University of Technology
الجامعة التكنولوجية

Computer Science Department
قسم علوم الحاسوب

Dynamic Web Programming
برمجة مواقع متغيرة

Dr. Shatha Habeeb

Dr. Athraa Jasim Mohammed

Dr. Muna Ghazi

cs.uotechnology.edu.iq

Dynamic Web programming

Introduction

A server-side dynamic web page is a web page whose construction is controlled by an application server processing server-side scripts. In server-side scripting, parameters determine how the assembly of every new web page proceeds, including the setting up of more client-side processing.

A client-side dynamic web page processes the web page using HTML scripting running in the browser as it loads. JavaScript and other scripting languages determine the way the HTML in the received page is parsed into the Document Object Model, or DOM, that represents the loaded web page.

A dynamic web page is then reloaded by the user or by a computer program to change some variable content. The updating information could come from the server, or from changes made to that page's, for its client, from an application server.

A program and web application is many things. It is a piece of text typed by a programmer, it is the directing force that makes the computer do what it does, it is data in the computer's memory, yet it controls the actions performed on this same memory. Analogies that try to compare programs to objects we are familiar with tend to fall short. A superficially fitting one is that of a machine—lots of separate parts tend to be involved and to make the whole thing tick, we have to consider the ways in which these parts interconnect and contribute to the operation of the whole. A computer is a physical machine that acts as a host for these immaterial machines. Computers themselves can do only stupidly straightforward things. The reason they are so useful is that they do these things at an incredibly high speed. A program can ingeniously combine an enormous number of these simple actions to do very complicated things. A program is a building of thought.

JavaScript

There are those who will say terrible things about JavaScript. Many of these things are true. When I was required to write something in JavaScript for the first time, I quickly came to despise it. It would accept almost anything I typed but interpret it in a way that was completely different from what I meant. This had a lot to do with the fact that I did not have a clue what I was doing, of course, but there is a real issue here: JavaScript is ridiculously liberal in what it allows. The idea behind this design was that it would make programming in JavaScript easier for beginners. In actuality, it mostly

makes finding problems in your programs harder because the system will not point them out to you.

JavaScript Functions

A function is a piece of program wrapped in a value. Such values can be applied in order to run the wrapper program. For example, in a browser environment, the binding prompt holds a function that shows a little dialog box asking for user input. It is used like this:

prompt("Enter pass code") ; Executing a function is called invoking, calling, or applying it. You can call a function by putting parentheses after an expression that produces a function value. Usually, you'll directly use the name of the binding that holds the function. The values between the parentheses are given to the program inside the function. In the example, the prompt function uses the string that we give it as the text to show in the dialog box. Values given to functions are called arguments. Different functions might need a different number or different types of arguments.

The prompt function isn't used much in modern web programming, mostly because you have no control over the way the resulting dialog looks but can be helpful in toy programs and experiments.

A function will be executed by an event or by a call to the function. To keep the browser from executing a script when the page loads, you can put your script into a function.

A function contains code that will be executed by an event or by a call to the function. You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file).

Functions can be defined both in the <**head**> and in the <**body**> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

How to Define a Function

Syntax

```
function function-name(var1,var2,...,varX)
{
some code
}
```

The parameters var1, var2, etc. are variables or values passed into the function. The {and the} defines the start and end of the function.

Note:

- A function with no parameters must include the parentheses () after the function name.
- Do not forget about the importance of capitals in JavaScript! The word function must be written in lowercase letters, otherwise a JavaScript error occurs! Also note that you must call a function with the exact same capitals as in the function name.

Example

```
1-<head>
<script type="text/javascript">
function displaymessage()
{
alert("Hello World!");   }
</script>
</head>
<body>
<form>
<input type="button" value="Click me!" onclick="displaymessage()" />
</form>
</body>

2- <"script type = "text/javascript>

 //Function definition
function welcomeMsg(name } (
document.write("Hello " + name + " welcome to GeeksforGeeks");
 {
 //creating a variable
var nameVal = "Admin";
 //calling the function
welcomeMsg(nameVal);
</script>
```

The Return Statement

The return statement is used to specify the value that is returned from the function. So, functions that are going to return a value must use the return statement. The example below returns the product of two numbers (a and b):

Example

```
<html>
<head>
<script type="text/javascript">
function product(a,b)
{
return a*b;
}
</script>
</head>
<body>
<script type="text/javascript">
document.write(product(4,3));
</script></body> </html>
```

The Lifetime of JavaScript Variables

If you declare a variable within a function, the variable can only be accessed within that function. When you exit the function, the variable is destroyed. These variables are called local variables. You can have local variables with the same name in different functions, because each is recognized only by the function in which it is declared.

If you declare a variable outside a function, all the functions on your page can access it. The lifetime of these variables starts when they are <html>

Example

```
<head>
<script type="text/javascript">
Function myfunction(txt)
{ alert(txt);}
</script>  </head>  <body>
<form>
<input type="button" onclick="myfunction('Hello')" value="Call function">
</form>
<p>By pressing the button above, a function will be called with "Hello" as a parameter.
The function will alert the parameter.</p>
</body> </html>
```

The HTML specification refers to these as intrinsic events and defines 18 as listed below:

Event Handler	Event that it handles
onChange	User has changed the object, then attempts to leave that field (i.e. clicks elsewhere).
onClick	User clicked on the object.
onDblClick	User clicked twice on the object.
onKeydown	A key was pressed over an element.
onKeyup	A key was released over an element.
onKeypress	A key was pressed over an element then released.
onLoad	The object has loaded.
onMousedown	The cursor moved over the object and mouse/pointing device was pressed down.
onMouseup	The mouse/pointing device was released after being pressed down.
onMouseover	The cursor moved over the object (i.e. user hovers the mouse over the object).
onMousemove	The cursor moved while hovering over an object.
onMouseout	The cursor moved off the object
onReset	User has reset a form.
onSelect	User selected some or all of the contents of the object. For example, the user selected some text within a text field.
onSubmit	User submitted a form.
onUnload	User left the window (i.e. user closes the browser window).

Array

JavaScript provides a data type specifically for storing sequences of values. It is called an *array* and is written as a list of values between square brackets, separated by commas. An array is a special variable, which can hold more than one value, at a time. If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
1- let listOfNumbers = [2, 3, 5, 7, 11];
2- cars1="Saab";
   cars2="Volvo";
   cars3="BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The best solution here is to use an array! An array can hold all your variable values under a single name. And you can access the values by referring to the array name. Each element in the array has its own ID so that it can be easily accessed.

Create an Array

An array can be defined in three ways. The following code creates an Array object called myCars

1:

```
var myCars=new Array(); // regular array (add an optional integer
myCars[0]="Saab"; // argument to control array's size)
myCars[1]="Volvo";
myCars[2]="BMW";
```

2:

```
var myCars=new Array("Saab","Volvo","BMW"); // condensed array
```

3:

```
var myCars=["Saab","Volvo","BMW"]; // literal array
```

Note: If you specify numbers or true/false values inside the array then the variable type will be Number or Boolean, instead of String.

Access an Array

You can refer to a particular element in an array by referring to the name of the array and the index number. The index number starts at 0. The following code line:

```
document.write(myCars[0]);
```

will result in the following output: Saab

Modify Values in an Array

To modify a value in an existing array, just add a new value to the array with a specified index number:

```
myCars[0]="Opel";
```

Now, the following code line:

```
document.write(myCars[0]);
```

will result in the following output: Opel

Arrays - concat()

```
var parents = ["Jani", "Tove"];
var children = ["Cecilie", "Lone"];
var family = parents.concat(children);
document.write(family);
```

Insert Special Characters

The backslash (\) is used to insert apostrophes, new lines, quotes, and other special characters into a text string.

Look at the following JavaScript code:

```
var txt="We are the so-called "Vikings" from the north.";
document.write(txt);
```

In JavaScript, a string is started and stopped with either single or double quotes. This means that the string above will be chopped to: We are the so-called

To solve this problem, you must place a backslash (\) before each double quote in "Viking". This turns each double quote into a string literal:

```
var txt="We are the so-called \"Vikings\" from the north.";
document.write(txt);
```

JavaScript will now output the proper text string: We are the so-called "Vikings" from the north.

Here is another example:

```
document.write ("You \& I are student");
```

The example above will produce the following output:

You & I are students!

The table below lists other special characters that can be added to a text string with the backslash sign:

Code	Outputs
\'	single quote
\"	Double quote
\&	Ampersand
\\	Backslash

\n	**new line**
\b	**Backspace**

String

We can read properties like length and toUpperCase from string values.

1- Return the length of a string

```
var txt = "Hello World!";  document.write(txt.length);
```

2- Style strings

```
var txt = "Hello World!";

document.write("<p>Big: " + txt.big() + "</p>");

document.write("<p>Bold: " + txt.bold() + "</p>");

document.write("<p>Italic: " + txt.italics() + "</p>");
```

3- The toLowerCase() and toUpperCase() methods

```
var txt="Hello World!";

document.write(txt.toLowerCase() + "<br />");

document.write(txt.toUpperCase());
```

4- The match() method

The match() method searches a string for a match against a regular expression, and returns the matches, the method will return only the first match in the string.

```
var str="Hello world!";

document.write(str.match("world")+"<br />");//output: world

document.write(str.match("World"));// output :Null
```

5- Replace characters in a string - replace()

```
var str="Visit Microsoft!";

document.write(str.replace("Microsoft"," Schools"));
```

6- The indexOf() method:

Return the position of the first found occurrence of a specified value in a string.

- This method is used to return the position of the first occurrence of a specified value inside a string variable.
- The return value of this method is -1 in case the specified value is not found.
- The indexOf() method is case sensitive.

```
var str="Hello world!";

document.write(str.indexOf("d") + "<br />");//10

document.write(str.indexOf("world")); //6

var beasts = ['ant', 'bison', 'camel', 'duck', 'bison'];

Document.write(beasts.indexOf('bison')); // expected output: 1

Document.write(beasts.indexOf('giraffe'));// expected output:-1
```

Create a Date Object

Objects and arrays (which are a specific kind of object) provide ways to group several values into a single value. Conceptually, this allows us to put a bunch of related things in a bag and run around with the bag, instead of wrapping our arms around all of the individual things and trying to hold on to them separately.

The Date object is used to work with dates and times. Date objects are created with the Date() constructor.

```
new Date() // current date and time
```

Once a Date object is created, a number of methods allow you to operate on it. Most methods allow you to get and set the year, month, day, hour, minute, second, and milliseconds of the object..

Some examples of instantiating a date:

```
today = new Date()
d1 = new Date("October 13, 1975 11:13:00")
d2 = new Date(79,5,24)
d3 = new Date(79,5,24,11,33,0)
```

Set and Get Dates

We can easily manipulate the date by using the methods available for the Date object. In the example below we set a Date object to a specific date (14th January 2010):

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
```

in the following example we set a Date object to be 5 days into the future:

```
var myDate=new Date();
myDate.setDate(myDate.getDate()+5);
```

Note: If adding five days to a date shifts the month or year, the changes are handled automatically by the Date object itself!

Compare Two Dates

The Date object is also used to compare two dates. The following example compares today's date with the 14th January 2010:

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
var today = new Date();
if (myDate>today)
 {
 alert("Today is before 14th January 2010");
 }
else {
 alert("Today is after 14th January 2010");
 }
```

Method of object set and get date

Where get is used to retrieve a specific component from a date, set is used to modify components of a date.

Function	Description	Returned Values
getDate()	Day of the month	1-31
getDay()	Day of the week (integer)	0-6
getFullYear()	Year (full four digit)	1900+
getHours()	Hour of the day (integer)	0-23
getMinutes()	Minutes (since last hour)	0-59
getMonth()	Month	0-11
getSeconds()	Seconds (since last minute)	0-59
getYear()	Year	0-99 for years
setDate()	Sets the day, given a number between 1-31	Date in milliseconds
setFullYear()	Sets the year, given a four digit number	Date in milliseconds
setHours()	Sets the hour, given a number between 0-23	Date in milliseconds
setMinutes()	Sets the minutes, given a number between 0-59	Date in milliseconds
setMonth()	Sets the month, given a number between 0-11	Date in milliseconds
setSeconds()	Sets the seconds,l given a number between 0-59	Date in milliseconds
setYear()	Sets the year, given either a two digit or four digit number	Date in milliseconds

JavaScript Math Object

round() Parameters: This function accepts a single parameter, var. It is the number which you want to **round** off. Return Value: The Math.**round()** function returns the value of the given number **rounded** to the nearest integer.

**document.write(Math.round(0.60) + "
");**

random()

The Math.random() function is used to return a floating-point random number between range [0,1) , 0 (inclusive) and 1 (exclusive).This random number can then be scaled according to the desired range

```
document.write(Math.random() + "<br />");

//return a random integer between 0 and 10

document.write(Math.floor(Math.random()*11));
```

max()

use max() to return the number with the highest value of two specified numbers.

```
document.write(Math.max(0,150,30,20,38) + "<br />");

document.write(Math.max(-5,10) + "<br />");
```

min()

use min() to return the number with the lowest value of two specified numbers.

```
document.write(Math.min(-5,-10) + "<br />");
document.write(Math.min(1.5,2.5));
```

charAt

charAt() gives you the character at a certain position. For instance, when you do

```
var b = 'I am a JavaScript hacker.'
document.write(b.charAt(5))
```

Split

split() is a specialized method that you need sometimes. It allows you to split a string at the places of a certain character. You must put the result in an array, not in a simple variable.

Note split() does not work in Netscape 2 and Explorer 3.

Let's split b on the spaces

```
var b = 'I am a JavaScript hacker.'
var temp = new Array();
temp = b.split(' ');
```

Now the string has been split into 5 strings that are placed in the array temp. The **spaces themselves are gone.**

```
temp[0] = 'I';
temp[1] = 'am';
temp[2] = 'a';
temp[3] = 'JavaScript';
temp[4] = 'hacker.';
```

String Search Function

The search() method searches a string for a specified value and returns the position of the match. If there is a match, it will return the position in the string where the match was found and returns -1 if no match is found. The search value can be a string or a regular expression.

Search Function Regular Expression

The most important thing to remember when creating a regular expression is that it must be surrounded with slashes ***/regular expression/***. With that knowledge let's search a string to see if a common name "Alex" is inside it.

```
<script type="text/javascript">
var myRegExp = /Alex/;
var string1 = "Today John went to the store and talked with Alex.";
var matchPos1 = string1.search(myRegExp);
if(matchPos1 != -1)
        document.write("There was a match at position " + matchPos1);
else
        document.write("There was no match in the first string");
</script>
```

JavaScript Form Validation

There's nothing more troublesome than receiving orders, guestbook entries, or other form submitted data that are incomplete in some way. You can avoid these headaches once and for all with JavaScript's amazing way to combat bad form data with a technique called "form validation".

JavaScript **document.getElementById**

JavaScript getElementById

Have you ever tried to use JavaScript to do some form validation? Did you have any trouble using JavaScript to grab the value of your text field? There's an easy way to access any HTML element, and it's through the use of *id* attributes and the *getElementById* function.

If you want to quickly access the value of an HTML input give it an *id* to make your life a lot easier. This small script below will check to see if there is any text in the text field "myText". The argument that *getElementById* requires is the *id* of the HTML element you wish to utilize.

```
<script type="text/javascript">
function notEmpty(){
	var myTextField = document.getElementById('myText');
	if(myTextField.value != "")
		alert("You entered: " + myTextField.value)
	else
		alert("Would you please enter some text?")
}
</script>
<input type='text' id='myText' />
<input type='button' onclick='notEmpty()' value='Form Checker' />
```

The quickest way to check if an input's string value is all numbers is to use a regular expression **/^[0-9]+$/** that will only match if the string is all .numbers and is at least one character long.

JavaScript Code:

```
// If the element's string matches the regular expression it is all numbers.
	var numericExpression = /^[0-9]+$/;
```

What we're doing here is using JavaScript existing framework to have it do all the hard work for us. Inside each string is a function called a match that you can use to see if the string matches a certain regular expression. We accessed this function like so**:** **elem.value.match(expression here)**.
We wanted to see if the input's string was all numbers so we made a regular expression to check for numbers [0-9] and stored it as **numericExpression**.
We then used the match function with our regular expression. If it is numeric then match will return true, making our if statement pass the test and our function **isNumeric** will also return true. However, if the expression fails because there is a letter or other character in our input's string then we'll display our **helperMsg** and return false.

```
    if(elem.value.match(numericExpression)){
		return true;
	else	return false;
```

Checking for All Letters

This function will be identical to isNumeric except for the change to the regular expression we use inside the match function. Instead of checking for numbers we will want to check for all letters.
If we wanted to see if a string contained only letters we need to specify an expression that allows for both lowercase and uppercase letters:
/^[a-zA-Z]+$/ .

JavaScript Code:
// If the element's string matches the regular expression it is all letters

```
function isAlphabet(elem, helperMsg){
        var alphaExp = /^[a-zA-Z]+$/;
        if(elem.value.match(alphaExp)){
                return true;
        }else{ alert(helperMsg);   return false;          } }
```

Form Validation - Restricting the Length

Being able to restrict the number of characters a user can enter into a field is one of the best ways to prevent bad data. For example, if you know that the zip code field should only be 5 numbers you know that 2 numbers is not sufficient.
Below we have created a lengthRestriction function that takes a text field and two numbers. The first number is the minimum number of characters and the second is the maximum number of a characters the input can be. If you just want to specify an exact number then send the same number for both minimum and maximum.

JavaScript Code:

```
function lengthRestriction(elem, min, max){
        var uInput = elem.value;
        if(uInput.length >= min && uInput.length <= max){
                return true;
        }else{
                alert("Please enter between " +min+ " and " +max+ " characters");
                elem.focus();
                return false;
        }
}
```

Form Validation - Email Validation

And for our grand finale we will be showing you how to check to see if a user's email address is valid. Every email is made up for 5 parts:
A combination of letters, numbers, periods, hyphens, plus signs, and/or underscores The at symbol @ A combination of letters, numbers, hyphens, and/or periods A period The top level domain (com, net, org, us, gov, ...)
.
Valid Examples:
bobby.jo@filltank.net
jack+jill@hill.com
the-stand@steven.king.com
Invalid Examples:
@deleted.net - no characters before the @
free!dom@bravehe.art - invalid character !
shoes@need_shining.com - underscores are not allowed in the domain name

The `innerHTML`

The innerHTML property can be used to modify your document's HTML on the fly. When you use innerHTML, you can change the page's content without refreshing the page. This can make your website feel quicker and more responsive to user input. The innerHTML property is used along with getElementById within your JavaScript code to refer to an HTML element and change its contents, Despite this, it is supported in all major browsers, which stands for Document Object Model, is the hierarchy that you can use to access and manipulate HTML objects from within your JavaScript.

The `innerHTML` Syntax

The syntax for using `innerHTML` goes like this:

```
document.getElementById('{ID of
element}').innerHTML = '{content}';
```

In this syntax example, `{ID of element}` is the ID of an HTML element and `{content}` is the new content to go into the element.

Basic `innerHTML` Example

Here's a basic example to demonstrate how `innerHTML` works.

This code includes two functions and two buttons. Each function displays a different message and each button triggers a different function.

In the functions, the `getElementById` refers to the HTML element by using its ID. We give the HTML element an ID of "myText" using `id="myText"`.
So in the first function for example, you can see that

`document.getElementById('myText')innerHTML='Thanks!';` is setting the innerHTML of the "myText" element to "Thanks!".

In the previous, we used an event handler to trigger off a call to our function. There are 18 event handlers that you can use to link your HTML elements to a piece of JavaScript. When you write a JavaScript function, you will need to determine when it will run. Often, this will be when a user does something like a click or hover over something, submit a form, double clicks on something etc.
Using JavaScript, you can respond to an event using event handlers. You can attach an event handler to the HTML element for which you want to respond to when a specific event occurs.
For example, you could attach JavaScript's onMouseover event handler to a button and specify some JavaScript to run whenever this event occurs against that button.

Examples :
Code:

```
<script type="text/javascript">
function Msg1(){
  document.getElementById('myText').innerHTML = 'Thanks!';
}
function Msg2(){
  document.getElementById('myText').innerHTML = 'Try message 1 again...';
}
</script>
<input type="button" onclick="Msg1()" value="Show Message 1" />
<input type="button"  onclick="Msg2()" value="Show Message 2" />
<p id="myText"></p>
```

Result:
Thanks!

Example 2: `innerHTML` With User Input

Here's an example of using `innerHTML` with a text field. Here, we display whatever the user has entered into the input field.

Code:

```
<script type="text/javascript">
function showMsg(){
  var userInput = document.getElementById('userInput').value;
  document.getElementById('userMsg').innerHTML = userInput;
}
</script>
<input type="input" maxlength="40" id="userInput"
  onkeyup="showMsg()" value="Enter text here..." />
<p id="userMsg"></p>
```

Result:

Example 3: Formatting with `getElementById`

In this example, we use the `getElementById` property to detect the color that the user has selected. We can then use `style.color` to apply the new color. In both cases, we refer to the HTML elements by their ID (using `getElementById`.)

Code:

```
<script type="text/javascript">
function changeColor(){
  var newColor = document.getElementById('colorPicker').value;
        document.getElementById('colorMsg').style.color = newColor;
}
</script>
<p id="colorMsg">Choose a color...</p>
<select id="colorPicker" onchange="JavaScript:changeColor()">
<option value="#000000">Black</option>
<option value="#000099">Blue</option>
<option value="#990000">Red</option>
<option value="#009900">Green</option>
</select>
```

Result:

Choose a color...

Sometimes, you may need to call some JavaSript from within a link. Normally, when you click a link, the browser loads a new page (or refreshes the same page).

This might not always be desirable. For example, you might only want to dynamically update a form field when the user clicks a link.

Example : JavaScript "On Double Click"

You could just have easily used another event to trigger the same JavaScript. For example, you could run JavaScript only when the double clicks the HTML element. We can do this using the **onDblClick** event handler.
Code:

```
<input type="button" onDblClick="alert('Hey, remember to tell your friends about Quackit.com!');" value="Double Click Me!" />
```

To prevent the load from refreshing, you could use the JavaScript void() function and pass a parameter of (zero).

Example of void(0): We have a link that should only do something (i.e. display a message) upon two clicks (i.e. a double click). If you click once, nothing should happen. We can specify the double click code by using JavaScript's "ondblclick" method. To prevent the page reloading upon a single click, we can use "JavaScript:void(0);" within the anchor link.
Code:

```
<a href="JavaScript:void(0);" ondblclick="alert('Well done!')">Double Click Me!</a>
```

Result:
Double Click Me!
Same Example, but without void(0):
Look at what would happen if we didn't use "JavaScript:void(0);" within the anchor link...
Code:

```
<a href="" ondblclick="alert('Well done!')">Double Click Me!</a>
```

Result:
Double Click Me!

Did you notice the page refresh as soon you clicked the link. Even if you double clicked and triggered the "ondbclick" event, the page still reloads!
Note: Depending on your browser, your browser might have redirected you to the "/javascript/tutorial/" index page. Either way, JavaScript's "void()" method will prevent this from happening.
Void(0) can become useful when you need to call another function that may have otherwise resulted in an unwanted page refresh.
Refresh code In JavaScript, you refresh the page using `location.reload`.

This code can be called automatically upon an event or simply when the user clicks on a link.

Example JavaScript Refresh code

Typing this code:

<input type="button" value="Reload Page" onClick = "document.location.reload (true)">

You can use JavaScript to automatically open print dialogue box so that users can print the page. Although most users know they can do something like "File > Print", a "Print this page" option can be nice, and may even encourage users to print the page. Also, this code can be triggered automatically upon loading a printer friendly version.

Creating a "Print this page"

The following code creates a hyperlink and uses the Javascript print function to print the current page:

```
<a href="JavaScript:window.print();">Print this page</a>
```

Open a new window in JavaScript

One of the more common requests I see is how to open a new Javascript window with a certain feature/argument. The Window.open method has been available since Netscape 2.0 (Javascript 1.0), but additional window decoration features have been added to the syntax since then.

Window.open method

The syntax to open a popup is: window.open(url, name, params):

url

An URL to load into the new window.

name

A name of the new window. Each window has a window.name, and here we can specify which window to use for the popup. If there's already a window with such name – the given URL opens in it, otherwise a new window is opened.

params

The configuration string for the new window. It contains settings, delimited by a comma. There must be no spaces in params, for instance: width:200,height=100.

```
Example let params = `scrollbars=no, resizable=no, status=no,
location=no, toolbar=no, menubar=no, width=0,height=0, left=-1000, top=-
1000`;

open('/', 'test', params);
```

```
<html>
<body>
<p>Click the button to open a new browser window.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
  window.open("https://www.yahoo.com");}
</script>
</body>
</html>
```

write to new window

How do I write script-generated content to another window?

To write script-generated content to another window, use the method ***winRef*.document.write()**, as returned by the `window.open()` method.

To make sure that your script's output actually shows up in the other window, use ***winRef*.document.close()** after writing the content. As an example, consider the following function that opens a new window with the title **Console** and writes the specified `content` to the new

Examples

```
<HTML>
<HEAD>
<TITLE>Writing to Subwindow</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var newWindow
function makeNewWindow() {
   newWindow = window.open("","","status,height=200,width=300")
}
function subWrite() {
   if (newWindow.closed) {
      makeNewWindow()
   }
```

```
    newWindow.focus()
    var newContent = "<HTML><HEAD><TITLE>A New Doc</TITLE></HEAD>"
    newContent += "<BODY BGCOLOR='coral'><H1>This document is brand new.</H1>"
    newContent += "</BODY></HTML>"
    newWindow.document.write(newContent)
    newWindow.document.close() // close layout stream
}
</SCRIPT>
</HEAD>
<BODY onLoad="makeNewWindow()">
<FORM>
<INPUT TYPE="button" VALUE="Write to Subwindow" onClick="subWrite()">
</FORM>
</BODY>
</HTML>
```

Moving and resizing windows

When you have created a window, you can use Javascript to **move** it or **resize** it.

- **Moving windows**

A window object has two methods for moving it**:** **`moveTo`** and **`moveBy`**. Both take two numbers as arguments.
The first function moves the window to the specified coordinates, measured in pixels from the top left corner of the screen. The first argument is the horizontal, or X, coordinat, and the second is the vertical, or Y, coordinate.
window.moveTo(*x*, *y*)
x is the horizontal coordinate to be moved to.
y is the vertical coordinate to be moved to.

Example

```
<script type="text/javascript">
function open_and_move1(){
win2=window.open("page.htm","","width=300,height=300")
win2.moveTo(0,0)
}
</script
```

The second method changes the current coordinates by the given amounts, which can be negative to move the window left or up.
Not all browsers allow you to move a window.

window.moveBy(deltaX, deltaY**)**
deltaX is the amount of pixels to move the window horizontally. Positive values are to the right, while negative values are to the left.

deltaY is the amount of pixels to move the window vertically. Positive values are down, while negative values are up.

Example

```
<script type="text/javascript">
function open_and_move2(){
   win3=window.open("page.htm","","width=600,height=500")
   win3.moveTo(screen.width/2-300,screen.height/2-250)
}
</script>
```

The center coordinates of the screen in the second example is calculated by determining the screen's dimensions, dividing that by 2, and subtracting half of either the window's width/height from it. In other words, in order to center a window,

- **Resizing windows**

Similar to the methods for moving, there are two methods for resizing a window: **`resizeTo`** and **`resizeBy`**. Like for moving, they either set the size absolutely or modify the size relatively to the current size.

Most browsers' standard security setting will not let you resize a window to less than 100 pixels in any direction.

```
window.resizeBy(100,-100);
```

Closing windows

Closing a window is simple when it works. All windows have a `close` method, so you can attempt to close any window you have a reference to.

```
myWindow.close();
```

Checking whether window has been closed

Given a reference to a window, it is possible to see whether the window has been closed.

```
if (myWindow.closed) { /* do something, e.g., open it again */ }
```

External JavaScript

You can place all your scripts into an external file (with a .js extension), then link to that file from within your HTML document. This is handy if you need to use the same scripts across multiple HTML pages, or a whole website.
To link to an external JavaScript file, you add a `src` attribute to your HTML `script` tag and specify the location of the external JavaScript file.

Linking to an external JavaScript file

<script type="text/javascript" src="external_javascript.js"></script>
Contents of your external JavaScript file
The code in **your .js file** should be no different to the code you would normally have placed in between the script tags. But remember, you don't need to create script tag again - it is already on the HTML page calling the external file! Let's create an external JavaScript file that prints Hello Javatpoint in a alert dialog box.

message.js

```
function msg(){
 alert("Hello Javatpoint");
}
```

Let's include the JavaScript file into html page. It calls the JavaScript function on button click.

index.html

```
<html>
<head>
<script type="text/javascript" src="message.js"></script>
</head>
<body>
<p>Welcome to JavaScript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```

Active Server Pages or Classic ASP

Introduction

Active Server Pages ASP, as it is more commonly known, is Microsoft's first server side scripting engine that enables you to make dynamic and interactive web pages.
ASP uses server-side scripting to dynamically produce web pages that are not affected by the type of browser the website visitor is using.
The **default** scripting language used for writing ASP is VBScript, although you can use other scripting languages like JScript (Microsoft's version of JavaScript).
ASP pages have the extension .asp instead of .htm, when a page with the extension .asp is requested by a browser the web server knows to interpret any ASP contained within the web page before sending the HTML produced to the browser. This way all the ASP is run on the web server and no ASP will ever be passed to the web browser. Any web pages containing ASP cannot be run by just simply opening the page in a web browser. The page must be requested through a web server that supports ASP, this is why ASP stands for Active Server Pages, no server, no active pages.
As ASP was first introduced by Microsoft on it's web server, Internet Information Services (IIS), that runs on all versions of Windows from NT4, including Windows 7, Vista, XP Pro, and Windows Server OS's like Windows 2000, 2003, 2008, it is this web server that ASP pages usually run best on.
For those of you running Windows and wish to play around with ASP on your own system you will need to install Microsoft's Internet Information Services (IIS). Lucky IIS or its micro version Personal Web Server (PWS) comes free with Windows, where it is installed in a server program features **a dynamic link library called ASP.DLL,** when a user requests a page extension asp service provider processes the orders existing ASP this and sends the result.

There are two software suitable:

1 - PWS program, a shortcut Personal Web Server

2 - Program IIS which is an a shortcut for Internet Information Server

The choice of one of these programs on the operating system you have

What is ASP?

ASP is a powerful tool for making dynamic and interactive Web pages.

- ASP stands for **A**ctive **S**erver **P**ages
- ASP is a Microsoft Technology
- ASP is a program that runs inside **IIS**

- IIS stands for **I**nternet **I**nformation **S**ervices
- IIS comes as a free component with **Windows 2000**

What is an ASP File?

- An ASP file is just the same as an HTML file
- An ASP file can contain text, HTML, XML, and scripts
- Scripts in an ASP file are executed on the server
- An ASP file has the file extension ".asp"

How Does ASP Differ from HTML?

- When a browser requests an 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

IIS - Internet Information Server

IIS is a set of Internet-based services for servers created by Microsoft for use with Microsoft Windows.IIS comes with Windows 2000, XP, Vista, and Windows 7. It is also available for Windows NT.

Steps to Install IIS on Windows XP and Windows 2000

1. On the Start menu, click Settings and select Control Panel
2. Double-click Add or Remove Programs
3. Click Add/Remove Windows Components
4. Click Internet Information Services (IIS)
5. Click Details
6. Select the check box for World Wide Web Service, and click OK
7. In Windows Component selection, click Next to install IIS

After you have installed IIS, make sure you install all patches for bugs and security problems. (Run Windows Update).

Steps Install IIS on Windows 7 and Windows Vista

1. Open the Control Panel from the Start menu
2. Double-click Programs and Features
3. Click "Turn Windows features on or off" (a link to the left)
4. Select the check box for Internet Information Services (IIS), and click OK

After you have installed IIS or PWS follow these steps

- Look for a new folder called **Inetpub** on your hard drive
- Open the Inetpub folder, and find a folder named **wwwroot**
- Create a new folder, like "MyWeb", under wwwroot
- Write some ASP code and save the file as "test1.asp" in the new folder
- Make sure your Web server is running (see below)
- Open your browser "http://localhost/MyWeb/test1.asp", to view your first web page

Script languages used with ASP:

There are two different types of (Script Languages) can be used to write the active code on the ASP pages are:
1 - VBScript (the default language permitted use in the pages of the ASP).
<%@ language= "VBscript" %>
2 - JavaScript.
<%@ language= "Javascript" %>

Write Output to a Browser

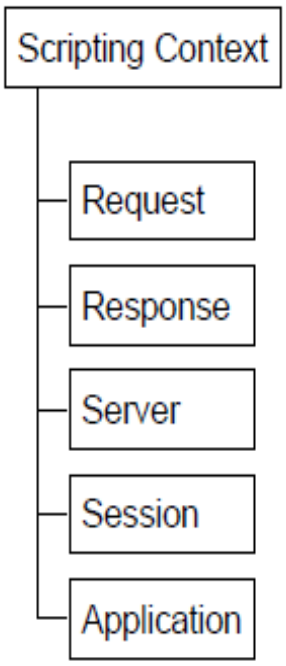
An ASP file normally contains HTML tags, just like an HTML file. However, an ASP file can also contain server scripts, surrounded by the delimiters <% and %>.

Server scripts are executed on the server, and can contain any expressions, statements, procedures, or operators valid for the scripting language you prefer to use.

What is localhost

Let us first see, what we mean by a hostname. Whenever you connect to a remote computer using it's URL, you are in effect calling it by its hostname.
For example, when you type in http://www.google.com/ you are really asking the network to connect to a computer named
www.google.com. It is called the "hostname" of that computer.
Localhost is a special hostname. It always references your own machine. So what you just did was to try to access a webpage on your own machine (which is what you wanted to do anyway.)
For testing all your pages, you will need to use localhost as the hostname.
When executing programs for the server, write hostname and the name of the program **:- http://localhost/path/simpleform.asp?**

ASP Objects



Request: To get information from the user

Response: To send information to the user

Server: To control the Internet Information Server

Session: To store information about and change settings for the user's current Web-server session

Application: To share application-level information and control settings for the lifetime of the application

1-Response Object

The ASP Response object is used to send output to the user from the server. Its collections, properties, and methods are described below:

Methods

Method	Description
Clear	Clears any buffered HTML output
End	Stops processing a script, and returns the current result
Redirect	Redirects the user to a different URL
Write	Writes a specified string to the output

Write method

The Write method writes a specified string to the output.

Syntax

Response.Write (variant)

Parameter	Description
Variant	Required. The data to write

```
<%
response.write("Hello World!")
%>
```

Variable in asp

A variable is used to store information. **Dim** variable

```
<%
dim i
for i=1 to 6
  response.write("<h" & i & ">Heading " & i & "</h" & i & ">")
next
%>
```

Redirect Method

The Redirect method redirects the user to a different URL.

Syntax

Response.Redirect URL

Parameter	Description
URL	Required. The URL that the user (browser) is redirected to

Examples

```
<%
Response.Redirect (http://www.webpage.com ) %>
```

Request object

When a browser asks for a page from a server, it is called a request. The Request object is used to get information from a visitor. Its collections, properties, and methods are described below:

Collections

Collection	Description
Cookies	Contains all the cookie values sent in a HTTP request

Form	Contains all the form (input) values from a form that uses the post method
QueryString	Contains all the variable values in a HTTP query string

Used to deliver data sent by the user's computer to a server through HTTP request.

Required object

There are two ways to get form information:

1- The Request.QueryString command

2- The Request.Form command.

1- Request.QueryString

The Request.QueryString command collects the values in a form as text. Information sent from a form with the GET method is visible to

everybody (in the address field). Remember that the GET method limits the amount of information to send.

Request.Querystring

Method="get"

action = "filename.asp"

Syntax

Request.QueryString(variable)[(index)|.Count]

2-Request.Form

Command Request.form, which is not much different from something Request.querystring, it is used to gather the values of the form with the use of method Post, which may not be visible to anyone, and it does not specify the amount of.

Syntax

Request.Form(element)[(index)|.Count]

Request.Form

Method="post"

action = "filename.asp"

GET and POST

One thing we ignored in our discussion about forms was that the METHOD by which the form is submitted may be one of the two: GET or POST.

When to use GET?

Any data that you pass via a GET can be retrieved in your script by using the Request.QueryString collection. GET may be used for small amounts of data

– the reason being that, the data items are appended to the URL by your

browser, and obviously, you cannot have an infinitely long URL (with the

QueryString).

When to use POST?

Almost always. Stick to POST for your forms, and be sure to use the Request.Form collection to access them (and *not* the Request.QueryString collection.)

Example1 Simpleform.asp

```
<html>
<body>
 <%
   request.querystring("fname“)
   request.querystring("lname“)
  %>
 <form method="get" action="simpleform.asp">
    fname : <input type="text" name="fname">
   </br >
 lname: <input type="text" name="lname">
    </br ></br >
    <input type="submit" value=“sent data” >
 </form>
</body>
</html>
```

Run

http://localhost/simpleform.asp?fname=shatha &lname=Habeeb

Example2 <html dir="rtl">

```
<body>
 <%
   request.form("username")
   request.form("pass")
 %>
 <form method="post" action="form2.asp">
    user name : <input type="text" name=“username"></br >
   password: <input type=“password" name=“pass">
   </br ></br >
   <input type="submit" value="sent">
 </form>
 <% ="username" &request.form("username") %></br>
 <% ="pass" & request.form("pass") %>
</body>
</html>
```

Example3

Form1.asp

```
<html>
<body>
  <form method="get" action="form2.asp">
      fname: <input type="text" name="fname">
      </br >
      lname: <input type="text" name="lname">
      </br ></br >
      <input type="submit" value=“send data">
  </form>
</body>
</html>
```

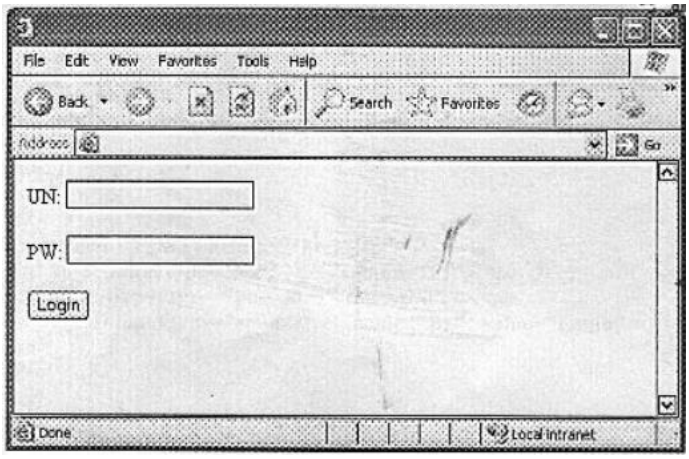
Form2.asp

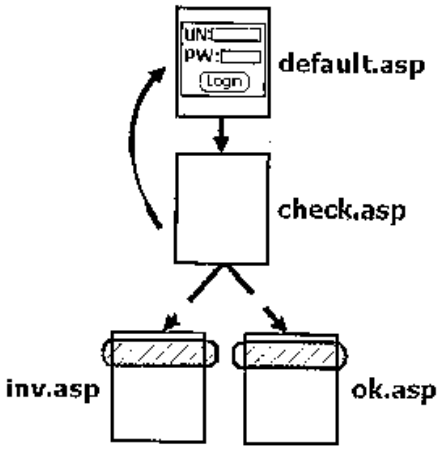
```
<html>
<body>
 <%
   dim fname,lname
  fname=request.querystring("fname“)
   lname=request.querystring("lname“)
```

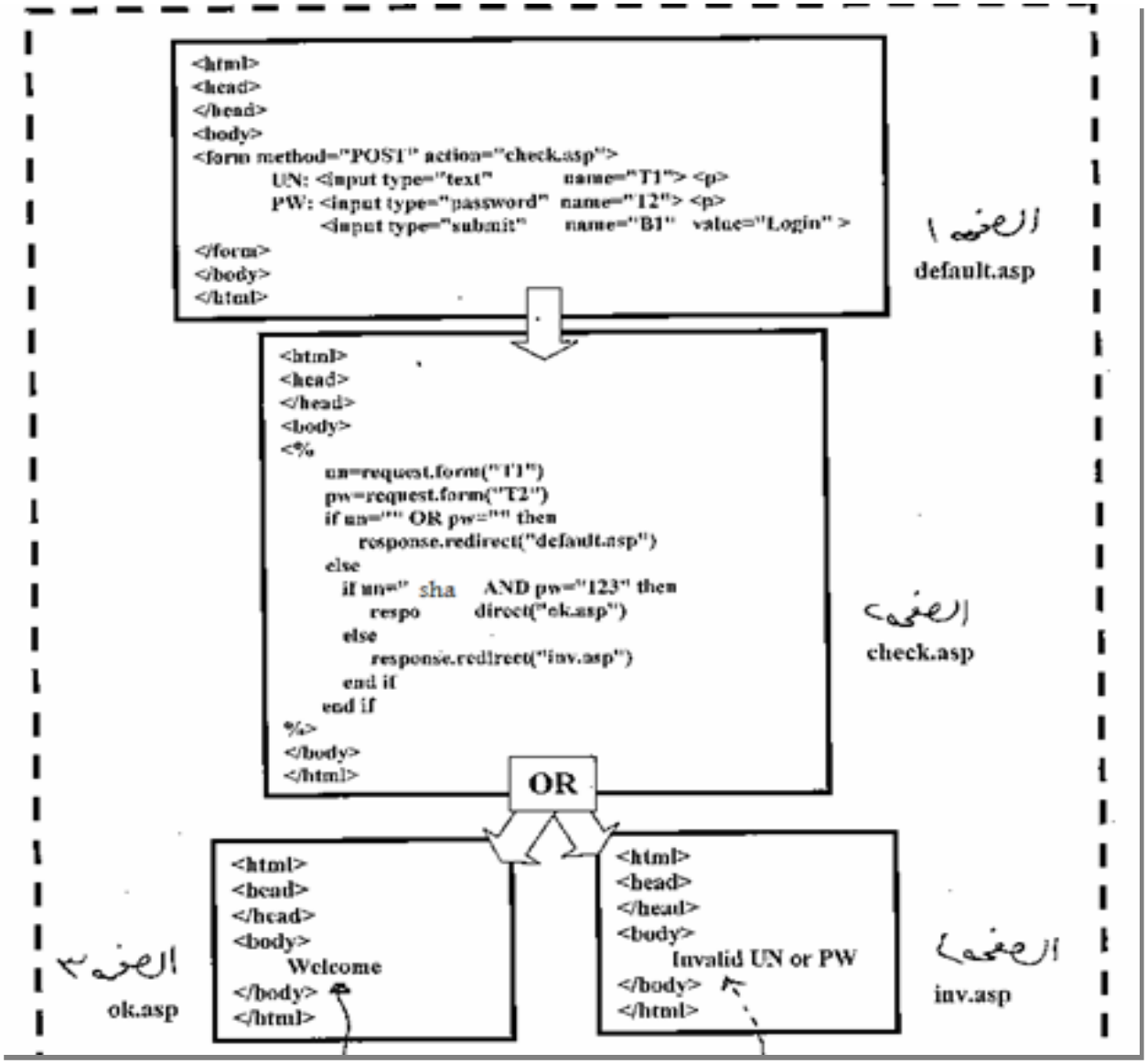
```
 %>
<%=“fname<B>"&fname&"</b>"%>
   <%=“lname<B>"&lname &"</B>"%>
</body>
</html>
```

Login Application:

Login Application: is used to allow the authorized users to access private and secured areas(web pages), by using User-Name and Password strings.







However, the internal pages ('ok. asp' and 'in .asp' pages) must be having "**Guarding Code**" to prevent any direct access to them. So if any user writes the following address (HTTP:// ..... / ok. asp) to access the 'ok. Asp' page without coming from login-form (in another word to bypassing the checking code), the guarding-code must prevent this unauthorized access and the process must be returned to the 'default, asp' page The "Guarding Code" must be written inside all internal pages.

Guarding-Code is work by check if the user access to this page by follows authorized-path or not. If the user access 'default.asp' page and enter the correct username and password strings, then the 'check.asp' the page must not only redirect the process to the 'ok.asp' page but must register the username inside cookie file. So when access the 'ok.asp' page the process must be check **cookies file**, if cookie hold username then this means that the user access 'ok.asp' page by follow the authorized path, else if cookie

hold nothing then this means that the user tries to access to the 'ok.asp' page by by-passing checking code.

Cookies

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With ASP, you can both create and retrieve cookie values.

How to Create a Cookie?

The "Response.Cookies" command is used to create cookies. we will create a cookie named "firstname" and assign the value "Alex" to it:

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

It is also possible to assign properties to a cookie, like setting a date when the cookie should expire:

```
<%
Response.Cookies("firstname")="Alex"
Response.Cookies("firstname").Expires=#May 10,2012#
%>
```

How to Retrieve a Cookie Value?

The "Request.Cookies" command is used to retrieve a cookie value.

```
<%
fname=Request.Cookies("firstname")
response.write("Firstname=" & fname)
%>
```

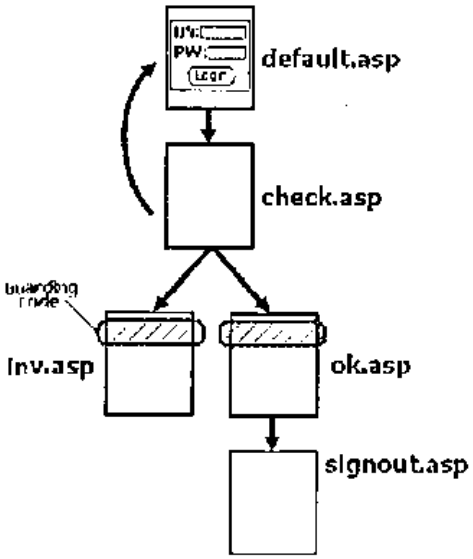
Output: Firstname=Alex

A Cookie with Keys

If a cookie contains 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")="John"
Response.Cookies("user")("lastname")="Smith"
Response.Cookies("user")("country")="Norway"
```

Response.Cookies("user")("age")="25"
%>
However, when user wants to leave 'ok.asp' page, so the user must be sign out this page, this means that the user must put nothing inside cookie file.



FileSystemObject

The FileSystemObject object is used to access the file system on a server. This object can manipulate files, folders, and directory paths. It is also possible to retrieve file system information with this object.

Scripting.FileSystemObject

```
Set fs = CreateObject("Scripting.FileSystemObject")
Set a = fs.CreateTextFile("c:\testfile.txt", True)
a.WriteLine("This is a test.")
a.Close
```

1- **Open and Read content from a text file**

```
<%
Set fs = CreateObject("Scripting.FileSystemObject")
Set wfile = fs.OpenTextFile("c:\Mydir\myfile.txt")
filecontent = wfile.ReadAll
wfile.close
Set wfile=nothing
Set fs=nothing
response.write(filecontent)
%>
```

ReadLine

```
<%
Set fs = CreateObject("Scripting.FileSystemObject")
Set wfile = fs.OpenTextFile("c:\Mydir\myfile.txt")
firstname = wfile.ReadLine
lastname = wfile.ReadLine
theage = wfile.ReadLine
wfile.close
Set wfile=nothing
Set fs=nothing
%>
Your first name is <% =firstname %><BR>
Your last name is <% =lastname %><BR>
Your are <% =thaage %> years old<BR>
```

AtEndOfStream

```
<%
Set fs = CreateObject("Scripting.FileSystemObject")
Set wfile = fs.OpenTextFile("c:\Mydir\myfile.txt")
    counter=0
    do while not wfile.AtEndOfStream
     counter=counter+1
     singleline=wfile.readline
     response.write (counter & singleline & "<br>")
    loop
    wfile.close
    Set wfile=nothing
    Set fs=nothing
    %>
```

2- **Create and Write a text file**

Example 1: The basic code we need to create a file is very similar to that one we have used to open a file:

```
<%
dim fs,f
set fs=Server.CreateObject("Scripting.FileSystemObject")
set f=fs.CreateTextFile("c:\test.txt",true)
f.WriteLine("Hello World!")
```

```
f.Close
set f=nothing
set fs=nothing
%>
```

ADO (ActiveX Data Objects)

ADO can be used to access databases from your web pages

ADO is a Microsoft technology
ADO stands for **A**ctiveX **D**ata **O**bjects
ADO is automatically installed with Microsoft IIS
ADO is a programming interface to access data in a database

Accessing a Database from an ASP Page

The common way to access a database from inside an ASP page is:

1. Create an ADO connection to a database
2. Open the database connection
3. Create an ADO recordset
4. Open the recordset
5. Extract the data you need from the recordset
6. Close the recordset
7. Close the connection

Before a database can be accessed from a web page, a database connection has to be established.
The ADO Connection Object is used to create an open connection to a data source. Through this connection, you can access and manipulate a database.
If you want to access a database multiple times, you should establish a connection using the Connection object. You can also make a connection to a database by passing a onnection string via a Command or Recordset object.

Set objConnection = Server.CreateObject("ADODB.connection")

Connection to your database can be done using either a DSN (Data Source Name) or without DSN aka DSN-less.
DSN stores all the information required to connect to your database, including your database type, server / file location, username and password as well as connection options is stored in the DSN itself. Your script simply refers to the connection using the DSN name.
On the contrary, when using DSN-less method, your script would have to specify all the above details in the connection string itself.

Create a DSN-less Database Connection

The easiest way to connect to a database is to use a DSN-less connection. A DSN-less connection can be used against any Microsoft Access database on your web site.

If you have a database called "northwind.mdb" located in a web directory like "c:/webdata/", you can connect to the database with the following ASP code:

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
%>
```

Note, from the example above that you have to specify the Microsoft Access database driver (Provider) and the physical path to the database on your computer.

ADO Recordset

The ADO Recordset object is used to hold a set of records from a database table. A Recordset object consist of records and columns (fields).

To be able to read database data, the data must first be loaded into a recordset.

Suppose we have a database named "Northwind", we can get access to the "Customers" table inside the database with the following lines:

```
set objRecordset=Server.CreateObject("ADODB.recordset")
```

When you first open a Recordset, the current record pointer will point to the first record and the BOF and EOF properties are False. If there are no records, the BOF and EOF property are True.

Create an ADO SQL Recordset

We can also get access to the data in the "Customers" table using SQL:

```
<%
set conn = Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs=Server.CreateObject("ADODB.recordset")
rs.Open "Select * from Customers", conn
%>
```

The Structured Query Language (SQL) forms the backbone of all relational databases. This language offers a flexible interface for databases of all shapes and sizes and is used as the basis for all user and administrator interactions with the database.

Examples The SELECT command retrieves data from one or more tables or views. It generally consists of the following language elements:

```
SELECT  <things_to_be_displayed> -- called 'Projection' - mostly a list of columns
FROM    <tablename>              -- table or view names and their aliases
WHERE   <where_clause>           -- the so called 'Restriction' or 'search condition'
ORDER BY <order_by_clause>
FETCH   <fetch_first_or_next_clause>;
```

Display the Field Names and Field Values

We have a database named "Northwind" and we want to display the data from the "Customers" table (remember to save the file with an .asp extension):

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

set rs = Server.CreateObject("ADODB.recordset")
rs.Open "SELECT * FROM Customers", conn

do until rs.EOF
 for each x in rs.Fields
  Response.Write(x.name)
  Response.Write(" = ")
  Response.Write(x.value & "<br>")
 next
 Response.Write("<br>")
 rs.MoveNext
loop

rs.close
conn.close
%>
```

Display Selected Data

We want to display only the records from the "Customers" table that have a "Companyname" that starts with an A (remember to save the file with an .asp extension):

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

set rs=Server.CreateObject("ADODB.recordset")
```

```
sql="SELECT Companyname, Contactname FROM Customers
WHERE CompanyName LIKE 'A%'"
rs.Open sql, conn
%>

<table border="1" width="100%">
 <tr>
 <%for each x in rs.Fields
  response.write("<th>" & x.name & "</th>")
 next%>
 </tr>
 <%do until rs.EOF%>
  <tr>
  <%for each x in rs.Fields%>
   <td><%Response.Write(x.value)%></td>
  <%next
  rs.MoveNext%>
  </tr>
 <%loop
 rs.close
 conn.close%>
```

Sort the Data

We want to display the "Companyname" and "Contactname" fields from the "Customers" table, ordered by "Companyname" (remember to save the file with an .asp extension

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

set rs = Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname, Contactname FROM
Customers ORDER BY CompanyName"
rs.Open sql, conn
%>
```

-
```
<table border="1" width="100%">
  <tr>
  <%for each x in rs.Fields
   response.write("<th>" & x.name & "</th>")
  next%>
```

```
 </tr>
 <%do until rs.EOF%>
  <tr>
  <%for each x in rs.Fields%>
    <td><%Response.Write(x.value)%></td>
  <%next
  rs.MoveNext%>
  </tr>
 <%loop
 rs.close
 conn.close%>
</table>

</body>
</html>
```

- ```
  <html>
  <body>

  <form method="post" action="demo_add.asp">
  <table>
  <tr>
  <td>CustomerID:</td>
  <td><input name="custid"></td>
  </tr><tr>
  <td>Company Name:</td>
  <td><input name="compname"></td>
  </tr><tr>
  <td>Contact Name:</td>
  <td><input name="contname"></td>
  </tr><tr>
  <td>Address:</td>
  <td><input name="address"></td>
  </tr><tr>
  <tr>
  <td>City:</td>
  <td><input name="city"></td>
  </tr><tr>
  <td>Postal Code:</td>
  <td><input name="postcode"></td>
  </tr><tr>
  ```
  ```

```
<td>Country:</td>
<td><input name="country"></td>
</tr>
</table>
<br><br>
<input type="submit" value="Add New">
<input type="reset" value="Cancel">
</form>

</body>
</html>
```

- ```
  <html>
  <body>
  <%
  set conn=Server.CreateObject("ADODB.Connection")
  conn.Provider="Microsoft.Jet.OLEDB.4.0"
  conn.Open "c:/webdata/northwind.mdb"

  sql="INSERT INTO customers (customerID,companyname,"
  sql=sql & "contactname,address,city,postalcode,country)"
  sql=sql & " VALUES "
  sql=sql & "('" & Request.Form("custid") & "',"
  sql=sql & "'" & Request.Form("compname") & "',"
  sql=sql & "'" & Request.Form("contname") & "',"
  sql=sql & "'" & Request.Form("address") & "',"
  sql=sql & "'" & Request.Form("city") & "',"
  sql=sql & "'" & Request.Form("postcode") & "',"
  sql=sql & "'" & Request.Form("country") & "')"
  ```
- ```
  on error resume next
  conn.Execute sql,recaffected
  if err<>0 then
    Response.Write("No update permissions!")
  else
    Response.Write("<h3>" & recaffected & "
    record added</h3>")
end if
conn.close
%>
</body>
</html>
```