JavaScript

What is JavaScript

- JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the web pages.
- It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document.
- It was introduced in the year 1995 for adding programs to the webpages and it is called as live script
- With JavaScript, users can build modern web applications to interact directly without reloading the page every time.
- The traditional website uses js to provide several forms of interactivity and simplicity.

Features of JavaScript

There are following features of JavaScript:

1. All popular web browsers support JavaScript as they provide built-in execution environments.
2. JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
3. JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
4. JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
5. It is a light-weighted and interpreted language.
6. It is a case-sensitive language.

7. JavaScript is supportable in several operating systems including, Windows, macOS, etc.
8. It provides good control to the users over the web browsers.

Applications of Java script Programming

As mentioned before, **Javascript** is one of the most widely used **programming languages** (Front-end as well as Back-end). It has it's presence in almost every area of software development. I'm going to list few of them here:

- **Client side validation** - This is really important to verify any user input before submitting it to the server and Java script plays an important role in validating those inputs at front-end itself.
- **Manipulating HTML Pages** - Javascript helps in manipulating HTML page on the fly. This helps in adding and deleting any HTML tag very easily using javascript and modify your HTML to change its look and feel based on different devices and requirements.
- **User Notifications** - You can use Javascript to raise dynamic pop-ups on the webpages to give different types of notifications to your website visitors.
- **Back-end Data Loading** - Javascript provides Ajax library which helps in loading back-end data while you are doing some other processing. This really gives an amazing experience to your website visitors.
- **Presentations** - JavaScript also provides the facility of creating presentations which gives website look and feel. JavaScript provides RevealJS and BespokeJS libraries to build a web-based slide presentations.
- **Server Applications** - Node JS is built on Chrome's Javascript runtime for building fast and scalable network applications. This is an event based library which helps in developing very sophisticated server applications including Web Servers.

This list goes on, there are various areas where millions of software developers are happily using Javascript to develop great websites and others softwares.

Advantages/merits/pros 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/demerits/cons of 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 multi-threading or multiprocessor capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

Scripting Laguages

Scripting languages are like programming languages that allow us to write programs in form of script. These scripts are interpreted not compiled and executed line by line.

Scripting language is used to create dynamic web pages.

Client-side Scripting

Client-side scripting refers to the programs that are executed on client-side. Client-side scripts contains the instruction for the browser to be executed in response to certain user's action.

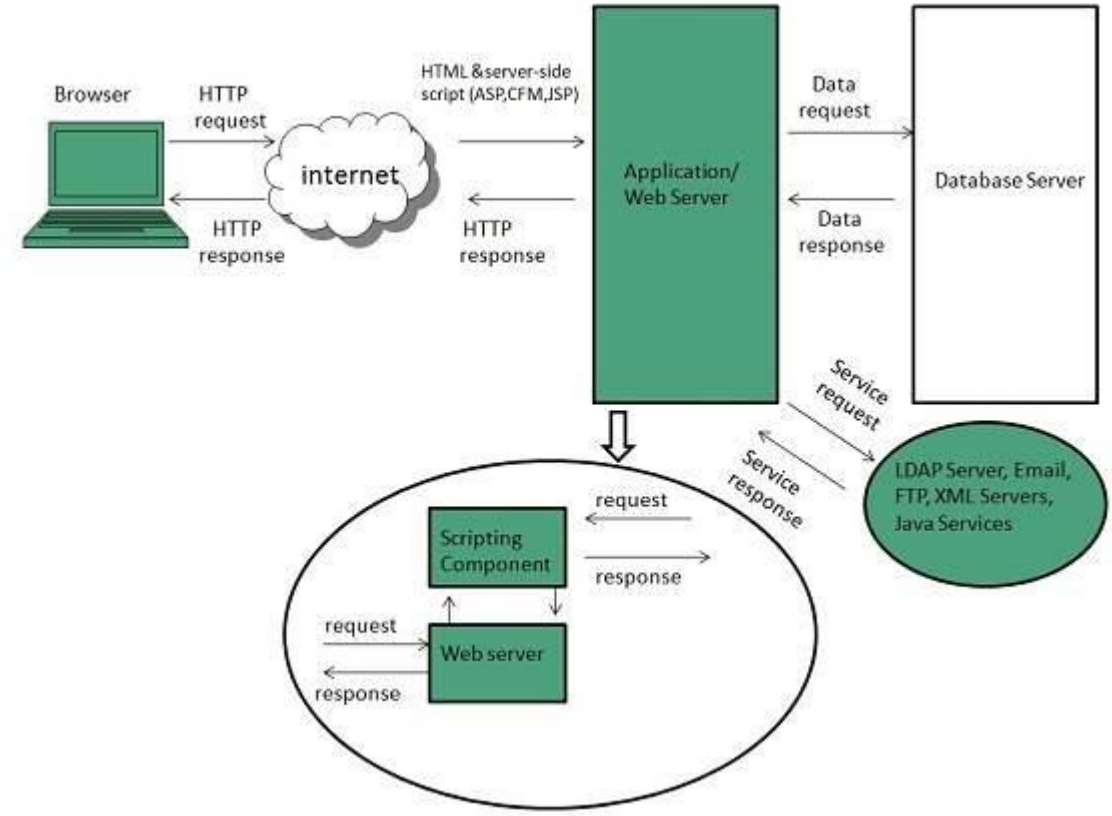
Client-side scripting programs can be embedded into HTML files or also can be kept as separate files.

Following table describes commonly used Client-Side scripting languages:

S.N.	Scripting Language Description
1.	**JavaScript** It is a prototype based scripting language. It inherits its naming conventions from java. All java script files are stored in file having **.js** extension.
2.	**ActionScript**It is an object oriented programming language used for the development of websites and software targeting Adobe flash player.
3.	**Dart** It is an open source web programming language developed by Google. It relies on source-to-source compiler to JavaScript.
4.	**VBScript** It is an open source web programming language developed by Microsoft. It is superset of JavaScript and adds optional static typing class-based object oriented programming.

Server-side Scripting

Sever-side scripting acts as an interface for the client and also limit the user access the resources on web server. It can also collects the user's characteristics in order to customize response.

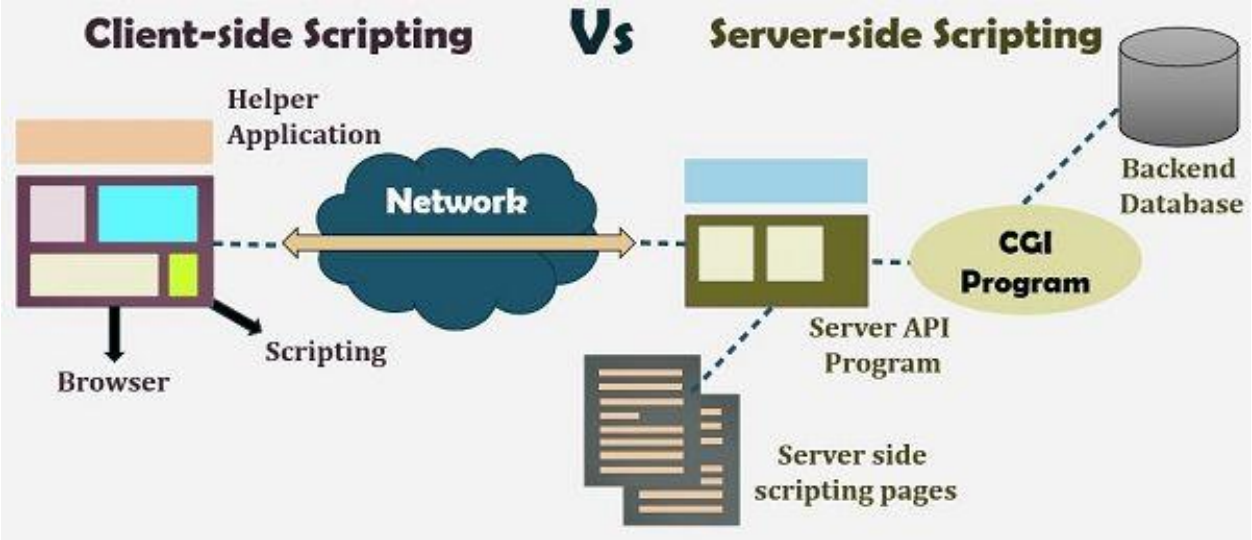


Following table describes commonly used Server-Side scripting languages:

S.N.	Scripting Language Description
1.	**ASP** Active Server Pages (ASP)is server-side script engine to create dynamic web pages. It supports **Component Object Model (COM)** which enables ASP web sites to access functionality of libraries such as DLL.
2.	**ActiveVFP** It is similar to PHP and also used for creating dynamic web pages. It uses native **Visual Foxpro** language and database.
3.	**ASP.net** It is used to develop dynamic websites, web applications, and web services.
4.	**Java** Java Server Pages are used for creating dynamic web applications. The Java code is compiled into byte code and run by **Java Virtual Machine (JVM).**
5.	**Python** It supports multiple programming paradigms such as object-oriented, and functional

	programming. It can also be used as non-scripting language using third party tools such as **Py2exe** or **Pyinstaller.**
6.	**WebDNA** It is also a server-side scripting language with an embedded database system.

3.1 Overview of Client side and Server side Scripting:



- The scripts can be written in two forms, at the server end (back end) or at the client end (server end).
- The main difference between server-side scripting and client-side scripting is that the server side scripting involves **server** for its processing.
- On the other hand, client-side scripting requires **browsers** to run the scripts on the client machine but does not interact with the server while processing the client-side scripts.
- Following table describe the difference between them

BASIS FOR COMPARISON	SERVER-SIDE SCRIPTING	CLIENT-SIDE SCRIPTING
Basic	Works in the back end which could not be visible at the client end.	Works at the front end and script are visible among the users.
Processing	Requires server interaction.	Does not need interaction with the server.
Languages involved	PHP, ASP.net, Ruby on Rails, ColdFusion, Python, etcetera.	HTML, CSS, JavaScript, etc.
Affect	Could effectively customize the web pages and provide dynamic websites.	Can reduce the load to the server.
Security	Relatively secure.	Insecure

BASIS FOR COMPARISON	SERVER-SIDE SCRIPTING	CLIENT-SIDE SCRIPTING
Source Code	It is not visible to the user because it's output of server side html page	It is visible to the user
Dependency	In this any server side technology can be used it does not depend on the client	It's usually depends on browser and it's version

DHTML stands for **Dynamic Hypertext Markup language** i.e., **Dynamic HTML**.

Dynamic HTML is not a markup or programming language but it is a term that combines the features of various web development technologies for creating the web pages dynamic and interactive.

The DHTML application was introduced by Microsoft with the release of the 4th version of IE (Internet Explorer) in 1997.

Components of Dynamic HTML

DHTML consists of the following four components or languages:

- HTML 4.0
- CSS
- JavaScript
- DOM.

HTML 4.0

HTML is a client-side markup language, which is a core component of the DHTML. It defines the structure of a web page with various defined basic elements or tags.

CSS

CSS stands for Cascading Style Sheet, which allows the web users or developers for controlling the style and layout of the HTML elements on the web pages.

JavaScript

JavaScript is a scripting language which is done on a client-side. The various browser supports JavaScript technology. DHTML uses the JavaScript technology for accessing, controlling, and manipulating the HTML elements. The statements in JavaScript are the commands which tell the browser for performing an action.

DOM

DOM is the **document object model**. It is a w3c standard, which is a standard interface of programming for HTML. It is mainly used for defining the objects and properties of all elements in HTML.

3.2 Structure of Java Script

JavaScript - Syntax

- javaScript can be implemented using JavaScript statements that are placed within the **<script>… </script>** HTML tags in a web page.
- You can place the **<script>** tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the **<head>** tags.
- The <script> tag alerts the browser program to start interpreting all the text between these tags as a script.
- A simple syntax of your JavaScript will appear 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.

- **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>
```

Places to put JavaScript code

1. Between the body tag of html
2. Between the head tag of html
3. **In .js file (external javaScript)**

JavaScript Example: code between the body tag

- In the above example, we have displayed the dynamic content using JavaScript. Let's see the simple example of JavaScript that displays alert dialog box.

```
<script type="text/javascript">
 alert("Hello Javatpoint");
</script>
```

JavaScript Example: code between the head tag

- Let's see the same example of displaying alert dialog box of JavaScript that is contained inside the head tag.
- In this example, we are creating a function msg().
- To create function in JavaScript, you need to write function with function_name as given below.
- To call function, you need to work on event. Here we are using onclick event to call msg() function.

```
<html>
<head>
<script type="text/javascript">
function msg(){
 alert("Hello Javatpoint");
}
```

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

External JavaScript file

We can create external JavaScript file and embed it in many html page.

It provides **code re usability** because single JavaScript file can be used in several html pages.

An external JavaScript file must be saved by **.js extension**. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.

Let's create an external JavaScript file that prints Hello Javatpoint in a alert dialog box.

message.js

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

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

index.html

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

Advantages of External JavaScript

There will be following benefits if a user creates an external javascript:

1. It helps in the reusability of code in more than one HTML file.

2. It allows easy code readability.
3. It is time-efficient as web browsers cache the external js files, which further reduces the page loading time.
4. It enables both web designers and coders to work with html and js files parallelly and separately, i.e., without facing any code conflictions.
5. The length of the code reduces as only we need to specify the location of the js file.

Disadvantages of External JavaScript

There are the following disadvantages of external files:

1. The stealer may download the coder's code using the url of the js file.
2. If two js files are dependent on one another, then a failure in one file may affect the execution of the other dependent file.
3. The web browser needs to make an additional http request to get the js code.
4. A tiny to a large change in the js code may cause unexpected results in all its dependent files.
5. We need to check each file that depends on the commonly created external javascript file.
6. If it is a few lines of code, then better to implement the internal javascript code.

JavaScript Comment

The **JavaScript comments** are meaningful way to deliver message. It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.

The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.

Advantages of JavaScript comments

There are mainly two advantages of JavaScript comments.

1. **To make code easy to understand** It can be used to elaborate the code so that end user can easily understand the code.
2. **To avoid the unnecessary code** It can also be used to avoid the code being executed. Sometimes, we add the code to perform some action. But after sometime, there may be need to disable the code. In such case, it is better to use comments.

Types of JavaScript Comments

There are two types of comments in JavaScript.

1. Single-line Comment
2. Multi-line Comment

JavaScript Single line Comment

It is represented by double forward slashes (//). It can be used before and after the statement.

Let's see the example of single-line comment i.e. added before the statement.

```
<script>
// It is single line comment
document.write("hello javascript");
</script>
```

JavaScript Multi line Comment

It can be used to add single as well as multi line comments. So, it is more convenient.

It is represented by forward slash with asterisk then asterisk with forward slash. For example:

```
/* your code here  */

It can be used before, after and middle of the statement.

<script>
/* It is multi line comment.
It will not be displayed */
document.write("example of javascript multiline comment");
</script>
```

JavaScript Variable

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

Correct JavaScript variables

```
var x = 10;
var _value="sonoo";
```

Incorrect JavaScript variables

```
var  123=30;
var *aa=320;
```

Example of JavaScript variable

Let's see a simple example of JavaScript variable.

```
<script>
var x = 10;
var y = 20;
var z=x+y;
document.write(z);
</script>
```

Output of the above example

```
30
```

JavaScript local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. For example:

```
<script>
function abc(){
var x=10;//local variable
}
</script>
```

Or,

```
<script>
If(10<13){
var y=20;//JavaScript local variable
}
</script>
```

JavaScript global variable

A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable.

A **JavaScript global variable** is declared outside the function or declared with window object. It can be accessed from any function.

For example:

```
<script>
```

```
var data=200;//gloabal variable
function a(){
document.writeln(data);
}
function b(){
document.writeln(data);
}
a();//calling JavaScript function
b();
</script>
```

Declaring JavaScript global variable within function

To declare JavaScript global variables inside function, you need to use **window object**. For example:

```
window.value=90;
```

Now it can be declared inside any function and can be accessed from any function. For example:

```
function m(){
      window.value=100;//declaring global variable by window object
}
function n(){
      alert(window.value);//accessing global variable from other function

}
```

Internals of global variable in JavaScript

When you declare a variable outside the function, it is added in the window object internally. You can access it through window object also. For example:

```
var value=50;
function a(){
      alert(window.value);//accessing global variable
}
```

Javascript Data Types

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use **var** here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

```
var a=40;//holding number
var b="Rahul";//holding string
```

JavaScript primitive data types

There are five types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

JavaScript non-primitive data types

The non-primitive data types are as follows:

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

What is an Operator?

Let us take a simple expression **4 + 5 is equal to 9**. Here 4 and 5 are called **operands** and '+' is called the **operator**. JavaScript supports the following types 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.

Arithmetic Operators

JavaScript supports the following arithmetic operators −

Assume variable A holds 10 and variable B holds 20, then −

Sr.No.	Operator & Description
1	**+ (Addition)** Adds two operands **Ex:** A + B will give 30
2	**- (Subtraction)** Subtracts the second operand from the first **Ex:** A - B will give -10
3	*** (Multiplication)** Multiply both operands **Ex:** A * B will give 200
4	**/ (Division)** Divide the numerator by the denominator **Ex:** B / A will give 2
5	**% (Modulus)** Outputs the remainder of an integer division **Ex:** B % A will give 0
6	**++ (Increment)** Increases an integer value by one

	Ex: A++ will give 11
7	**-- (Decrement)** Decreases an integer value by one **Ex:** A-- will give 9

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

Example

The following code shows how to use arithmetic operators in JavaScript.

```
<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>

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

Output

```
a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
++a = 35
--b = 8
Set the variables to different values and then try...
```

Comparison Operators

JavaScript supports the following comparison operators −

Assume variable A holds 10 and variable B holds 20, then −

Sr.No.	Operator & Description
1	**= = (Equal)** Checks if the value of two operands are equal or not, if yes, then the condition becomes true. **Ex:** (A == B) is not true.
2	**!= (Not Equal)** Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true. **Ex:** (A != B) is true.
3	**> (Greater than)** Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A > B) is not true.
4	**< (Less than)** Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A < B) is true.
5	**>= (Greater than or Equal to)** Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.

	Ex: (A >= B) is not true.
6	**<= (Less than or Equal to)** Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true. **Ex:** (A <= B) is true.

Example

The following code shows how to use comparison operators in JavaScript.

Live Demo

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

Output

```
(a == b) => false
```

```
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
a <= b) => true
Set the variables to different values and different operators and then try...
```

Logical Operators

JavaScript supports the following logical operators −

Assume variable A holds 10 and variable B holds 20, then −

Sr.No.	Operator & Description
1	**&& (Logical AND)** If both the operands are non-zero, then the condition becomes true. **Ex:** (A && B) is true.
2	**|| (Logical OR)** If any of the two operands are non-zero, then the condition becomes true. **Ex:** (A || B) is true.
3	**! (Logical NOT)** Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. **Ex:** ! (A && B) is false.

Example

Try the following code to learn how to implement Logical Operators in JavaScript.

Live Demo

```
<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>
```

Output

```
(a && b) => false
(a || b) => true
!(a && b) => true
Set the variables to different values and different operators and then try...
```

Example

Try the following code to implement Bitwise operator in JavaScript.

Live Demo

```
<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>
```

```
(a & b) => 2
(a | b) => 3
(a ^ b) => 1
(~b) => -4
(a << b) => 16
(a >> b) => 0
Set the variables to different values and different operators and then try...
```

Assignment Operators

JavaScript supports the following assignment operators −

Sr.No.	Operator & Description
1	**= (Simple Assignment)** Assigns values from the right side operand to the left side operand **Ex:** C = A + B will assign the value of A + B into C
2	**+= (Add and Assignment)** It adds the right operand to the left operand and assigns the result to the left operand. **Ex:** C += A is equivalent to C = C + A
3	**−= (Subtract and Assignment)** It subtracts the right operand from the left operand and assigns the result to the left operand. **Ex:** C -= A is equivalent to C = C - A
4	***= (Multiply and Assignment)** It multiplies the right operand with the left operand and assigns the result to the left operand. **Ex:** C *= A is equivalent to C = C * A
5	**/= (Divide and Assignment)** It divides the left operand with the right operand and assigns the result to the left operand. **Ex:** C /= A is equivalent to C = C / A
6	**%= (Modules and Assignment)** It takes modulus using two operands and assigns the result to the left operand. **Ex:** C %= A is equivalent to C = C % A

Note − Same logic applies to Bitwise operators so they will become like <<=, >>=, >>=, &=, |= and ^=.

Example

Try the following code to implement assignment operator in JavaScript.

Live Demo

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = 33;
            var b = 10;
            var linebreak = "<br />";

            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>
```

Output

```
Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0
```

Control Structure

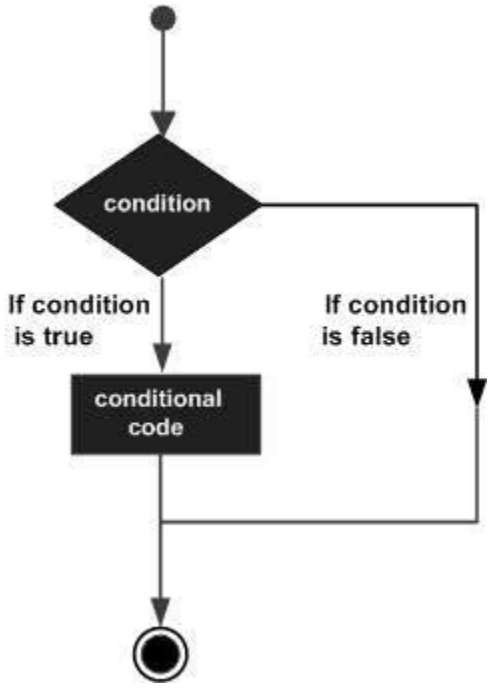
JavaScript - if...else Statement

While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use 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 the **if..else** statement.

Flow Chart of if-else

The following flow chart shows how the if-else statement works.



JavaScript supports the 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

The syntax for a basic if statement is as follows −

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

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

Example

Try the following example to understand how the **if** statement works.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var age = 20;

            if( age > 18 ) {
               document.write("<b>Qualifies for driving</b>");
            }
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
Qualifies for driving
Set the variable to different value and then try...
```

if...else statement

The **'if...else'** statement is the next form of control statement that allows JavaScript to execute statements in a 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, the given statement(s) in the 'if' block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example

Try the following code to learn how to implement an if-else statement in JavaScript.

```
<html>
   <body>
      <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>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
Does not qualify for driving
```

```
Set the variable to different value and then try...
```

if...else if... statement

The **if...else if...** statement is an advanced form of **if…else** that allows JavaScript to make a correct decision out of several conditions.

Syntax

The syntax of an if-else-if statement is as follows −

```
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 a part of the **else** clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the **else** block is executed.

Example

Try the following code to learn how to implement an if-else-if statement in JavaScript.

```
<html>
   <body>
      <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>
      <p>Set the variable to different value and then try...</p>
   </body>
<html>
```

Output

Maths Book

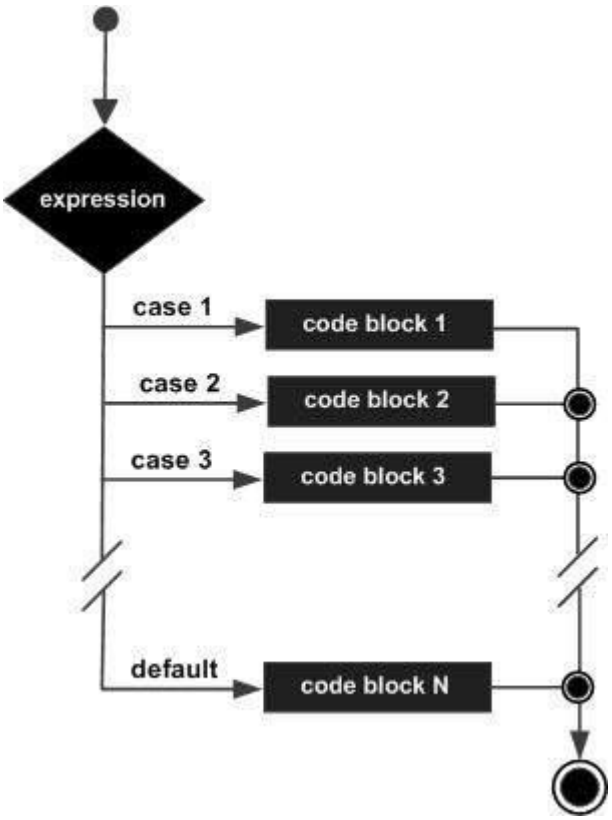
JavaScript - Switch Case

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.

Flow Chart

The following flow chart explains a switch-case statement works.



Syntax

The objective of a **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 the end of a 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

Try the following example to implement switch-case statement.

```
<html>
   <body>
      <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>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
Entering switch block
Good job
Exiting switch block
Set the variable to different value and then try...
```

Break statements play a major role in switch-case statements. Try the following code that uses switch-case statement without any break statement.

```
<html>
   <body>
      <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>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

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

JavaScript - While Loops

While writing a program, you may encounter a situation where you need to perform an action over and over again. In such situations, you would need to write loop statements to reduce the number of lines.

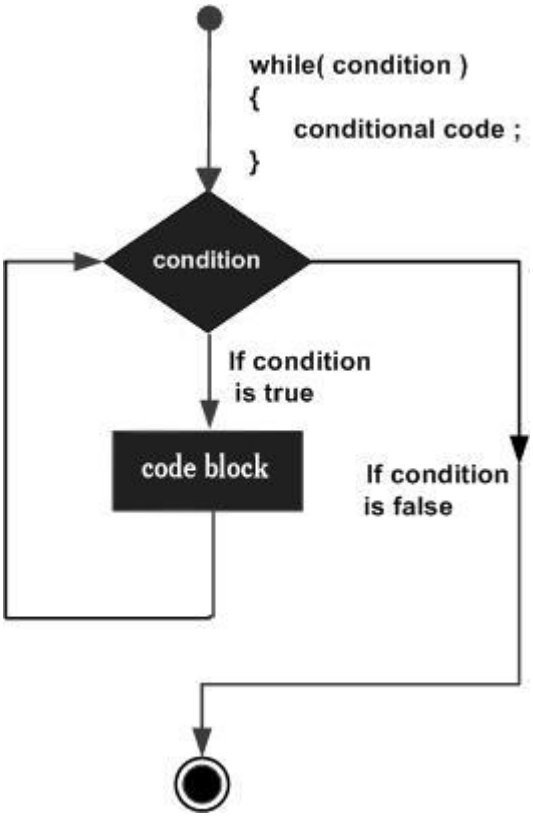
JavaScript supports all the necessary loops to ease down the pressure of programming.

The while Loop

The most basic loop in JavaScript is the **while** loop which would be discussed in this chapter. The purpose of a **while** loop is to execute a statement or code block repeatedly as long as an **expression** is true. Once the expression becomes **false,** the loop terminates.

Flow Chart

The flow chart of **while loop** looks as follows −



Syntax

The syntax of **while loop** in JavaScript is as follows −

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

}

Example

Try the following example to implement while loop.

Live Demo

```
<html>
   <body>

      <script type = "text/javascript">
         <!--
            var count = 0;
            document.write("Starting Loop ");

            while (count < 10) {
               document.write("Current Count : " + count + "<br />");
               count++;
            }

            document.write("Loop stopped!");
         //-->
      </script>

      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

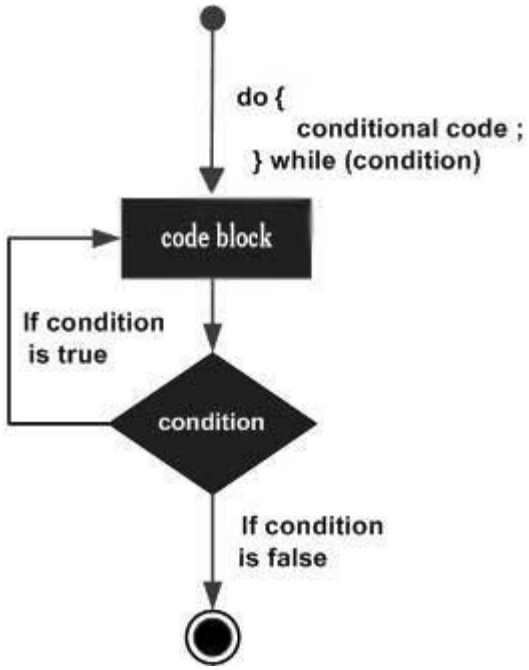
```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
Set the variable to different value and then try...
```

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**.

Flow Chart

The flow chart of a **do-while** loop would be as follows −



Syntax

The syntax for **do-while** loop in JavaScript is as follows −

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

Note − Don't miss the semicolon used at the end of the **do...while** loop.

Example

Try the following example to learn how to implement a **do-while** loop in JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var count = 0;

            document.write("Starting Loop" + "<br />");
            do {
               document.write("Current Count : " + count + "<br />");
               count++;
            }

            while (count < 5);
            document.write ("Loop stopped!");
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
```

```
Loop Stopped!
```

JavaScript - For Loop

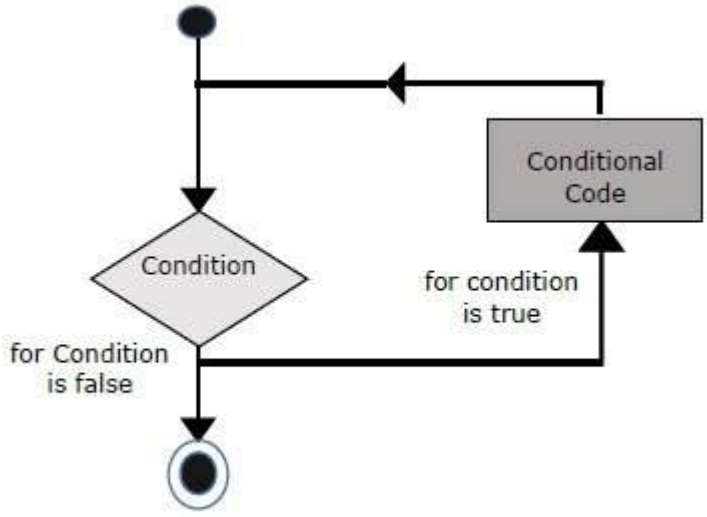
he '**for**' loop is the most compact form of looping. It includes the following three important parts −

- 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 a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The **iteration statement** where you can increase or decrease your counter.

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

Flow Chart

The flow chart of a **for** loop in JavaScript would be as follows −



Syntax

The syntax of **for** loop is JavaScript is as follows −

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

Example

Try the following example to learn how a **for** loop works in JavaScript.

Live Demo

```
<html>
   <body>
      <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>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
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 - Loop Control

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

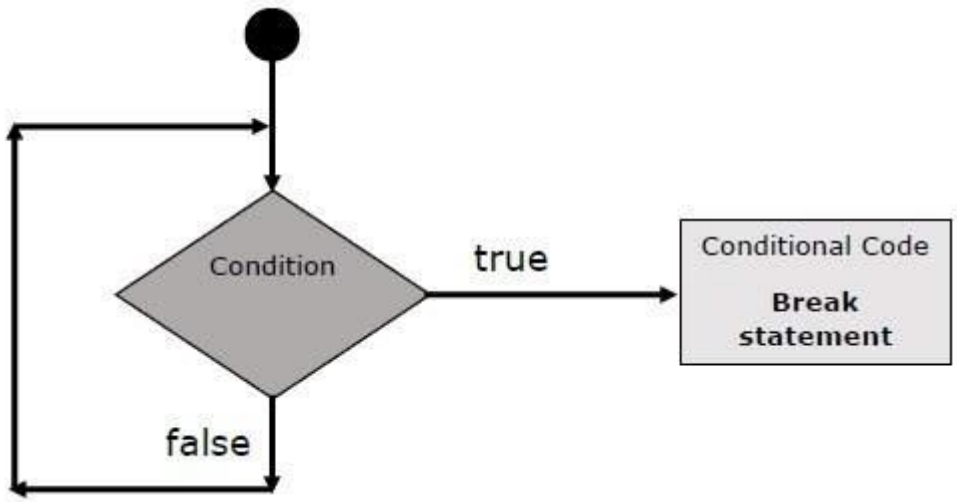
To handle all such situations, JavaScript provides **break** and **continue** statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

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.

Flow Chart

The flow chart of a break statement would look as follows −



Example

The following 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 the closing curly brace −

Live Demo

```
<html>
   <body>
      <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>

      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
Entering the loop
2
3
4
5
Exiting the loop!
Set the variable to different value and then try...
```

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 the remaining code block. When a **continue** statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the 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 −

Live Demo

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var x = 1;
            document.write("Entering the loop<br /> ");
```

```
            while (x < 10) {
               x = x + 1;

               if (x == 5) {
                  continue;    // skip rest of the loop body
               }
               document.write( x + "<br />");
            }
            document.write("Exiting the loop!<br /> ");
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
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 a block of code. We will see two different examples to understand how to use labels 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.

Try the following two examples for a better understanding of Labels.

Example 1

The following example shows how to implement Label with a break statement.

Live Demo

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

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

Output

```
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!
```

JavaScript - Functions

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Like any other advanced 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 the earlier chapters. We were using these functions again and again, but they had been written in core JavaScript only once.

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

Function Definition

Before we use a function, we need to define it. 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.

Syntax

The basic syntax is shown here.

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

Example

Try the following example. It defines a function called sayHello that takes no parameters −

```
<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 shown in the following code.

Live Demo

```
<html>
   <head>
      <script type = "text/javascript">
         function sayHello() {
            document.write ("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 write method and then try...</p>
   </body>
</html>
```

Output

Function Parameters

Till now, we have seen functions without 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

Try the following example. We have modified our **sayHello** function here. Now it takes two parameters.

```
<html>
   <head>
      <script type = "text/javascript">
         function sayHello(name, age) {
            document.write (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>
```

Output

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 the function to return their multiplication in your calling program.

Example

Try the following example. It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program.

Live Demo

```
<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');
            document.write (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>
```

Output

There is a lot to learn about JavaScript functions, however we have covered the most important concepts in this tutorial.

- JavaScript Nested Functions
- JavaScript Function( ) Constructor
- JavaScript Function Literals

What is an Event ?

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

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a 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.

Events are a part of the Document Object Model (DOM) Level 3 and **every HTML element contains a set of event**s which can trigger JavaScript Code.

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

When a user visit your website, they do things like click on text and images and given links, hover over things etc. These are examples of what JavaScript calls events.

We can write our event handlers in Javascript or vbscript and can specify these event handlers as a value of event tag attribute. The HTML 4.01 specification defines 19 event attributes as listed below

<body> and <frameset> Level Events

There are only two attributes which can be used to trigger any javascript or vbscript code when there is any event occurs on document level.

Attribute	Value	Description
onload	script	Script runs when a HTML document loads
onunload	script	Script runs when a HTML document unloads

NOTE − Here script refer to any VBScript or JavaScript function or piece of code.

<form> Level Events

There are following six attributes which can be used to trigger any javascript or vbscript code when there is any event occurs on form level.

Attribute	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

Keyboard Events

There are following three events which are generated by keyboard. These events are not valid in base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style, and title elements.

Attribute	Value	Description
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

Other Events(mouse events)

There following other 7 events which are generated by **mouse** when it comes in contact of any HTML tag. These events are not valid in base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style, title elements.

Attribute	Value	Description
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

DHTML Events

An event is defined as changing the occurrence of an object.

It is compulsory to add the events in the DHTML page. Without events, there will be no dynamic content on the HTML page. The event is a term in the HTML, which triggers the actions in the web browsers.

Suppose, any user clicks an HTML element, then the JavaScript code associated with that element is executed. Actually, the event handlers catch the events performed by the user and then execute the code.

Example of events:

1. Click a button.
2. Submitting a form.
3. An image loading or a web page loading, etc.

Following table describes the Event Handlers used in the DHTML:

Following table describes the Event Handlers used in the DHTML:

S.No.	Event	When it occurs
1.	onabort	It occurs when the user aborts the page or media file loading.
2.	onblur	It occurs when the user leaves an HTML object.
3.	onchange	It occurs when the user changes or updates the value of an object.
4.	onclick	It occurs or triggers when any user clicks on an HTML element.
5.	ondblclick	It occurs when the user clicks on an HTML element two times together.
6.	onfocus	It occurs when the user focuses on an HTML element. This event handler works opp to onblur.
7.	onkeydown	It triggers when a user is pressing a key on a keyboard device. This event handler v for all the keys.
8.	onkeypress	It triggers when the users press a key on a keyboard. This event handler is not trig for all the keys.
9.	onkeyup	It occurs when a user released a key from a keyboard after pressing on an object o element.
10.	onload	It occurs when an object is completely loaded.
11.	onmousedown	It occurs when a user presses the button of a mouse over an HTML element.
12.	onmousemove	It occurs when a user moves the cursor on an HTML object.
13.	onmouseover	It occurs when a user moves the cursor over an HTML object.
14.	onmouseout	It occurs or triggers when the mouse pointer is moved out of an HTML element.
15.	onmouseup	It occurs or triggers when the mouse button is released over an HTML element.
16.	onreset	It is used by the user to reset the form.
17.	onselect	It occurs after selecting the content or text on a web page.
18.	onsubmit	It is triggered when the user clicks a button after the submission of a form.
19.	onunload	It is triggered when the user closes a web page.

Following are the different examples using the different event handlers, which helps us to understand the concept of DHTML events:

Example 1: This example uses the onclick event handler, which is used to change the text after clicking.

```
<html>
```

```
<head>
<title>
Example of onclick event
</title>
<script type="text/javascript">
function ChangeText(ctext)
{
ctext.innerHTML=" Hi JavaTpoint! ";
}
</script>
</head>
<body>
<font color="red"> Click on the Given text for changing it: <br>
</font>
<font color="blue">
<h1 onclick="ChangeText(this)"> Hello World! </h1>
</font>
</body>
</html>
```

onclick Event Type

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

Example

Try the following example.

Live Demo

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

   <body>
      <p>Click the following button and see result</p>
      <form>
         <input type = "button" onclick = "sayHello()" value = "Say Hello" />
      </form>
   </body>
</html>
```

Output

onsubmit Event Type

onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

Example

The following example shows how to use onsubmit. 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.

Try the following example.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function validate () {
               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 create nice effects with images or even with text as well. The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element. Try the following example.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function over() {
               document.write ("Mouse Over");
            }
            function out() {
               document.write ("Mouse Out");
            }
         //-->
      </script>
   </head>

   <body>
      <p>Bring your mouse inside the division to see the result:</p>
      <div onmouseover = "over()" onmouseout = "out()">
         <h2> This is inside the division </h2>
      </div>
```

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

Output

Mouse Over

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 discuss each dialog box one by one.

Alert Dialog Box

An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

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

Example

Live Demo

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function Warn() {
               alert ("This is a warning message!");
               document.write ("This is a warning message!");
            }
         //-->
      </script>
   </head>

   <body>
      <p>Click the following button to see the result: </p>
      <form>
         <input type = "button" value = "Click Me" onclick = "Warn();" />
      </form>
   </body>
</html>
```

Output

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

Example

Live Demo

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

   <body>
      <p>Click the following button to see the result: </p>
      <form>
         <input type = "button" value = "Click Me" onclick =
"getConfirmation();" />
      </form>
   </body>
</html>
```

Output

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 enables 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 and (ii) a default string to display in the text box.

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

Example

The following example shows how to use a prompt dialog box −

Live Demo

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function getValue() {
               var retVal = prompt("Enter your name : ", "your name here");
               document.write("You have entered : " + retVal);
            }
         //-->
      </script>
   </head>
```

```
   <body>
      <p>Click the following button to see the result: </p>
      <form>
         <input type = "button" value = "Click Me" onclick = "getValue();" />
      </form>
   </body>
</html>
```

Output

Click the following button to see the result:

JavaScript - Void Keyword

void is an important keyword in JavaScript which can be used as a unary operator that appears before its single operand, which may be of any type. This operator specifies an expression to be evaluated without returning a value.

Syntax

The syntax of **void** can be either of the following two −

```
<head>
   <script type = "text/javascript">
      <!--
         void func()
         javascript:void func()
         or:
         void(func())
         javascript:void(func())
      //-->
   </script>
</head>
```

Document Object Model (DOM)

The **document object** represents the whole html document.

When html document is loaded in the browser, it becomes a document object. It is the **root element** that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

As mentioned earlier, it is the object of window. So

window.document

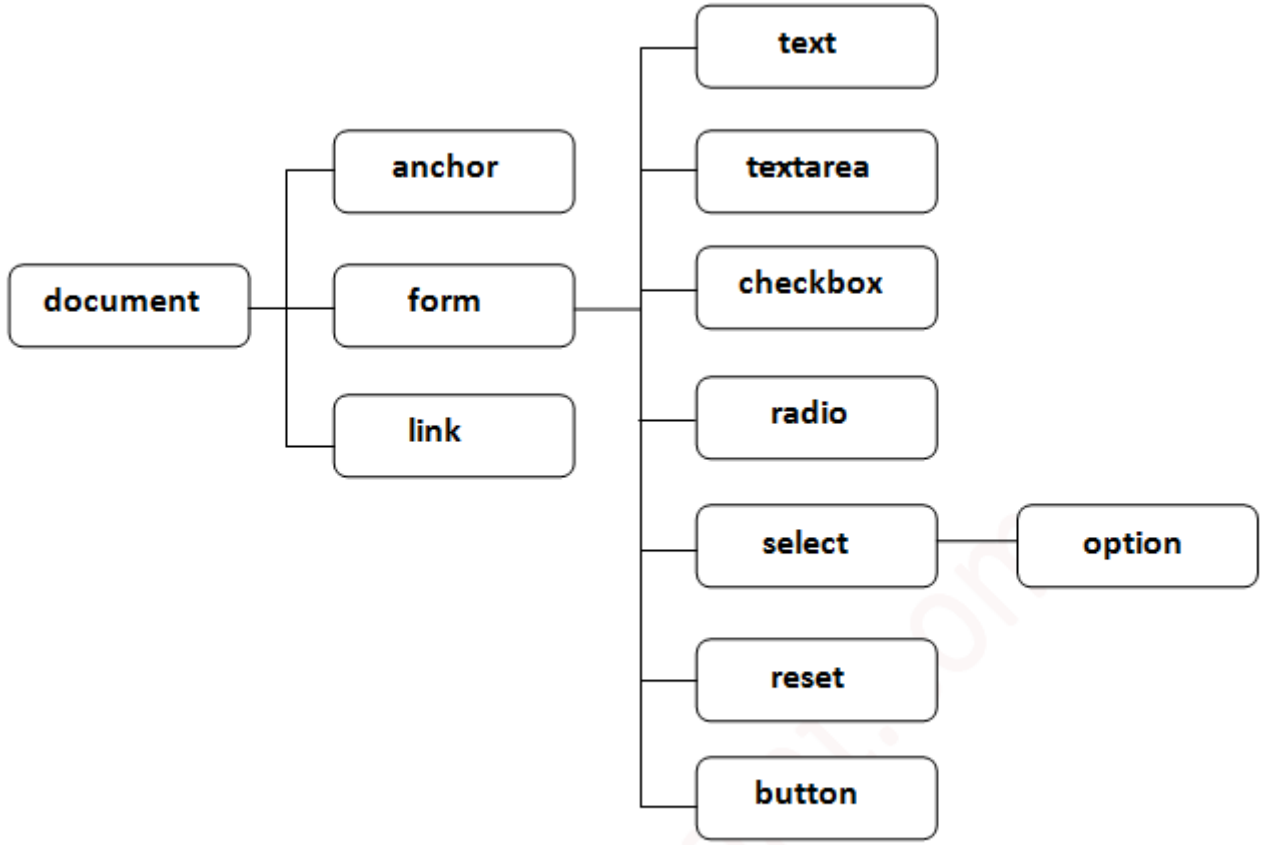
Is same as

document

According to W3C - *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*

Properties of document object

Let's see the properties of document object that can be accessed and modified by the document object.



Methods of document object

We can access and change the contents of document by its methods.

The important methods of document object are as follows:

Method	Description
write("string")	writes the given string on the doucment.
writeln("string")	writes the given string on the doucment with newline character at the end.
getElementById()	returns the element having the given id value.

getElementsByName()	returns all the elements having the given name value.
getElementsByTagName()	returns all the elements having the given tag name.
getElementsByClassName()	returns all the elements having the given class name.

Accessing field value by document object

In this example, we are going to get the value of input text by user. Here, we are using **document.form1.name.value** to get the value of name field.

Here, **document** is the root element that represents the html document.

form1 is the name of the form.

name is the attribute name of the input text.

value is the property, that returns the value of the input text.

Let's see the simple example of document object that prints name with welcome message.

```
<script type="text/javascript">
function printvalue(){
var name=document.form1.name.value;
alert("Welcome: "+name);
}
</script>

<form name="form1">
Enter Name:<input type="text" name="name"/>
<input type="button" onclick="printvalue()" value="print name"/>
</form>
```

Javascript - document.getElementById() method

The **document.getElementById()** method returns the element of specified id.

In the previous page, we have used **document.form1.name.value** to get the value of the input value. Instead of this, we can use document.getElementById() method to get value of the input text. But we need to define id for the input field.

Let's see the simple example of document.getElementById() method that prints cube of the given number.

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

```
function getcube(){
var number=document.getElementById("number").value;
alert(number*number*number);
}
</script>
<form>
Enter No:<input type="text" id="number" name="number"/><br/>
<input type="button" value="cube" onclick="getcube()"/>
</form>
```

Javascript - document.getElementsByName() method

The **document.getElementsByName()** method returns all the element of specified name.

The syntax of the getElementsByName() method is given below:

1. document.getElementsByName("name")

Here, name is required.

Example of document.getElementsByName() method

In this example, we going to count total number of genders. Here, we are using getElementsByName() method to get all the genders.

```
<script type="text/javascript">
function totalelements()
{
var allgenders=document.getElementsByName("gender");
alert("Total Genders:"+allgenders.length);
}
</script>
<form>
Male:<input type="radio" name="gender" value="male">
Female:<input type="radio" name="gender" value="female">

<input type="button" onclick="totalelements()" value="Total Genders">
</form>
```

Javascript - document.getElementsByTagName() method

The **document.getElementsByTagName()** method returns all the element of specified tag name.

The syntax of the getElementsByTagName() method is given below:

1. document.getElementsByTagName("name")

Here, name is required.

Example of document.getElementsByTagName() method

In this example, we going to count total number of paragraphs used in the document. To do this, we have called the document.getElementsByTagName("p") method that returns the total paragraphs.

```
<script type="text/javascript">
function countpara(){
var totalpara=document.getElementsByTagName("p");
alert("total p tags are: "+totalpara.length);

}
</script>
<p>This is a pragraph</p>
<p>Here we are going to count total number of paragraphs by getElementByTa
gName() method.</p>
<p>Let's see the simple example</p>
<button onclick="countpara()">count paragraph</button>
```

Javascript - innerHTML

The **innerHTML** property can be used to write the dynamic html on the html document.

It is used mostly in the web pages to generate the dynamic html such as registration form, comment form, links etc.

Example of innerHTML property

In this example, we are going to create the html form when user clicks on the button.

In this example, we are dynamically writing the html form inside the div name having the id mylocation. We are identifing this position by calling the document.getElementById() method.

```
<script type="text/javascript" >
function showcommentform()
{
      var data="Name:<input type='text' name='name'><br>Comment:<br><texta
      rea rows='5' cols='80'></textarea>
      <br><input type='submit' value='Post Comment'>";
      document.getElementById('mylocation').innerHTML=data;
}
```

```
</script>

<form name="myForm">
<input type="button" value="comment" onclick="showcommentform()">
<div id="mylocation"> </div>
</form>
```

Javascript - innerText

The **innerText** property can be used to write the dynamic text on the html document. Here, text will not be interpreted as html text but a normal text. It is used mostly in the web pages to generate the dynamic content such as writing the validation message, password strength etc.

Javascript innerText Example

In this example, we are going to display the password strength when releases the key after press.

```
<script type="text/javascript" >
function validate() {
var msg;
if(document.myForm.userPass.value.length>5){
msg="good";
}
else{
msg="poor";
}
document.getElementById('mylocation').innerText=msg;
 }

</script>
<form name="myForm">
<input type="password" value="" name="userPass" onkeyup="validate()">
Strength:<span id="mylocation">no strength</span>
</form>
```

JavaScript Form Validation

It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.

JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation. Most of the web developers prefer JavaScript form validation.

Through JavaScript, we can validate name, password, email, date, mobile numbers and more fields.

JavaScript Form Validation Example

In this example, we are going to validate the name and password. The name can’t be empty and password can’t be less than 6 characters long.

Here, we are validating the form on form submit. The user will not be forwarded to the next page until given values are correct.

```
<script>
function validateform(){
var name=document.myform.txtName.value;
var password=document.myform.pwd.value;

if (name==null || name==""){
  alert("Name can't be blank");
  return false;
}else if(password.length<6){
  alert("Password must be at least 6 characters long.");
  return false;
  }
}
</script>
<body>
<form name="myform" method="post" action="abc.jsp" onsubmit="return valida
teform()" >
Name: <input type="text" name="txtName"><br/>
Password: <input type="password" name="pwd"><br/>
<input type="submit" value="register">
</form>
```

JavaScript Retype Password Validation

```
<script type="text/javascript">
function matchpass(){
var firstpassword=document.f1.password.value;
var secondpassword=document.f1.password2.value;

if(firstpassword==secondpassword){
return true;
}
else{
alert("password must be same!");
return false;
}
}
</script>
```

```
<form name="f1" action="register.jsp" onsubmit="return matchpass()">
Password:<input type="password" name="password" /><br/>
Re-enter Password:<input type="password" name="password2"/><br/>
<input type="submit">
</form>
```

JavaScript Number Validation

Let's validate the textfield for numeric value only. Here, we are using isNaN() function.

```
<script>
function validate(){
var num=document.myform.num.value;
if (isNaN(num)){
  document.getElementById("numloc").innerHTML="Enter Numeric value only";

  return false;
}else{
  return true;
  }
}
</script>
<form name="myform" onsubmit="return validate()" >
Number: <input type="text" name="num"><span id="numloc"></span><br/>
<input type="submit" value="submit">
</form>
```

JavaScript validation with image

Let’s see an interactive JavaScript form validation example that displays correct and incorrect image if input is correct or incorrect.

```
<script>
function validate(){
var name=document.f1.name.value;
var password=document.f1.password.value;
var status=false;

if(name.length<1){
document.getElementById("nameloc").innerHTML=
" <img src='unchecked.gif'/> Please enter your name";
```

```
status=false;
}else{
document.getElementById("nameloc").innerHTML=" <img src='checked.gif'/>";

status=true;
}
if(password.length<6){
document.getElementById("passwordloc").innerHTML=
" <img src='unchecked.gif'/> Password must be at least 6 char long";
status=false;
}else{
document.getElementById("passwordloc").innerHTML=" <img src='checked.gif'/
>";
}
return status;
}
</script>

<form name="f1" action="#" onsubmit="return validate()">
<table>
<tr><td>Enter Name:</td><td><input type="text" name="name"/>
<span id="nameloc"></span></td></tr>
<tr><td>Enter Password:</td><td><input type="password" name="password"/>
<span id="passwordloc"></span></td></tr>
<tr><td colspan="2"><input type="submit" value="register"/></td></tr>
</table>
</form>
```

Output:

Enter Name:

Enter Password:

register