By: N. Selvakumar
Email: anselva@yahoo.com
Web: www.nselva.com

ASP

1. Introduction to ASP

What is ASP?

- ASP stands for Active Server Pages (ASP 1.0, ASP 2.0, ASP 3.0, ASP+ now renamed ASP.Net)
- ASP is a program that runs inside IIS or PWS. IIS stands for Internet Information Server & PWS stands for Personal Web Server
- IIS comes as a free component with Windows 2000/XP
- IIS is also a part of the Windows NT 4.0 Option Pack. It can be downloaded from Microsoft website.
- PWS is a smaller, but fully functional version of IIS
- PWS can be found on your Windows 95/98 CD

ASP compatibility

- ASP is a Microsoft Technology
- To run IIS you must have Windows NT Server/ 2000/ XP
- To run PWS you must have Windows 95/ 98/ NT Workstation/ Me
- ChiliASP is a technology that runs ASP without Windows OS. (Unix, Linux OS)

What is an ASP file?

- An ASP file has the file extension ".asp"
- An ASP file is just the same as an HTML file
- An ASP file can contain text, HTML tags, client scripts (JavaScript) and server scripts (VBScript)
- ASP scripts (VBScript or JavaScript) are executed on the server.

How does ASP differ from HTML?

- When a browser requests a HTML file, the server returns the file.
- When a browser requests an ASP file, IIS passes the request to the ASP engine. The ASP engine reads the ASP file, line by line, and executes the scripts in the file. Finally, the ASP file is returned to the browser as plain HTML.

What can ASP do for you?

- Dynamically change, delete or add any content of a web page
- Respond to user queries or data submitted from HTML forms
- Access any data or databases and return the results to a browser
- Customize a Web page to make it more useful for individual users
- Provides security since your ASP code can not be viewed from the browser
- Since ASP files are returned as plain HTML, they can be viewed in any browser
- Clever ASP programming can minimize the network traffic

2. Installation (Web Server – IIS)

How to run ASP on your own PC?

You can run ASP on your own PC without an external server. Internet Information Server (IIS) or Microsoft's Personal Web Server (PWS) is normally not pre-installed on your PC.

How to install IIS and run ASP on Windows 2000/ XP?

- Insert the Windows 2000/ XP Professional CD-Rom into your CD-Rom Drive
- From your **Start Button**, go to **Settings**, and **Control Panel**
- In the Control Panel window select **Add/Remove Programs**
- In the Add/Remove window select **Add/Remove Windows Components**
- In the Wizard window check **Internet Information Services**, **click OK**

How to install PWS and run ASP on Windows 95/ 98/ NT Workstation/ Me?

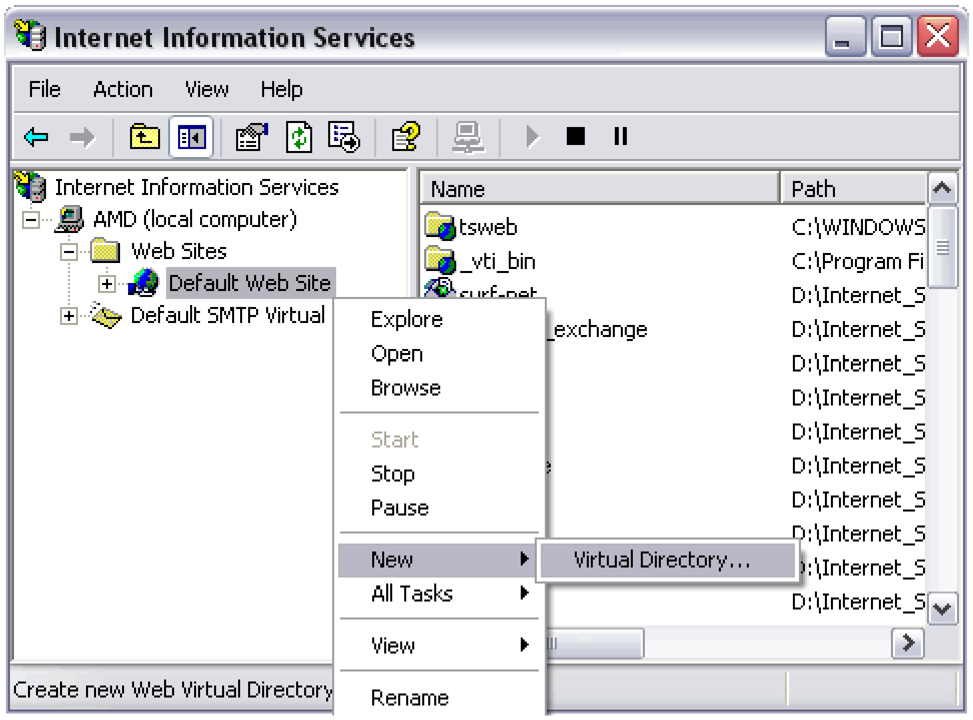
- Open the **Add-ons** folder on your Windows 95/ 98/ NT Workstation/ Me CD, find the **PWS** folder and run the **setup.exe** file.

Once IIS is installed on your machine you can view your home page in a web browser by typing *'http://localhost'* (you can substitute *'localhost'* for the name of your computer or http://127.0.0.1) into the address bar of your web browser.

Note: If you are not sure of the name of your computer right-click on the *'My Computer'* icon on your desktop, select *'Properties'* from the shortcut menu, and click on the *'Computer Name'* tab.

Go to IIS server:
The *'Internet Information Services'* console can be found in the *'Administration Tools'* in the *'Control Panel'*.

Double-click on the *'Internet Information Services'* icon then following screen appear.



To add a new virtual directory right click on *'Default Web Site'* and select *'New'*, followed by *'Virtual Directory'*, from the drop down list.

Next you will see the *'Virtual Directory Creation Wizard'* from the first screen click the *'next'* button.

You will then be asked to type in an *'Alias'* by which you will access the virtual directory from your web browser (this is the name you will type into your web browser after *'localhost'* to view any web pages you place in the directory).

Next you will see a *'Browse...'* button, click on this to select the directory your web site pages are in on your computer, after which click on the *'next'* button to continue.

On the final part of the wizard you will see a series of boxes, if you are not worried about security then select them all, if you are and want to run ASP scripts then check the first two, followed by the *'next'* button.

Once the virtual directory is created you can view the web pages in the folder by typing *'http://localhost/aliasName'* (where *'aliasName'* is, place the alias you called the virtual directory) into the address bar of your web browser (you can substitute *'localhost'* for the name of your computer if you wish)

If *aliasName is test, computer name Selva & asp file name exp.asp*
http://localhost/test/exp.asp or http://selva/test/exp.asp or http://127.0.0.1/test//exp.asp

You can set a default page, right click on particular virtual directory and select properties, select Documents tag. On here click add button and create a default page.

3. ASP Syntax

An ASP file can contain text, HTML tags, client side scripts (eg: JavaScript) and server side scripts (eg: VBScript). Server side scripts in an ASP file are executed on the server. You cannot view server side scripts in a browser; you will only see the output from ASP, which is plain HTML. This is because the scripts are executed on the server before the result is sent to the browser.

The Basic Syntax Rule
ASP file normally contains HTML tags, just as a standard HTML file. In addition, an ASP file can contain server scripts, surrounded by the delimiters <% and %>. Server scripts are executed on the server, and can contain any expressions, statements, procedures, or operators that are valid for the scripting language you use.

How to write text on an ASP code?
The Write method of the ASP Response Object is used to send content to the browser. For example, the following statement sends the text "Hello World" to the browser: Response.Write("Hello World").

Language Independence:
Active Server Pages (ASP) technology is language-independent ASP comes with VBScript and JavaScript. If you want to script in another language, like PERL or Python, you have to install scripting engines for them. Because the scripts are executed on the server, the browser that requests the ASP file does not need to support scripting. Whatever scripting language you use, you can simply enclose script statements in special delimiters for ASP. The starting delimiter is <%, and the closing delimiter is %>.

VBScript
In ASP it is possible to use different scripting languages. The default language in ASP is VBScript, as in this example:

```
<html>
<body>
<%
  Response.write("Hello World!")
%>
</body>
</html>
```

The example above uses the response.write function to write Hello World! into the body of the HTML document.

JavaScript
To use JavaScript as the default scripting language, insert a language specification at the top of the page:

```
<%@ language="javascript" %>
<html>
<body>
<%
  Response.Write("Hello World!")
%>
</body>
</html>
```

Note that - unlike VBScript - JavaScript is case sensitive. You will have to write your ASP code with uppercase letters and lowercase letters when the language requires it.

4. VBScript Variables & Array

Variables are used to store information. Dim is used to declare variables. VBScript variables are variants, which means that they do not have to have a fixed data type. This example demonstrates how to create a variable, assign a value to it, and insert the variable value into a text.

```
dim x
x = 34
x = ”siva”
Dim  strName, intAge
strName = ”Sivakumar”
intAge = 45
```

A variable declared outside a procedure can be accessed and changed by any script in the ASP page in which it is declared. A variable declared inside a procedure is created and destroyed every time the procedure is executed. To make a variable accessible to several ASP pages, declare it either as a session variable or as an application variable.

Meaningful variable names:
String data type variables: strName, strAddress, strFirstName, strMessage, strDescription
Decimal data type variables: dblSalary, dblTax, dblProfit, dblAmount
Integer data type variables: intAge, intYear, intHours
Boolean data type variables: blnStatus, blnFlag

Session Variables
Session variables store information about one single user, and are available to all pages in an application. Common information stored in session variables is username and userid. To create a session variable, store it in a Session Object. (Will see later)

Application Variables
Application variables are also available to all pages in one application. Application variables are used to hold information about all users in a specific application. To create an application variable, store it in an Application Object. (Will see later)

Arrays:
Arrays are used to store a series of related data items. This example demonstrates how you can make an array that stores names.

```
Dim arrayDay(7)
arrayDay(0) = “Sun”
arrayDay(1) = ”Mon”
arrayDay(6) = “Sat”

Dim arrayDay
arrayDay =Array("Sun","Mon","Tues”,"Wednes","Thurs","Fri","Satur")
```

5. VBScript Operators

List of operators supported in VBScript.

Exponentiation - ^, Multiplication - *, Division - /, Integer Division - \, Modulus –Mod, Addition - +, Subtraction (-), String Concatenation - & or +, Comparison Operators (=, <>, <, >, <=, >=), not, and, or

6. VBScript Control Structures

Control structures allow you to control the flow of execution of your scripts. You can specify that some code should be executed only under certain condition, using conditional structures. You can specify that some code should be executed repeatedly, using looping structures. Lastly, you can specify that code from somewhere else in the script should be executed using branching controls.

Conditional Structures

The If…Then…Else construct allows you to choose which block of code to execute based on a condition or series of conditions.

<% If conditions Then codeblock End If %>	<% If conditions Then codeblock1 Else codeblock2 End If %>	<% If conditions Then codeblock1 ElseIf condition2 Then codeblock2 Else codeblock3 End If %>

If condition1 is true, codeblock1 is executed. If it is false, and condition2 is true, codeblock2 is executed. If condition1 and condition2 are both false, codeblock3 executes. An If-Then construct may have zero or more ElseIf statements, and zero or one Else statements.

In place of some really complex If...Then constructs, you can use a Select Case statement. It takes the following form:

```
Select Case Variable
  Case Choice1
   'CodeBlock1
  Case Choice2
    'CodeBlock2
  Case ChoiceN
    'CodeBlockN
  Case Else
    'Default Code Block
End Select
```

This compares the value of variable with choice1, choice2, ... and so on. If it finds a match, it executes the code associated with that choice. If it does not, it executes the default code.

Looping Structures

Looping structures allow you to execute the same block of code repeatedly. The number of times it executes may be fixed or may be based on one or more conditions.
The For…Next looping structure takes the following form:

```
For Counter = Start To Stop
       CodeBlock
Next
```

CodeBlock is executed with Counter having the value Start, then with Counter having the value Start + 1, then Start + 2, and so forth through the value Stop.
Optionally, you may specify a different value to increment Counter by. In this case the form looks like this:

```
For Counter = Start To Stop Step StepValue
       CodeBlock
Next
```

Now counter will take the values Start + StepValue, Start + StepValue + StepValue, and so forth. Notice that if StepValue is negative, Stop should be less than Start.

The Do While-Loop looping structure has the following form:

```
Do While BooleanValue                    While BooleanValue
    CodeBlock                                CodeBlock
Loop                                     wend
```

CodeBlock is executed as long as BooleanValue is True. If it is False to begin with, the loop is not executed at all.

The Do-Loop While looping structure has the following form:

```
Do
    CodeBlock
Loop While BooleanValue
```

CodeBlock is executed as long as BooleanValue is True. The loop is executed at least once no matter what.

7. VBScript Functions

Repeating the same script more than once in a page, then it is a good idea to convert the script into either a function or subroutine. This has the dual benefit of reducing typing and making your code easier to read.

Functions come in two varieties:

- User-defined functions
- Built-in functions.

User-defined functions

- Subroutine
- Function

Function:
You declare a function by using the FUNCTION statement like this:

```
FUNCTION addNumbers( myVar1, myVar2 )
  addNumbers = myVar1 + myVar2
END FUNCTION
```

This function takes two numbers as inputs and returns the sum of the numbers as the output. After you have declared this function anywhere within a page, you can use the function anywhere in the page like this:

```
Response.Write addNumbers(10,13)
```

This statement simply displays the number 23.

Subroutine:
A VBScript subroutine is similar to a VBScript function, except that it usually is not used to return a value. Like functions, subroutines can make your code easier to write and maintain. You declare a subroutine with the SUB statement. Here is a simple example:

```
SUB repeatText( theText, theCount )
  FOR k = 1 TO theCount
    Response.Write theText
  Next
END SUB
```

This subroutine accepts two arguments as inputs. The first argument represents text to be displayed and the second argument represents the number of times the text is displayed. After you have declared a subroutine anywhere within an Active Server Page, you can execute the subroutine by just using its name.

```
<%
        repeatText("Jaffna ",50)
        repeatText("-",50)
        repeatText("*",25)
%>
```

Built-in functions
Quick look at the more important Built in VBScript functions. They include functions for type checking, formatting, math, date manipulation, string manipulation, and more.

Type Checking Functions:
IsNumeric (expression)
    returns a Boolean value of True if the expression is numeric data, and False otherwise.
IsArray (expression)
    returns a Boolean value of True if the expression is an array, and False otherwise.
IsDate(expression)
    returns a Boolean value of True if the expression is date/time data, and False otherwise.
IsEmpty (expression)
    returns a Boolean value of True if the expression is an empty value (uninitialized variable), and False otherwise.
IsNull (expression)
    returns a Boolean value of True if the expression contains no valid data, and False otherwise.

Math Functions:
Abs(number)
    returns the absolute value of number.
Rnd(number)
    returns a random number less than one and greater than or equal to zero.
    If the argument number is less than 0, the same random number is always returned, using number as a seed. If number is greater than zero, or not provided, Rnd generates the next random number in the sequence. If number is 0, Rnd returns the most recently generated number.
Round(number)
    returns number rounded to an integer.
Round(number, dec)
    returns number rounded to dec decimal places.
Sqr(number)
    returns the square root of number, number must be positive.

Date Functions
Date() - returns the current date on the server.
Time() - returns the current time on the server.
Now() - returns the current date and time on the server.

DateAdd(interval, number, date)
    is used to add to the date specified by date. Interval is a string that represents whether you want to add days, months, years, and so on. Number indicates the number of intervals you want to add; that is, the number of days, months, years, and so on.

DateDiff(interval, date1, date2, firstDOW, firstWOY)
    is used to find the time between two dates. DateDiff returns the number of intervals elapsed between date1 and date2. The optional integer firstDOW specifies what day of the week to treat as the first. The optional firstWOY specifies which week of the year to treat as the first.

DateSerial(year, month, day)
    takes the integers year, month, and day and puts them together into a date value. They may be negative.

TimeSerial(hour, minute, second)
  is similar to DateSerial. Timer returns the number of seconds elapsed since midnight.
Year(date)
  returns the year portion from date as a number.
Month(date) – return 0, 1, 2, …. 11
  returns the month portion from date as a number. (0, 1, 2, ….. 11)
MonthName(date)
  returns the month portion from date. Eg: MonthName(1) – return January
Day(date)
  returns the day portion from date as a number.
Weekday(date) – return 0, 1, 2, …. 6
  returns the day of the week of date as a number. (0- sun, 1-mon,…..6-Saturday)
Hour(time)
  returns the hour portion from time.
Minute(time)
  returns the minute portion from time.
Second(time)
  returns the second portion from time.

String Functions

UCase(string)
  returns string with all its lowercase letters converted to uppercase letters.
LCase(string)
  returns string with all its uppercase letters converted to lowercase letters.
LTrim(string)
  removes all the spaces from the left side of string.
RTrim(string)
  removes all the spaces from the right side of string.
Trim(string)
  removes spaces from both the left and the right sides.
Len(string)
  returns the number of characters in string.
Right(string, number)
  returns the number rightmost characters of string.
Left(string, number),
  as you may guess, returns the number leftmost characters of string.
Replace(string, find, replace, start, count, comparetype)
  is used to replace occurrences of find with replace in string
Split(expression,delimiter,count,comparetype)
  takes a string and splits it into an array of strings. Expression is the string to be split up. If expression is zero length. Split returns an array of no elements,
Join(stringarray, delimiter)
  does just the opposite of Split. It takes an array of strings and joins them into one string, using delimiter to separate them. delimiter is optional; the space is the default.

Other functions
LBound(array)
  returns the smallest valid index for array.
UBound(array)
  returns the largest valid index for array.
Asc(string)
  returns the ANSI character code for the first character of string.
Chr(integer)
  returns a string consisting of the character that matches the ANSI character.

8. Using forms in ASP
How to get information from forms? Form validation & on URL passing values.

Forms are a convenient way to communicate with visitors to your Web site. One way that browsers can send specific information to the server is to use a <FORM> element. When they fill out the form, you can process the results automatically. ASP pages can collect the information that the client has typed into the <FORM> elements and then decode and process this information. It is therefore important to assign meaningful values to the attributes of the <FORM> and its elements.

The attributes of the <FORM> element are:

ACTION: is the URL of the CGI (Common Gateway Interface) program that is going to accept the data from the form, process it, and send a response back to the browser.

METHOD: GET (default) or POST specifies which HTTP method will be used to send the form's contents to the web server. The CGI application should be written to accept the data from either method.

NAME: is a form name used by VBScript or by JavaScript.

Tip: A good way to format a FORM and its sub elements is to use a table. With a table, you have more control over the layout of your page. The table can be created inside the <FORM></FORM> tags.

In forms, there are two steps: first, you create the form, and then you process it. To create a form for an Active Server Page, just create a standard HTML form.
To try out this example, create an HTML file ("form_response.html").

```
form_response.html
<html>
<head><title>Asking for information</title></head>
<body>
<form method="post" action="form_response.asp">
      Your name: <input type="text" name="name" size="20"><BR>
      Your email: <input type="password" name="email" size="15"><BR>
      <input type="Submit" value="Submit">
</form>
</body>
```

Active Server Pages provide a mechanism for processing forms that, unlike CGI scripting, doesn't involve serious programming: the Request.Form.
Considering the form above, we may create the file bellow and get a response.

```
form_response.asp
<html>
<head><title>Responding to a form</title></head>
<body>
   Your name is <%=Request.Form("name") %><BR> or Request("name")
   Your email is <%=Request.Form("email") %>
</body>
</html>
```

To display the contents of each field in the form, type:

```
      <%= Request.Form(fieldname) %>
      where fieldname is the name of the field.
```

Form Validation
The form input should be validated on the browser, by client side scripts. Browser validation has a faster response time, and reduces the load on the server.

You should consider using server validation if the input from a form is inserted into a database. A good way to validate the form on a server is to post the form into itself, instead of jumping to a different page. The user will then get the error messages on the same page as the form. This makes it easier to discover the error.

```
        if not result.eof then
                Response.write(“Someone else has already chosen user name<br>”)
                Response.write(“Please go back and try another user name.<br>”)
        end if
```

URL passing values:

```
        <a href=”contents.asp?var1=jaffna&var2=srilanka”>Jaffna</a>
        <a href=”contents.asp?var1=colombo&var2=srilanka”>Colombo</a>
        <a href=”contents.asp?var1=vanni&var2=srilanka”>Vanni</a>
        <a href=”contents.asp?var1=trinco&var2=srilanka”>Trinco</a>
        <a href=”contents.asp?var1=kandy&var2=srilanka”>Kandy</a>
```

9. Using ASP to connect to database (MS Access & SQL 2000)

The power of Active Server Pages comes through when we tie databases to our web site. ASP uses an object library called **A**ctiveX **D**ata **O**bjects, or ADO. ADO can be used to connect to any ODBC-compliant database. ODBC-compliant databases include MS-SQL Server, MS Access, Informix, Oracle, FoxPro, Excel, etc.

Connecting to a Database:

- Using a System **DSN** (Data Source Name)
- Using a **DSN Less** connection

<u>Using a System DSN</u>

1) Create a system DSN using the ODBC Wizard in the Control Panel
2) Connect to the database using a simple connection string: "DSN=*System DSN Name*"

<u>Using a DSN-less connection</u>

1.) Place the Access database file (the *.mdb file) in a directory
2.) The connection string is a bit more complex. You need to specify two things:
The driver and the physical location of the file (eg: C:\WebShare\wwwroot\db\uoct.mdb) or relative path by using Server.MapPath eg: Server.MapPath("db/uoct.mdb")

With native OLE (Object Linking & Embedding) DB Provider (Best way):
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=c:\uoctamil\uoct.mdb

Using ODBC connection without specifying a DSN:
Driver={Microsoft Access Driver (*.mdb)}; DBQ=c:\uoctamil\uoct.mdb

Here is the list of steps we need to follow to connect to a database:

- Create an object to connect to the database
- Provide that object with information regarding the location and type of our database
- Command that object to open a connection with the database

To do the first step, we will use one of ADO's objects, the Connection Object. To create an instance of the connection object in ASP, we need only issue the following commands:

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

To do the second step, Provide that object with information regarding the location and type of our database & Command that object to open a connection with the database.

MS Access

DSN Connection with out User Name & Password:

```
<%
  set conn = Server.CreateObject("ADODB.Connection")
  conn.open "DSNname"
%>
```

DSN Connection with User Name & Password:

```
<%
  set conn = Server.CreateObject("ADODB.Connection")
  conn.open "DSNname","username","password"
%>
```

DSN less Connection

Using physical path as a reference:

```
<%
  Set conn = Server.CreateObject("ADODB.Connection")

  DSNtemp="DRIVER={Microsoft Access Driver (*.mdb)}; DBQ=e:\DataArea\financials.mdb"
  OR
  DSNtemp="DRIVER={Microsoft Access Driver (*.mdb)}; DBQ=" & Server.MapPath("db/uoct.mdb")
  OR
  DSNtemp= "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" & Server.MapPath("db/uoct.mdb")

conn.Open DSNtemp
%>
```

MS SQL Server

DSN Connection with User Name & Password:

```
<%
   set conn = Server.CreateObject("ADODB.Connection")
   conn.open "DSN=MyDSN;UID=user;PWD=password;DATABASE=databasename"
%>
```

DSN Less Connection with User Name & Password:

```
<%
Set conn = Server.CreateObject("ADODB.Connection")
DSNtemp="DRIVER={SQLServer};SERVER=ServerName;UID=USER;PWD=password;DATABASE=da
tabasename"
conn.open DSNtemp
%>
```

Close the Connection to the Datasource

Should be done at the end of each page for each Datasource opened.

```
<%
    conn.close
    set conn = Nothing
%>
```

Working With Recordsets:

In order to read information from a Datasource, you need to open a 'Recordset' - a set of database records based on some type of criteria, either all of the records in a table or those matching some condition or set of conditions. After a recordset is opened, we can extract data from recordset.

To create a recordset containing all of the records in one table of the database, without sorting them:

```
<%
    set objCon = Server.CreateObject("ADODB.Connection")
    objCon.open "DSNname","username","password"

    Set objRec = Server.CreateObject("ADODB.Recordset")
    strSqlList = "SELECT * FROM tablename"
    set objRec= objCon.execute(strSqlList)
%>
```

For example:
To create a recordset containing all of the records in students table of the database with sorting them

```
<%
    set objCon = Server.CreateObject("ADODB.Connection")
    objCon.open "uoctDsn","admin","123"

    Set objRec = Server.CreateObject("ADODB.Recordset")
    strSqlList = "SELECT * FROM students order by stuName"
    set objRec = objCon.execute(strSqlList)
%>
```

To loop through a recordset, writing a field from each record to the web page:

```
<%
' Open the recordset first (see above)
do while not objRec.eof
       response.write(objRec("FieldName") & "<br>")
       objRec.movenext
loop
%>
```

The code used to move around in a recordset is:

```
        objRec.movefirst  ' Move to the first record
        objRec.movelast ' Move to the last record
        objRec.movenext  ' Move to the next record
        objRec.moveprevious  ' Move to the previous record
```

ADO Recordset Object

The Recordset Object is used to hold a set of records from a database table.

Standard SQL queries can be used to update, add or delete record.
For example:

```
<%
strSqlDel = "DELETE * FROM tablename WHERE Name = 'siva'"
objCon.execute(strSqlDel)
%>
<%
strSqlEdit = "UPDATE tablename SET FieldName1 = " & strValue1 & " WHERE FieldName2 = '" & strValue2 & "'"
objCon.execute(strSqlEdit)
%>
```

```
<%
strSqlAdd = "INSERT INTO tablename (FieldName1, FieldName2) VALUES (" & strVal1 & ", " & strVal2 & ")"
objCon.execute(strSqlAdd)
%>
```

10. How to send emails by using ASP?

Many Web-based email providers (as well as ISPs), allow individuals to view their email through the POP3 protocol. This protocol specifies how emails should be retrieved from an email inbox.

ASP provides no mechanism for retrieving emails via the POP3 protocol. Therefore, to access a POP3 email account via an ASP page you will either need to build or buy a POP3 component. But ASP has a facility send an email with the CDONTS Object (To, From, Subject, Cc, Bcc and Body properties)

```
        Set objCDO = Server.CreateObject("CDONTS.NewMail")
        objCDO.From = “anselva@yahoo.com”
        objCDO.To = “anselvas@hotmail.com”
        objCDO.Subject ="Contact form from nselva.com"
        msg=”Name:” & request(“name”) & “<br><br>Comments: & request(“comments”)
        objCDO.Body = msg
        objCDO.BodyFormat = 0 'Body Format is a HTML file format
        objCDO.MailFormat = 0 ' Mail Format Mime
        objCDO.Send
```

11. ASP Session:

The Session object is used to store information about, or change settings for a user session. Variables stored in the Session object hold information about one single user, and are available to all pages in one application. When you are working with an application, you open it, do some changes and then you close it. This is much like a Session. Common information stored in session variables are name, id, and preferences. The server creates a new Session object for each new user, and destroys the Session object when the session expires.

The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are and what you do because the HTTP address doesn't maintain state. ASP solves this problem by creating a unique cookie for each user. The cookie is sent to the client and it contains information that identifies the user. This interface is called the Session object.

When does a Session End?

A session ends if a user has not requested or refreshed a page in the application for a specified period. By default, this is 20 minutes.
If you want to set a timeout interval that is shorter or longer than the default, you can set the Timeout property. The example below sets a timeout interval of 5 minutes:
`<% Session.Timeout=5 %>`
To end a session immediately, you may use the Abandon method:
`<% Session.Abandon %>`

Note: The main problem with sessions is WHEN they should end. We do not know if the user's last request was the final one or not. So we do not know how long we should keep the session "alive". Waiting too long uses up resources on the server. But if the session is deleted too fast you risk that the user is coming back and the server has deleted all the information, so the user has to start all over again. Finding the right timeout interval can be difficult.

Store and Retrieve Session Variables

```
<%
Session("username")="Sivakumar"
Session("age")=36
%>
```
When the value is stored in a session variable it can be reached from ANY page in the ASP application:

```
Welcome <%Response.Write(Session("username"))%>
```
The line above returns: Welcome Sivakumar

You can also store user preferences in the Session object, and then access that preference to choose what page to return to the user.

The example below specifies a text-only version of the page if the user has a low screen resolution:

```
<%If Session("screenres")="low" Then%>
  This is the text version of the page
<%Else%>
  This is the multimedia version of the page
<%End If%>
```

12. ASP Cookies

What is a Cookie?
A cookie can be used to identify a user. It is a small file that the server embeds on the user's computer. Each time the same computer asks for a page through a browser, it will send the cookie too.
With ASP, you can both create cookies and retrieve the value of cookies.

How to Create a Cookie:
The "Response.Cookies" command is used to create a cookie:

```
<%Response.Cookies("firstname")="Siva" %>
```

In the code above, we have created a cookie named "firstname" and assigned the value "Siva" to it.
Note: The Response.Cookies command must appear before the <html> tag.
It is also possible to assign some properties to a cookie, like setting a date when a cookie should expire:

```
<%
Response.Cookies("firstname")="Siva"
Response.Cookies("firstname").Expires=#May 10,2002#
%>
```
Now the cookie named "firstname" has the value of "Siva", and it will expire from the user's computer at May 10, 2002.

How to Get a Cookie Value:
The "Request.Cookies" command is used to get a cookie value.
In the example below, we retrieve the value of the cookie "firstname" and display it on a page:

```
fname=Request.Cookies("firstname")
response.write("Firstname=" & fname)
```
A Cookie with Keys:
A cookie can also contain a collection of multiple values. We say that the cookie has Keys.
In the example below, we will create a cookie-collection named "user". The "user" cookie has Keys that contains information about a user:

```
Response.Cookies("user")("firstname")="Siva"
Response.Cookies("user")("lastname")="Kumar"
```

13. ASP Server Side Includes

The #include directive is used to create functions, headers, footers, or elements that will be reused on multiple pages. You can insert the content of one ASP file into another ASP file before the server executes it, with the #include directive.

```
<p><!--#include file="top.inc"--></p>
<h3>The time is:</h3>
<p><!--#include file="time.inc"--></p>
```

The Virtual Keyword
Use the virtual keyword to indicate a path beginning with a virtual directory.
If a file named "header.inc" resides in a virtual directory named /html, the following line would insert the contents of "header.inc":
`<!-- #include virtual ="/html/header.inc" -->`

The File Keyword
Use the file keyword to indicate a relative path. A relative path begins with the directory that contains the including file.
If you have a file in the html directory, and the file "header.inc" resides in html/headers, the following line would insert "header.inc" in your file:
`<!-- #include file ="headers/header.inc" -->`

14. ASP Application Object

An application on the Web is a group of asp files. The files work together to perform some purpose. The Application object in ASP is used to tie these files together.

The Application object is used to store variables and access variables from any page, just like the Session object. The difference is that all users share ONE Application object, while with Sessions there is one Session object for each user.

Like Session variables, Application variables are created in your Web server's memory and they persist across multiple pages. However, unlike Session variables, Application variables are not tied to a particular user. Once created, Application variable remains in memory until it is explicitly removed or your Web server crashes.

Unlike Session variables, Application variables can be read and modified by different users at the same time. Furthermore, unlike Session variables, Application variables do not rely on cookies.

The Application object should hold information that will be used by many pages in the application. This means that you can access the information from any page. It also means that you can change the information in one place and the new information will automatically be reflected on all pages.

Store and Retrieve Variable Values
Application variables can be accessed and changed by any page in the application.
You can create Application variables in Global.asa like this:

```
Sub Application_OnStart
    application("vartime")=""
    application("whoon")=1
End Sub
```

In the example above we have created two Application variables: The first is named vartime, and the second is named whoon.

You can access the value of an Application variable like this:

```
There are <% Response.Write(Application("whoon")) %> active connections.
```

The Application object should hold information that will be used by many pages in the application (like database connection information). This means that you can access the information from any page. It also means that you can change the information in one place and the new information will automatically be reflected on all pages.

15. ASP Global.asa

The Global.asa file is an optional file that can contain declarations of objects, variables, and methods that can be accessed by every page in an ASP application.
All valid browser scripts (JavaScript, VBScript, JScript, PerlScript, etc.) can be used within Global.asa.

Global.asa files can contain only the following:
Application events
Session events
<object> declarations
TypeLibrary declarations

Note: The Global.asa file must be stored in the root directory of the ASP application, and each application can only have one Global.asa file. Changes to the Global.asa file require a reboot for the server to recognize them.

Standard Events in Global.asa
In the Global.asa file you can tell the application and session objects what to do when the application/session starts and what to do when the application/session ends. The code for this is placed into event handlers. The Global.asa file can contain four types of events:

Application_OnStart - This event occurs when the FIRST user calls the first page from an ASP application. This event occurs after the Web server is restarted or after the Global.asa file is edited. When this procedure is complete, the "Session_OnStart" procedure runs.

Session_OnStart - This event occurs EVERY time a new user requests the first page in the ASP application.

Session_OnEnd - This event occurs EVERY time a user ends a session. A user ends a session after a page has not been requested by the user for a specified time (by default this is 20 minutes). A session also ends if the user closes the web browser, or goes to someone else's web page.

Application_OnEnd - This event occurs after the LAST user has ended the session. Typically, this event occurs when a Web server stops. This procedure is used to clean up settings after the Application stops, like delete records or write information to text files.

Global.asa file (This is a common file for all pages)

```
<script language="vbscript" runat = "server">
Sub Application_OnStart
   'initialize variable
   Application("visitors_online") = 1
End Sub
Sub Session_OnStart
   Session.Timeout = 30 '20 minute timeout
   Application.Lock
   Application("visitors_online") = Application("visitors_online") + 1
   Application.Unlock
End Sub
Sub Session_OnEnd
```

```
    Application.Lock
    Application("visitors_online") = Application("visitors_online") - 1
    Application.Unlock
End Sub
</script>
```

index.asp page
There are currently <b><%=Application("visitors_online")%></b> visitor(s)

Note: We do not use the ASP script delimiters, <% and %>, to insert scripts in the Global.asa file, we have to put the subroutines inside the HTML <script> element.

16. ASP Objects

Built-In and Installable Objects:
Most of the functionality you can build into an ASP page comes from objects on the server. IIS 4.0 comes with some built-in objects, as well as a number of installable objects. You can also use objects created by a developer you know, or create and use your own objects.

Built-In Objects:
Five objects come built-in with IIS 4.0. These objects enable your pages to communicate effectively with the server, the application, the current session, and the user:

- Response object -- Sends information to the user. You can send text to the page, redirect to another URL, and set cookies. (main methods – write, end, redirect)
- Request object -- Gets information from the user. You can get FORM data, read cookies, … (main methods – form, querystring, servervariables, cookies)

  Your IP address is: <%Response.Write(Request.ServerVariables("remote_addr"))%>
  The server's domain name:<%Response.Write(Request.ServerVariables("server_name"))%>

- Server object -- Interacts with the server. You can access a database, read files, and find out about the capabilities of the browser.

  CreateObject - Creates an instance of an object. Eg: Server.CreateObject("ADODB.Connection")

  MapPath - Maps a specified path to a physical path. Eg: Server.MapPath("db/uoct.mdb")

  Execute - Executes an ASP file from inside another ASP file. Eg: Server.execute(“1.asp”)

  ScriptTimeout - Sets or returns the maximum number of seconds a script can run before it is terminated (Session.Timeout[=nMinutes]). Eg: Server.ScriptTimeout=200

  URLEncode - Applies URL encoding rules to a specified string. Eg: Server.URLEncode(“1 2.asp”)

  HTMLEncode- Applies HTML encoding to a specified string. Eg: Server.URLEncode(“we se”)

- Session object -- Enables you to manage information about the current session (each user has one session per open browser). – main method – timeout, abandon
- Application object -- Can store information for all sessions.