JAVA SCRIPT & DHTML

CONTENTS

- Introduction to Java Scripts
  - Variable , operators
  - Conditional statements ,loops
  - Functions
  - Events
-
  - Window
  - Navigator
  - Document
  - form
  - Date
  - String
  - arrays
- Dynamic HTML with Java Script

Introduction to JavaScript

A number of technologies are present that develops the static web page, but we require a language that is dynamic in nature to develop web pages a client side. Dynamic HTML is a combination of content formatted using HTML, cascading stylesheets, a scripting language and DOM.

JavaScript originates from a language called LiveScript. The idea was to find a language which can be used at client side, but not complicated as Java. JavaScript is a simple language which is only suitable for simple tasks.

Benefits of JavaScript

Following are some of the benefits that JavaScript language possess to make the web site

- dynamic. It is widely supported in browser
- It gives easy access to document object and can manipulate most of them.
- JavaScript can give interesting animations with many multimedia
- datatypes. Special plug-in are not required to use JavaScript JavaScript is
- secure language
- JavaScript code resembles the code of C language, The syntax of both the language is very
- close to each other. The set of tokens and constructs are same in both the language.

A Sample JavaScript program

```
<html>
<head><title>java script program</title>
<script languague="javascript">
function popup()
{
var major=parseInt(navigator.appVersion);
var minor=parseInt(navigator.appVersion);
var agent=navigator.userAgent.toLowerCase();
document.write(agent+" "+major);
window.alert(agent+" "+major);
}
function farewell()
{
window.alert("Farewell and thanks for visiting");
}
</script>
</head>
<body onLoad="popup()" onUnload="farewell()">
</body>
</html>
```

- JavaScript program contains variables, objects and functions.
- Each line is terminated by a semicolon. Blocks of code must be surrounded by curly brackets.
- Functions have parameters which are passed inside parenthesis
- Variables are declared using the keyword var.
- Script does not require main function and exit condition.

JavaScript program that shows the use of variables, datatypes

```
<html>
<head>
<title> My Sample JavaScript program</title>
<script language="javascript">
function disp()
{
var rno,sname,br,pr;
rno=prompt("Enter your registration number");
sname=prompt("Enter your Name");
br=prompt("Enter your branch Name");
pr=prompt("Enter the percentage");
document.writeln("<h2> Your Registration No. is :</h2>"+ rno.toUpperCase());
document.writeln("<h2> Your Name is :</h2>"+sname.toUpperCase());
document.writeln("<h2> Your Branch Name is :</h2>"+br.toUpperCase());
document.writeln("<h2> Your Overall Percentage is :</h2>"+pr);
document.close();
}
</script>
</head>
<body onLoad="disp()">
</body>
</html>
```

JavaScript program showing the using of constructs

```
<html>
<head> <title> Factorial</title> </head.
<body>
<script language="javascript">
function fact(n)
{
var i,f=1;
for(i=1;i<=n;i++)
{
f=f*i;
}
return(f);
}
var x,n,f;
x=prompt("Enter the number");
f=fact(x);
document.writeln("Factorial of "+x+" is "+f);
document.close();
</script>
</body>
</html>
```

Variables

Variables are like storage units. You can create variables to hold values. It is ideal to name a variable something that is logical, so that you'll remember what you are using it for. For example, if you were writing a program to divide 2

numbers, it could be confusing if you called your variables numberOne, numberTwo, numberThree because you may forget which one is the divisor, which one is the dividend, and which one is the quotient. A more logical approach would be to name them just that: divisor, dividend, quotient.
It is important to know the proper syntax to which variables must conform:

- They must start with a letter or underscore ("_")
- Subsequent characters can also be digits (0-9) or letters (A-Z and/or a-z). Remember, JavaScript is case-sensitive. (That means that MyVariable and myVariable are two different names to JavaScript, because they have different capitalization.)

Some examples of legal names are Number_hits, temp99, and _name.

When you declare a variable by assignment outside of a function, it is called a global variable, because it is available everywhere in the current document. When you declare a variable within a function, it is called a local variable, because it is available only within the function. Using var is optional, but you need to use it if you have a variable that has been declared global and you want to re-declare it as a local variable inside a function.

Variables can store all kinds of data (see below, Values of Variables, section 3.2). To assign a value to a variable, you use the following notation:

```
dividend = 8;
divisor = 4;

myString = "I may want to use this message multiple
times"; message = myString;
```

Let's say the main part of the function will be dividing the dividend by the divisor and storing that number in a variable called quotient. I can write this line of code in my program: quotient = divisor*dividend, and I have both stored the value of the quotient to the variable quotient and I have declared the variable at the same time. If I had wanted to, I could have declared it along with my other assigned variables above, with a value of null. After executing the program, the value of quotient will be 2.

It is important to think about the design of your program before you begin. You should create the appropriate variables so that it makes your life easier when you go to write the program. For instance, if you know that you will be coding a lot of the same strings in a message, you may want to create a variable called message and give it the value of your message. That way, when you call it in your program, you do not have to retype the same sentence over and over again, and if you want to change the content of that message, you only have to change it once -- in the variable declaration.

Values of Variables

JavaScript recognizes the following types of values:

- Numbers, such as 42 or 3.14159
- Logical (Boolean) values, either true or false
- Strings, such as "Howdy!"
- null, a special keyword which refers to nothing

This relatively small set of types of values, or data types, enables you to perform useful functions with your applications. There is no explicit distinction between integer and real-valued numbers.

Data Type Conversion

JavaScript is a loosely typed language. That means you do not have to specify the data type of a variable when you declare it, and data types are converted automatically as needed during script execution. So, for example, you could define a variable as follows:

```
var answer = 42
```

And later, you could assign the same variable a string value, for example, answer = "Thanks for all the fish..."

Because JavaScript is loosely typed, this assignment does not cause an error message. However, this is not good coding! You should create variables for a specific type, such as an integer, string, or array, and be consistent in the values that you store in the variable. This prevents confusion when you are writing your program.
In expressions involving numeric and string values, JavaScript converts the numeric values to strings. For example, consider the following statements:
x = "The answer is " + 42
y = 42 + " is the answer."
(The + sign tells JavaScript to concatenate, or stick together, the two strings. For example, if you write:
message = "Hello" + "World"
...then the variable message becomes the string "Hello World")
In the first statement, x becomes the string "The answer is 42." In the second, y becomes the string "42 is the answer."

Literals

You use literals to represent values in JavaScript. These are fixed values, not variables, that you literally provide in your script. Examples of literals include: 1234, "This is a literal," and true.

Integers

Integers can be expressed in decimal (base 10), hexadecimal (base 16), and octal (base 8). A decimal integer literal consists of a sequence of digits without a leading 0 (zero). A leading 0 (zero) on an integer literal indicates it is in octal; a leading 0x (or 0X) indicates hexadecimal. Hexadecimal integers can include digits (0-9) and the letters a-f and A-F. Octal integers can include only the digits 0-7.
Some examples of integer literals are: 42, 0xFFF, and -345.

Floating-point literals

A floating-point literal can have the following parts: a decimal integer, a decimal point ("."), a fraction (another decimal number), an exponent, and a type suffix. The exponent part is an "e" or "E" followed by an integer, which can be signed (preceded by "+" or "-"). A floating-point literal must have at least one digit, plus either a decimal point or "e" (or "E").
Some examples of floating-point literals are 3.1415, -3.1E12, .1e12, and 2E-12

Boolean literals

The Boolean type has two literal values: true and false.

String literals

A string literal is zero or more characters enclosed in double (") or single (') quotation marks. A string must be delimited by quotation marks of the same type; that is, either both single quotation marks or double quotation marks. The following are examples of string literals:

- "blah"
- 'blah'
- "1234"
- "one line \n another line"

In addition to ordinary characters, you can also include special characters in strings, as shown in the last element in the preceding list. The following table lists the special characters that you can use in JavaScript strings.

Character	Meaning
\b	backspace
\f	form feed
\n	new line
\r	carriage return
\t	tab

\\	backslash character

Escaping characters

For characters not listed in the preceding table, a preceding backslash is ignored, with the exception of a quotation mark and the backslash character itself.

You can insert quotation marks inside strings by preceding them with a backslash. This is known as escaping the quotation marks. For example,

```
var quote = "He read \"The Cremation of Sam McGee\" by R.W. Service."
document.write(quote)
```

The result of this would be

He read "The Cremation of Sam McGee" by R.W. Service.

To include a literal backslash inside a string, you must escape the backslash character. For example, to assign the file path c:\temp to a string, use the following:

```
var home = "c:\\temp"
```

Arrays

An Array is an object which stores multiple values and has various properties. When you declare an array, you must declare the name of it, and then how many values it will need to store. It is important to realize that each value is stored in one of the elements of the array, and these elements start at 0. This means that the first value in the array is really in the 0 element, and the second number is really in the first element. So for example, if I want to store 10 values in my array, the storage elements would range from 0-9. The notation for declaring an array looks like this:

```
myArray = new Array(10); foo = new Array(5);
```

Initially, all values are set to null. The notation for assigning values to each unit within the array looks like this:

```
myArray[0] = 56;
myArray[1] = 23;
myArray[9] = 44;
```

By putting the element number in brackets [ ] after the array's name, you can assign a value to that specific element. Note that there is no such element, in this example, as myArray[10]. Remember, the elements begin at myArray[0] and go up to myArray[9].

In JavaScript, however, an array's length increases if you assign a value to an element higher than the current length of the array. The following code creates an array of length zero, then assigns a value to element 99. This changes the length of the array to 100.

```
colors = new Array();
colors[99] = "midnightblue";
```

Be careful to reference the right cells, and make sure to reference them properly!

Because arrays are objects, they have certain properties that are pre-defined for your convenience. For example, you can find out how many elements myArray has and store this value in a variable called numberOfElements by using:

```
numberOfElements = myArray.length;
```

Operators

JavaScript has many different operators, which come in several flavors, including binary. This tutorial will cover some of the most essential assignment, comparison, arithmetic and logical operators.

Selected assignment operators

An assignment operator assigns a value to its left operand based on the value of its right operand. The basic assignment operator is equal (=), which assigns the value of its right operand to its left operand. The other operators are shorthand for standard operations. Find an abridged list of shorthand operators below:

Shorthand operator	Meaning
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y

Note that the = sign here refers to assignment, not "equals" in the mathematical sense. So if x is 5 and y is 7, x = x + y is not a valid mathematical expression, but it works in JavaScript. It makes x the value of x + y (12 in this case).
EXAMPLES: If x = 10 and y = 2, x += y would mean x = x + y, hence x's new value would be the sum of x's previous value plus y's previous value. Upon executing x = x + y = 10 + 2, x's new value becomes 12, while y's new value remains equal to its old value. Similarly, x -= y would change x's value to 8. Calculate x *= y and x /= y to get a better idea of how these operators work.

Comparison operators

A comparison operator compares its operands and returns a logical value based on whether the comparison is true or not. The operands can be numerical or string values. When used on string values, the comparisons are based on the standard lexicographical ordering. They are described in the following table.

Operator	Description	Example
Equal (= =)	Evaluates to true if the operands are equal.	x == y evaluates to true if x equals y.
Not equal (!=)	Evaluates to true if the operands are not equal.	x != y evaluates to true if x is not equal to y.
Greater than (>)	Evaluates to true if left operand is greater than right operand.	x > y evaluates to true if x is greater than y.
Greater than or equal (>=)	Evaluates to true if left operand is greater than or equal to right operand.	x >= y evaluates to true if x is greater than or equal to y.
Less than (<)	Evaluates to true if left operand is less than right operand.	x < y evaluates to true if x is less than y.
Less than or equal (<=)	Evaluates to true if left operand is less than or equal to right operand.	x <= y evaluates to true if x is less than or equal to y.

EXAMPLES: 5 == 5 would return TRUE. 5 != 5 would return FALSE. (The statement 'Five is not equal to five.' is patently false.) 5 <= 5 would return TRUE. (Five is less than or equal to five. More precisely, it's exactly equal to five, but JavaScript could care less about boring details like that.)

Selected Arithmetic Operators

Arithmetic operators take numerical values (either literals or variables) as their operands and return a single numerical value. The standard arithmetic operators are addition (+), subtraction (-), multiplication (*), division (/) and remainder (%). These operators work as they do in other programming languages, as well as in standard algebra.
Since programmers frequently need to add or subtract 1 from a variable, JavaScript has shortcuts for doing this. myVar++ adds one to the value of myVar, while myVar-- subtracts one from myVar.
EXAMPLES: Let x = 3. x++ bumps x up to 4, while x-- makes x equal to 2.

Logical Operators

Logical operators take Boolean (logical) values as operands and return a Boolean value. That is, they evaluate whether each subexpression within a Boolean expression is true or false, and then execute the operation on the respective truth values. Consider the following table:

Operator	Usage	Description
and (&&)	expr1 && expr2	True if both logical expressions expr1 and expr2 are true. False otherwise.
or (||)	expr1 || expr2	True if either logical expression expr1 or expr2 is true. False if both expr1 and expr2 are false.
not (!)	!expr	False if expr is true; true if expr is false.

EXAMPLES: Since we have now learned to use the essential operators, we can use them in conjunction with one another. See if you can work out why the following examples resolve the way they do: If x = 4 and y = 7, ((x + y + 2) == 13) && (((x + y) / 2) == 2) returns FALSE.
If x = 4 and y = 7, ((y - x + 9) == 12) || ((x * y) == 2) returns TRUE.
If x = 4 and y = 7, !((x/2 + y) == 9) || ((x * (y/2)) == 2) returns FALSE.

Using JavaScript Objects

When you load a document in your web browser, it creates a number of JavaScript objects with properties and capabilities based on the HTML in the document and other pertinent information. These objects exist in a hierarchy that reflects the structure of the HTML page itself.
The pre-defined objects that are most commonly used are the window and document objects. The window has methods that allow you to create new windows with the open() and close() methods. It also allows you to create message boxes using alert(), confirm(), and prompt(). Each displays the text that you put between the parentheses. For example, the following code:
alert("This is an alert box")
...pops up an alert box displaying the given message. Try it yourself by clicking on this link.

The document object models the HTML page. The document object contains arrays which store all the components constituting the contents of your web page, such as images, links, and forms. You can access and call methods on these elements of your web page through the arrays.

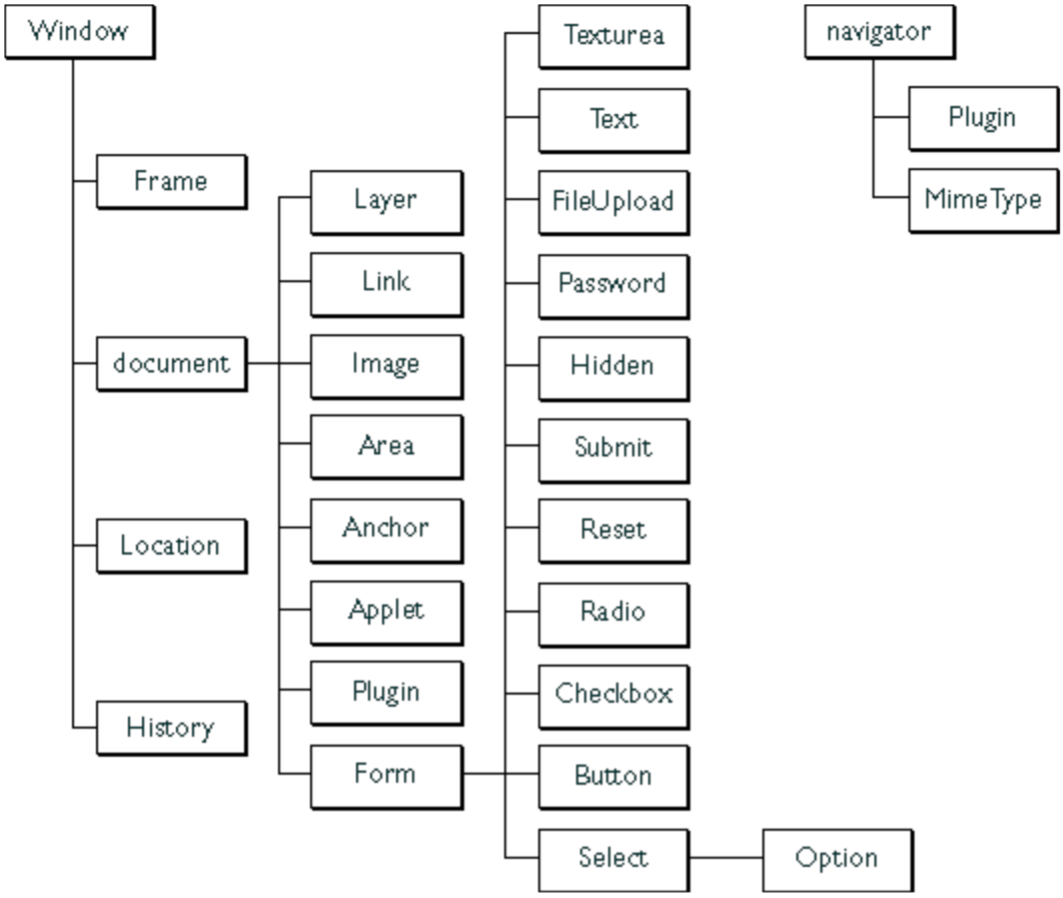
The objects in this pre-defined hierarchy can be accessed and modified. To refer to specific properties, you must specify the property name and all its ancestors, spelling out the complete hierarchy until the document object. A period, '.', is used in between each object and the name of its property. Generally, a property / object gets its name from the NAME attribute of the HTML tag. For example, the following refers to the *value* property of a text field named *text1* in a form named *myform* in the current document. document.myform.text1.value

Form elements can also be accessed through the aforementioned forms array of the document object. In the above example, if the form named *myform* was the first form on the page, and *text1* was the third field in the form, the following also refers to that field's value property.

document.forms[0].elements[2].value
Functions (capabiltiies) of an object can similarly be accessed using the period notation. For example, the following instruction resets the 2nd form in the document.

document.forms[2].reset();

Click on one of the objects below to view the Netscape documentation on the specific properties and methods that that object has:



Functions

Functions are one of the fundamental building blocks in JavaScript. A function is a JavaScript procedure -- a set of statements that performs a specific task. A function definition has these basic parts:

- The function keyword
- A function name
- A comma-separated list of arguments to the function in parentheses
- The statements in the function in curly braces: { }

Defining a Function

When defining a function, it is *very* important that you pay close attention to the syntax. Unlike HTML, JavaScript is case sensitive, and it is very important to remember to enclose a function within curly braces { }, separate parameters with commas, and use a semi-colon at the end of your line of code.

It's important to understand the difference between *defining* and *calling* a function.

Defining the function names the function and specifies what to do when the function is called. You define a function within the <SCRIPT>...</SCRIPT> tags within the <HEAD>...</HEAD> tags. In defining a function, you must also declare the variables which you will be calling in that function. Here's an example of *defining* a function:

```
function popupalert() {

                alert('This is an alert box.');

}
```

Notice the parentheses after the function name. It is imperative that you include these parentheses, even if they are empty. If you want to pass a *parameter* into the function, you would include that parameter inside of the parentheses. A parameter is a bit of extra information that cn be different each time the function is run. It is stored in a variable and can be accessed just like any other variable. Here's an example of a function that takes a parameter:

```
function anotherAlert(word) {

                alert(word + ' is the word that you clicked on');

}
```

When you call this function, you need to pass a parameter (such as the word that the user clicked on) into the function. Then the function can use this information. You can pass in a different word as a parameter each time you call the function, and the alert box will change appropriately. You'll see how to pass a parameter a little later on. You can pass in multiple parameters, by separating them with a comma. You would want to pass in a few parameters if you have more than one variable that you either want to change or use in your function. Here are two examples of passing in multiple parameters when you are defining the function:

```
function secondAlert(word,password) {

                confirm(word + ' is the word that you clicked on. The

                                secret password is ' + password);

}
function thirdAlert(word,password) {

                confirm(word + ' is the word you clicked on. Please

                                take note of the password, ' + password);

}
```

You'll notice that the same parameters are passed into both of these functions. However, you can pass in whatever values you want to use (see this same example below in calling the function).

Calling a Function

Calling the function actually performs the specified actions. When you call a function, this is usually within the BODY of the HTML page, and you usually pass a parameter into the function. A parameter is a variable from outside of the defined function on which the function will act.

Here's an example of calling the same function:

```
popupalert();
```

For the other example, this is how you may call it:

```
<A HREF="#top" onClick="anotherAlert('top')">top</A>
```

This would bring you to the top of the page, and bring up an alert box that said: "top is the word you clicked on" Try it for yourself: top

Here is the same example with multiple parameters that was shown above:

```
<A HREF="#top" onClick="secondAlert('awesome','pandas')">awesome</A>

<A HREF="#top" onClick="thirdAlert('computers','insert')">computers</A>
```

You'll notice in the code that different values for the variables word and password are passed in. These values here are what the function will need to perform the actions in the function. Make sure that the values you pass in are in the correct order because the function will take them in and assign these values to the parameters in the parentheses of the function declaration. Once you pass values into your function, you can use them however you want within your function.

Try it for yourself:

When you click on the words below, a confirmation box will pop up and then the link will bring you to the top of the page.

awesome

computers

If/Else Statements

if statements execute a set of commands if a specified condition is true. If the condition is false, another set of statements can be executed through the use of the else keyword.

The main idea behind if statements is embodied by the sentence: "If the weather's good tomorrow, we'll go out and have a picnic and Lisa will do cartwheels -- else, we'll stay in and Catherine will watch TV." As you can see, the idea is quite intuitive and, surprisingly enough, so is the syntax:

```
if (condition) {
        statements1
}
-or-
if (condition) {
        statements1
}
else {
        statements2
}
```

(An **if** statement does not require an **else** statement following it, but an **else** statement must be preceded by an **if** statement.)

condition can be any JavaScript expression that evaluates to true or false. Parentheses are required around the condition. If *condition* evaluates to true, the statements in *statements1* are executed.

statements1 and *statements2* can be any JavaScript statements, including further nested **if** statements. Multiple statements must be enclosed in braces.

Here's an example:

```
if (weather == 'good') {
        go_out(we);
        have_a_picnic(we);
        do_cartwheels(Lisa);
}
else {
        stay_in(we);
```

```
        watch_TV(Catherine);
}
```

Loops

Loops are an incredibly useful programming tool. Loops handle repetitive tasks extremely well, especially in the context of consecutive elements. Arrays immediately spring too mind here, since array elements are numbered consecutively. It would be quite intuitive (and equally practical), for instance, to write a loop that added 1 to each element within an array. Don't worry if this doesn't make a lot of sense now, it will, after you finish reading the tutorial.

The two most common types of loops are for and while loops:

for Loops

A for loop constitutes a statement which consists of three expressions, enclosed in parentheses and separated by semicolons, followed by a block of statements executed in the loop.

This definition may, at first, sound confusing. Indeed, it is hard to understand for loops without seeing them in action.

A for loop resembles the following:

for (*initial-expression*; *condition*; *increment-expression*) {
        statements
}

The *initial-expression* is a statement or variable declaration. (See the section on variables for more information.) It is typically used to initialize a counter variable. This expression may optionally declare new variables with the var keyword.

The *condition* is evaluated on each pass through the loop. If this condition evaluates to true, the statements in *statements* are performed. When the condition evaluates to false, the execution of the for loop stops. This conditional test is optional. If omitted, the condition always evaluates to true.

The *increment-expression* is generally used to update or increment the counter variable.

The *statements* constitute a block of statements that are executed as long as *condition* evaluates to true. This can be a single statement or multiple statements. Although not required, it is good practice to indent these statements from the beginning of the for statement to make your code more readable.

Check out the following for statement. It starts by declaring the variable *i* and initializing it to zero. It checks whether *i* is less than nine, performs the two successive statements, and increments *i* by one after each pass through the loop:

```
var n = 0;
for (var i = 0; i < 3; i++) {
        n += i;
        alert("The value of n is now " + n);
}
```

while Loops

The while loop, although most people would not recognize it as such, is for's twin. The two can fill in for one another - using either one is only a matter of convenience or preference according to context. while creates a loop that evaluates an expression, and if it is true, executes a block of statements. The loop then repeats, as long as the specified condition is true.

The syntax of while differs slightly from that of for:

while (*condition*) {
        statements
}

condition is evaluated before each pass through the loop. If this condition evaluates to true, the statements in the succeeding block are performed. When *condition* evaluates to false, execution continues with the statement following *statements*.

statements is a block of statements that are executed as long as the *condition* evaluates to true. Although not required, it is good practice to indent these statements from the beginning of the statement. The following while loop iterates as long as *n* is less than three.

```
var n = 0;
var x = 0;
while(n < 3) {
        n++;
        x += n;
        alert("The value of n is " + n + ". The value of x is " + x);
}
```

Try it for yourself: Click this link

Commenting

Comments allow you to write notes to yourself within your program. These are important because they allow someone to browse your code and understand what the various functions do or what your variables represent. Comments also allow you to understand your code if it's been a while since you last looked at it.

In JavaScript, you can write both one-line comments and multiple line comments. The notation for each is different though. For a one line comment, you precede your comment with //. This indicates that everything written on that line, after the //, is a comment and the program should disregard it.

For a multiple-line comment, you start with /* and end with */ . It is nice to put an * at the beginning of each line just so someone perusing your code realizes that he/she is looking at a comment (if it is really long this helps). This is not necessary though.

The following are examples of comments in JavaScript.

```
// This is a single line comment.

/* This is a multiple line comment with only one line. */

/* This is a multiple line comment.
 * The star (*) at the beginning of this line is optional.
 * So is the star at the beginning of this line. */
```

JavaScript program using objects

```
<html>
<head>
<script language="javascript">
function demo1()
{
Popup("Hello");
Obj= new sample (2, 4);
alert(obj.x + obj.y);
}
function sample(x,y)
{
this.x=x;
this.y=y;
}
</script>
</head.
```

```
<body onLoad="demo1( )">
</body>
</html>
```

Regular Expression

A script language may take name data from a user and have to search through the string one character at a time. The usual approach in scripting language is to create a pattern called a regular expression which describes a set of characters that may be present in a string.

```
var pattern = "target";
var string = "can you find the target";
string.match(pattern);
```

But the above code can also be written using regular expression as a parameter, as shown below.

```
var pattern = new RegExp("target");
var string = "can you find the target";
pattern.exec(string);
```

Regular expression is a javascript object. Dynamic patterns are created using the keyword new.

```
regex = new RegExp("feroz | btech");
```

JavaScript code to implement RegExp

```
<html>
<head>
<body>
<script language="javascript">
var re = new RegExp("*A | a+mer");
var msg=" Have you met Btech recently";
var res= re.exec(msg);
if(res)
{
alert( " I found " + res*0+);
}
else
{
alert(" I didn't find it");
}
</script>
</body>
</html>
```

Functions:

Regular Expressions are manipulated using the functions which belong to either the RegExp or String class.

Class String functions

match(pattern)

This function searches a matching pattern. Returns array holding the results.

replace(pattern1, pattern2)

Searches for pattern1. If the search is successful pattern1 is replaced with pattern2.

search(pattern)

Searches for a pattern in the string. If the match is successful, the index of the start of the match is returned. If the search fails, the function returns -1.

Class RegExp functions

exec(string)

Executes a search for a matching pattern in its parameter string. Returns an array holding the results of the operation.

test(string)

Searches for a match in its parameter string. Returns true if a match is found, otherwise returns false.

Built in objects:

The document object

A document is a web page that is being either displayed or created. The document has a number of properties that can be accessed by JavaScript programs and used to manipulate the content of the page.

Write or writeln

Html pages can be created using JavaScript. This is done by using the write or writeln methods of the document object.

Document.write(“<body>”);

Document.write(“<h1> Hello </h1>”);

The form object

Two aspects of the form can be manipulated through JavaScript. First, most commonly and probably most usefully, the data that is entered onto your form can be checked at submission. Second you can actually build forms through JavaScript.

Example : Validate.js

```
function validate()
{
var t1=document.forms[0].elements;
var t2=parent.frames['f4'].document;
var bg1=t1.bg.value;
var c1=t1.c.value;
t2.open();
t2.write("<body bgcolor="+bg1+">");
t2.write("Candidate name is : "+c1);
t2.write("</body>");
t2.close();
}
```

Mypage.html

```
<html>
<head>
<script language = "javascript src= "D:\Documents and Settings \ p6 \ validate.js">
</script>
</head>
<body>
<form>
Background Color: <input type="text" size=16 name="bg" value="white">
Candidate's name:<input type="text" size=16 name="c">
```

<input type="button" value="showit" onClick="validate()">
</form>
</body>
</html>

The browser object

Some of the properties of the browser object is as follows

Navigator.appCodeName : The internal name for the browser.
Navigator.appVersion:This is the public name of the browser.
Navigator.appVersion:The version number, platform on which the browser is running.
Navigator.userAgent :The strings appCodeName and appVersion concatenated together.

The Date object

JavaScript provides functions to perform many different date manipulation. Some of the functions are mentioned below.
Date( ) : Construct an empty date object.

Date(year, month, day [,hour, minute, second]) :Create a new Date object based upon numerical values for the year, month and day. Optional time values may also be supplied. getDate( ):Return the day of the month

getDay( ):Return an integer representing the day of the week.
getFullYear( ):Return the year as a four digit number.
getHours( ):Return the hour field of the Date object.
getMinutes( ):Return the minutes field of the Date object.
getSeconds( ):Return the second field of the Date object.
setDate(day):Set the day value of the object. Accepts values in the range 1 to 31.
setFullYear(year [,month, day]):Set the year value of the object. Optionally also sets month and day values.
toString( ):Returns the Date as a string.

Events:

JavaScript is a event-driven system. Nothing happens unless it is initiated by an event outside the script. The table below shows event, event handler and the description about the event handler.
The following are events used with the elements

Event Attribute	Description	Tags/elements used
onblur	When the field lost focuses and Enters into another field usually By tag or mouse click	Area,button,input,select, textarea
onchange	Whenever the text or options are Modified	Input,select,textarea
onclick	On clicking the button or reset or submit etc	Most elements
ondblclick	Clicking twice	Most elements
onfocus	Got focus in the field or entering into the field	Area,button,input,select, textarea
Onkeydown	Key pressed down	Most elements
onkeyup	Pressing up	Most elements

onload	When the document is loaded	Body,frameset
Onmousedown	Moving mousedown	Most elements
Onmousemove	Moving mouse	Most element
Onmouseout	Mouse moved away	Mostelement
Onmouseover	Placingmouse on it	Most
Onreset	Clicking on reset button	Form
Onselect	Selecting text	Input.textarea
Onsubmit	Clicking on submit button	Form
Onunload	When unloaded from memory	Body ,frameset

Dynamic HTML with JavaScript

Data Validation

Data validation is the common process that takes place in the web sites. One common request is for a way of validating the username and password. Following program shows the validation of data which uses two frames, in one frame user is going to enter the data and in the other frame equivalent result is going to be displayed.

Example JavaScript code for data validation
Mypage.html

```
<html>
<head>
<title>frame page </title>
</head>
<frameset rows="20%,*">
<frame name="f1" src="">
<frameset cols="20%,*">
<frame name="f2" src="">
<frameset cols="50%,*">
<frame name="f3" src="D:\Documents and Settings\Btech\Desktop\btech\p6\reg.html">
<frame name="f4" src="D:\Documents and
Settings\Btech\Desktop\btech\p6\profile.html">
</frameset>
</frameset>
</frameset>
</html>
```

Myform.html

```
<html>
<head>
<script language = "javascript" src = "D:\ Documents and Settings \ Btech\ Desktop\btech\ p6\ validate.js">
</script>
</head>
<body>
<form>
Background Color: <input type="text" size=16 name="bg" value="white">
```

```
Candidate's name:<input type="text" size=16 name="c">
<input type="button" value="showit" onClick="validate()">
</form>
</body>
</html>
```

Validate.js

```
function validate()
{
var t1=document.forms[0].elements;
var t2=parent.frames['f4'].document;
var bg1=t1.bg.value;
var c1=t1.c.value;
t2.open();
t2.write("<body bgcolor="+bg1+">");
t2.write("Candidate name is : "+c1);
t2.write("</body>");
t2.close();
}
```