Set Conn = Server.CreateObject("ADODB.Connection")

  Conn.Open("DRIVER={Microsoft Access Driver (*.mdb)}; DBQ=" &
Server.MapPath("\db\datos.mdb"))

  strSQL = "insert into prueba (nombre, apellidos) values (" &
Request.Form("nombre") & ", " & Request.Form("apellidos") & ")"
  Conn.Execute(strSQL)
  Conn.Close
  set Conn = nothing

  Response.Redirect("DatosInsertar.asp")
%>
```

La primera página ASP *DatosInsertar.asp* es un formulario que nos permite introducir nombre y apellido para añadirlo a la base de datos, seguido de una consulta que nos muestra el contenido de la tabla prueba. El formulario llama a la página *insertar.asp* que añadirá los datos a la tabla.

La segunda página *insertar.asp* se conecta a la base de datos y añade un nuevo registro con la instrucción *insert* del lenguaje de base de datos *SQL*. Una vez el registro se ha añadido se vuelve a cargar la página *DatosInsertar.asp*

### 6.3.5 Borrado de registros

Para cerrar el ciclo, nos queda el borrado de registros. El borrado de registros es uno de los procesos más sencillos.

Para indicar que elemento vamos a borrar hemos usado un enlace a la página *borrar.asp* pasándole el *ID\_Prueba* de cada registro, de esta manera la página *borrar.asp* sabe que elemento de la tabla ha de borrar.

#### DatosBorrar.asp

```
<html>
<head>
  <title>Ejemplo de ASP</title>
</head>
<body>
<H1>Ejemplo de uso de bases de datos con ASP y ADO</H1>
<%
  Dim Conn,strSQL, RS

  Set Conn = Server.CreateObject("ADODB.Connection")

  Conn.Open("DRIVER={Microsoft Access Driver (*.mdb)}; DBQ=" &
Server.MapPath("\db\datos.mdb"))

  strSQL = "SELECT * FROM prueba"
  Set RS = Conn.Execute(strSQL)

%>
<TABLE BORDER=1 CELSPACING=1 CELLPADDING=1>
  <TR><TD>Nombre</TD><TD>Apellidos</TD><TD>Borrar</TD></TR>
<%

  while (not RS.EOF)
    Response.Write("<tr><td>" & RS("Nombre") & "<td><td>" &
RS("Apellidos") & "</td><td><a href=" & "borrar.asp?id=" &
RS("id_prueba") & "">borrar</a></td></tr>")
    RS.MoveNext
  wend

  Conn.Close
  set RS = nothing
  set Conn = nothing

%>
</table>
</body>
</html>
```

#### borrar.asp

```
<%
  Dim Conn,strSQL
```

```
Set Conn = Server.CreateObject("ADODB.Connection")

Conn.Open("DRIVER={Microsoft Access Driver (*.mdb)}; DBQ=" &
Server.MapPath("\db\datos.mdb"))

strSQL = "delete from prueba where id_prueba = " &
Request.QueryString("id")
Conn.Execute(strSQL)
Conn.Close
set Conn = nothing

Response.Redirect("EnviarDatosBorrar.asp")
%>
```

La página *borrar.asp* se conecta a la base de datos y borra el registro indicado en la variable *id* que ha sido pasado desde la página *DatosBorrar.asp*. Una vez el registro se ha borrado se vuelve a cargar la página *DatosBorrar..asp*