

**Further
Inside****Scripting Languages****At This
Level****Introduction To VB Script**

Visual Basic Scripting Edition (VBScript) is the default scripting language of Active Server Pages. VBScript extends HTML with variables, operators, loops, conditionals, functions and subroutines.

VBScript is a scripting language. A scripting language is typically easier to use when creating simple programs.

VBScripts can be directly embedded into the HTML files. This allows you to extend HTML into something more than a page-formatting language.

Other scripting languages like JScript, Perl or REXX can also be used with ASP

A client-side programming language is a language that can be interpreted and executed by a browser. When a program written in any of these languages is loaded into a compatible browser, the browser will automatically execute the program. Java and JScript/JavaScript are additional examples of client-side programming language.

The advantage of client-side programming language is that browsers do all the work. This places less of a burden on the server. Client-side programs can also be much faster than server-side programs.

A server-side programming executes on the server that serves a Web site's file, rather than on the browsers. It performs all the work on the Web site's computer and more burdens on the server.

The advantage of using VBScript is that the scripts work regardless of the browser in use. The scripts are processed before the pages are sent out across the Internet to browsers. Web browsers receive nothing more than normal HTML files.

The easiest way to add a script to an ASP is by using script delimiters <% and %>. Any text enclosed within these delimiters will be processed as a script.

Example:

```
<HTML>
<HEAD><TITLE>My ASP</TITLE></HEAD>
<BODY>
My very first
<% FOR i=1 TO 5 %>
first,
<% NEXT %>
page.
</BODY>
</HTML>
```

The two characters <% and %> function very much like the two characters < and > in HTML. Whereas the < and > characters are used to indicate HTML tags, the <% and %> characters are used to indicate scripts. Whenever a Web server sees these two special characters, it attempts to interpret what they contain as a script.

Like this you can mix scripts and HTML freely in an ASP.

You can use the delimiters <%= and %> to print the value of a variable or function.

This example shows how to use <%= and %> delimiters to print a value of a variable.

```
<HTML>
<HEAD><TITLE>My ASP</TITLE></HEAD>
<BODY>
<% FOR var1=1 TO 5 %>
<%=var1%><B> Hello ! </B>
<% NEXT %>
</BODY>
</HTML>
```

This example shows how to use <%= and %> delimiters to print a value of a function.

```
<HTML>
<HEAD><TITLE>My ASP</TITLE></HEAD>
<BODY>
This page was created on : <%= NOW %>
<BR>Have a nice day!
</BODY>
</HTML>
```

You don't need to explicitly declare a variable before you use it in VBScript.

```
<HTML>
<HEAD><TITLE>Variables</TITLE></HEAD>
<BODY>
<%var1="Hello !" %>
<%= var1 %>
</BODY>
</HTML>
```

While creating long or complicated programs, it is good to declare a variable. The advantage of doing this is that it makes the scripts easier to debug.

<% OPTION EXPLICIT %>

```
<HTML>
<HEAD><TITLE>Variables</TITLE></HEAD>
<BODY>
<% DIM var1
var1= "Hello !" %>
<%= var1 %>
</BODY>
</HTML>
```

Here the OPTION EXPLICIT statement forces all variables to be explicitly declared.

Making Comment Lines

To place a comment in a script, you must Write REM or put a Single Inverted Quotes(') at the beginning of the line do it like this:

```
<%
REM This is a VBScript comment.
' This is also a VBScript comment.
%>
```

Both types of comments can only be used to comment a single line.

VBScript Functions –Day function

The NOW function retrieves the current date and time. If you want to retrieve only the current date, you can use the DATE function.

You can break a date into smaller parts by using the MONTH(), DAY(), WEEKDAY() and YEAR() functions. All of these functions take a date expression as an argument. All of these functions return a whole number.

Example:

The Month is: <%=MONTH(DATE)%>

The Day is: <%=DAY(DATE)%>

The weekday is: <%=WEEKDAY(DATE)%>

The year is: <%=YEAR(DATE)%>

Here the WEEKDAY() function will return an integer. By default,

- 1 - is Sunday
- 2 - is Monday
- 3 - is Tuesday
- 4 - is Wednesday
- 5 - is Thursday
- 6 - is Friday and
- 7 - is Saturday

If you want the first day of the week to be Wednesday, you can use the following statement:

The weekday is:

```
<%=WEEKDAY(DATE, vbWednesday)%>
```

You can make the first day of the week any day you want. To specify a particular starting day for the week, substitute vbSunday, vbMonday, vbTuesday, vbWednesday, vbThursday, vbFriday or vbSaturday as the second argument in the WEEKDAY() function

Time Functions - To retrieve the current time

-
If you want to retrieve only the current time, you can use the TIME function.

You can also break the time into smaller parts. You can return parts of times by using the HOUR(), MINUTE() AND SECOND() functions.

Example:

The hour is : <%=HOUR(TIME)%>

The minute is : <%=MINUTE(TIME)%>

The second is : <%=SECOND(TIME)%>

The HOUR() function returns a number between 0 and 23, the MINUTE() function returns a number between 0 and 59 and the SECOND() function returns a number between 0 and 59.

VBScript includes two functions for comparing dates and times.

1. **DATEADD()** - function to add dates and times
2. **DATEDIFF()** - function to calculate the difference between two dates and times

Example:

<%=DATEADD("w", 6, DATE)%>

<%=DATEADD("s", 15, TIME)%>

The DATEADD() function takes three arguments:

1. The first argument specifies an interval of time.
2. The second argument specifies a multiplier for that interval.
3. The third argument is a variant or literal date or time.

Example:

<%=DATEDIFF("d", "1/1/2000", DATE)%>

<%=DATEDIFF("s", DATE, "1/1/2000")%>

The DATEDIFF() functions take three arguments:

1. The first argument is a date or time interval.
2. The final two arguments are two dates. To avoid a negative number, the first date argument listed should be an earlier date than the second.

Date and time intervals:

<u>Interval</u>	<u>Description</u>
yyyy	- Year
q	- Quarter
m	- Month
y	- Day of year
d	- Day
w	- Weekday
ww	- Week of year
h	- Hour
m	- Minute
s	- Second

Math Functions The function includes

- 1 ATN() - for the arctangent
- 2 MOD() - To obtain the remainder (eg. 8 MOD 3 will be giving you the remainder as 2)
- 3 COS() - for the cosine
- 4 SIN() - for the sine
- 5 TAN() - for the tangent
- 6 LOG() - for natural logarithms
- 7 SQR() - for square roots

8	EXP()	- antilogarithms
9	ABS()	- absolute value
10	INT()	- for the nearest integer after rounding
11	FIX()	- returns the nearest integer
12	SGN()	- returns 1, 0 or -1, depending on whether its argument is positive, zero or negative
13	ROUND()	- lets to specify the number of decimal places to round

VBScript includes a number of comparison operators.

1	(=)	The equal operator
2	(<>)	The inequality operator
3	(<) &	The less-than and greater-than operator
4	(<=)	The less-than-or-equal operator
5	(>=)	The greater-than-or-equal operator
6	(NOT)	Negations
7	(AND)	Conjunctions
8	(XOR)	Exclusive disjunctions
9	(OR)	Nonexclusive disjunctions
10	(IMP)	Material implications
11	(EQV)	Material equivalencies

VBScript Control Structures

IF-Then-Else

```
<%
IF condition1 = 10 THEN
    value1 = 10
ELSE
    value1 = 20
END IF
%>
```

Looping

When you need to repeat a group of statements, a set number of times, you can use a FOR...NEXT loop.

```
<%
FOR i=1 to 20 STEP 2
%>
<%=i%><BR>
<%
NEXT
%>
```

VBScript Subroutines

If you need to execute the same group of statements in more than one place in a script, use subroutine. A subroutine can contain any collection of VBScript statements. You can call the same subroutine as many times as you need.

```
<HTML>
<HEAD><TITLE> Subroutine </TITLE></HEAD>
<BODY>
<% SUB show1 %>
This sentence was created by a subroutine
<% END SUB
show1
show1
%>
</BODY>
</HTML>
```

Redirecting a User to Another Page

It is very easy to redirect a user to a new page using Active Server Pages. The Redirect method of the Response object allows you to redirect a user to a new page

```
<%
```

```

IF Request.Form("FirstName") = " " THEN Response.Redirect"/register.asp"
%>
<HTML>
<HEAD><TITLE> Registration Results </TITLE></HEAD>
<BODY>
Thank you <%=Request.Form("FirstName")%> for registering
n</BODY>
n</HTML>

```

You must use the Response, Redirect method before any text is outputted to the browser. Therefore, it's good idea to place this method in a script that appears above the <HTML> tag.

You can use the Response, Redirect method to redirect a user to any valid URL. This could be another page on your Web site or even a page located at another Web site on the Internet

Including Files

You can easily include one file in another Active Server Page by using a server-side INCLUDE directive. A server-side INCLUDE directive shouldn't appear within a script; use it outside a script as part of the HTML code:

```

<HTML>
<HEAD><TITLE>Welcome =/TITLE></HEAD>
<body>
<!--#INCLUDE VIRTUAL = "mybanner...inc"...>
Welcome To Our Web Site
</BODY>
</HTML>

```

ASP Session Object – What are Sessions?

A session is something that starts the moment a user requests a page from your Web site and ends soon after the user leaves. Each visitor to your Web site is given an individual session. Sessions can be used to store a visitor's preferences.

Sessions can also be used to create virtual shopping carts.

Sessions can be used to keep track of the habits of your visitors. Sessions were invented to address a limitation of the HTTP protocol.

All interaction between a browser and a Web server takes place in discrete request and response pairs.

Nothing in the HTTP protocol allows a server to keep track of the users making the requests. After a server has finished responding to a request, the server can't continue to identify the browser that made it. From the perspective of a Web server, each new request is made by a new individual. For this reason, the HTTP protocol is called a stateless protocol.

The HTTP protocol can't be used to retain the state of a user. This is a very serious limitation because it means that you can't identify a user over multiple Web pages.

Sessions were introduced to fix this problem. By using sessions, you can store information about a user over multiple Web pages.

Sessions allow you to do many things that would otherwise be very difficult or completely impossible. You can control all aspects of a session by using the Active Server Pages session object. If you need to store data that will persist throughout a user session, you simply store that data in a collection of the Session object.

The Stateliness of Sessions

Sessions were invented to address a limitation of the HTTP protocol. All interaction between a browser and a Web server takes place in discrete request and response pairs. Nothing in the HTTP protocol allows a server to keep track of the users making the requests. After a server has finished responding to a request, the server can't continue to identify the browser that made it. From the perspective of a Web server, each new request is made by a new individual. For this reason, the HTTP protocol is called a stateless protocol. The HTTP protocol can't be used to retain the state of a user. This is a very serious limitation because it means that you can't identify a user over multiple Web pages. Sessions were introduced to fix this problem. By using sessions, you can store information about a user over multiple Web pages. Sessions allow you to do many things that would otherwise be very difficult or completely impossible.

```

<HTML>
<HEAD><TITLE> Session Example </TITLE></HEAD>
<BODY>
<%
Session("Greeting") = "Welcome"
Response.Write(Session("Greeting"))
%>
<?BODY>
</HTML>

```

For example, imagine that the user requests the following page:

```

<HTML>
<HEAD><TITLE> another Page</TITLE></HEAD>
<BODY>
<%=Session("Greeting")%>
</BODY>
</HTML>

```

The lifetime of a normal variable extends only throughout a single page. A session variable, on the other hand, persists until the user leaves the Web site.

Most session variables are actually stored in a collection of the Session object named contents.

```

<% Session ("MyVar") = "some data"%>
<%session.contents("MyVar") = "Some data"%>

```

As in the collections discussed previously, you can use the Count property to determine the number of items in the contents collection.

You can also display all the items contained in the contents collection by using either a FOR...EACH or a FOR...NEXT loop.

```

<%
Session("FavouriteColor") = "blue"
Session("FavouriteFont")="Comic Sans MS"
%>

```

There are

```

<%=Session.Contents.Count %>
Items in the Session Contents collection
<HR>

```

```

<%
FOR EACH thing IN Session.contents
Response.Write("<HR>")
For I = 1 TO Session.contents.Count
Response.Write("<BR>"&Session.Contents(I))
NEXT
%>

```

Active Server Pages assign each user session a unique identifier. To retrieve the session ID, you use the SessionID property of the Session object.

```

<HTML>
<HEAD><TITLE> session ID </TITLE></HEAD>
<BODY>
Your session ID is:<%=Session.SessionID%>
</BODY>
</HTML>

```

One use for the SessionID property is to track the movements of your visitors

```

<%
Who=Session.SessionID
currentPage=Request.ServerVariables("SCRIPT_NAME")
Response.AppendToLog.who & ":"&CurrentPage
%>

```

This script uses the AppendToLog method of the Response object to append an entry in the server log file.

In this example, the string added to the log file contains the session ID that was retrieved from the sessionID property.

The string also contains the path to the current page, which was retrieved from the SCRIPT_NAME server variable.

How does the server know when a session ends? In other words, how does the server know whether a user has left your Web site for another one or has turned off his or her computer.

The server assumes that if someone hasn't requested or refreshed a page for a period of more than 20 minutes, that person has left, and times out that user's session. This strategy allows the server to recover resources that it has been using to track the user's session.

For certain Web site applications, this timeout period of 20 minutes is too short.

For other Web site applications, the session timeout period of 20 minutes is too short.

For other Web site applications, the session timeout period of 20 minutes is too long.

You can control the maximum amount of time for which a user is allowed to be inactive before a session times out.

For example, the following script sets the Timeout property to 60 minutes

```
<% Session.Timeout=60%>
```

When a user session times out and the user makes a new request, the server treats the user as a new user.

The server creates a new session, and all the old session information is lost. You can force this to happen manually by using the abandon method of the Session object.

```
<HTML>
<HEAD><TITLE> abandon Session </TITLE></HEAD>
<BODY>
<BR> The user is <%=Session.SessionID %>
<%Session.abandon%>
<BR>the user is <%=session.SessionID%>
</BODY>
</HTML>
```

After the Abandon method is called, the server treats the user as a new user.

Session Object's Events

The Session object has events. Two of them, in fact: the **Session_OnStart** event, which is triggered when a session begins, and the **Session_Onend** event, which is triggered when a session ends.

The statements in the script are executed when the event is triggered. Both of these scripts must be located in a special file named **Global.asa**.

Every Web site application can have only one **Global.asa** file. The file is located in the root directory of your Web site application.

The Global.asa file contains information that's global to your Web site application. The Global.asa file has the following structure:

```
<SCRIPT LANGUAGE = VBScript RUNAT=Server>
SUB Application_ONStart
END SUB
</SCRIPT>
<SCRIPT LANGUAGE=VBScript RUNAT =Server>
SUB Application_OnEnd
END SUB
</SCRIPT>

<SCRIPT LANGUAGE=VBScript RUNAT=Server>
SUB Session_Onstart
END SUB
</SCRIPT>
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
SUBSession_Onend
END SUB
</SCRIPT>
```

Notice that Global.asa uses the Microsoft extended HTML <SCRIPT> tag syntax to specify the scripts. You must use this method of indicating a script within the global.asa file instead of using the normal script delimiters <% and %>.

To create script that executes whenever a new session is started, you simply add the script to the session_OnStart section of the global.asa file, as in this example:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
SUBSession_Onstart
Session("UserName")="Unknown"
Session("UserPassword")="Unknown"
END SUB
</SCRIPT>
```

Session Management –*Roll Of Cookies*

ASP Sessions use cookies (see the following section for details on cookies). When a user first requests a page from your Web site, the server creates a single cookie in the user's browser to track the session. When the session ends, the cookies expire as well.

The cookie created for each user is named ASPSESSIONID. The only purpose of this cookie is to provide a unique identifier for each user.

The session ID stored in the ASPSESSIONID cookie is not the same as the sessionId property.

Because cookies aren't compatible with all browsers, you should be cautious in using the Session object when building your Web site.

What are cookies? Browsers that support cookies maintain one or more special files.

These files, called cookie files on windows machines and magic cookie files on the Macintosh, are used to store data from web sites. A Web server can insert pieces of information into these cookie files.

Certain cookies are temporary; others are persistent. For example, the cookies used by Active Server Pages to track user sessions expire after a visitor leaves the 'web site. Other cookies can remain in the cookie files to be read by the server when a user returns. Cookies work on the majority of browsers, but fail completely when used with certain browsers, and this means those sessions will fail as well.

Cookies are passed back and forth between a browser and server through HTTP headers. The server first creates a cookie by using the Set.cookie header in a response. Subsequent requests from the browser will return this cookie in the Cookie header.

To create this cookie, the server would send a header like this:

```
Set.cookie; Username=BILL+Gates; path=/; domain=aspsite.com;
    expires=Tuesday, 01-Jan-99 00:00:01 GMT
```

The browser adds the cookie named UserName with the value Bill Gates.

If the path attribute were set to another value such as /private, the cookie would only be returned in requests to this path.

The domain attribute further restricts where the cookie can be sent by the browser. In this example, the cookie can be sent only to the www.aspitie.com Web site. The cookie will never be sent to www.yahoo.com or any other Web site on the Internet.

The Expiry of the sessions:

The Expires attribute specifies when the cookie should expire.

Once the browser has created a cookie, the browser returns the cookie in every request it makes to the Web site that is, every request that satisfies the path requirement.

The browser continues to send the cookie until the cookie expires.

To create a cookie with Active Server Pages, you use the Cookies collection of the Response object. You can create two types of cookies: a cookie with a single value, or a cookie dictionary, which contains multiple name and value pairs.

To create a cookie with a single value, you can use a script like this:

```
<%
Response.Cookies("UserName")="Bill Gates"
Response.Cookies("UserName").Expires="Jan 1, 1999"
%>
```

Because the example script actually creates a header, you must place the script before any output statements in your active Server Pages file.

The preceding script is a simple example of how you can create a cookie. The example uses only the Expires attribute of the cookies collection. However, the cookies collection has a number of additional attributes.

```
<%
Response.Cookies("UserName")="Steeve Jobs"
Response.Cookies("UserName").Expires="Jan 1, 1999"
Response.Cookies("UserName").Path="/examples"
Response.Cookies("UserName").domain="aspite.com"
Response.Cookies("UserName").Secure=True
%>
```

The path attribute is used to specify more exactly when the browser should send the cookie.

The domain attribute also specifies when the cookie should be sent.

Finally, the secure attribute specifies that the cookie should only be sent in an encrypted transmission.

To read a cookie within an Active Server Page, you use the Cookies collection of the Request object.

```
<%=Request.Cookies("UserName")%>
```

```
<%
FOR EACH thing IN Request.Cookies
Response.write("<BR>&thing&"="&Request.Cookies(thing)
NEXT
%>
```

You can create more than one cookie, by simply creating multiple cookie with the Response.cookies collection as in the previous examples.

An alternative method is available for creating multiple cookies. You can create a cookie dictionary. **A cookie dictionary is actually a single cookie with multiple name and value pairs**

```
<%
Response.Cookies("User")("Name")="Bill Gates"
Response.Cookies("User")("Password")="billions"

%>
```

To determine whether a cookie is a cookie dictionary, Use the HasKkeys attribute. For example, the following script returns True if the cookie is a cookie dictionary and False otherwise.

```
<%=Request.Cookies"User".Haskeys%>
```

ASP Application Object:

When you create a single Active Server Page, you're creating something like a procedure or sub-routine.

When you create a group of related Active Server Pages, you're creating an application.

Following is a list of some features of an active Server Pages application:

- 1) Data can be shared among the pages in an application, and therefore among more than one user of a Web site.
- 2) An application has events that can trigger special application scripts.
- 3) An instance of an object can be shared among all the pages in an application.
- 4) Separate applications can be configured with the Internet Service Manager to have different properties.
- 5) Separate applications can be isolated to execute in their own memory space. This means that if one application crashes, the others won't also crash.
- 6) You can stop one application (unloading all of its components from memory) without affecting other applications.
- 7) A Web site can have more than one application.
- 8) An application is defined by using the Internet Service Manager to specify a root directory for the application.
- 9) An application consists of a particular directory and all of its subdirectories. If one of these subdirectories is also defined to be an application, then it constitutes a separate application.

The application object's all the collections, methods, and events

The application object has all the collections, methods, and events related to applications.

An application variable contains data that can be used on all the pages and by all the users of an application. Application variables can contain any type of data, including arrays and objects. An

application variable differs from a session variable in two ways:

The Comparison Between Session & Application Variables

Unlike a session variable, an application variable doesn't depend on cookies.

Unlike a session variable, the data in an application variable can be shared among multiple users.

An application variable can be used to display transient information on every Web page.

An application variable can be used to record the number of times that a banner advertisement on your Web site has been clicked.

An application variable can hold data retrieved from a database.

An application variable can contain a running count of the number of visitors at your Web site.

An application variable can be used to enable communication between the users of your Web site.

To create a new application variable, you can simply pass the name of the new variable to the Application object, as in the following example:

```
<HTML>
<HEAD><TITLE> Application Example </TITLE></HEAD>
<BODY>
<%
Application("Greeting")="Welcome!"
%>
<%=application["Greeting"]%>
</BODY>
</HTML>
```

Once an application variable has been assigned a value, the value can be displayed on all the pages in an application.

```
<HTML>
<HEAD><TITLE> Another Page </TITLE></HEAD>
<BODY>
<%Application("Greeting")%>
</BODY>
</HTML>
```

****Note:**

It's important to understand that, unlike session variables, application variables don't die when a user leaves. Once an application variable has been assigned a value, it retains that value until the Web server is shut down or the application is unloaded.

Most application variables are actually stored in the **contents collection** of the application object. Whenever you create a new application variable, a new item is added to this collection.

```
<%Application("FavoriteColor")="Blue"%>
<%application.Contents("FavoriteColor")="Blue"%>
```

You can display all the items contained in the Contents collection by using either a FOR...EACH loop or a FOR...NEXT loop.

```
<%
FOR EACH thing IN Application.Contents
Response.Write("<BR>&thing&"="&Application.Contents(thing)
NEXT

%>
```

Like the session object, the application object has two events: application_OnStart and application_Onend. One event is fired when an Active Server Pages application starts; the other event is fired when it ends.

An application doesn't start until the first page is requested from the application

The application_OnEnd event is fired when the Web server is shut down or the application is unloaded

The `Application_Onstart` and `Application_OnEnd` events each trigger one and only one script. Both of these scripts must be located in the `Global.asa` file. These special scripts can't be called from any other Active Server Page.

```
<SCRIPT LANGUAGE = VBScript RUNAT = Server>
SUB Application_OnStart
END SUB
</SCRIPT>
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
SUB Application_OnEnd
END SUB
</SCRIPT>
```

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
SUB Session_OnStart
END SUB
</SCRIPT>
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
SUB Session_OnEnd
END SUB
</SCRIPT>
```

You are very restricted in what you can include in these `Application_OnStart` and `Application_OnEnd` scripts. **You can't place any statements that output content. For example, you can't use HTML or the `Response.Write` method.**

The `application_OnStart` script is valuable for initializing application wide variables. For example, one common use of the `Application_OnStart` and `Session_OnStart` scripts is for tracking the total number of visitors since the application was started. Here's an example of how you can do this:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
SUB Application_OnStart
Application("TotalUsers")=0
END SUB
</SCRIPT>
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
SUB Session-OnStart
Application.Lock
Application("TotalUsers")=application("TotalUsers")+1
Application.Unlock
END SUB
</SCRIPT>
```

After you've modified the `Global.asa` file, you can display the total number of visitors on any Active Server Page by including the following line:

```
<%=Application("TotoalUsers")%>
```

ASP Application

Concurrency: (Lock & Unlock Methods)

If a shared ASP application is running on a web server, the ASP application object has to ensure the real time updation individually from every user who are sharing the particular application. The **Lock** method restricts the concurrent user from accessing the same application until the application updation has not taken place (ie; Until the Application Object has not been set free by **Unlock** Method).

The **Unlock Method** frees the Locked Application Object Activity thus by ensuring the Concurrency of the ASP Application and giving the Updated results for the Concurrent Users.

Example –The Hit Counter Program

```
<%@ LANGUAGE = VBScript %>
<% Option Explicit %>
<%
'Ensure that this page is not cached.
Response.Expires = 0
%>
```

```

<HTML>
<HEAD>
<TITLE>Using Application Variables</TITLE>
</HEAD>
<BODY BGCOLOR="#000000" TOPMARGIN="10" LEFTMARGIN="10">
<FONT SIZE="4" FACE="ARIAL, HELVETICA">
<B>Using Application Variables</B></FONT><BR>
<HR SIZE="1" COLOR="#000000">
<%
'If this is the first time any user has visited
'the page, initialize Application Value.
If (Application("AppPageCountVB") = "") Then
Application("AppPageCountVB") = 0
End If
'Increment the Application AppPageCount by one.
'Note that this AppPageCount value is being
'shared, locking must be used to prevent
'two sessions from simultaneously attempting
'to update the value. However, for a single statement you
'don't need to apply locks (it already is applied.)

Application.Lock
Application("AppPageCountVB") = Application("AppPageCountVB") + 1
Application.Unlock
%>
<!-- Output the Application Page Counter value. -->
<!-- Note that locking does not need to be used -->
<!-- because the value is not being changed by -->
<!-- this user session. --><font color="#FFFF00">
Users have visited this page</font><font color="#FF0000"><b><u>
<%= Application("AppPageCountVB") %> </u></b></font><font
color="#FFFF00">times!</font>
<!-- Provide a link to revisit the page. -->
<P><A HREF="avb.asp"><font color="#00FFFF">Click here to visit it again</font></A>
</BODY>
</HTML>
▲▲▲▲▲▲▲▲▲▲▲▲▲▲▲▲

```

This site is authored and maintained by [M/s Cyber Planet](#) for [Abhinav Vidyalay & Jr. College](#). The said institution provides educational services in the area of primary, secondary & higher secondary education and thus the content is the intellectual property of the staff and students of this institution. No part of it may be reproduced without the consent of the institution.