What is JavaScript ?

JavaScript started life as LiveScript, but Netscape changed the name, possibly because of the excitement being generated by Java.to JavaScript. JavaScript made its first appearance in Netscape 2.0 in 1995 with a name *LiveScript*.

JavaScript is a lightweight, interpreted programming language with object-oriented capabilities that allows you to build interactivity into otherwise static HTML pages.

The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers

JavaScript is:

- JavaScript is a lightweight, interpreted programming language
- Designed for creating network-centric applications
- Complementary to and integrated with Java
- Complementary to and integrated with HTML
- Open and cross-platform

Client-side JavaScript:

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need no longer be static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism features many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user explicitly or implicitly initiates.

Advantages of JavaScript:

The merits of using JavaScript are:

- **Less server interaction:** You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors:** They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity:** You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces:** You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations with JavaScript:

We cannot treat JavaScript as a full fledged programming language. It lacks the following important features:

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocess capabilities.

A JavaScript consists of JavaScript statements that are placed within the <script>... </script> HTML tags in a web page.

You can place the <script> tag containing your JavaScript anywhere within you web page but it is preferred way to keep it within the <head> tags.

The <script> tag alert the browser program to begin interpreting all the text between these tags as a script. So simple syntax of your JavaScript will be as follows

```
<script ...>
  JavaScript code
</script>
```

The script tag takes two important attributes:

- **language:** This attribute specifies what scripting language you are using. Typically, its value will be *javascript*. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **type:** This attribute is what is now recommended to indicate the scripting language in use and its value should be set to *"text/javascript"*.

So your JavaScript segment will look like:

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

Your First JavaScript Script:

Let us write our class example to print out "Hello World".

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

Whitespace and Line Breaks:

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs.

Because you can use spaces, tabs, and newlines freely in your program so you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Semicolons are Optional:

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if your statements are each placed on a separate line. For example, the following code could be written without semicolons

```
<script language="javascript" type="text/javascript">
<!--
  var1 = 10
  var2 = 20
//-->
</script>
```

But when formatted in a single line as follows, the semicolons are required:

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

```
<!--
  var1 = 10; var2 = 20;
//-->
</script>
```

Note: It is a good programming practice to use semicolons.

Case Sensitivity:

JavaScript is a case-sensitive language. This means that language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So identifiers *Time*, *TIme* and *TIME* will have different meanings in JavaScript.

NOTE: Care should be taken while writing your variable and function names in JavaScript.

Comments in JavaScript:

JavaScript supports both C-style and C++-style comments, Thus:

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

Example:

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

// This is a comment. It is similar to comments in C++

/*
 * This is a multiline comment in JavaScript
 * It is very similar to comments in C Programming
 */
//-->
</script>
```

JavaScript Placement in HTML File

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 and 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:

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:

```
Advertisements

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:

```
Advertisements

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 DataTypes:

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types:

- Numbers eg. 123, 120.50 etc.
- Strings of text e.g. "This text string" etc.
- Boolean e.g. true or false.

JavaScript also defines two trivial data types, *null* and *undefined*, each of which defines only a single value.

In addition to these primitive data types, JavaScript supports a composite data type known as*object*. We will see an object detail in a separate chapter.

Note: Java does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format defined by the IEEE 754 standard.

JavaScript Variables:

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows:

```
<script type="text/javascript">
<!--
var money;
var name;
//-->
</script>
```

You can also declare multiple variables with the same **var** keyword as follows:

```
<script type="text/javascript">
<!--
var money, name;
//-->
</script>
```

Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or later point in time when you need that variable as follows:

For instance, you might create a variable named *money* and assign the value 2000.50 to it later. For another variable you can assign a value the time of initialization as follows:

```
<script type="text/javascript">
<!--
var name = "Ali";
var money;
money = 2000.50;
//-->
</script>
```

Note: Use the **var** keyword only for declaration or initialization.once for the life of any variable name in a document. You should not re-declare same variable twice.

JavaScript is *untyped* language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable Scope:

The scope of a variable is the region of your program in which it is defined. JavaScript variable will have only two scopes.

- **Global Variables:** A global variable has global scope which means it is defined everywhere in your JavaScript code.
- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Following example explains it:

```
<script type="text/javascript">
<!--
var myVar = "global"; // Declare a global variable
function checkscope( ) {
   var myVar = "local";  // Declare a local variable
   document.write(myVar);
}
```

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

This produces the following result:

```
local
```

JavaScript Variable Names:

While naming your variables in JavaScript keep following rules in mind.

- You should not use any of the JavaScript reserved keyword as variable name. These keywords are mentioned in the next section. For example, *break* or *boolean* variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or the underscore character. For example, *123test* is an invalid variable name but *_123test* is a valid one.
- JavaScript variable names are case sensitive. For example, *Name* and *name* are two different variables.

JavaScript Reserved Words:

The following are reserved words in JavaScript. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

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
- Comparision Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Let's have a look on all operators one by one.

The Arithmatic Operators:

There are following arithmatic 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 denumerator	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".

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.

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 becomes true.	(A && B) is 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.

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 is a unary operator and operates by reversing	(~B) is -4 .

	all bits in the operand.	
<<	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.
>>	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.	(A >> 1) is 1.

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
+=	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 ^=.

Miscellaneous Operator

The Conditional Operator (? :)

There is an oprator 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 conditioanl operator has this syntax:

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

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"

JavaScript if...else Statements

While writing a program, there may be a situation when you need to adopt one path out of the given two paths. So you need to make use of conditional statements that allow your program to make correct decisions and perform right actions.

JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain **if..else** statement.

JavaScript supports following forms of **if..else** statement:

- if statement
- if...else statement
- if...else if... statement.

if statement:

The **if** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax:

```
if (expression){
   Statement(s) to be executed if expression
is true
}
```

Here JavaScript *expression* is evaluated. If the resulting value is *true*, given *statement(s)* are executed. If *expression* is *false* then no statement would be not executed. Most of the times you will use comparison operators while making decisions.

Example:

```
<script type="text/javascript">
<!--
var age = 20;
if( age > 18 ){
   document.write("<b>Qualifies          for
driving</b>");
}
//-->
</script>
```

This will produce following result:

```
Qualifies for driving
```

if...else statement:

The **if...else** statement is the next form of control statement that allows JavaScript to execute statements in more controlled way.

Syntax:

```
if (expression){
   Statement(s) to be executed if expression
is true
}else{
   Statement(s) to be executed if expression
is false
}
```

Here JavaScript *expression* is evaluated. If the resulting value is *true*, given *statement(s)* in the *if*block, are executed. If *expression* is *false* then given *statement(s)* in the *else* block, are executed.

Example:

```
<script type="text/javascript">
<!--
var age = 15;
if( age > 18 ){
   document.write("<b>Qualifies          for
driving</b>");
}else{
   document.write("<b>Does  not  qualify  for
driving</b>");
}
//-->
</script>
```

This will produce following result:

```
Does not qualify for driving
```

if...else if... statement:

The **if...else if...** statement is the one level advance form of control statement that allows JavaScript to make correct decision out of several conditions.

Syntax:

```
if (expression 1){
   Statement(s) to be executed if expression
1 is true
}else if (expression 2){
   Statement(s) to be executed if expression
2 is true
}else if (expression 3){
   Statement(s) to be executed if expression
3 is true
```

```
}else{
   Statement(s)   to   be   executed   if   no
expression is true
}
```

There is nothing special about this code. It is just a series of *if* statements, where each *if* is part of the *else* clause of the previous statement. Statement(s) are executed based on the true condition, if non of the condition is true then *else* block is executed.

Example:

```
<script type="text/javascript">
<!--
var book = "maths";
if( book == "history" ){
   document.write("<b>History Book</b>");
}else if( book == "maths" ){
   document.write("<b>Maths Book</b>");
}else if( book == "economics" ){
   document.write("<b>Economics Book</b>");
}else{
  document.write("<b>Unknown Book</b>");
}
//-->
</script>
```

This will produce following result:

```
Maths Book
```

JavaScript Switch Case

You can use multiple *if...else if* statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Starting with JavaScript 1.2, you can use a **switch** statement which handles exactly this situation, and it does so more efficiently than repeated *if...else if* statements.

Syntax:

The basic syntax of the **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each**case** against the value of the expression until a match is found. If nothing matches, a **default**condition will be used.

```
switch (expression)
{
  case condition 1: statement(s)
                    break;
  case condition 2: statement(s)
                    break;
   ...
  case condition n: statement(s)
                    break;
  default: statement(s)
}
```

The **break** statements indicate to the interpreter the end of that particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

We will explain **break** statement in *Loop Control* chapter.

Example:

Following example illustrates a basic while loop:

```
<script type="text/javascript">
<!--
var grade='A';
document.write("Entering   switch   block<br
/>");
switch (grade)
{
  case   'A':   document.write("Good   job<br
/>");
            break;
  case  'B':  document.write("Pretty  good<br
/>");
            break;
  case 'C': document.write("Passed<br />");
            break;
  case  'D':  document.write("Not  so  good<br
/>");
            break;
  case 'F': document.write("Failed<br />");
            break;
  default:  document.write("Unknown grade<br
/>")
}
document.write("Exiting switch block");
//-->
</script>
```

This will produce following result:

```
Entering switch block
Good job
Exiting switch block
```

Example:

Consider a case if you do not use **break** statement:

```
<script type="text/javascript">
<!--
var grade='A';
document.write("Entering   switch   block<br
/>");
switch (grade)
{
  case   'A':   document.write("Good   job<br
/>");
  case  'B':  document.write("Pretty  good<br
/>");
  case 'C': document.write("Passed<br />");
  case  'D':  document.write("Not  so  good<br
/>");
  case 'F': document.write("Failed<br />");
  default:  document.write("Unknown grade<br
/>")
}
document.write("Exiting switch block");
//-->
</script>
```

This will produce following result:

```
Entering switch block
Good job
Pretty good
Passed
Not so good
Failed
Unknown grade
Exiting switch block
```

JavaScript while Loops

While writing a program, there may be a situation when you need to perform some action over and over again. In such situation you would need to write loop statements to reduce the number of lines.

JavaScript supports all the necessary loops to help you on all steps of programming.

The *while* Loop

The most basic loop in JavaScript is the **while** loop which would be discussed in this tutorial.

Syntax:

```
while (expression){
   Statement(s) to be executed if expression
is true
}
```

The purpose of a **while** loop is to execute a statement or code block repeatedly as long as*expression* is true. Once expression becomes *false*, the loop will be exited.

Example:

Following example illustrates a basic while loop:

```
<script type="text/javascript">
<!--
var count = 0;
document.write("Starting Loop" + "<br />");
while (count < 10){
  document.write("Current Count : " + count
+ "<br />");
  count++;
}
document.write("Loop stopped!");
//-->
</script>
```

This will produce following result:

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
```

```
Loop stopped!
```

The *do...while* Loop:

The **do...while** loop is similar to the **while** loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is *false*.

Syntax:

```
do{
   Statement(s) to be executed;
} while (expression);
```

Note the semicolon used at the end of the **do...while** loop.

Example:

Let us write above example in terms of **do...while** loop.

```
<script type="text/javascript">
<!--
var count = 0;
document.write("Starting Loop" + "<br />");
do{
  document.write("Current Count : " + count
+ "<br />");
  count++;
}while (count < 0);
document.write("Loop stopped!");
//-->
</script>
```

This will produce following result:

```
Starting Loop
Current Count : 0
Loop stopped!
```

JavaScript *for* Loops

We have seen different variants of **while** loop. This chapter will explain another popular loop called**for** loop.

The *for* Loop

The **for** loop is the most compact form of looping and includes the following three important parts:

- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if the given condition is true or not. If condition is true then code given inside the loop will be executed otherwise loop will come out.
- The iteration statement where you can increase or decrease your counter.

You can put all the three parts in a single line separated by a semicolon.

Syntax:

```
for   (initialization;   test   condition;
iteration statement){
     Statement(s)  to  be  executed  if  test
condition is true
}
```

Example:

Following example illustrates a basic for loop:

```
<script type="text/javascript">
<!--
var count;
document.write("Starting Loop" + "<br />");
for(count = 0; count < 10; count++){
  document.write("Current Count : " + count
);
  document.write("<br />");
}
document.write("Loop stopped!");
//-->
</script>
```

This will produce following result which is similar to **while** loop:

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
```

```
Current Count : 9
Loop stopped!
```

JavaScript *for...in* loop

There is one more loop supported by JavaScript. It is called **for...in** loop. This loop is used to loop through an object's properties.

Because we have not discussed Objects yet, so you may not feel comfortable with this loop. But once you will have understanding on JavaScript objects then you will find this loop very useful.

Syntax:

```
for (variablename in object){
  statement or block to execute
}
```

In each iteration one property from *object* is assigned to *variablename* and this loop continues till all the properties of the object are exhausted.

Example:

Here is the following example that prints out the properties of a Web browser's **Navigator** object:

```
<script type="text/javascript">
<!--
var aProperty;
document.write("Navigator            Object
Properties<br /> ");
for (aProperty in navigator)
{
  document.write(aProperty);
  document.write("<br />");
}
document.write("Exiting from the loop!");
//-->
</script>
```

This will produce following result:

```
Navigator Object Properties
appCodeName
appName
appMinorVersion
cpuClass
```

```
platform
plugins
opsProfile
userProfile
systemLanguage
userLanguage
appVersion
userAgent
onLine
cookieEnabled
mimeTypes
Exiting from the loop!
```

JavaScript Loop Control

JavaScript provides you full control to handle your loops and switch statement. There may be a situation when you need to come out of a loop without reaching at its bottom. There may also be a situation when you want to skip a part of your code block and want to start next iteration of the look.

To handle all such situations, JavaScript provides **break** and **continue** statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

The *break* Statement:

The **break** statement, which was briefly introduced with the *switch* statement, is used to exit a loop early, breaking out of the enclosing curly braces.

Example:

This example illustrates the use of a **break** statement with a while loop. Notice how the loop breaks out early once x reaches 5 and reaches to *document.write(..)* statement just below to closing curly brace:

```
<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop<br /> ");
while (x < 20)
{
  if (x == 5){
     break;     //   breaks   out   of   loop
completely
  }
  x = x + 1;
  document.write( x + "<br />");
}
```

```
document.write("Exiting the loop!<br /> ");
//-->
</script>
```

This will produce following result:

```
Entering the loop
2
3
4
5
Exiting the loop!
```

We already have seen the usage of **break** statement inside a *switch* statement.

The *continue* Statement:

The **continue** statement tells the interpreter to immediately start the next iteration of the loop and skip remaining code block.

When a **continue** statement is encountered, program flow will move to the loop check expression immediately and if condition remain true then it start next iteration otherwise control comes out of the loop.

Example:

This example illustrates the use of a **continue** statement with a while loop. Notice how the**continue** statement is used to skip printing when the index held in variable x reaches 5:

```
<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop<br /> ");
while (x < 10)
{
  x = x + 1;
  if (x == 5){
     continue;   // skill rest of the loop
body
  }
  document.write( x + "<br />");
}
document.write("Exiting the loop!<br /> ");
//-->
</script>
```

This will produce following result:

```
Entering the loop
2
3
4
6
7
8
9
10
Exiting the loop!
```

Using Labels to Control the Flow:

Starting from JavaScript 1.2, a label can be used with **break** and **continue** to control the flow more precisely.

A **label** is simply an identifier followed by a colon that is applied to a statement or block of code. We will see two different examples to understand label with break and continue.

Note: Line breaks are not allowed between the *continue* or **break** statement and its label name. Also, there should not be any other statement in between a label name and associated loop.

Example 1:

```
<script type="text/javascript">
<!--
document.write("Entering the loop!<br /> ");
outerloop:   // This is the label name
for (var i = 0; i < 5; i++)
{
  document.write("Outerloop: " + i + "<br
/>");
  innerloop:
  for (var j = 0; j < 5; j++)
  {
     if (j >  3 ) break ;          // Quit
the innermost loop
     if (i == 2) break innerloop; // Do the
same thing
     if (i == 4) break outerloop; // Quit
the outer loop
     document.write("Innerloop: " + j + "
<br />");
   }
}
```

```
document.write("Exiting the loop!<br /> ");
//-->
</script>
```

This will produce following result:

```
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 2
Outerloop: 3
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 4
Exiting the loop!
```

Example 2:

```
<script type="text/javascript">
<!--
document.write("Entering the loop!<br /> ");
outerloop:   // This is the label name
for (var i = 0; i < 3; i++)
{
   document.write("Outerloop: " + i + "<br
/>");
   for (var j = 0; j < 5; j++)
   {
      if (j == 3){
         continue outerloop;
      }
      document.write("Innerloop:  "  +  j  +
"<br />");
   }
}
document.write("Exiting the loop!<br /> ");
//-->
</script>
```

This will produce following result:

```
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 2
Innerloop: 0
Innerloop: 1
Innerloop: 2
Exiting the loop!
```

JavaScript Functions

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 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 simple need to write the name of that function as follows:

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

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:

```
<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>
```

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:

```
<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>
```

The *function* statement is not the only way to define a new function but also, you can define your function dynamically using **Function( )** constructor along with the **new** operator.

Note: This is the terminology from Object Oriented Programming. You may not feel comfortable for the first time, which is OK.

Syntax:

Following is the syntax to create a function using **Function( )** constructor along with the **new**operator.

```
<script type="text/javascript">
<!--
var   variablename   =   new   Function(Arg1,
Arg2..., "Function Body");
//-->
</script>
```

The **Function()** constructor expects any number of string arguments. The last argument is the body of the function - it can contain arbitrary JavaScript statements, separated from each other by semicolons.

Notice that the **Function()** constructor is not passed any argument that specifies a name for the function it creates. The *unnamed* functions created with the **Function()** constructor are called*anonymous* functions.

Example:

Here is an example of creating a function in this way:

```
<script type="text/javascript">
<!--
var func = new Function("x", "y", "return
x*y;");
//-->
</script>
```

This line of code creates a new function that is more or less equivalent to a function defined with the familiar syntax:

```
<script type="text/javascript">
<!--
function f(x, y){
   return x*y;
}
//-->
</script>
```

It means you can call above function as follows:

```
<script type="text/javascript">
<!--
func(10,20); // This will produce 200
//-->
</script>
```

JavaScript Events

What is an Event ?

JavaScript's interaction with HTML is handled through events that occur when the user or browser manipulates a page.

When the page loads, that is an event. When the user clicks a button, that click, too, is an event. Another example of events are like pressing any key, closing window, resizing window etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable to occur.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element have a certain set of events which can trigger JavaScript Code.

Please go through this small tutorial for a better understanding HTML Event Reference. Here we will see few examples to understand a relation between Event and JavaScript:

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>
```

This will produce following result and when you click Hello button then *onclick* event will occur which will trigger *sayHello()* function.

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.

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>
```

You can change different images using these two event types or you can create help baloon to help your users.

HTML 4 Standard Events

The standard HTML 4 events are listed here for your reference. Here *script* indicates a Javascript function to be executed agains that event.

Event	**Value**	**Description**
onchange	script	Script runs when the element changes
onsubmit	script	Script runs when the form is submitted
onreset	script	Script runs when the form is reset
onselect	script	Script runs when the element is selected
onblur	script	Script runs when the element loses focus
onfocus	script	Script runs when the element gets focus
onkeydown	script	Script runs when key is pressed
onkeypress	script	Script runs when key is pressed and released
onkeyup	script	Script runs when key is released

onclick	script	Script runs when a mouse click
ondblclick	script	Script runs when a mouse double-click
onmousedown	script	Script runs when mouse button is pressed
onmousemove	script	Script runs when mouse pointer moves
onmouseout	script	Script runs when mouse pointer moves out of an element
onmouseover	script	Script runs when mouse pointer moves over an element
onmouseup	script	Script runs when mouse button is released

JavaScript - Page Redirection

What is page redirection ?

When you click a URL to reach to a page X but internally you are directed to another page Y that simply happens because of page re-direction. This concept is different from JavaScript Page Refresh.

There could be various reasons why you would like to redirect from original page. I'm listing down few of the reasons:

- You did not like the name of your domain and you are moving to a new one. Same time you want to direct your all visitors to new site. In such case you can maintain your old domain but put a single page with a page re-direction so that your all old domain visitors can come to your new domain.
- You have build-up various pages based on browser versions or their names or may be based on different countries, then instead of using your server side page redirection you can use client side page redirection to land your users on appropriate page.

- The Search Engines may have already indexed your pages. But while moving to another domain then you would not like to lose your visitors coming through search engines. So you can use client side page redirection. But keep in mind this should not be done to make search engine a fool otherwise this could get your web site banned.

How Page Re-direction works ?

Example 1:

This is very simple to do a page redirect using JavaScript at client side. To redirect your site visitors to a new page, you just need to add a line in your head section as follows:

```
<head>
<script type="text/javascript">
<!--

window.location="http://www.newlocation.com";
//-->
</script>
</head>
```

Example 2:

You can show an appropriate message to your site visitors before redirecting them to a new page. This would need a bit time delay to load a new page. Following is the simple example to implement the same:

```
<head>
<script type="text/javascript">
<!--
function Redirect()
{

window.location="http://www.newlocation.com";
}

document.write("You  will  be  redirected  to
main page in 10 sec.");
setTimeout('Redirect()', 10000);
//-->
</script>
</head>
```

Here *setTimeout()* is a built-in JavaScript function which can be used to execute another function after a given time interval.

Example 3:

Following is the example to redirect site visitors on different pages based on their browsers :

```
<head>
<script type="text/javascript">
<!--
var browsername=navigator.appName;
if( browsername == "Netscape" )
{
   window.location="http://www.location.com/ns.htm";
}
else   if   (   browsername   =="Microsoft   Internet
Explorer")
{
   window.location="http://www.location.com/ie.htm";
}
else
{

window.location="http://www.location.com/other.htm";
}
//-->
</script>
</head>
```

JavaScript - Dialog Boxes

JavaScript supports three important types of dialog boxes. These dialog boxes can be used to raise and alert, or to get confirmation on any input or to have a kind of input from the users.

Here we will see each dialog box one by one:

Alert Dialog Box:

An alert dialog box is mostly used to give a warning message to the users. Like if one input field requires to enter some text but user does not enter that field then as a part of validation you can use alert box to give warning message as follows:

```
<head>
<script type="text/javascript">
<!--
   alert("Warning Message");
```

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

Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed.

Confirmation Dialog Box:

A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: **OK** and **Cancel**.

If the user clicks on OK button the window method *confirm()* will return true. If the user clicks on the Cancel button *confirm()* returns false. You can use confirmation dialog box as follows:

```
<head>
<script type="text/javascript">
<!--
   var  retVal  =  confirm("Do  you  want  to
continue ?");
   if( retVal == true ){
      alert("User wants to continue!");
         return true;
   }else{
      alert("User   does   not   want   to
continue!");
         return false;
   }
//-->
</script>
</head>
```

Prompt Dialog Box:

The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus it enable you to interact with the user. The user needs to fill in the field and then click OK.

This dialog box is displayed using a method called *prompt()* which takes two parameters (i) A label which you want to display in the text box (ii) A default string to display in the text box.

This dialog box with two buttons: **OK** and **Cancel**. If the user clicks on OK button the window method *prompt()* will return entered value from the text box. If the user clicks on the Cancel button the window method *prompt()* returns *null*.

You can use prompt dialog box as follows:

```
<head>
<script type="text/javascript">
<!--
   var retVal = prompt("Enter your name : ",
"your name here");
   alert("You have entered : " +  retVal );
//-->
</script>
</head>
```

JavaScript Built-in Functions

Javascript String - split() Method

Description:

This method splits a String object into an array of strings by separating the string into substrings.

Syntax:

```
string.split([separator][, limit]);
```

Here is the detail of parameters:

- **separator :** Specifies the character to use for separating the string. If *separator* is omitted, the array returned contains one element consisting of the entire string.
- **limit :** Integer specifying a limit on the number of splits to be found.

Return Value:

- The split method returns the new array. Also, when the string is empty, split returns an array containing one empty string, rather than an empty array.

Example:

```
<html>
<head>
<title>JavaScript       String       split()
Method</title>
</head>
<body>
<script type="text/javascript">

var str = "Apples are round, and apples are
juicy.";
```

```
var splitted = str.split(" ", 3);

document.write( splitted );

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

This will produce following result:

```
Apples,are,round,
```

Javascript String - match() Method

Description:

This method is used to retrieve the matches when matching a string against a regular expression.

Syntax:

```
string.match( param )
```

Here is the detail of parameters:

- **param :** A regular expression object.

Return Value:

- If the regular expression does not include the g flag, returns the same result as regexp.exec(string).
- If the regular expression includes the g flag, the method returns an Array containing all matches.

Example:

```
<html>
<head>
<title>JavaScript       String       match()
Method</title>
</head>
<body>
<script type="text/javascript">
var str = "For more information, see Chapter
3.4.5.1";
var re = /(chapter \d+(\.\d)*)/i;
var found = str.match( re );
```

```
document.write(found );

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

This returns the array containing Chapter 3.4.5.1,Chapter 3.4.5.1,.1:

Javascript String - length Property

Description:

This property returns the number of characters in the string.

Syntax:

```
string.length
```

Here is the detail of parameters:

- A string

Return Value:

Returns the number of characters in the string.

Example:

```
<html>
<head>
<title>JavaScript       String       length
Property</title>
</head>
<body>
<script type="text/javascript">
   var str = new String( "This is string" );
   document.write("str.length     is:"     +
str.length);
</script>
</body>
</html>
```

This will produce following result:

```
str.length is:14
```

Description:

This method returns a subset of a String object.

Syntax:

```
string.substring(indexA, [indexB])
```

Here is the detail of parameters:

- **indexA :** An integer between 0 and one less than the length of the string.
- **indexB :** (optional) An integer between 0 and the length of the string.

Return Value:

- The substring method returns the new sub string based on given parameters.

Example:

```
<html>
<head>
<title>JavaScript     String     substring()
Method</title>
</head>
<body>
<script type="text/javascript">

var str = "Apples are round, and apples are
juicy.";

document.write("(1,2):    "                +
str.substring(1,2));
document.write("<br   />(0,10):   "        +
str.substring(0, 10));
document.write("<br  />(5):  "             +
str.substring(5));
</script>
</body>
</html>
```

This will produce following result:

```
(1,2): p
(0,10): Apples are
(5): s are round, and apples are juicy.
```

Javascript String - substr() Method

Description:

This method returns the characters in a string beginning at the specified location through the specified number of characters.

Syntax:

```
string.substr(start[, length]);
```

Here is the detail of parameters:

- **start :** Location at which to begin extracting characters (an integer between 0 and one less than the length of the string).
- **length :** The number of characters to extract.

Note: If start is negative, substr uses it as a character index from the end of the string.

Return Value:

- The substr method returns the new sub string based on given parameters.

Example:

```
<html>
<head>
<title>JavaScript String substr() Method</title>
</head>
<body>
<script type="text/javascript">

var str = "Apples are round, and apples are juicy.";

document.write("(1,2): "    + str.substr(1,2));
document.write("<br />(-2,2): "   + str.substr(-2,2));
document.write("<br />(1): "      + str.substr(1));
document.write("<br />(-20, 2): " + str.substr(-20,2));
document.write("<br />(20, 2): "  + str.substr(20,2));

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

This will produce following result:

```
(1,2): pp
(-2,2): Ap
(1): pples are round, and apples are juicy.
(-20, 2): Ap
(20, 2): d
```

Javascript Array slice() Method

Description:

Javascript array **slice()** method extracts a section of an array and returns a new array.

Syntax:

```
array.slice( begin [,end] );
```

Here is the detail of parameters:

- **begin** : Zero-based index at which to begin extraction. As a negative index, start indicates an offset from the end of the sequence.
- **end** : Zero-based index at which to end extraction.

Return Value:

Returns the extracted array based on the passed parameters.

Example:

```
<html>
<head>
<title>JavaScript Array slice Method</title>
</head>
<body>
<script type="text/javascript">
var arr = ["orange", "mango", "banana", "sugar", "tea"];
document.write("arr.slice( 1, 2) : " + arr.slice( 1, 2) );
document.write("<br />arr.slice( 1, 3) : " + arr.slice( 1, 3) );
</script>
</body>
</html>
```

This will produce following result:

```
arr.slice( 1, 2) : mango
arr.slice( 1, 3) : mango,banana
```

Javascript Array reverse() Method

Description:

Javascript array **reverse()** method reverses the element of an array. The first array element becomes the last and the last becomes the first.

Syntax:

```
array.reverse();
```

Here is the detail of parameters:

- **NA**

Return Value:

Returns the reversed single value of the array.

Example:

```
<html>
<head>
<title>JavaScript Array reverse Method</title>
</head>
<body>
<script type="text/javascript">
var arr = [0, 1, 2, 3].reverse();
document.write("Reversed array is : " + arr );
</script>
</body>
</html>
```

This will produce following result:

```
Reversed array is : 3,2,1,0
```

Javascript Array sort() Method

Description:

Javascript array **sort()** method sorts the elements of an array.

Syntax:

```
array.sort( compareFunction );
```

Here is the detail of parameters:

- **compareFunction** : Specifies a function that defines the sort order. If omitted, the array is sorted lexicographically.

Return Value:

Returns a sorted array.

Example:

```
<html>
<head>
<title>JavaScript Array sort Method</title>
</head>
<body>
<script type="text/javascript">
var arr = new Array("orange", "mango", "banana", "sugar");

var sorted = arr.sort();
document.write("Returned string is : " + sorted );

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

This will produce following result:

```
Returned array is : banana,mango,orange,sugar
```

Document **getElementById()** Method

Definition and Usage

The getElementById() method accesses the first element with the specified id.

Syntax

```
document.getElementById("id")
```

Example

Alert innerHTML of an element with a specific ID:

```
<html>
<head>
<script>
function getValue()
  {
  var x=document.getElementById("myHeader");
  alert(x.innerHTML);
  }
</script>
</head>
<body>

<h1 id="myHeader" onclick="getValue()">Click me!</h1>

</body>
</html>
```

getAttribute and setAttribute

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

```
 var x = document.createElement("INPUT");
 x.setAttribute("type", "radio");
 document.write(x.getAttribute("type"));
</script>
</body>
</html>
```

Assign value to TAG and textarea

1.

```
<html>
<script language="javascript" type="text/javascript">
function show()
{
 var x = document.getElementById("Myid1").value;
 document.getElementById("Myid").innerHTML = x;
}
</script>
<body>
<input type="text"  id="Myid1">
<input type="button" onClick="show()" value="Click">
<div id = "Myid"></div>
</body>
</html>
```

2.

```
<html>
<script language="javascript" type="text/javascript">
function show()
{
 var x = document.getElementById("Myid1").value;
 document.getElementById("Myid").innerHTML = x;
}
```

```
</script>
<body>
<input type="text"  id="Myid1">
<input type="button" onClick="show()" value="Click">
<textarea id="Myid"></textarea>

</body>
</html>
```

Assign value to TAG parameter <input>

```
<html>
<script language="javascript" type="text/javascript">
function show()
{
 var x = document.getElementById("Myid1").value;
 document.getElementById("bk").value = x;
}
</script>
<body>
<input type="text"  id="Myid1">
<input type="button" onClick="show()" value="Click">
<div id = "Myid"></div>
<input type="text" name="bk" id="bk">
</body>
</html>
```