



JAVASCRIPT

Tutorial By

Hafizur Rahman

Lecturer

Daffodil Institute of IT

Hafizur Rahman

hafizur.kl@diit.info

01739-981172

Javascript - Intro

What is JavaScript?

- JavaScript was designed to add interactivity to HTML pages
- JavaScript is a scripting language
- A scripting language is a lightweight programming language
- JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license

JavaScript – Syntax

```
<html>  
<body>  
<script type="text/javascript">  
...  
</script>  
</body>  
</html>
```

JavaScript – Statements

Unlike HTML, JavaScript is case sensitive - therefore watch your capitalization closely when you write JavaScript statements, create or call variables, objects and functions.

A JavaScript statement is a command to a browser. The purpose of the command is to tell the browser what to do.

JavaScript code (or just JavaScript) is a sequence of JavaScript statements.

Each statement is executed by the browser in the sequence they are written.

This example will write a heading and two paragraphs to a web page:

```
<html>
```

```
<body>
```

```
<script type="text/javascript">
```

```
document.write("<h1>This is a heading</h1>");
```

```
document.write("<p>This is a paragraph.</p>");
```

```
document.write("<p>This is another paragraph.</p>");
```

```
</script>
```

</body>

</html>

JavaScript statements can be grouped together in blocks.

Blocks start with a left curly bracket { and end with a right curly bracket }.

The purpose of a block is to make the sequence of statements execute together.

This example will write a heading and two paragraphs to a web page:

```
<html>
```

```
<body>
```

```
<script type="text/javascript">
```

```
{
```

```
document.write("<h1>This is a heading</h1>");
```

```
document.write("<p>This is a paragraph.</p>");
```

```
document.write("<p>This is another paragraph.</p>");
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

JavaScript Comments

1. Singl Line Comments
2. Multiline comments

Single Line Comments Examples:

```
<html>
```

```
<head>
```

```
<title>A Simple Page</title>
```

```
<script language="javascript">
```

```
<!--
```

```
// The first alert is below
```

```
alert("An alert triggered by JavaScript!");
```

```
// Here is the second alert
```

```
alert("A second message appears!");
```

```
// -->
```

```
</script>
```

```
</head>
```

```
<body>
```

```
</body>
```

```
</html>
```

Multiline comments Examples:

```
<html>
```

```
<head>
```

```
<title>A Simple Page</title>
```

```
<script language="javascript">
```

```
<!--
```

```
/*
```

Below two alert() methods are used to

fire up two message boxes - note how the

second one fires after the OK button on the

first has been clicked

```
*/
```

```
alert("An alert triggered by JavaScript!");
```

```
alert("A second message appears!");
```

```
// -->
```

```
</script>
```

```
</head>
```

```
<body>
```

```
</body>
```

```
</html>
```

JavaScript Keywords

Keywords are reserved and cannot be used as variable or function names.

Here is the complete list of JavaScript keywords:

break	else	New	var
case	finally	return	void
catch	for	switch	while
continue	function	This	with
default	if	throw	
delete	in	try	
do	instanceof	typeof	

JavaScript Reserved Words

The reserved words are words that are reserved for future use as keywords.

The reserved words cannot be used as variable or function names.

The complete list of reserved words is as follows:

abstract	enum	int	short
boolean	Export	interface	static
byte	extends	long	super
char	final	native	synchronized

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

class	float	package	throws
const	goto	private	transient
debugger	implements	protected	volatile
double	Import	public	

JavaScript Semicolon

JavaScript executes your code accordingly even if you forget a semicolon at the end of the line.

You must use a semicolon to separate the two pieces of code when putting two independent pieces of code on one line.

```
<html>
<body>
<SCRIPT LANGUAGE="JavaScript">

<!--
// Declare 2 numeric variables on the same line
var varA = 5; var varB = 0.06;

;
document.write(varA*varB);
-->

</SCRIPT>
</body>
</html>
```

JavaScript Enabling

All the modern browsers come with built-in support for JavaScript. Many times you may need to enable or disable this support manually.

This tutorial will make you aware the procedure of enabling and disabling JavaScript support in your browsers : Internet Explorer, Firefox and Opera.

JavaScript in Internet Explorer:

Here are simple steps to turn on or turn off JavaScript in your Internet Explorer:

1. Follow **Tools-> Internet Options** from the menu
2. Select **Security** tab from the dialog box
3. Click the **Custom Level** button
4. Scroll down till you find **Scripting option**
5. Select *Enable* radio button under **Active scripting**
6. Finally click OK and come out

Hafizur Rahman
hafizur.kl@diit.info
 01739-981172

To disable JavaScript support in your Internet Explorer, you need to select *Disable* radio button under **Active scripting**.

JavaScript in Firefox:

Here are simple steps to turn on or turn off JavaScript in your Firefox:

1. Follow **Tools-> Options**
from the menu
2. Select **Content** option from the dialog box
3. Select *Enable JavaScript* checkbox
4. Finally click OK and come out

To disable JavaScript support in your Firefox, you should not select *Enable JavaScript* checkbox.

JavaScript in Opera:

Here are simple steps to turn on or turn off JavaScript in your Opera:

1. Follow **Tools-> Preferences**
from the menu
2. Select **Advanced** option from the dialog box
3. Select **Content** from the listed items
4. Select *Enable JavaScript* checkbox
5. Finally click OK and come out

To disable JavaScript support in your Opera, you should not select *Enable JavaScript* checkbox.

Warning for Non-JavaScript Browsers:

If you have to do something important using JavaScript then you can display a warning message to the user using `<noscript>` tags.

You can add a *noscript* block immediately after the script block as follows:

```
<html>
<body>

<script language="javascript" type="text/javascript">
<!--
    document.write("Hello World!")
//-->
</script>
```

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

```
<noscript>
  Sorry...JavaScript is needed to go ahead.
</noscript>
</body>
</html>
```

Now, if user's browser does not support JavaScript or JavaScript is not enabled then message from `</noscript>` will be displayed on the screen.

JavaScript Location/Placement

There is a flexibility given to include JavaScript code anywhere in an HTML document. But there are following most preferred ways to include JavaScript in your HTML file.

- Script in `<head>...</head>` section.
- Script in `<body>...</body>` section.
- Script in `<body>...</body>` and `<head>...</head>` sections.
- Script in an external file and then include in `<head>...</head>` section.

In the following section we will see how we can put JavaScript in different ways:

JavaScript in `<head>...</head>` section:

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows:

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
    alert("Hello World")
}
//-->
</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

JavaScript in `<body>...</body>` section:

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

If you need a script to run as the page loads so that the script generates content in the page, the script goes in the <body> portion of the document. In this case you would not have any function defined using JavaScript:

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello World")
//-->
</script>
<p>This is web page body </p>
</body>
</html>
```

This will produce following result:

```
Hello World
This is web page body
```

JavaScript in <body> and <head> sections:

You can put your JavaScript code in <head> and <body> section altogether as follows:

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
    alert("Hello World")
}
//-->
</script>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello World")
//-->
</script>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

This will produce following result:

```
Hello World
```

JavaScript in External File :

As you begin to work more extensively with JavaScript, you will likely find that there are cases where you are reusing identical JavaScript code on multiple pages of a site.

You are not restricted to be maintaining identical code in multiple HTML files. The *script* tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

Here is an example to show how you can include an external JavaScript file in your HTML code using *script* tag and its *src* attribute:

```
<html>
<head>
<script type="text/javascript" src="filename.js" ></script>
</head>
<body>
.....
</body>
</html>
```

To use JavaScript from an external file source, you need to write your all JavaScript source code in a simple text file with extension ".js" and then include that file as shown above.

For example, you can keep following content in filename.js file and then you can use *sayHello* function in your HTML file after including filename.js file:

```
function sayHello() {
    alert("Hello World")
}
```

JavaScript Variable

Do You Remember Algebra From School?

Do you remember algebra from school? $x=5$, $y=6$, $z=x+y$

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

Do you remember that a letter (like x) could be used to hold a value (like 5), and that you could use the information above to calculate the value of z to be 11?

These letters are called **variables**, and variables can be used to hold values ($x=5$) or expressions ($z=x+y$).

JavaScript Variables

As with algebra, JavaScript variables are used to hold values or expressions.

A variable can have a short name, like x, or a more descriptive name, like carname.

Rules for JavaScript variable names:

- Variable names are case sensitive (y and Y are two different variables)
- Variable names must begin with a letter, the \$ character, or the underscore character

Note: Because JavaScript is case-sensitive, variable names are case-sensitive.

Example

A variable's value can change during the execution of a script. You can refer to a variable by its name to display or change its value.

```
<html>
<body>

<script type="text/javascript">
var firstname;
firstname="Hege";
document.write(firstname);
document.write("<br />");
firstname="Tove";
document.write(firstname);
</script>
```

<p>The script above declares a variable, assigns a value to it, displays the value, changes the value, and displays the value again.</p>

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

Declaring (Creating) JavaScript Variables

Creating variables in JavaScript is most often referred to as "declaring" variables.

You declare JavaScript variables with the **var** keyword:

```
var x;  
var carname;
```

After the declaration shown above, the variables are empty (they have no values yet).

However, you can also assign values to the variables when you declare them:

```
var x=5;  
var carname="Volvo";
```

After the execution of the statements above, the variable **x** will hold the value **5**, and **carname** will hold the value **Volvo**.

Note: When you assign a text value to a variable, put quotes around the value.

Note: If you redeclare a JavaScript variable, it will not lose its value.

<html>

<body>

<script type="text/javascript">

```
var money;
```

```
var name;
```

```
money = 2000.50;
```

```
name = "Ali";
```

```
document.write(money);
```

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

```
document.write(name);
```

</script>

<p>Set the variables to different values and then try...</p>

</body>

</html>

Local JavaScript Variables

A variable declared within a JavaScript function becomes **LOCAL** and can only be accessed within that function. (the variable has local scope).

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

Local variables are deleted as soon as the function is completed.

Global JavaScript Variables

Variables declared outside a function become **GLOBAL**, and all scripts and functions on the web page can access it.

Global variables are deleted when you close the page.

Assigning Values to Undeclared JavaScript Variables

If you assign values to variables that have not yet been declared, the variables will automatically be declared as global variables.

These statements:

```
x=5;  
carname="Volvo";
```

will declare the variables x and carname as global variables (if they don't already exist).

More About JavaScript Variable Scope

A variable can be either global or local in JavaScript.

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

All variables are global unless they are declared in a function.

Variables declared in a function are local to that function.

It is possible for two variables with the same name to exist if one is global and the other is local.

When accessing the variable within the function, you are accessing the local variable.

If the variable is accessed outside the function, the global variable is used.

Always use the var keyword to declare local variables in functions.

Without var, JavaScript will create a global variable.

Variable names are case sensitive (y and Y are two different variables)

Variable names must begin with a letter or the underscore character

Variables in JavaScript are defined by using the var operator (short for variable), followed by the variable name, such as:

```
var test = "hi";
```

In this example, the variable test is declared and given an initialization value of "hi" (a string).

You can also define two or more variables using the same var statement:

```
var test = "hi", test2 = "hola";
```

Variables using the same var statement don't have to be of the same type:

```
var test = "hi", age = 25;
```

variables in JavaScript do not require initialization:

```
var test;
```

A variable can be initialized with a string value, and later on be set to a number value.

```
var test = "hi";  
alert(test);  
test = 55;  
alert(test);
```

Code blocks indicates a series of statements that should be executed in sequence.

Code blocks are enclosed between an opening brace ({} and a closing brace ({}).

```
if (test1 == "red") {  
    test1 = "blue";  
    alert(test1);  
}
```

Example of JavaScript Local and Global Variables:

```
<SCRIPT LANGUAGE="JavaScript">  
  
<!--  
  
function function_1 ()  
  
{  
  
    var x = 5;  
  
    document . write ("<br>Inside function_1, x=" + x);  
  
    function_2 ();  
  
    document . write ("<br>Inside function_1, x=" + x);  
  
    }  
  
    function function_2 () {  
  
        document . write ("<br>Inside function_2, x=" + x);  
  
        }  
  
        x = 1;  
  
        document . write ("<br>Outside, x=" + x);  
  
        function_1 ();  
  
        document . write ("<br>Outside, x=" + x)
```

-->

</SCRIPT>

JavaScript Operators

What is an operator?

Simple answer can be given using expression *4 + 5 is equal to 9*. Here 4 and 5 are called operands and + is called operator. JavaScript language supports following type of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Lets have a look on all operators one by one.

The Arithmetic Operators:

There are following arithmetic operators supported by JavaScript language:

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Note: Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

```
<html>
```

```
<body>
```

```
<script type="text/javascript">
```

```
<!--
```

```
var a = 33;
```

```
var b = 10;
```

```
var c = "Test";
```

```
var linebreak = "<br />";
```

```
document.write("a + b = ");
```

```
result = a + b;
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("a - b = ");
```

```
result = a - b;
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("a / b = ");
```

```
result = a / b;
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("a % b = ");  
  
result = a % b;  
  
document.write(result);  
  
document.write(linebreak);
```

```
document.write("a + b + c = ");  
  
result = a + b + c;  
  
document.write(result);  
  
document.write(linebreak);
```

```
a = a++;  
  
document.write("a++ = ");  
  
result = a++;  
  
document.write(result);  
  
document.write(linebreak );
```

```
b = b--;  
  
document.write("b-- = ");  
  
result = b--;  
  
document.write(result);  
  
document.write(linebreak);
```

//-->

</script>

<p>Set the variables to different values and then try...</p>

</body>

</html>

The Comparison Operators:

There are following comparison operators supported by JavaScript language

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

```
<html>
```

```
<body>
```

```
<script type="text/javascript">
```

```
<!--
```

```
var a = 10;
```

```
var b = 20;
```

```
var linebreak = "<br />";
```

```
document.write("(a == b) => ");
```

```
result = (a == b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("(a < b) => ");
```

```
result = (a < b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("(a > b) => ");
```

```
result = (a > b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("(a != b) => ");
```

```
result = (a != b);
```

```
document.write(result);  
  
document.write(linebreak);  
  
  
document.write("(a >= b) => ");  
  
result = (a >= b);  
  
document.write(result);  
  
document.write(linebreak);
```

```
document.write("(a <= b) => ");  
  
result = (a <= b);  
  
document.write(result);  
  
document.write(linebreak);
```

```
//-->  
  
</script>
```

<p>Set the variables to different values and different operators and then try...</p>

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

The Logical Operators:

There are following logical operators supported by JavaScript language

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then then condition	(A && B) is true.

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

	becomes true.	
	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false.


```
<html>
```

```
<body>
```

```
<script type="text/javascript">
```

```
<!--
```

```
var a = true;
```

```
var b = false;
```

```
var linebreak = "<br />";
```

```
document.write("(a && b) => ");
```

```
result = (a && b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("(a || b) => ");
```

```
result = (a || b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("! (a && b) => ");
```

```
result = (!(a && b));
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
//-->
```

```
</script>
```

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>

The Bitwise Operators:

There are following bitwise operators supported by JavaScript language

Assume variable A holds 2 and variable B holds 3 then:

Operator	Description	Example
&	Called Bitwise AND operator. It performs a Boolean AND operation on each bit of its integer arguments.	(A & B) is 2 .
	Called Bitwise OR Operator. It performs a Boolean OR operation on each bit of its integer arguments.	(A B) is 3.
^	Called Bitwise XOR Operator. It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.	(A ^ B) is 1.
~	Called Bitwise NOT Operator. It is a unary operator and operates by reversing all bits in the operand.	(~B) is -4 .
<<	Called Bitwise Shift Left Operator. It moves all bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying by 2, shifting two positions is equivalent to multiplying by 4, etc.	(A << 1) is 4.

>>	<p>Called Bitwise Shift Right with Sign Operator. It moves all bits in its first operand to the right by the number of places specified in the second operand. The bits filled in on the left depend on the sign bit of the original operand, in order to preserve the sign of the result. If the first operand is positive, the result has zeros placed in the high bits; if the first operand is negative, the result has ones placed in the high bits. Shifting a value right one place is equivalent to dividing by 2 (discarding the remainder), shifting right two places is equivalent to integer division by 4, and so on.</p>	(A >> 1) is 1.
>>>	<p>Called Bitwise Shift Right with Zero Operator. This operator is just like the >> operator, except that the bits shifted in on the left are always zero,</p>	(A >>> 1) is 1.

```
<html>
```

```
<body>
```

```
<script type="text/javascript">
```

```
<!--
```

```
var a = 2; // Bit presentation 10
```

```
var b = 3; // Bit presentation 11
```

```
var linebreak = "<br />";
```

```
document.write("(a & b) => ");
```

```
result = (a & b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("(a | b) => ");
```

```
result = (a | b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("(a ^ b) => ");
```

```
result = (a ^ b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("(~b) => ");
```

```
result = (~b);
```

```
document.write(result);  
document.write(linebreak);
```

```
document.write("(a << b) => ");  
result = (a << b);  
document.write(result);  
document.write(linebreak);
```

```
document.write("(a >> b) => ");  
result = (a >> b);  
document.write(result);  
document.write(linebreak);
```

```
//-->  
</script>
```

<p>Set the variables to different values and different operators and then try...</p>

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

The Assignment Operators:

There are following assignment operators supported by JavaScript language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assigne value of A + B into C

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	$C += A$ is equivalent to $C = C + A$
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	$C -= A$ is equivalent to $C = C - A$
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	$C *= A$ is equivalent to $C = C * A$
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	$C /= A$ is equivalent to $C = C / A$
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	$C \% = A$ is equivalent to $C = C \% A$

Note: Same logic applies to Bitwise operators so they will become like $<<=$, $>>=$, $&=$, $|=$ and $\wedge=$.

<html>

<body>

<script type="text/javascript">

<!--

var a = 33;

var b = 10;

var linebreak = "
";

document.write("Value of a => (a = b) => ");

result = (a = b);

document.write(result);

document.write(linebreak);

document.write("Value of a => (a += b) => ");

result = (a += b);

document.write(result);

document.write(linebreak);

document.write("Value of a => (a -= b) => ");

result = (a -= b);

document.write(result);

document.write(linebreak);

document.write("Value of a => (a *= b) => ");

result = (a *= b);

```
document.write(result);  
document.write(linebreak);
```

```
document.write("Value of a => (a /= b) => ");  
result = (a /= b);  
document.write(result);  
document.write(linebreak);
```

```
document.write("Value of a => (a %= b) => ");  
result = (a %= b);  
document.write(result);  
document.write(linebreak);
```

```
//-->  
</script>
```

```
<p>Set the variables to different values and different operators and then try...</p>  
</body>  
</html>
```

Miscellaneous Operator

The Conditional Operator (? :)

There is an operator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation. The conditional operator has this syntax:

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

Operator	Description	Example
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

<html>

<body>

<script type="text/javascript">

<!--

var a = 10;

var b = 20;

var linebreak = "
";

document.write("((a > b) ? 100 : 200) => ");

result = (a > b) ? 100 : 200;

document.write(result);

document.write(linebreak);

document.write("((a < b) ? 100 : 200) => ");

result = (a < b) ? 100 : 200;

document.write(result);

document.write(linebreak);

//-->

</script>

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>

The *typeof* Operator

The *typeof* is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is the list of return values for the *typeof* Operator :

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

<html>

<body>

<script type="text/javascript">

<!--

var a = 10;

var b = "String";

var linebreak = "
";

result = (typeof b == "string" ? "B is String" : "B is Numeric");

document.write("Result => ");

document.write(result);

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

```

document.write(linebreak);

result = (typeof a == "string" ? "A is String" : "A is Numeric");
document.write("Result => ");
document.write(result);
document.write(linebreak);

//-->
</script>

<p>Set the variables to different values and different operators and then try...</p>
</body>
</html>

```

JavaScript Conditionals Statement

1. if..... else statement

The format of a simple if statement looks like the following:

```

if (expression)
    statement;

```

If the 'expression' in parentheses evaluates to true, the statement is executed; otherwise, the statement is skipped.

If two or more lines of code are to be executed, curly braces { } must be used to designate what code belongs in the if statement.

The keyword else extends the functionality of the basic if statement by providing other alternatives.

The format of an if...else combination looks like the following:

```

if (expression)
    statement1;
else
    statement2;

```

if the expression evaluates to true, statement1 is executed, otherwise, statement2 is executed.

```

<html>
<SCRIPT LANGUAGE='JavaScript'>
  <!--
    var varA = 2;

```

Hafizur Rahman
hafizur.kl@diit.info
 01739-981172

```

var varB = 5;

if (varA == 0)
    document.write("varA == 0");
else {
    if ((varA * 2) >= varB)
        document.write("(varA * 2) >= varB");
    else
        document.write("(varA * 2) < varB");
    document.write(varB);
}
//-->
</SCRIPT>
</html>

```

Another Example:

```

<html>
<body>
<script language="JavaScript">
<!--
var temp = 1;

if(temp == true){
    document.write("Statement is true");
} else{
    document.write("Statement is false");
}

-->
</script>
</body>
</html>

```

2. for Loops....

The for loop is a structure that loops for a preset number of times.

The for loop is made up of two parts: condition and statement.

The condition structure determines how many times the loop repeats.

The statement is what is executed every time the loop occurs.

The condition structure is contained within parentheses and is made up of three parts.

Each part is separated by a semicolon (;).

The first part of the condition structure initializes a variable to a starting value.

In most cases, the variable is declared within this section as well as initialized.

The second part is the conditional statement.

The third and final part determines how the variable should be changed each time the loop is iterated.

The format of the for loop looks like the following:

```
for (initialize; condition; adjust) {
    statement;
}

<html>
<SCRIPT LANGUAGE='JavaScript'>
    <!--
    document.write("<H2>Multiplication table for 4</H2>");

    for (var aNum = 0; aNum <= 10; aNum++)
    {
        document.write("4 X ", aNum, " = ", 4*aNum, "<BR>");
    }
    //-->
</SCRIPT>
</html>
```

Or

```
<html>
<SCRIPT LANGUAGE='JavaScript'>
    <!--
    document.write("<H2>Multiplication table for 4</H2>");

    for (var aNum = 0; aNum <= 10;)
    {
        document.write("4 X ", aNum, " = ", 4*aNum, "<BR>");
        aNum++;
    }
    //-->
</SCRIPT>
</html>
```

Another for loop example

```
<HTML>
<HEAD>
<TITLE>
Using the JavaScript for Loop
</TITLE>
</HEAD>

<BODY>
```

```

<H1>Using the JavaScript for Loop</H1>
<SCRIPT LANGUAGE="JavaScript">
<!--
var i, total
total = 0
for (i = 1; i < 5; i++) {
    total += i
    document.write("Loop iteration " + i + " (Cumulative total = " + total + ")<BR
>")
}
// -->
</SCRIPT>
</BODY>
</HTML>

```

Reversed for loop example

```

<html>
<head>
<title>A Simple Page</title>
<script language="JavaScript">
<!--
for (x = 10; x >= 0; x = x - 2)
{
    document.write(x+" ");
}
// -->
</script>
</head>
<body>

</body>
</html>

```

JavaScript Switch

JavaScript offers the switch statement as an alternative to using the if...else structure.

The switch statement is useful when testing all the possible results of an expression.

The format of a switch structure looks like the following:

```

switch (expression)
{
    case label1:

```

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

```

        statement1;
    break;
case label2:
    statement2;
    break;
default:
    statement3;
}

```

The switch statement evaluates an expression placed between parentheses.

The result is compared to labels associated with case structures that follow the switch statement.

If the result is equal to a label, the statement(s) in the corresponding case structure are executed.

A default is used at the end of a switch structure to catch results that do not match any of the case labels.

A colon always follows a label.

Curly brackets { } are used to hold all the case structures together, but they are not used within a case structure.

The keyword break is used to break out of the entire switch statement once a match is found, thus preventing the default structure from being executed accidentally.

Example 1

```

<html>
<SCRIPT LANGUAGE='JavaScript'>
<!--
    var color = "green";
    switch (color)
    {
        case "red":
            document.write("The car is red.");
            break;
        case "blue":
            document.write("The car is blue.");
            break;
        case "green":
            document.write("The car is green.");
            break;
        default:
            document.write("The car is purple.");
    }
    //-->
</SCRIPT>
</html>

```

JavaScript While Loop

The while loop can be summarized as while the expression evaluates to true, execute the statements in the loop.

Once the last statement in the loop is executed, go back to the top of the loop and evaluate the expression again.

When the expression evaluates to false, the next line of code following the while loop structure is executed.

Because the expression is evaluated before the loop, it is possible the loop will never be executed.

The format of the while loop looks like the following:

```
while (expression)
{
    statement;
}
```

Example 1

```
<html>
  <SCRIPT LANGUAGE='JavaScript'>
  <!--
    var counter = 1;
    document.write("While loop is Starting<br>");
    while (counter <= 5)
    {
      document.write("Counter Now",counter,"<BR>");
      counter++; //Next car
    }
    document.write("While loop is finished!");
  //-->
  </SCRIPT>
</html>
```

JavaScript Do While Loop

do...while loop always executes the loop once before evaluating the expression for the first time.

This difference is seen in the following format:

Hafizur Rahman
hafizur.kl@diit.info
01739-981172


```
do
{
    statement;
}
while (expression);
```

Once the loop has executed for the first time, the expression is evaluated.

If true, the loop is executed again.

When the expression evaluates to false, the next line of code following the while structure is executed.

A semicolon (;) must be placed after the rightmost parenthesis.

Example 1:

```
<html>
<SCRIPT LANGUAGE='JavaScript'>
<!--
var counter = 1;
do
{
    document.write(counter + "<BR>");
    counter++;
}
while (counter < 5);
//-->
</SCRIPT>
</html>
```

Example 2:

```
<html>
<head>
<title>A Simple Page</title>
<script language="JavaScript">
<!--
var x = 1
do
{
    ++x;
    document.write(x+" ");
}
while (x < 10);

// -->
</script>
</head>
<body>
```

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

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

JavaScript Continue

The continue statement forces the execution of the code to continue at the beginning of the loop.

When a label is used, JavaScript immediately jumps to the beginning of the loop designated by a label and begins executing code.

Example 1:

```
<html>
<head>
<title>A Simple Page</title>
<script language="JavaScript">
<!--
var x = 0;
while (x < 10)
{
    x++;
    document.write(x);
    continue;
    document.write("You never see this!");
}
// -->
</script>
</head>
<body>

</body>
</html>
```

Example 2:

```
<html>
<head>
<title>A Simple Page</title>
<script language="JavaScript">
<!--
var x = 0;
while (x < 10) {
    x++;
    if (x == 5) {
        continue;
    }
}
```

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

```
//break;
}
document.write(x);
}

// -->
</script>
</head>
<body>

</body>
</html>
```

JavaScript Array

Array Constructors:

```
var variable = new Array()
var variable = new Array(int)
var variable = new Array(arg1, ..., argN)
```

These constructors create a new array and initialize the Array object based on the arguments passed in the parameter list.

The constructor that has no arguments sets the length property to 0.

int-- An array is created and its length property is set to the value int.

arg1,...argN--An array is created and the array is populated with the arguments. The array length property is set to the number of arguments in the parameter list.

The newly created array is returned from the constructor

Use Array's Constructor

Arrays stores multiple data based on a numbered position into one storage structure.

The index starts at 0 and goes up.

JavaScript supports having arrays within arrays, called multidimensional arrays.

One-Dimensional array

To create an instance of an array, you use the new operator along with the Array object.

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

There are four ways to declare an array.

First, an empty array can be created by leaving the constructor parameters empty:

```
var x = new Array();
```

The second way to create an array is to fill in the constructor parameters with the array elements.

An array can contain elements of various types:

```
var x = new Array("A", "BB", "CCC", 1, 5, 8);
```

The third way to create an array is to fill in the constructor parameter with the size of the array.

This initializes the array to hold the number of elements specified, but does not specify the actual elements.

```
var x = new Array(6);
```

The fourth way to create an array is to use the array square brackets to fill in the array elements directly.

```
var x = ["A", "BB", "CCC", 1, 5, 8];
```

Creating an Array and Accessing Its Elements:

```
<html>
  <h2>Creating and Accessing Arrays</h2>
  <script language="JavaScript">
    <!--
    numArray = new Array(4,6,3);

    document.write("[0]="+numArray[0]+<br>");
    document.write("[1]="+numArray[1]+<br>");
    document.write("[2]="+numArray[2]);
    //End Hide-->
  </script>
</html>
```

Create an array using the Object() constructor:

```
<html>
<SCRIPT LANGUAGE="JavaScript">
<!--
```

```

var myArray = Object();
myArray.length=3; //array position zero
myArray[1]="A";
myArray[2]="B";
myArray[3]="C";

document.write("The myArray holds: ");

for(x=1; x<=myArray.length; x++) {
    document.write(myArray[x]," ");
}
-->
</SCRIPT>
</html>

```

Array declaration with element count:

```

<html>
<head>
<title>A Simple Page</title>
<script language="javascript">
<!--
var days_of_week = new Array(7);
days_of_week[0] = "Sunday";
days_of_week[1] = "Monday";
days_of_week[2] = "Tuesday";
days_of_week[3] = "Wednesday";
days_of_week[4] = "Thursday";
days_of_week[5] = "Friday";
days_of_week[6] = "Saturday";
var x = 2;
alert(days_of_week[x]);
// -->
</script>
</head>
<body>

</body>
</html>

```

Initialize a string array during declaration and check the array size:

```

<html>
<head>
<title>A Simple Page</title>
<script language="javascript">

```

```

<!--
var days_of_week = new
Array("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday");
alert(days_of_week.length);
// -->
</script>
</head>
<body>

</body>
</html>

```

Define two arrays and use for statement to output their values:

```

<html>
<head>
<title>Define two arrays and use for statement to ouput their values</title>

</head>
<body>

<h1>Define two arrays and use for statement to ouput their values</h1>

<table border="1" width="200">

<script language="javascript" type="text/javascript">
<!--
var myArray  = new Array();
    myArray[0] = "A";
    myArray[1] = "B";
    myArray[2] = "C";
    myArray[3] = "D";
    myArray[4] = "E";

var myArray2  = new Array();
    myArray2[0] = "1";
    myArray2[1] = "2";
    myArray2[2] = "3";
    myArray2[3] = "4";
    myArray2[4] = "5";

for (var i=0; i<myArray.length; i++) {
    document.write("<tr><td>" + myArray[i] + "</td>");
    document.write("<td>" + myArray2[i] + "</td></tr>");
}

//-->

```

```
</script>
```

```
</table>
```

```
</body>
```

```
</html>
```

Store strings in array:

```
<html>
```

```
<head>
```

```
<title>Store strings in array</title>
```

```
</head>
```

```
<body>
```

```
<P>
```

```
<script language="javascript" type="text/javascript">
```

```
<!--
```

```
var myArray = new Array();
```

```
myArray[0] = "AAA";
```

```
myArray[1] = "BBB";
```

```
myArray[2] = "CCC";
```

```
for (var i=0; i<myArray.length; i++) {
```

```
    document.write("Element " +i+ " contains: " +myArray[i]+ "<br />");  
}
```

```
//-->
```

```
</script>
```

```
</p>
```

```
</body>
```

```
</html>
```

JavaScript Array Sorting

Imagine that you wanted to sort an array alphabetically before you wrote the array to the browser. Well, this code has already been written and can be accessed by using the Array's *sort* method.

```
<script type="text/javascript">
```

```
<!--
```

```
var myArray2= new Array();
```

```
myArray2[0] = "Football";
```

```
myArray2[1] = "Baseball";
```

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

```
myArray2[2] = "Cricket";

document.write(myArray2.sort());

//-->
</script>
```

Output:

Baseball,Cricket,Football

Get Array length:

```
<html>
<head>
<title>Get Array length</title>

<script language="javascript" type="text/javascript">
<!--

var x = new Array(2,3,4);

alert(x.length);

//-->
</script>

</head>
<body>

</body>
</html>
```

JavaScript Alert

JavaScript Alert Example 1:

```
<HTML>
<HEAD>
<TITLE>Using alert Boxes</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function displayer()
{
    alert("Hello from JavaScript!")
}
}
```

Hafizur Rahman
hafizur.kl@diit.info
01739-981172


```

    //-->
  </SCRIPT>
</HEAD>

<BODY>
  <H1>Using alert Boxes</H1>
  <FORM>
    <INPUT TYPE="BUTTON" ONCLICK="displayer()" VALUE="Click Me!">
  </FORM>
</BODY>
</HTML>

```

JavaScript Prompt

```

<html>
<head>
<script type="text/javascript">
<!--
function prompter() {
var reply = prompt("Hey there, good looking stranger! What's your name?", "")
alert ( "Nice to see you around these parts " + reply + "!")
}
//-->
</script>
</head>
<body>
<input type="button" onclick="prompter()" value="Say my name!">
</body>
</html>

```

JavaScript Confirm function

```

<html>
<head>
<title>A Simple Page</title>
<script language="javascript">
<!--
confirm("Which one will you choose?")
// -->
</script>
</head>
<body>

```

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

Confirm dialog returns boolean value:

```
<html>
<head>
<title>A Simple Page</title>
<script language="JavaScript">
<!--
var response = confirm("Do you want to proceed with this book? Click OK to proceed otherwise click Cancel.");
alert(response)
// -->
</script>
</head>
<body>

</body>
</html>
```

Use Confirm dialog in if statement:

```
<html>
<head>
<script Language="Javascript" type = "text/javascript">
<!--
var newurl

function CheckRequest(newurl)
{
    if (confirm("Do you want to visit " + newurl + " site. ")) {
        return true
    } else {
        return false
    }
}

//-->
</script>
<title>Capturing Links</title>
</head>
<P><A href="http://www.google.com" onClick = "return CheckRequest('java2s')">Java</A></P>
<P><A href="http://www.netscape.com" onClick = "return
CheckRequest('Netscape')">Netscape</A></P>
</body>
</html>
```

Use confirm method in if statement:

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

```
<script language="JavaScript" type = "text/javascript">
<!--
var username = prompt("Type your name:", "");

if (confirm("Your name is: " + username + ". Is that correct?") == true )
{
    alert("Hello " + username);
}
else
{
    alert("Hi");
}
//-->
</script>
</head>
</html>
```

JavaScript For...In

The for...in statement loops through the properties of an object.

Syntax

```
for (variable in object)
{
    code to be executed
}
```

Note: The code in the body of the for...in loop is executed once for each property.

Example

```
<html>

<body>

<script type="text/javascript">

var person={ fname:"John",lname:"Doe",age:25 };

var x;

for (x in person)

{

document.write(person[x] + " ");

}

</script>

</body>

</html>
```

JavaScript Try...Catch

JavaScript - Catching Errors

When browsing Web pages on the internet, we all have seen a JavaScript alert box telling us there is a runtime error and asking "Do you wish to debug?". Error message like this may be useful for developers but not for users. When users see errors, they often leave the Web page.

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

This chapter will teach you how to catch and handle JavaScript error messages, so you don't lose your audience.

The try...catch Statement

The try...catch statement allows you to test a block of code for errors. The try block contains the code to be run, and the catch block contains the code to be executed if an error occurs.

Syntax

```
try
{
  //Run some code here
}
catch(err)
{
  //Handle errors here
}
```

Note that try...catch is written in lowercase letters. Using uppercase letters will generate a JavaScript error!

Examples

The example below is supposed to alert "Welcome guest!" when the button is clicked. However, there's a typo in the message() function. alert() is misspelled as adddler(). A JavaScript error occurs. The catch block catches the error and executes a custom code to handle it. The code displays a custom error message informing the user what happened:

Example

```
<html>
<head>
<script type="text/javascript">
var txt="";
function message()
{
try
{
  adddler("Welcome guest!");
}
catch(err)
{
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.message + "\n\n";
  txt+="Click OK to continue.\n\n";
}
```

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

```
    alert(txt);
  }
}
</script>
</head>

<body>
<input type="button" value="View message" onclick="message()" />
</body>

</html>
```

Events

onclick Event Type:

This is the most frequently used event type which occurs when a user clicks mouse left button. You can put your validation, warning etc against this event type.

Example:

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
    alert("Hello World")
}
//-->
</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

onsubmit event type:

Another most important event type is *onsubmit*. This event occurs when you try to submit a form. So you can put your form validation against this event type.

Here is simple example showing its usage. Here we are calling a *validate()* function before submitting a form data to the webserver. If *validate()* function returns true the form will be submitted otherwise it will not submit the data.

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

Example:

```
<html>
<head>
<script type="text/javascript">
<!--
function validation() {
    all validation goes here
    .....
    return either true or false
}
//-->
</script>
</head>
<body>
<form method="POST" action="t.cgi" onsubmit="return validate()">
.....
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

onmouseover and onmouseout:

These two event types will help you to create nice effects with images or even with text as well. The *onmouseover* event occurs when you bring your mouse over any element and the *onmouseout* occurs when you take your mouse out from that element.

Example:

Following example shows how a division reacts when we bring our mouse in that division:

```
<html>
<head>
<script type="text/javascript">
<!--
function over() {
    alert("Mouse Over");
}
function out() {
    alert("Mouse Out");
}
//-->
</script>
</head>
<body>
<div onmouseover="over()" onmouseout="out()">
<h2> This is inside the division </h2>
</div>
```

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

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.

JavaScript document.getElementById

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' />
```

A screenshot of a web form. It consists of a single-line text input field on the left and a button labeled "Form Checker" on the right. The button has a light blue border and a slight 3D effect.

document.getElementById returned a reference to our HTML element *myText*. We stored this reference into a variable, *myTextField*, and then used the *value* property that all input elements have to use to grab the value the user enters.

There are other ways to accomplish what the above script does, but this is definitely a straight-forward and browser-compatible approach.

Things to Remember About getElementById

When using the *getElementById* function, you need to remember a few things to ensure that everything goes smoothly. You always need to remember that *getElementById* is a method (or function) of the *document* object. This means you can only access it by using *document.getElementById*.

Also, be sure that you set your HTML elements' *id* attributes if you want to be able to use this function. Without an *id*, you'll be dead in the water.

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

If you want to access the text within a non-input HTML element, then you are going to have to use the *innerHTML* property instead of *value*. The next lesson goes into more detail about the uses of *innerHTML*.

JavaScript innerHTML

Ever wonder how you could change the contents of an HTML element? Maybe you'd like to replace the text in a paragraph to reflect what a visitor has just selected from a drop down box. By manipulating an element's *innerHTML* you'll be able to change your text and HTML as much as you like.

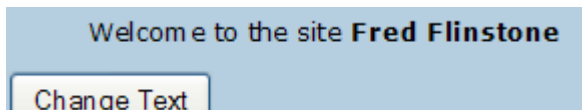
Changing Text with innerHTML

Each HTML element has an *innerHTML* property that defines both the HTML code and the text that occurs between that element's opening and closing tag. By changing an element's *innerHTML* after some user interaction, you can make much more interactive pages.

However, using *innerHTML* requires some preparation if you want to be able to use it easily and reliably. First, you must give the element you wish to change an id. With that *id* in place you will be able to use the *getElementById* function, which works on all browsers.

After you have that set up you can now manipulate the text of an element. To start off, let's try changing the text inside a bold tag.

```
<script type="text/javascript">
function changeText() {
    document.getElementById('boldStuff').innerHTML = 'Fred Flinstone';
}
</script>
<p>Welcome to the site <b id='boldStuff'>dude</b> </p>
<input type='button' onclick='changeText()' value='Change Text' />
```



You now know how to change the text in any HTML element, but what about changing the text in an element based on user input? Well, if we combine the above knowledge with a text input...

Updating Text Based on User Input

By adding a Text Input, we can take to updating our bold text with whatever the user types into the text input. Note: We updated the function a bit and set the id to *boldStuff2*.

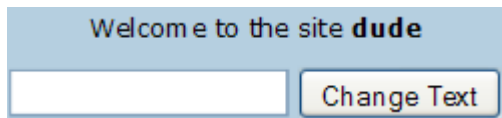
```
<script type="text/javascript">
function changeText2() {
    var userInput = document.getElementById('userInput').value;
    document.getElementById('boldStuff2').innerHTML = userInput;
}
```

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

```

</script>
<p>Welcome to the site <b id='boldStuff2'>dude</b> </p>
<input type='text' id='userInput' value='Enter Text Here' />
<input type='button' onclick='changeText2()' value='Change Text' />

```



Changing HTML with innerHTML

You can also insert HTML into your elements in the exact same way. Let's say we didn't like the text that was displayed in our paragraph and wanted to update it with some color. The following code will take the old black text and make it bright white. The only thing we're doing different here is inserting the html element *span* to change the color.

```

<script type="text/javascript">
function changeText3(){
    var oldHTML = document.getElementById('para').innerHTML;
    var newHTML = "<span style='color:#ffffff'>" + oldHTML + "</span>";
    document.getElementById('para').innerHTML = newHTML;
}
</script>
<p id='para'>Welcome to the site <b id='boldStuff3'>dude</b> </p>
<input type='button' onclick='changeText3()' value='Change Text' />

```

Function

A function is a group of reusable code which can be called anywhere in your programme. This eliminates the need of writing same code again and again. This will help programmers to write modular code. You can divide your big programme in a number of small and manageable functions.

Like any other advance programming language, JavaScript also supports all the features necessary to write modular code using functions.

You must have seen functions like *alert()* and *write()* in previous chapters. We are using these function again and again but they have been written in core JavaScript only once.

JavaScript allows us to write our own functions as well. This section will explain you how to write your own functions in JavaScript.

Function Definition:

Before we use a function we need to define that function. The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of

Hafizur Rahman
hafizur.kl@diit.info
 01739-981172

parameters (that might be empty), and a statement block surrounded by curly braces. The basic syntax is shown here:

```
<script type="text/javascript">
<!--
function functionname(parameter-list)
{
    statements
}
//-->
</script>
```

Example:

A simple function that takes no parameters called sayHello is defined here:

```
<script type="text/javascript">
<!--
function sayHello()
{
    alert("Hello there");
}
//-->
</script>
```

Calling a Function:

To invoke a function somewhere later in the script, you would simply need to write the name of that function as follows:

```
<script type="text/javascript">
<!--
sayHello();
//-->
</script>
```

```
<html>

<head>

<script type="text/javascript">

function sayHello()

{

    alert("Hello there!");

}

</script>

</head>


<body>

<p>Click the following button to call the function</p>

<form>

<input type="button" onclick="sayHello()" value="Say Hello">

</form>


<p>Use different text in alert box and then try...</p>

</body>

</html>
```

Function Parameters:

Till now we have seen function without a parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters.

A function can take multiple parameters separated by comma.

Example:

Let us do a bit modification in our *sayHello* function. This time it will take two parameters:

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

```
<script type="text/javascript">
<!--
function sayHello(name, age)
{
    alert( name + " is " + age + " years old.");
}
//-->
</script>
```

Note: We are using + operator to concatenate string and number all together. JavaScript does not mind in adding numbers into strings.

Now we can call this function as follows:

```
<script type="text/javascript">
<!--
sayHello('Zara', 7 );
//-->
</script>
```

```
<html>

<head>

<script type="text/javascript">

function sayHello(name, age)

{

    alert( name + " is " + age + " years old.");

}

</script>

</head>


<body>

<p>Click the following button to call the function</p>

<form>

<input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">

</form>


<p>Use different parameters inside the function and then try...</p>

</body>

</html>
```

The *return* Statement:

A JavaScript function can have an optional *return* statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

For example you can pass two numbers in a function and then you can expect from the function to return their multiplication in your calling program.

Example:

This function takes two parameters and concatenates them and return resultant in the calling program:

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

```
<script type="text/javascript">
<!--
function concatenate(first, last)
{
    var full;

    full = first + last;
    return full;
}
//-->
</script>
```

Now we can call this function as follows:

```
<script type="text/javascript">
<!--
    var result;
    result = concatenate('Zara', 'Ali');
    alert(result );
//-->
</script>
```

```
<html>

<head>

<script type="text/javascript">

function concatenate(first, last)

{

    var full;


    full = first + last;

    return full;

}

function secondFunction()

{

    var result;

    result = concatenate('Zara', 'Ali');

    alert(result );

}

</script>

</head>


<body>

<p>Click the following button to call the function</p>

<form>

<input type="button" onclick="secondFunction()" value="Call Function">

</form>


<p>Use different parameters inside the function and then try...</p>
```


</body>

</html>

Advanced Concepts for Functions:

There is lot to learn about JavaScript functions. But I have put following important concepts in this tutorial. If you are not in hurry then I would suggest to go through them at least once.

- [JavaScript Nested Functions](#)
- [JavaScript Function\(\) Constructor](#)
- [JavaScript Function Literals](#)

JavaScript Print Function

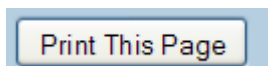
The JavaScript print function performs the same operation as the print option that you see at the top of your browser window or in your browser's "File" menu. The JavaScript print function will send the contents of the webpage to the user's printer.

JavaScript Print Script - window.print()

The JavaScript print function *window.print()* will print the current webpage when executed. In this example script, we will be placing the function on a JavaScript button that will perform the print operation when the *onClick* event occurs.

```
<form>  
<input type="button" value="Print This Page" onClick="window.print()" />  
</form>
```

Display:



If you click this button you should be prompted with whatever application your computer uses to handle its print functionality.

JavaScript Redirect

You're moving to a new domain name. You have a time-delay placeholder on your download site. You have a list of external web servers that are helping to mirror your site. What will help you deal with and/or take advantage of these situations? JavaScript redirects will.

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

When your webpage is moved, you'd probably want to notify your visitors of the change. One good way is to place a "redirect page" at the old location which, after a timed delay, will forward visitors to the new location of your webpage. You can do just this with a JavaScript redirection.

JavaScript Window.Location

Control over what page is loaded into the browser rests in the JavaScript property *window.location*. By setting *window.location* equal to a new URL, you will in turn change the current webpage to the one that is specified. If you wanted to redirect all your visitors to www.google.com when they arrived at your site, you would just need the script below:

```
<script type="text/javascript">
<!--
window.location = "http://www.google.com/"
//-->
</script>
```

JavaScript Time Delay

Implementing a timed delay in JavaScript is useful in the following situations:

- Showing an "Update your bookmark" page when you have to change URLs or page locations
- For download sites that wish to have a timed delay before the download starts
- To refresh a webpage once every specified number of seconds

The code for this timed delay is slightly involved and is beyond the scope of this tutorial. However, we have tested it and it seems to function properly.

```
<html>
<head>
<script type="text/javascript">
<!--
function delayer(){
    window.location = "../javascriptredirect.php"
}
//-->
</script>
</head>
<body onLoad="setTimeout('delayer()', 5000)">
<h2>Prepare to be redirected!</h2>
<p>This page is a time delay redirect, please update your bookmarks to our new
location!</p>

</body>
</html>
```

The most important part of getting the delay to work is being sure to use the JavaScript function *setTimeout*. We want the *delayer()* function to be used after 5 seconds or 5000 milliseconds (5 seconds), so we pass the *setTimeout()* two arguments.

- **'delayer()'** - The function we want *setTimeout()* to execute after the specified delay.
- **5000** - the number of milliseconds we want *setTimeout()* to wait before executing our function.
1000 milliseconds = 1 second.

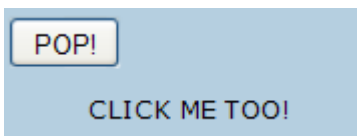
JavaScript Popups

However, we will show you how to make windows popup for reasonable occasions, like when you want to display extra information, or when you want to have something open a new window that isn't an HTML anchor tag (i.e. a hyperlink).

JavaScript window.open Function

The *window.open()* function creates a new browser window, customized to your specifications, without the use of an HTML anchor tag. In this example, we will be making a function that utilizes the *window.open()* function.

```
<head>
<script type="text/javascript">
<!--
function myPopup() {
window.open( "http://www.google.com/" )
}
//-->
</script>
</head>
<body>
<form>
<input type="button" onClick="myPopup()" value="POP!">
</form>
<p onClick="myPopup()">CLICK ME TOO!</p>
</body>
```



This works with pretty much any tag that can be clicked on, so please go ahead and experiment with this fun little tool. Afterwards, read on to learn more about the different ways you can customize the JavaScript window that **pops** up.

JavaScript Window.Open Arguments

There are three arguments that the *window.open* function takes:

1. The relative or absolute URL of the webpage to be opened.

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

2. The text name for the window.
3. A long string that contains all the different properties of the window.

Naming a window is very useful if you want to manipulate it later with JavaScript. However, this is beyond the scope of this lesson, and we will instead be focusing on the different properties you can set with your brand spanking new JavaScript window. Below are some of the more important properties:

- **dependent** - Subwindow closes if the parent window (the window that opened it) closes
- **fullscreen** - Display browser in fullscreen mode
- **height** - The height of the new window, in pixels
- **width** - The width of the new window, in pixels
- **left** - Pixel offset from the left side of the screen
- **top** - Pixel offset from the top of the screen
- **resizable** - Allow the user to resize the window or prevent the user from resizing, currently broken in Firefox.
- **status** - Display or don't display the status bar

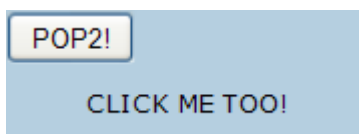
Dependent, fullscreen, resizable, and status are all examples of ON/OFF properties. You can either set them equal to zero to turn them off, or set them to one to turn them ON. There is no inbetween setting for these types of properties.

Upgraded JavaScript Popup Window!

Now that we have the tools, let's make a sophisticated JavaScript popup window that we can be proud of!

HTML & JavaScript Code:

```
<head>
<script type="text/javascript">
<!--
function myPopup2 () {
window.open( "http://www.google.com/", "myWindow",
"status = 1, height = 300, width = 300, resizable = 0" )
}
//-->
</script>
</head>
<body>
<form>
<input type="button" onClick="myPopup2 ()" value="POP2!">
</form>
<p onClick="myPopup2 ()">CLICK ME TOO!</p>
</body>
```



Hafizur Rahman
hafizur.kl@diit.info
01739-981172

Now, that is a prime example of a worthless popup! When you make your own, try to have them relate to your content, like a small popup with no navigation that just gives the definition or explanation of a word, sentence, or picture!

JavaScript Date and Time Object

The Date object is useful when you want to display a date or use a timestamp in some sort of calculation. In Java, you can either make a Date object by supplying the date of your choice, or you can let JavaScript create a Date object based on your visitor's system clock. It is usually best to let JavaScript simply use the system clock.

When creating a Date object based on the computer's (not web server's!) internal clock, it is important to note that if someone's clock is off by a few hours or they are in a different time zone, then the Date object will create a different times from the one created on your own computer.

JavaScript Date Today (Current)

To warm up our JavaScript Date object skills, let's do something easy. If you do not supply any arguments to the Date constructor (this makes the Date object) then it will create a Date object based on the visitor's internal clock.

```
<h4>It is now
<script type="text/javascript">
<!--
var currentTime = new Date()
//-->
</script>
</h4>
```

Nothing shows up! That's because we still don't know the methods of the Date object that let us get the information we need (i.e. Day, Month, Hour, etc).

Get the JavaScript Time

The Date object has been created, and now we have a variable that holds the current date! To get the information we need to print out, we have to utilize some or all of the following functions:

- **getTime()** - Number of milliseconds since 1/1/1970 @ 12:00 AM
- **getSeconds()** - Number of seconds (0-59)
- **getMinutes()** - Number of minutes (0-59)
- **getHours()** - Number of hours (0-23)
- **getDay()** - Day of the week(0-6). 0 = Sunday, ... , 6 = Saturday
- **getDate()** - Day of the month (0-31)
- **getMonth()** - Number of month (0-11)
- **getFullYear()** - The four digit year (1970-9999)

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

Now we can print out the date information. We will be using the *getDate*, *getMonth*, and *getFullYear* methods in this example.

```
<h4>It is now
<script type="text/javascript">
<!--
var currentTime = new Date()
var month = currentTime.getMonth() + 1
var day = currentTime.getDate()
var year = currentTime.getFullYear()
document.write(month + "/" + day + "/" + year)
//-->
</script>
</h4>
```

Output

It is now 1/31/2012 !

Notice that we added 1 to the *month* variable to correct the problem with January being 0 and December being 11. After adding 1, January will be 1, and December will be 12.

JavaScript Current Time Clock

Now, instead of displaying the date we, will display the format you might see on a typical digital clock -- HH:MM AM/PM (H = Hour, M = Minute).

```
<h4>It is now
<script type="text/javascript">
<!--
var currentTime = new Date()
var hours = currentTime.getHours()
var minutes = currentTime.getMinutes()
if (minutes < 10){
minutes = "0" + minutes
}
document.write(hours + ":" + minutes + " ")
if(hours > 11){
document.write("PM")
} else {
document.write("AM")
}
//-->
</script>
</h4>
```

Output:

It is now 12:51 PM

Above, we check to see if either the *hours* or *minutes* variable is less than 10. If it is, then we need to add a zero to the beginning of minutes. This is not necessary, but if it is 1:01 AM, our clock would output "1:1 AM", which doesn't look very nice at all!

Creating a JavaScript Clock

One of the great things about JavaScript is that it lets you manipulate the contents of a Web page in real-time. This means that we can use JavaScript to create a digital clock, embedded in the Web page, that updates every second. This article shows you how to do just that.

Getting the current time

If we want to create a clock then obviously we need to retrieve the current time. We can do this via JavaScript's `Date` class. First, we create a new `Date` object with no parameters, which gives us a `Date` object containing the current date and time on the visitor's computer:

```
var currentTime = new Date ( );
```

Next, we extract the hours, minutes and seconds components of the current time from our `Date` object:

```
var currentHours = currentTime.getHours ( );  
var currentMinutes = currentTime.getMinutes ( );  
var currentSeconds = currentTime.getSeconds ( );
```

Formatting the time

Now that we have the three component values of our current time, let's format them into a nice string for displaying in the Web page. We want it to be in the format "HH:MM:SS XX", where XX is either "AM" or "PM" (assuming we're after a 12-hour clock format).

First we'll add a leading zero to the minutes and seconds values, if required:

```
currentMinutes = ( currentMinutes < 10 ? "0" : "" ) + currentMinutes;  
currentSeconds = ( currentSeconds < 10 ? "0" : "" ) + currentSeconds;
```

The `?` and `:` symbols used above comprise the *ternary operator*. This is a special operator that returns the value before the colon if the condition before the query (`?`) is true, or the value after the colon if the condition is false. It's a great way to write an `if` block in shorthand, provided you only need to return a single value.

Next we'll set a variable, `timeOfDay`, to "AM" or "PM" as appropriate, and subtract 12 from the hours component, if required, to convert it to 12-hour format. We'd also like the hours component to show 12 rather than 0, so we need to add a check for this too:

```
var timeOfDay = ( currentHours < 12 ) ? "AM" : "PM";  
currentHours = ( currentHours > 12 ) ? currentHours - 12 : currentHours;
```

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

```
currentHours = ( currentHours == 0 ) ? 12 : currentHours;
```

Finally, we'll join all our formatted components together into a single string, in the format "HH:MM:SS XX":

```
var currentTimeString = currentHours + ":" + currentMinutes + ":" + currentSeconds  
+ " " + timeOfDay;
```

Displaying the clock

Now that we have our time string ready for display, the next step is to display it in the Web page. To do this, we first create a `span` container in the markup to hold the time display:

```
<span id="clock">&nbsp;</span>
```

By placing the ` ` inside the `span` element, we're creating a child text node for the `span` in the DOM. We can then populate the container with our time string by retrieving this child text node then setting its `nodeValue` property, as follows:

```
document.getElementById("clock").firstChild.nodeValue = currentTimeString;
```

Putting it all together

Here's our finished JavaScript code, ready to be placed in the `head` element of the Web page. We've wrapped all the above code in a JavaScript function, `updateClock()`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">  
<head>  
<title>JavaScript Clock</title>  
<style type="text/css">  
#clock{  
    font-family:Arial,Helvetica,sans-serif;  
    font-size:0.8em;  
    color:white;  
    background-color:black;  
    border:2px solid purple;  
    padding:4px;  
}  
</style>  
<script type="text/javascript">  
<!--
```

```
function updateClock()
```

Hafizur Rahman
hafizur.kl@diit.info
01739-981172


```

{
    var currentTime=new Date();
    var currentHours=currentTime.getHours();
    var currentMinutes=currentTime.getMinutes();
    var currentSeconds=currentTime.getSeconds();
    currentMinutes=(currentMinutes<10?"0:"")+currentMinutes;
    currentSeconds=(currentSeconds<10?"0:"")+currentSeconds;
    var timeOfDay=(currentHours<12)?"AM":"PM";
    currentHours=(currentHours>12)?currentHours-
12:currentHours;
    currentHours=(currentHours==0)?12:currentHours;
    var
currentTimeString=currentHours+":"+currentMinutes":"+currentSeconds
+" "+timeOfDay;

    document.getElementById("clock").firstChild.nodeValue=currentTi
meString;}
//-->
</script>
</head>
<body onload="updateClock(); setInterval('updateClock()', 1000 )">
    <span id="clock">&nbsp;</span>
</body>
</html>

```

To make the clock update every second, we need to use the `Window.setInterval()` method from within the page's `body` tag to call our `updateClock()` function once per second. We should also call `updateClock()` at the moment the page loads, so that the clock appears straightaway:

```

<body onload="updateClock(); setInterval('updateClock()', 1000 )">

```

JavaScript Form Validation

The idea behind JavaScript form validation is to provide a method to check the user entered information before they can even submit it. JavaScript also lets you display helpful alerts to inform the user what information they have entered incorrectly and how they can fix it. In this lesson we will be reviewing some basic form validation, showing you how to check for the following:

- If a text input is empty or not
- If a text input is all numbers
- If a text input is all letters
- If a text input is all alphanumeric characters (numbers & letters)
- If a text input has the correct number of characters in it (useful when restricting the length of a username and/or password)
- If a selection has been made from an HTML select input (the drop down selector)
- If an email address is valid

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

- How to check all above when the user has completed filling out the form

Form Validation - Checking for Non-Empty

This has to be the most common type of form validation. You want to be sure that your visitors enter data into the HTML fields you have "required" for a valid submission. Below is the JavaScript code to perform this basic check to see if a given HTML input is empty or not.

```
// If the length of the element's string is 0 then display helper message
function notEmpty(elem, helperMsg){
    if(elem.value.length == 0){
        alert(helperMsg);
        elem.focus(); // set the focus to this input
        return false;
    }
    return true;
}
```

The function *notEmpty* will check to see that the HTML input that we send it has something in it. *elem* is a HTML text input that we send this function. JavaScript strings have built in properties, one of which is the *length* property which returns the length of the string. The chunk of code *elem.value* will grab the string inside the input and by adding on length *elem.value.length* we can see how long the string is.

As long as *elem.value.length* isn't 0 then it's not empty and we return true, otherwise we send an alert to the user with a *helperMsg* to inform them of their error and return false.

```
<script type='text/javascript'>
function notEmpty(elem, helperMsg){
    if(elem.value.length == 0){
        alert(helperMsg);
        elem.focus();
        return false;
    }
    return true;
}
</script>
<form>
Required Field: <input type='text' id='req1' />
<input type='button'
    onclick="notEmpty(document.getElementById('req1'), 'Please Enter a Value')"
    value='Check Field' />
</form>
```



The screenshot shows a web form with a label "Required Field:" followed by a text input field. To the right of the input field is a button labeled "Check Field". The entire form is enclosed in a light blue border.

Form Validation - Checking for All Numbers

If someone is entering a credit card, phone number, zip code, similar information you want to be able to ensure that the input is all numbers. The quickest way to check if an input's string value is all

Hafizur Rahman
hafizur.kl@diit.info
 01739-981172

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.

```
// If the element's string matches the regular expression it is all numbers
function isNumeric(elem, helperMsg){
    var numericExpression = /^[0-9]+$/;
    if(elem.value.match(numericExpression)){
        return true;
    }else{
        alert(helperMsg);
        elem.focus();
        return false;
    }
}
```

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 *match* that you can use to see if the string matches a certain regular expression. We accessed this function like so: `elem.value.match(expressionhere)`.

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.

```
<script type='text/javascript'>
function isNumeric(elem, helperMsg){
    var numericExpression = /^[0-9]+$/;
    if(elem.value.match(numericExpression)){
        return true;
    }else{
        alert(helperMsg);
        elem.focus();
        return false;
    }
}
</script>
<form>
Numbers Only: <input type='text' id='numbers' />
<input type='button'
    onclick="isNumeric(document.getElementById('numbers'), 'Numbers Only
Please')"
    value='Check Field' />
</form>
```

The screenshot shows a web form with a light blue border. On the left, the text "Numbers Only:" is displayed. To its right is a white text input field. Further right is a button with a light blue border and the text "Check Field" in a bold, sans-serif font.

Form Validation - 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]+$/.`

```
// 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);
        elem.focus();
        return false;
    }
}
```

Working Example:

```
<script type='text/javascript'>
function isAlphabet(elem, helperMsg){
    var alphaExp = /^[a-zA-Z]+$/.;
    if(elem.value.match(alphaExp)){
        return true;
    }else{
        alert(helperMsg);
        elem.focus();
        return false;
    }
}
</script>
<form>
Letters Only: <input type='text' id='letters' />
<input type='button'
    onclick="isAlphabet(document.getElementById('letters'), 'Letters Only
Please') "
    value='Check Field' />
</form>
```



The screenshot shows a web form with a label "Letters Only:" followed by a text input field. To the right of the input field is a button labeled "Check Field". The entire form is contained within a light blue rectangular border.

Form Validation - Checking for Numbers and Letters

By combining both the *isAlphabet* and *isNumeric* functions into one we can check to see if a text input contains only letters and numbers.

```
// If the element's string matches the regular expression it is numbers and
letters
function isAlphanumeric(elem, helperMsg){
    var alphaExp = /^[0-9a-zA-Z]+$/;
    if(elem.value.match(alphaExp)){
        return true;
    }else{
        alert(helperMsg);
        elem.focus();
        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.

```
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;
    }
}
```

Here's an example of this function for a field that requires 6 to 8 characters for a valid username.

Working Example:

```
<script type='text/javascript'>
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;
    }
}
```

Hafizur Rahman
hafizur.kl@diit.info
01739-981172

```

</script>
<form>
Username(6-8 characters): <input type='text' id='restrict' />
<input type='button'
    onclick="lengthRestriction(document.getElementById('restrict'), 6, 8)"
    value='Check Field' />
</form>

```



Form Validation - Selection Made

To be sure that someone has actually selected a choice from an HTML select input you can use a simple trick of making the first option as helpful prompt to the user and a red flag to you for your validation code.

By making the first option of your select input something like "Please Choose" you can spur the user to both make a selection and allow you to check to see if the default option "Please Choose" is still selected when the submit the form.

```

function madeSelection(elem, helperMsg){
    if(elem.value == "Please Choose"){
        alert(helperMsg);
        elem.focus();
        return false;
    }else{
        return true;
    }
}

```

Working Example:

```

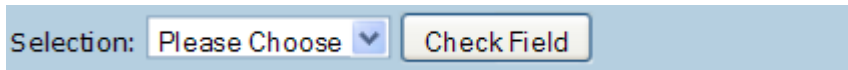
<script type='text/javascript'>
function madeSelection(elem, helperMsg){
    if(elem.value == "Please Choose"){
        alert(helperMsg);
        elem.focus();
        return false;
    }else{
        return true;
    }
}
</script>
<form>
Selection: <select id='selection'>
<option>Please Choose</option>
<option>CA</option>
<option>WI</option>
<option>XX</option>
</select>
<input type='button'

```

```

        onclick="madeSelection(document.getElementById('selection'), 'Please Choose
Something')"
        value='Check Field' />
</form>

```



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:

1. A combination of letters, numbers, periods, hyphens, plus signs, and/or underscores
2. The at symbol @
3. A combination of letters, numbers, hyphens, and/or periods
4. A period
5. 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 regular expression to check for all of this is a little overkill and beyond the scope of this tutorial to explain thoroughly. However, test it out and you'll see that it gets the job done.

```

function emailValidator(elem, helperMsg){
    var emailExp = /^[w\-\.\.]+\@\[a-zA-Z0-9\.\-]+\.[a-zA-z0-9]{2,4}$/;
    if(elem.value.match(emailExp)){
        return true;
    }else{
        alert(helperMsg);
        elem.focus();
        return false;
    }
}

```

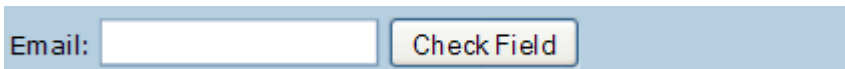
Working Example:

Hafizur Rahman
hafizur.kl@diit.info
 01739-981172

```

<script type='text/javascript'>
function emailValidator(elem, helperMsg){
    var emailExp = /^[w\-\.\.]+\@[a-zA-Z0-9\.\-]+\.[a-zA-z0-9]{2,4}$/;
    if(elem.value.match(emailExp)){
        return true;
    }else{
        alert(helperMsg);
        elem.focus();
        return false;
    }
}
</script>
<form>
Email: <input type='text' id='emailer' />
<input type='button'
    onclick="emailValidator1(document.getElementById('emailer'), 'Not a Valid
Email')"
    value='Check Field' />
</form>

```



Validating a Form - All at Once

```

<script type='text/javascript'>
function formValidator(){
    // Make quick references to our fields
    var firstname = document.getElementById('firstname');
    var addr = document.getElementById('addr');
    var zip = document.getElementById('zip');
    var state = document.getElementById('state');
    var username = document.getElementById('username');
    var email = document.getElementById('email');

    // Check each input in the order that it appears in the form!
    if(isAlphabet(firstname, "Please enter only letters for your name")){
        if(isAlphanumeric(addr, "Numbers and Letters Only for Address")){
            if(isNumeric(zip, "Please enter a valid zip code")){
                if(madeSelection(state, "Please Choose a State")){
                    if(lengthRestriction(username, 6, 8)){
                        if(emailValidator(email, "Please
enter a valid email address")){
                            return true;
                        }
                    }
                }
            }
        }
    }

    return false;
}

```



```

function notEmpty(elem, helperMsg){
    if(elem.value.length == 0){
        alert(helperMsg);
        elem.focus(); // set the focus to this input
        return false;
    }
    return true;
}

function isNumeric(elem, helperMsg){
    var numericExpression = /^[0-9]+$/;
    if(elem.value.match(numericExpression)){
        return true;
    }else{
        alert(helperMsg);
        elem.focus();
        return false;
    }
}

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

function isAlphanumeric(elem, helperMsg){
    var alphaExp = /^[0-9a-zA-Z]+$/;
    if(elem.value.match(alphaExp)){
        return true;
    }else{
        alert(helperMsg);
        elem.focus();
        return false;
    }
}

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;
    }
}

function madeSelection(elem, helperMsg){
    if(elem.value == "Please Choose"){
        alert(helperMsg);
        elem.focus();
        return false;
    }
}

```

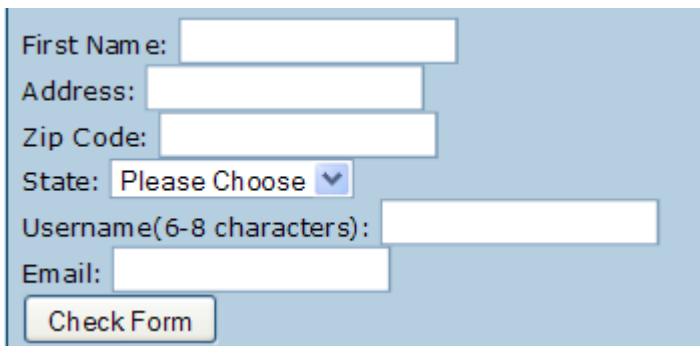
```

        }else{
            return true;
        }
    }

function emailValidator(elem, helperMsg){
    var emailExp = /^[\\w\\-\\.\\+]+\\@[a-zA-Z0-9\\.\\-]+\\. [a-zA-z0-9]{2,4}$ /;
    if(elem.value.match(emailExp)){
        return true;
    }else{
        alert(helperMsg);
        elem.focus();
        return false;
    }
}
</script>

<form onsubmit='return formValidator()' >
First Name: <input type='text' id='firstname' /><br />
Address: <input type='text' id='addr' /><br />
Zip Code: <input type='text' id='zip' /><br />
State: <select id='state'>
    <option>Please Choose</option>
    <option>AL</option>
    <option>CA</option>
    <option>TX</option>
    <option>WI</option>
</select><br />
Username(6-8 characters): <input type='text' id='username' /><br />
Email: <input type='text' id='email' /><br />
<input type='submit' value='Check Form' />
</form>

```



First Name:

Address:

Zip Code:

State:

Username(6-8 characters):

Email: