JavaScript is most commonly used as a client side scripting language. This means that JavaScript code is written into an HTML page. When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it's up to the browser to do something with it.

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 not a programming language in strict sense. Instead, it is a scripting language because it uses the browser to do the dirty work. If you command an image to be replaced by another one, JavaScript tells the browser to go do it. Because the browser actually does the work, you only need to pull some strings by writing some relatively easy lines of code. Thats what makes JavaScript an easy language to start with.

Features

- 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 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.

COPYRIGHT ©TALENTSPRINT, 2015. ALL RIGHTS RESERVED.

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

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

- We can not 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 can not be used for Networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocess capabilities.

How to use Javascript

A JavaScript consists of JavaScript statements that are placed within the

```
<script>... </script>
```

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>
```

First JavaScript code

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

We added an optional HTML comment that surrounds our Javascript code. This is to save our code from a browser that does not support Javascript. The comment ends with a "//–>". Here "//" signifies a comment in Javascript, so we add that to prevent a browser from reading the end of the HTML comment in as a piece of Javascript code.

Note
The function *document.write* which writes a string into our HTML document. This function can be used to write text, HTML, or both.

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>
```

Comments

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 at a time.

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 sample code showing how to use comments in script file

```
<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.

JavaScript in $< head >$ 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 $< body >$ 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>
```

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>
```

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")
}
```

Data Types

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 e.g., 123, 120.50 etc.
- Strings of text e.g. “This text string” etc.
- Boolean - Values are *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.

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:

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>
```

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. 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.

Operators

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

Arithmatic Operators

There are following arithmatic operators supported by JavaScript language:

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

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

Operator	Example
+	A + B will give 30
-	A - B will give -10
*	A * B will give 200
/	B / A will give 2
%	B % A will give 0
++	A++ will give 11
–	A– will give 9

Comparison Operators

There are following comparison operators supported by JavaScript language

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

Operator	Example
==	(A == B) is not true.
!=	(A != B) is true.
>	(A > B) is not true.
<	(A < B) is true.
>=	(A >= B) is not true.
<=	(A <= B) is true.

Logical Operators

There are following logical operators supported by JavaScript language

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

Operator	Example
&&	(A && B) is true.
||	(A || B) is true.
!	!(A && B) is false.

Bitwise Operators

There are following bitwise operators supported by JavaScript language Assume variable A holds 2 and variable B holds 3 then:

Operator	Example
&	(A & B) is 2 .
|	(A | B) is 3.
∧	(A ∧ B) is 1.
∼	∼ B is -4 .
<<	(A << 1) is 4.
>>	(A >> 1) is 1.
>>>	(A >>> 1) is 1.

Assignment Operators

There are following assignment operators supported by JavaScript language

Operator	Example
=	C = A + B will assigne value of A + B into C
+=	C += A is equivalent to C = C + A
-=	C -= A is equivalent to C = C- A
*=	C *= A is equivalent to C = C * A
/=	C /= A is equivalent to C = C / A
%=	C %= A is equivalent to C = C % A

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	Example
? :	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"

0.1 Dialog Boxes

Introduction

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.

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.

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

```
        <!--
            alert("Warning Message");
         //-->
        </script>
    </head>
```

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.

```
    <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.

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

0.2 Control Statements

Conditional Statements

JavaScript supports conditional statements which are used to perform different actions based on different conditions.

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

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 language = "javascript" 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>
```

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
}
```

Example

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

Looping Statements

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.

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 language = "javascript" 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>
```

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

Semicolon used at the end of the do while loop.

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

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

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 will test whether given condition is *true* or *false*. If condition gives *true* then code given inside the loop will be executed otherwise loop will be terminated.

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 language = "javascript" 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>
```

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.

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

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