Introduction to JavaScript

JavaScript is a lightweight, cross-platform, and interpreted compiled programming language which is also known as the scripting language for webpages. It is well-known for the development of web pages, many non-browser environments also use it. JavaScript can be used for **Client-side** developments as well as **Server-side** developments. Javascript is both imperative and declarative type of language. JavaScript contains a standard library of objects, like **Array**, **Date**, and **Math**, and a core set of language elements like **operators**, **control structures**, and **statements**.

- **Client-side:** It supplies objects to control a browser and its Document Object Model (DOM). Like if client-side extensions allow an application to place elements on an HTML form and respond to user events such as **mouse clicks**, **form input**, and **page navigation**. Useful libraries for the client-side are **AngularJS**, **ReactJS**, **VueJS** and so many others.
- **Server-side:** It supplies objects relevant to running JavaScript on a server. Like if the server-side extensions allow an application to communicate with a database, and provide continuity of information from one invocation to another of the application, or perform file manipulations on a server. The useful framework which is the most famous these days is **node.js**.
- **Imperative language –** In this type of language we are mostly concern about how it is to be done . It simply control the flow of computation . The procedural programming approach , object, oriented approach comes under this like async await we are thinking what it is to be done further after async call.
- **Declarative programming –** In this type of language we are concern about how it is to be done , basically here logical computation require . Here main goal is to describe the desired result without direct dictation on how to get it like arrow function do .

JavaScript can be added to your HTML file in two ways:

- **Internal JS:** We can add JavaScript directly to our HTML file by writing the code inside the <script> tag. The <script> tag can either be placed inside the <head> or the <body> tag according to the requirement.
- **External JS:** We can write JavaScript code in other file having an extension.js and then link this file inside the <head> tag of the HTML file in which we want to add this code.

Syntax:

```
<script>
  // JavaScript Code
</script>
```

Example:

```
<!DOCTYPE html>
<html lang="en">

<head>
   <title>
      Basic Example to Describe JavaScript
   </title>
</head>
```

```
<body>

   <!-- JavaScript code can be embedded inside
      head section or body section -->
   <script>
      console.log("Welcome to GeeksforGeeks");
   </script>
</body>

</html>
```

Output: The output will display on the console.

```
Welcome to GeeksforGeeks
```

History of JavaScript: It was created in 1995 by Brendan Eich while he was an engineer at Netscape. It was originally going to be named LiveScript but was renamed. Unlike most programming languages, the JavaScript language has no concept of input or output. It is designed to run as a scripting language in a host environment, and it is up to the host environment to provide mechanisms for communicating with the outside world. The most common host environment is the browser.

Features of JavaScript: According to a recent survey conducted by **Stack Overflow**, JavaScript is the most popular language on earth.
With advances in browser technology and JavaScript having moved into the server with Node.js and other frameworks, JavaScript is capable of so much more. Here are a few things that we can do with JavaScript:

- JavaScript was created in the first place for DOM manipulation. Earlier websites were mostly static, after JS was created dynamic Web sites were made.
- Functions in JS are objects. They may have properties and methods just like another object. They can be passed as arguments in other functions.
- Can handle date and time.
- Performs Form Validation although the forms are created using HTML.
- No compiler is needed.

Applications of JavaScript:

- **Web Development:** Adding interactivity and behavior to static sites JavaScript was invented to do this in 1995. By using AngularJS that can be achieved so easily.
- **Web Applications:** With technology, browsers have improved to the extent that a language was required to create robust web applications. When we explore a map in Google Maps then we only need to click and drag the mouse. All detailed view is just a click away, and this is possible only because of JavaScript. It uses Application Programming Interfaces(APIs) that provide extra power to the code. The Electron and React is helpful in this department.
- **Server Applications:** With the help of Node.js, JavaScript made its way from client to server and node.js is the most powerful on the server-side.
- **Games:** Not only in websites, but JavaScript also helps in creating games for leisure. The combination of JavaScript and HTML 5 makes JavaScript popular in game development as well. It provides the EaseJS library which provides solutions for working with rich graphics.
- **Smartwatches:** JavaScript is being used in all possible devices and applications. It provides a library PebbleJS which is used in smartwatch applications. This framework works for applications that require the internet for its functioning.

- **Art:** Artists and designers can create whatever they want using JavaScript to draw on HTML 5 canvas, and make the sound more effective also can be used **p5.js** library.
- **Machine Learning:** This JavaScript ml5.js library can be used in web development by using machine learning.
- **Mobile Applications:** JavaScript can also be used to build an application for non-web contexts. The features and uses of JavaScript make it a powerful tool for creating mobile applications. This is a Framework for building web and mobile apps using JavaScript. Using React Native, we can build mobile applications for different operating systems. We do not require to write code for different systems. Write once use it anywhere!

Limitations of JavaScript:

- **Security risks:** JavaScript can be used to fetch data using AJAX or by manipulating tags that load data such as <img>, <object>, <script>. These attacks are called cross site script attacks. They inject JS that is not the part of the site into the visitor's browser thus fetching the details.
- **Performance:** JavaScript does not provide the same level of performance as offered by many traditional languages as a complex program written in JavaScript would be comparatively slow. But as JavaScript is used to perform simple tasks in a browser, so performance is not considered a big restriction in its use.
- **Complexity:** To master a scripting language, programmers must have a thorough knowledge of all the programming concepts, core language objects, client and server-side objects otherwise it would be difficult for them to write advanced scripts using JavaScript.
- **Weak error handling and type checking facilities:** It is weakly typed language as there is no need to specify the data type of the variable. So wrong type checking is not performed by compile.

Why JavaScript is known as a lightweight programming language?

JavaScript is considered as lightweight due to the fact that it has low CPU usage, is easy to implement and has a minimalist syntax. Minimalist syntax as in, it has no data types. Everything is treated here as an object. It is very easy to learn because of its syntax similar to C++ and Java.

A lightweight language does not consume much of your CPU's resources. It doesn't put excess strain on your CPU or RAM. JavaScript runs in the browser even though it has complex paradigms and logic which means it uses fewer resources than other languages. For example, NodeJs, a variation of JavaScript not only performs faster computations but also uses less resources than its counterparts such as Dart or Java.

Additionally, when compared with other programming languages, it has less in-built libraries or frameworks, contributing as another reason for it to be lightweight. However, this brings it a drawback that we need to incorporate external libraries and frameworks.

Is JavaScript compiled or interpreted or both?

JavaScript is both compiled and interpreted. In the earlier versions of JavaScript, it used only the interpreter that executed code line by line and shows the result immediately. But with time the performance became an issue as interpretation is quite slow. Therefore, in the newer versions of JS, probably after the V8, JIT compiler was also incorporated to optimize the execution and display the result more quickly. This JIT Just in time compiler generates a bytecode that is relatively easier to code. This bytecode is a set of highly optimized instructions.

JavaScript Example

Javascript example is easy to code. JavaScript provides 3 places to put the JavaScript code: within body tag, within head tag and external JavaScript file.

Let’s create the first JavaScript example.

1. **<script** type="text/javascript">
2. document.write("JavaScript is a simple language for javatpoint learners");
3. **</script>**

The **script** tag specifies that we are using JavaScript.

The **text/javascript** is the content type that provides information to the browser about the data.

The **document.write()** function is used to display dynamic content through JavaScript. We will learn about document object in detail later.

3 Places to put JavaScript code

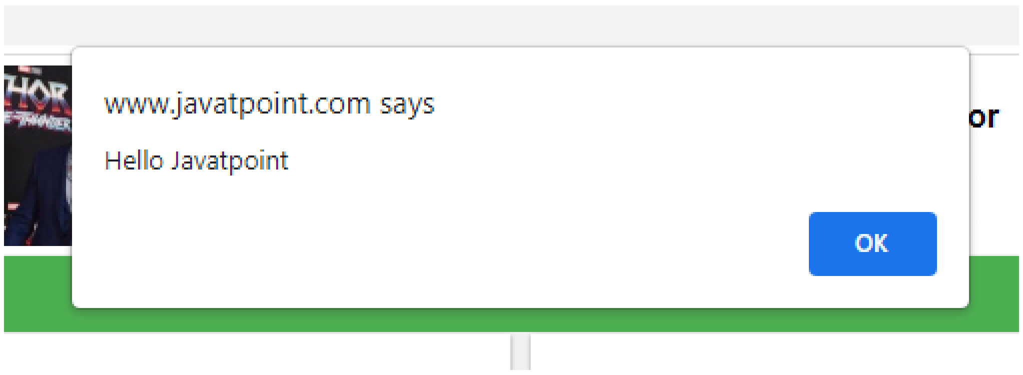
1. Between the body tag of html
2. Between the head tag of html
3. In .js file (external javaScript)

1) 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.

```
<html>

<body>

<script type="text/javascript">

 alert("Hello Javatpoint");

</script>

</body>
```

```
</html>
```


www.javatpoint.com says

Hello Javatpoint

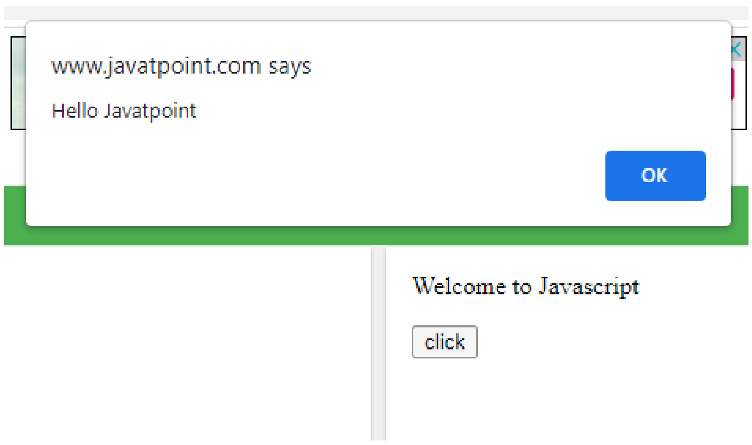
2) JavaScript Example : code between the head tag

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>
```


www.javatpoint.com says

Hello Javatpoint

Welcome to Javascript

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.

```
<html>

<body>

<script>

// It is single line comment

document.write("hello javascript");

</script>

</body>

</html>
```

Let’s see the example of single-line comment i.e. added after the statement.

```
<html>

<body>

<script>

var a=10;

var b=20;

var c=a+b;//It adds values of a and b variable
```

```
document.write(c);//prints sum of 10 and 20
</script>
</body>
</html>
```

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.

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

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.

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

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. For example:

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

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

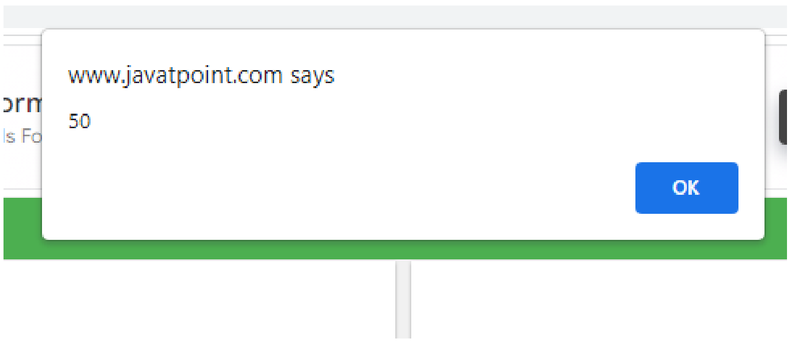
JavaScript Global Variable

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

Let’s see the simple example of global variable in JavaScript.

```
<html>
<body>
<script>
	var value=50;//global variable
	function a()
	{
	alert(value);
	}
	function b()
	{
		alert(value);
	}

a();
</script>
</body>
```

```
</html>
```



www.javatpoint.com says

50

Declaring JavaScript global variable within function

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

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

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

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

JavaScript Operators

JavaScript operators are symbols that are used to perform operations on operands. For example:

var sum=10+20;

Here, + is the arithmetic operator and = is the assignment operator.

There are following types of operators in JavaScript.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

avaScript Comparison Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example

==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

JavaScript Bitwise Operators

The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

Operator	**Description**	**Example**
&	Bitwise AND	(10==20 & 20==33) = false
|	Bitwise OR	(10==20 | 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

JavaScript Assignment Operators

The following operators are known as JavaScript assignment operators.

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

JavaScript Special Operators

The following operators are known as JavaScript special operators.

Operator	Description

(?:)	Conditional Operator returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
delete	Delete Operator deletes a property from the object.
in	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type
new	creates an instance (object)
typeof	checks the type of object.
void	it discards the expression's return value.
yield	checks what is returned in a generator by the generator's iterator.

JavaScript If-else

The **JavaScript if-else statement** is used *to execute the code whether condition is true or false*. There are three forms of if statement in JavaScript.

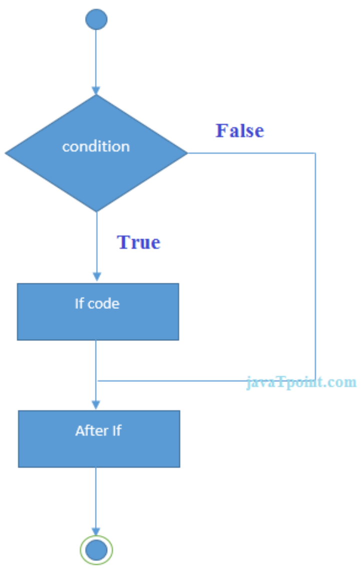
1. If Statement
2. If else statement
3. if else if statement

JavaScript If statement

It evaluates the content only if expression is true. The signature of JavaScript if statement is given below.

```
if(expression){
//content to be evaluated
}
```

Flowchart of JavaScript If statement



Let’s see the simple example of if statement in javascript.

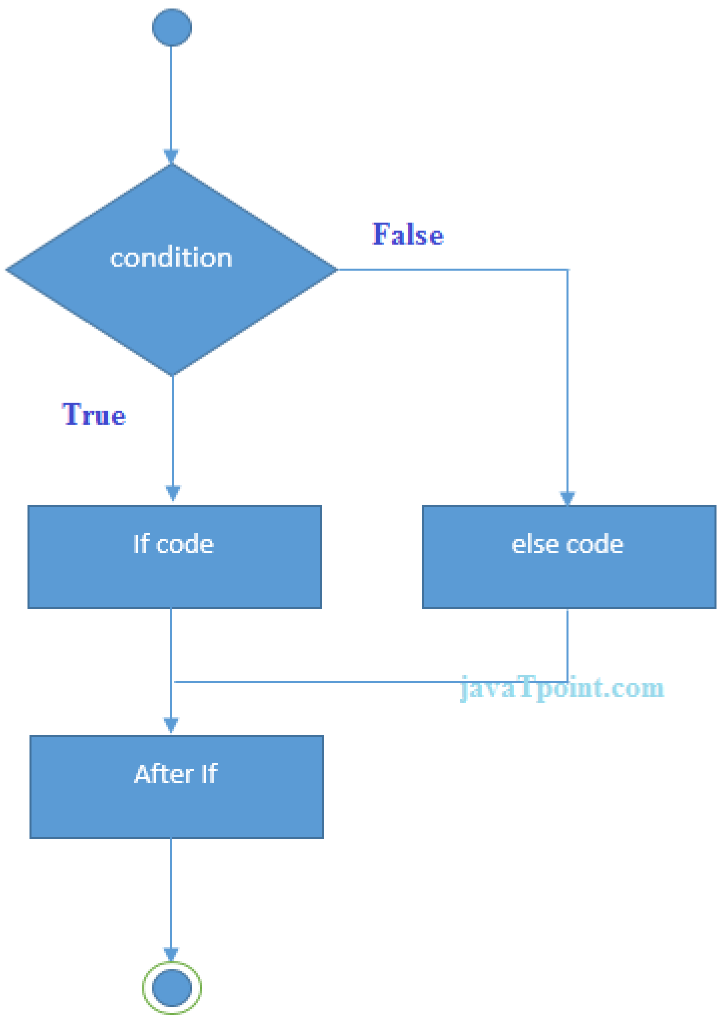
```
<html>
<body>
<script>
var a=20;
if(a>10){
document.write("value of a is greater than 10");
}
</script>
</body>
</html>
```

JavaScript If...else Statement

It evaluates the content whether condition is true of false. The syntax of JavaScript if-else statement is given below

```
if(expression){
//content to be evaluated if condition is true
}
else{
//content to be evaluated if condition is false
}
```

Flowchart of JavaScript If...else statement



Let's see the example of if-else statement in JavaScript to find out the even or odd number.

```
<html>
<body>
<script>
var a=20;
if(a%2==0){
document.write("a is even number");
}
else{
document.write("a is odd number");
}
</script>
</body>
```

```
</html>
```

JavaScript If...else if statement

It evaluates the content only if expression is true from several expressions. The signature of JavaScript if else if statement is given below.

```
if(expression1){
//content to be evaluated if expression1 is true
}
else if(expression2){
//content to be evaluated if expression2 is true
}
else if(expression3){
//content to be evaluated if expression3 is true
}
else{
//content to be evaluated if no expression is true
}
<html>
<body>
<script>
var a=20;
if(a==10){
document.write("a is equal to 10");
}
else if(a==15){
document.write("a is equal to 15");
}
else if(a==20){
document.write("a is equal to 20");
}
else{
document.write("a is not equal to 10, 15 or 20");
}
</script>
</body>
</html>
```

JavaScript Switch

The **JavaScript switch statement** is used *to execute one code from multiple expressions*. It is just like else if statement that we have learned in previous page. But it is convenient than *if..else..if* because it can be used with numbers, characters etc.

The signature of JavaScript switch statement is given below.

```
switch(expression){
case value1:
 code to be executed;
 break;
case value2:
 code to be executed;
 break;
......

default:
 code to be executed if above values are not matched;
}
```

Let’s see the simple example of switch statement in javascript.

```
<!DOCTYPE html>
<html>
<body>
<script>
var grade='B';
var result;
switch(grade){
case 'A':
result="A Grade";
break;
case 'B':
result="B Grade";
break;
case 'C':
result="C Grade";
break;
default:
result="No Grade";
}
```

```
document.write(result);
</script>
</body>
</html>
```

JavaScript Loops

The **JavaScript loops** are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. for loop
2. while loop
3. do-while loop
4. for-in loop

---

1) JavaScript For loop

The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

```
for (initialization; condition; increment)
{
    code to be executed
}
```

Let’s see the simple example of for loop in javascript.

```
<!DOCTYPE html>
<html>
<body>
<script>
for (i=1; i<=5; i++)
{
document.write(i + "<br/>")
}
</script>
</body>
</html>
```

2) JavaScript while loop

The **JavaScript while loop** *iterates the elements for the infinite number of times*. It should be used if number of iterations is not known. The syntax of while loop is given below.

```
while (condition)
{
    code to be executed
}
```

```
<!DOCTYPE html>
<html>
<body>
<script>
var i=11;
while (i<=15)
{
document.write(i + "<br/>");
i++;
}

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

3) JavaScript do while loop

The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least* once whether condition is true or false. The syntax of do while loop is given below.

```
do{
    code to be executed
}while (condition);

```

```
<!DOCTYPE html>
<html>
<body>
<script>
var i=21;
do{
document.write(i + "<br/>");
i++;
}while (i<=25);
```

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

JavaScript Functions

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

1. **Code reusability**: We can call a function several times so it save coding.
2. **Less coding**: It makes our program compact. We don’t need to write many lines of code each time to perform a common task.

JavaScript Function Syntax

The syntax of declaring function is given below.

```
1. function functionName([arg1, arg2, ...argN]){
2.  //code to be executed
3. }
```

```
<html>
<body>
<script>
function msg(){
alert("hello! this is message");
}
</script>
<input type="button" onclick="msg()" value="call function"/>
</body>
</html>
```

JavaScript Function Arguments

We can call function by passing arguments. Let’s see the example of function that has one argument.

```
<html>
<body>
<script>
function getcube(number){
```

```
alert(number*number*number);
}
</script>
<form>
<input type="button" value="click" onclick="getcube(4)"/>
</form>
</body>
</html>
```

Function with Return Value

We can call function that returns a value and use it in our program. Let’s see the example of function that returns value.

```
<html>
<body>
<script>
function getInfo(){
return "hello javatpoint! How r u?";
}
</script>
<script>
document.write(getInfo());
</script>
</body>
</html>
```

JavaScript Function Object

In JavaScript, the purpose of **Function constructor** is to create a new Function object. It executes the code globally. However, if we call the constructor directly, a function is created dynamically but in an unsecured way.

Syntax

1. new Function ([arg1[, arg2[, ...argn]],] functionBody)

Parameter

arg1, arg2, ... , argn - It represents the argument used by function.

functionBody - It represents the function definition.

JavaScript Function Methods

Let's see function methods with description.

```
<!DOCTYPE html>
<html>
<body>

<script>
var add=new Function("num1","num2","return num1+num2");
document.writeln(add(2,5));
</script>

</body>
</html>
```

JavaScript Objects

A javaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

Creating Objects in JavaScript

There are 3 ways to create objects.

1. By object literal
2. By creating instance of Object directly (using new keyword)
3. By using an object constructor (using new keyword)

1) JavaScript Object by object literal

The syntax of creating object using object literal is given below:

1. object={property1:value1,property2:value2.....propertyN:valueN}

As you can see, property and value is separated by : (colon).

Let’s see the simple example of creating object in JavaScript.

```
<html>

<body>

<script>

emp={id:102,name:"Shyam Kumar",salary:40000}

document.write(emp.id+" "+emp.name+" "+emp.salary);

</script>

</body>

</html>
```

2) By creating instance of Object

The syntax of creating object directly is given below:

1. var objectname=new Object();

Here, **new keyword** is used to create object.

Let’s see the example of creating object directly.

```
<html>
```

```
<body>

<script>

var emp=new Object();

emp.id=101;

emp.name="Ravi Malik";

emp.salary=50000;

document.write(emp.id+" "+emp.name+" "+emp.salary);

</script>

</body>

</html>
```

3) By using an Object constructor

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

The **this keyword** refers to the current object.

The example of creating object by object constructor is given below.

```
<html>

<body>

<script>

function emp(id,name,salary){

this.id=id;

this.name=name;

this.salary=salary;

}

e=new emp(103,"Vimal Jaiswal",30000);
```

```
document.write(e.id+" "+e.name+" "+e.salary);

</script>

</body>

</html>
```

Defining method in JavaScript Object

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.

The example of defining method in object is given below.

```
<html>

<body>

<script>

function emp(id,name,salary){

this.id=id;

this.name=name;

this.salary=salary;


this.changeSalary=changeSalary;

function changeSalary(otherSalary){

this.salary=otherSalary;

}

}

e=new emp(103,"Sonoo Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);

e.changeSalary(45000);
```

```
document.write("<br>"+e.id+" "+e.name+" "+e.salary);

</script>

</body>

</html>
```

JavaScript Object Methods

The various methods of Object are as follows:

S.No	Methods	Description
1	Object.assign()	This method is used to copy enumerable and own properties from a source object to a target object
2	Object.create()	This method is used to create a new object with the specified prototype object and properties.
3	Object.defineProperty()	This method is used to describe some behavioral attributes of the property.
4	Object.defineProperties()	This method is used to create or configure multiple object properties.
5	Object.entries()	This method returns an array with arrays of the key, value pairs.
6	Object.freeze()	This method prevents existing properties from being removed.
7	Object.getOwnPropertyDescriptor()	This method returns a property descriptor for the specified property of the specified object.
8	Object.getOwnPropertyDescriptors()	This method returns all own property descriptors of a given object.

9	Object.getOwnPropertyNames()	This method returns an array of all properties (enumerable or not) found.
10	Object.getOwnPropertySymbols()	This method returns an array of all own symbol key properties.
11	Object.getPrototypeOf()	This method returns the prototype of the specified object.
12	Object.is()	This method determines whether two values are the same value.
13	Object.isExtensible()	This method determines if an object is extensible
14	Object.isFrozen()	This method determines if an object was frozen.
15	Object.isSealed()	This method determines if an object is sealed.
16	Object.keys()	This method returns an array of a given object's own property names.
17	Object.preventExtensions()	This method is used to prevent any extensions of an object.
18	Object.seal()	This method prevents new properties from being added and marks all existing properties as non-configurable.
19	Object.setPrototypeOf()	This method sets the prototype of a specified object to another object.
20	Object.values()	This method returns an array of values.

JavaScript Array

JavaScript array is an object that represents a collection of similar type of elements.

There are 3 ways to construct array in JavaScript

1. By array literal
2. By creating instance of Array directly (using new keyword)
3. By using an Array constructor (using new keyword)

1) JavaScript array literal

The syntax of creating array using array literal is given below:

1. var arrayname=[value1,value2.....valueN];

As you can see, values are contained inside [ ] and separated by , (comma).

Let's see the simple example of creating and using array in JavaScript.

```
<html>
<body>
<script>
var emp=["Sonoo","Vimal","Ratan"];
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br/>");
}
</script>
</body>
</html>
```

The .length property returns the length of an array.

2) JavaScript Array directly (new keyword)

The syntax of creating array directly is given below:

1. var arrayname=new Array();

Here, **new keyword** is used to create instance of array.

Let's see the example of creating array directly.

```
<html>
<body>
<script>
var i;
var emp = new Array();
emp[0] = "Arun";
emp[1] = "Varun";
emp[2] = "John";

for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
</body>
</html>
```

3) JavaScript array constructor (new keyword)

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

The example of creating object by array constructor is given below.

```
<html>

<body>

<script>

var emp=new Array("Jai","Vijay","Smith");

for (i=0;i<emp.length;i++){

document.write(emp[i] + "<br>");

}

</script>

</body>

</html>
```

JavaScript Array Methods

Let's see the list of JavaScript array methods with their description.

Methods	Description
concat()	It returns a new array object that contains two or more merged arrays.
copywithin()	It copies the part of the given array with its own elements and returns the modified array.
entries()	It creates an iterator object and a loop that iterates over each key/value pair.
every()	It determines whether all the elements of an array are satisfying the provided function conditions.
flat()	It creates a new array carrying sub-array elements concatenated recursively till the specified depth.
flatMap()	It maps all array elements via mapping function, then flattens the result into a new array.
fill()	It fills elements into an array with static values.
from()	It creates a new array carrying the exact copy of another array element.
filter()	It returns the new array containing the elements that pass the provided function conditions.
find()	It returns the value of the first element in the given array that satisfies the specified condition.
findIndex()	It returns the index value of the first element in the given array that satisfies the specified condition.
forEach()	It invokes the provided function once for each element of an array.
includes()	It checks whether the given array contains the specified element.

indexOf()	It searches the specified element in the given array and returns the index of the first match.
isArray()	It tests if the passed value ia an array.
join()	It joins the elements of an array as a string.
keys()	It creates an iterator object that contains only the keys of the array, then loops through these keys.
lastIndexOf()	It searches the specified element in the given array and returns the index of the last match.
map()	It calls the specified function for every array element and returns the new array
of()	It creates a new array from a variable number of arguments, holding any type of argument.
pop()	It removes and returns the last element of an array.
push()	It adds one or more elements to the end of an array.
reverse()	It reverses the elements of given array.
reduce(function, initial)	It executes a provided function for each value from left to right and reduces the array to a single value.
reduceRight()	It executes a provided function for each value from right to left and reduces the array to a single value.
some()	It determines if any element of the array passes the test of the implemented function.
shift()	It removes and returns the first element of an array.
slice()	It returns a new array containing the copy of the part of the given array.

sort()	It returns the element of the given array in a sorted order.
splice()	It add/remove elements to/from the given array.
toLocaleString()	It returns a string containing all the elements of a specified array.
toString()	It converts the elements of a specified array into string form, without affecting the original array.
unshift()	It adds one or more elements in the beginning of the given array.
values()	It creates a new iterator object carrying values for each index in the array.

JavaScript String

The **JavaScript string** is an object that represents a sequence of characters.

There are 2 ways to create string in JavaScript

1. By string literal
2. By string object (using new keyword)

---

1) By string literal

The string literal is created using double quotes. The syntax of creating string using string literal is given below:

1. var stringname="string value";

Let's see the simple example of creating string literal.

```
<!DOCTYPE html>

<html>

<body>

<script>

var str="This is string literal";
```

```
document.write(str);
</script>
</body>
</html>
```

2) By string object (using new keyword)

The syntax of creating string object using new keyword is given below:

1. var stringname=new String("string literal");

Here, **new keyword** is used to create instance of string.

Let's see the example of creating string in JavaScript by new keyword.

```
<!DOCTYPE html>
<html>
<body>
<script>
var stringname=new String("hello javascript string");
document.write(stringname);
</script>
</body>
</html>
```

JavaScript String Methods

Let's see the list of JavaScript string methods with examples.

Methods	**Description**
charAt()	It provides the char value present at the specified index.

charCodeAt()	It provides the Unicode value of a character present at the specified index.
concat()	It provides a combination of two or more strings.
indexOf()	It provides the position of a char value present in the given string.
lastIndexOf()	It provides the position of a char value present in the given string by searching a character from the last position.
search()	It searches a specified regular expression in a given string and returns its position if a match occurs.
match()	It searches a specified regular expression in a given string and returns that regular expression if a match occurs.
replace()	It replaces a given string with the specified replacement.
substr()	It is used to fetch the part of the given string on the basis of the specified starting position and length.
substring()	It is used to fetch the part of the given string on the basis of the specified index.
slice()	It is used to fetch the part of the given string. It allows us to assign positive as well negative index.
toLowerCase()	It converts the given string into lowercase letter.
toLocaleLowerCase()	It converts the given string into lowercase letter on the basis of host?s current locale.
toUpperCase()	It converts the given string into uppercase letter.
toLocaleUpperCase()	It converts the given string into uppercase letter on the basis of host?s current locale.
toString()	It provides a string representing the particular object.

valueOf()	It provides the primitive value of string object.
split()	It splits a string into substring array, then returns that newly created array.
trim()	It trims the white space from the left and right side of the string.

1) JavaScript String charAt(index) Method

The JavaScript String charAt() method returns the character at the given index.

```
<!DOCTYPE html>
<html>
<body>
<script>
var str="javascript";
document.write(str.charAt(2));
</script>
</body>
</html>
```

2) JavaScript String concat(str) Method

The JavaScript String concat(str) method concatenates or joins two strings.

```
<!DOCTYPE html>
<html>
<body>
<script>
var s1="javascript ";
var s2="concat example";
var s3=s1+s2;
document.write(s3);
</script>
</body>
</html>
```

3) JavaScript String indexOf(str) Method

The JavaScript String indexOf(str) method returns the index position of the given string.

```
<!DOCTYPE html>
<html>
<body>
<script>
var s1="javascript from javatpoint indexof";
var n=s1.indexOf("from");
document.write(n);
</script>
</body>
</html>
```

4) JavaScript String lastIndexOf(str) Method

The JavaScript String lastIndexOf(str) method returns the last index position of the given string.

```
<!DOCTYPE html>
<html>
<body>
<script>
var s1="javascript from javatpoint indexof";
var n=s1.lastIndexOf("java");
document.write(n);
</script>
</body>
</html>
```

5) JavaScript String toLowerCase() Method

The JavaScript String toLowerCase() method returns the given string in lowercase letters.

```
<!DOCTYPE html>
<html>
<body>
<script>
```

```
var s1="JavaScript toLowerCase Example";
var s2=s1.toLowerCase();
document.write(s2);
</script>
</body>
</html>
```

6) JavaScript String toUpperCase() Method

The JavaScript String toUpperCase() method returns the given string in uppercase letters.

```
<!DOCTYPE html>
<html>
<body>
<script>
var s1="JavaScript toUpperCase Example";
var s2=s1.toUpperCase();
document.write(s2);
</script>
</body>
</html>
```

7) JavaScript String slice(beginIndex, endIndex) Method

The JavaScript String slice(beginIndex, endIndex) method returns the parts of string from given beginIndex to endIndex. In slice() method, beginIndex is inclusive and endIndex is exclusive.

```
<!DOCTYPE html>

<html>

<body>

<script>

var s1="abcdefgh";
var s2=s1.slice(2,5);

document.write(s2);

</script>

</body>

</html>
```

8) JavaScript String trim() Method

The JavaScript String trim() method removes leading and trailing whitespaces from the string.

```
<!DOCTYPE html>

<html>

<body>

<script>

var s1="     javascript trim     ";

var s2=s1.trim();

document.write(s2);

</script>

</body>

</html>
```

9) JavaScript String split() Method

```
<script>
var str="This is JavaTpoint website";
document.write(str.split(" ")); //splits the given string.
</script>
```

JavaScript Date Object

The **JavaScript date** object can be used to get year, month and day. You can display a timer on the webpage by the help of JavaScript date object.

You can use different Date constructors to create date object. It provides methods to get and set day, month, year, hour, minute and seconds.

Constructor

You can use 4 variant of Date constructor to create date object.

1. Date()
2. Date(milliseconds)
3. Date(dateString)
4. Date(year, month, day, hours, minutes, seconds, milliseconds)

JavaScript Date Methods

Let's see the list of JavaScript date methods with their description

Methods	**Description**
getDate()	It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of local time.
getDay()	It returns the integer value between 0 and 6 that represents the day of the week on the basis of local time.
getFullYears()	It returns the integer value that represents the year on the basis of local time.
getHours()	It returns the integer value between 0 and 23 that represents the hours on the basis of local time.
getMilliseconds()	It returns the integer value between 0 and 999 that represents the milliseconds on the basis of local time.
getMinutes()	It returns the integer value between 0 and 59 that represents the minutes on the basis of local time.
getMonth()	It returns the integer value between 0 and 11 that represents the month on the basis

	of local time.
getSeconds()	It returns the integer value between 0 and 60 that represents the seconds on the basis of local time.
getUTCDate()	It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of universal time.
getUTCDay()	It returns the integer value between 0 and 6 that represents the day of the week on the basis of universal time.
getUTCFullYears()	It returns the integer value that represents the year on the basis of universal time.
getUTCHours()	It returns the integer value between 0 and 23 that represents the hours on the basis of universal time.
getUTCMinutes()	It returns the integer value between 0 and 59 that represents the minutes on the basis of universal time.
getUTCMonth()	It returns the integer value between 0 and 11 that represents the month on the basis of universal time.
getUTCSeconds()	It returns the integer value between 0 and 60 that represents the seconds on the basis of universal time.
setDate()	It sets the day value for the specified date on the basis of local time.
setDay()	It sets the particular day of the week on the basis of local time.
setFullYears()	It sets the year value for the specified date on the basis of local time.
setHours()	It sets the hour value for the specified date on the basis of local time.
setMilliseconds()	It sets the millisecond value for the specified date on the basis of local time.
setMinutes()	It sets the minute value for the specified date on the basis of local time.

setMonth()	It sets the month value for the specified date on the basis of local time.
setSeconds()	It sets the second value for the specified date on the basis of local time.
setUTCDate()	It sets the day value for the specified date on the basis of universal time.
setUTCDay()	It sets the particular day of the week on the basis of universal time.
setUTCFullYears()	It sets the year value for the specified date on the basis of universal time.
setUTCHours()	It sets the hour value for the specified date on the basis of universal time.
setUTCMilliseconds()	It sets the millisecond value for the specified date on the basis of universal time.
setUTCMinutes()	It sets the minute value for the specified date on the basis of universal time.
setUTCMonth()	It sets the month value for the specified date on the basis of universal time.
setUTCSeconds()	It sets the second value for the specified date on the basis of universal time.
toDateString()	It returns the date portion of a Date object.
toISOString()	It returns the date in the form ISO format string.
toJSON()	It returns a string representing the Date object. It also serializes the Date object during JSON serialization.
toString()	It returns the date in the form of string.
toTimeString()	It returns the time portion of a Date object.
toUTCString()	It converts the specified date in the form of string using UTC time zone.
valueOf()	It returns the primitive value of a Date object.

JavaScript Date Example

Let's see the simple example to print date object. It prints date and time both.

```
<html>

<body>

Current Date and Time: <span id="txt"></span>

<script>

var today=new Date();

document.getElementById('txt').innerHTML=today;

</script>

</body>

</html>
```

Output:

```
Current Date and Time: Mon Dec 19 2022 11:06:05 GMT+0530 (India Standard Time)
```

Let's see another code to print date/month/year.

```
<script>
var date=new Date();
var day=date.getDate();
var month=date.getMonth()+1;
var year=date.getFullYear();
document.write("<br>Date is: "+day+"/"+month+"/"+year);
</script>
```

JavaScript Current Time Example

Let's see the simple example to print current time of system.

```
<html>

<body>

Current Time: <span id="txt"></span>
```

```
<script>

var today=new Date();

var h=today.getHours();

var m=today.getMinutes();

var s=today.getSeconds();

document.getElementById('txt').innerHTML=h+":"+m+":"+s;

</script>

</body>

</html>
```

Output:

```
Current Time: 11:6:5
```

JavaScript Digital Clock Example

Let's see the simple example to display digital clock using JavaScript date object.

There are two ways to set interval in JavaScript: by setTimeout() or setInterval() method.

```
<html>

<body>

Current Time: <span id="txt"></span>

<script>

window.onload=function(){getTime();}

function getTime(){

var today=new Date();

var h=today.getHours();

var m=today.getMinutes();

var s=today.getSeconds();

// add a zero in front of numbers<10
m=checkTime(m);

s=checkTime(s);

document.getElementById('txt').innerHTML=h+":"+m+":"+s;

setTimeout(function(){getTime()},1000);

}

//setInterval("getTime()",1000);//another way

function checkTime(i){

if (i<10){

  i="0" + i;

 }

return i;

}

</script>

</body>

</html>
```

Output:

```
Current Time: 11:10:17
```

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.

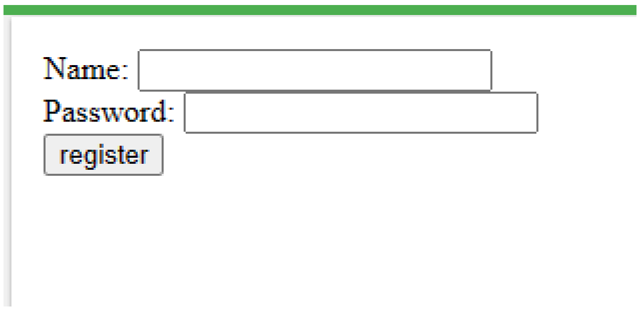
JavaScript Form Validation Example

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.

```
<html>

<body>

<script>

function validateform(){

var name=document.myform.name.value;

var password=document.myform.password.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="http://www.javatpoint.com/javascriptpages/valid.jsp" onsubmit="return
validateform()" >

Name: <input type="text" name="name"><br/>

Password: <input type="password" name="password"><br/>

<input type="submit" value="register">
```

```
</form>

</body>

</html>
```



JavaScript Retype Password Validation

```
<!DOCTYPE html>

<html>

<head>

<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>

</head>
```

```
<body>


<form name="f1" action="http://www.javatpoint.com/javascriptpages/valid.jsp"
onsubmit="return matchpass()">

Password:<input type="password" name="password" /><br/>

Re-enter Password:<input type="password" name="password2"/><br/>

<input type="submit">

</form>


</body>

</html>
```

JavaScript Number Validation

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

```
<!DOCTYPE html>

<html>

<head>

<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>

</head>


<body>

<form name="myform" action="http://www.javatpoint.com/javascriptpages/valid.jsp"
onsubmit="return validate()" >

Number: <input type="text" name="num"><span id="numloc"></span><br/>

<input type="submit" value="submit">

</form>


</body>

</html>
```

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.

```
<html>

<body>

<script type="text/javascript">

function validate(){

var name=document.f1.name.value;

var passwordlength=document.f1.password.value.length;

var status=false;

if(name==""){

document.getElementById("namelocation").innerHTML=

" <img src='http://www.javatpoint.com/javascriptpages/images/unchecked.gif'/> Please enter
your name";

status=false;
}else{

document.getElementById("namelocation").innerHTML=" <img
src='http://www.javatpoint.com/javascriptpages/images/checked.gif'/>";

status=true;

}



if(passwordlength<6){

document.getElementById("passwordlocation").innerHTML=

" <img src='http://www.javatpoint.com/javascriptpages/images/unchecked.gif'/> Password
must be greater than 6";

status=false;

}else{

document.getElementById("passwordlocation").innerHTML=" <img
src='http://www.javatpoint.com/javascriptpages/images/checked.gif'/>";

}



return status;

}

</script>

<form name="f1" action="http://www.javatpoint.com/javascriptpages/valid.jsp"
onsubmit="return validate()">

<table>

<tr><td>Name:</td><td><input type="text" name="name"/>

<span id="namelocation" style="color:red"></span></td></tr>

<tr><td>Password:</td><td><input type="password" name="password"/>

<span id="passwordlocation" style="color:red"></span></td></tr>

<tr><td colspan="2"><input type="submit" value="register"/>  </td></tr>

</table>
```

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

JavaScript email validation

We can validate the email by the help of JavaScript.

There are many criteria that need to be follow to validate the email id such as:

- email id must contain the @ and . character
- There must be at least one character before and after the @.
- There must be at least two characters after . (dot).

Let's see the simple example to validate the email field.

```
<script>
function validateemail()
{
var x=document.myform.email.value;
var atposition=x.indexOf("@");
var dotposition=x.lastIndexOf(".");
if (atposition<1 || dotposition<atposition+2 || dotposition+2>=x.length){
  alert("Please enter a valid e-mail address \n atpostion:"+atposition+"\n dotposition:"+dotposition);
  return false;
  }
}
</script>
<body>
<form name="myform"  method="post" action="#" onsubmit="return validateemail();">
```

```
Email: <input type="text" name="email"><br/>

<input type="submit" value="register">
</form>
```

JavaScript Classes

In JavaScript, classes are the special type of functions. We can define the class just like function declarations and function expressions.

The JavaScript class contains various class members within a body including methods or constructor. The class is executed in strict mode. So, the code containing the silent error or mistake throws an error.

The class syntax contains two components:

- Class declarations
- Class expressions

Class Declarations

A class can be defined by using a class declaration. A class keyword is used to declare a class with any particular name. According to JavaScript naming conventions, the name of the class always starts with an uppercase letter.

Class Declarations Example

Let's see a simple example of declaring the class.

```
<!DOCTYPE html>

<html>

<body>



<script>

//Declaring class

class Employee

  {
```

```
//Initializing an object
    constructor(id,name)
    {
      this.id=id;
      this.name=name;
    }
//Declaring method
    detail()
    {
  document.writeln(this.id+" "+this.name+"<br>")
    }
  }
//passing object to a variable
var e1=new Employee(101,"Martin Roy");
var e2=new Employee(102,"Duke William");
e1.detail(); //calling method
e2.detail();
</script>


</body>
</html>
```

Class Expression Example: Re-declaring Class

Unlike class declaration, the class expression allows us to re-declare the same class. So, if we try to declare the class more than one time, it throws an error.

```
<!DOCTYPE html>
```

```
<html>
<body>

<script>
//Declaring class
var emp=class
  {
//Initializing an object
    constructor(id,name)
    {
      this.id=id;
      this.name=name;
    }
//Declaring method
detail()
    {
  document.writeln(this.id+" "+this.name+"<br>")
    }
  }
//passing object to a variable
var e1=new emp(101,"Martin Roy");
var e2=new emp(102,"Duke William");
e1.detail(); //calling method
e2.detail();

//Re-declaring class
var emp=class
```

```
  {
//Initializing an object
    constructor(id,name)
    {
      this.id=id;
      this.name=name;
    }
    detail()
    {
  document.writeln(this.id+" "+this.name+"<br>")
    }
  }
//passing object to a variable
var e1=new emp(103,"James Bella");
var e2=new emp(104,"Nick Johnson");
e1.detail(); //calling method
e2.detail();
</script>

</body>
</html>
```