JavaScript Tutorial

Our **JavaScript Tutorial** is designed for beginners and professionals both. JavaScript is used to create client-side dynamic pages.

JavaScript is *an object-based scripting language* which is lightweight and cross-platform.

JavaScript is not a compiled language, but it is a translated language. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.

What is JavaScript

JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. 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 in the Netscape Navigator browser. Since then, it has been adopted by all other graphical web browsers. 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.

Although, JavaScript has no connectivity with Java programming language. The name was suggested and provided in the times when Java was gaining popularity in the market. In addition to web browsers, databases such as CouchDB and MongoDB uses JavaScript as their scripting and query language.

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.

History of JavaScript

In 1993, **Mosaic**, the first popular web browser, came into existence. In the **year 1994**, **Netscape** was founded by **Marc Andreessen**. He realized that the web needed to become more dynamic. Thus, a 'glue language' was believed to be provided to HTML to make web designing easy for designers and part-time programmers. Consequently, in 1995, the company recruited **Brendan Eich** intending to implement and embed Scheme programming language to the browser. But, before Brendan could start, the company merged with **Sun Microsystems** for adding Java into its Navigator so that it could compete with Microsoft over the web technologies and platforms. Now, two languages were there: Java and the scripting language. Further, Netscape decided to give a similar name to the scripting language as Java's. It led to 'Javascript'. Finally, in May 1995, Marc Andreessen coined the first code of Javascript named '**Mocha**'. Later, the marketing team replaced the name with '**LiveScript**'. But, due to trademark reasons and certain other reasons, in December 1995, the language was finally renamed to 'JavaScript'. From then, JavaScript came into existence.

Application of JavaScript

JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation,
- Dynamic drop-down menus,
- Displaying date and time,

- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- Displaying clocks etc.

JavaScript Example

```
<script>
document.write("Hello JavaScript by JavaScript");
</script>
```

JavaScript Example

1. JavaScript Example
2. Within body tag
3. Within head tag

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.

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

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

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>
```

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

```
1. function msg(){
2.  alert("Hello Javatpoint");
3. }
```

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

index.html

```
1. <html>
2. <head>
3. <script type="text/javascript" src="message.js"></script>
4. </head>
5. <body>
6. <p>Welcome to JavaScript</p>
7. <form>
8. <input type="button" value="click" onclick="msg()"/>
9. </form>
10. </body>
11. </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

1. JavaScript comments
2. Advantage of javaScript comments
3. Single-line and Multi-line comments

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>
```

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

```
<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>
```

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

1. JavaScript variable
2. JavaScript Local variable
3. JavaScript Global 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

```
1. var  123=30;
2. var *aa=320;
```

Example of JavaScript variable

Let's see a simple example of JavaScript variable.

```
1. <script>
2. var x = 10;
3. var y = 20;
4. var z=x+y;
5. document.write(z);
6. </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:

```
1. <script>
2. function abc(){
3. var x=10;//local variable
4. }
5. </script>
```

Or,

```
1. <script>
2. If(10<13){
3. var y=20;//JavaScript local variable
4. }
5. </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:

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

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.

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

Declaring JavaScript global variable within function

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

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

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

```
1. function m(){
2. window.value=100;//declaring global variable by window object
3. }
4. function n(){
5. alert(window.value);//accessing global variable from other function
6. }
```

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:

```
1. var value=50;
2. function a(){
3. alert(window.value);//accessing global variable
4. }
```

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:

```
1. var a=40;//holding number
2. 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:

```
1. 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

JavaScript 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.

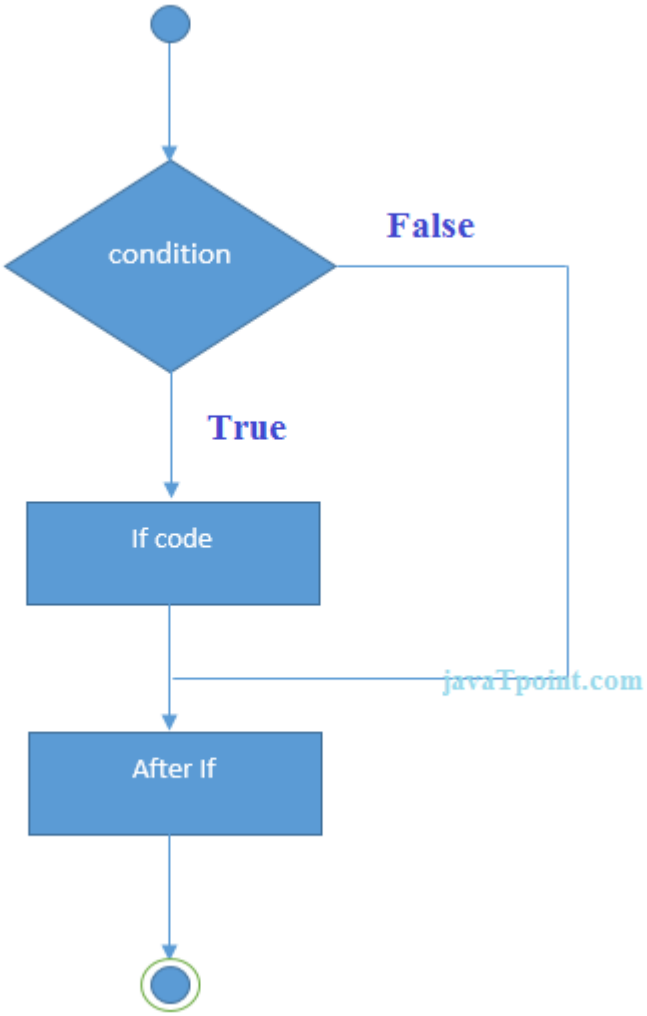
1. If Statement
2. If else statement
3. if else if statement

JavaScript If statement

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

```
1. if(expression){
2. //content to be evaluated
3. }
```

Flowchart of JavaScript If statement



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

```
1. <script>
2. var a=20;
3. if(a>10){
4. document.write("value of a is greater than 10");
5. }
6. </script>
```

Output of the above example

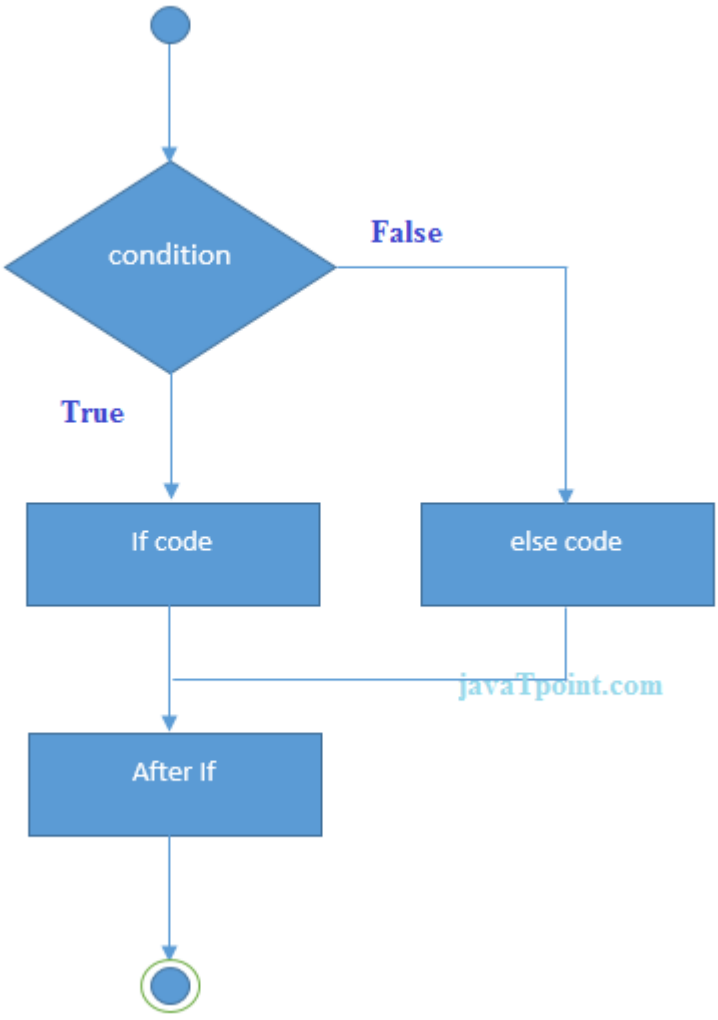
```
value of a is greater than 10
```

JavaScript If...else Statement

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

```
1. if(expression){
2. //content to be evaluated if condition is true
3. }
4. else{
5. //content to be evaluated if condition is false
6. }
```

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.

```
1. <script>
2. var a=20;
3. if(a%2==0){
```

```
4. document.write("a is even number");
5. }
6. else{
7. document.write("a is odd number");
8. }
9. </script>
```

Output of the above example

```
a is even number
```

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.

```
1. if(expression1){
2. //content to be evaluated if expression1 is true
3. }
4. else if(expression2){
5. //content to be evaluated if expression2 is true
6. }
7. else if(expression3){
8. //content to be evaluated if expression3 is true
9. }
10. else{
11. //content to be evaluated if no expression is true
12. }
```

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

```
1. <script>
2. var a=20;
3. if(a==10){
4. document.write("a is equal to 10");
5. }
6. else if(a==15){
7. document.write("a is equal to 15");
8. }
```

```
9.  else if(a==20){
10. document.write("a is equal to 20");
11. }
12. else{
13. document.write("a is not equal to 10, 15 or 20");
14. }
15. </script>
```

Output of the above example

```
a is equal to 20
```

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.

```
1.  switch(expression){
2.  case value1:
3.   code to be executed;
4.   break;
5.  case value2:
6.   code to be executed;
7.   break;
8.  ......
9.
10. default:
11.  code to be executed if above values are not matched;
12. }
```

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

```
1.  <script>
2.  var grade='B';
3.  var result;
4.  switch(grade){
5.  case 'A':
```

```
6.  result="A Grade";
7.  break;
8.  case 'B':
9.  result="B Grade";
10. break;
11. case 'C':
12. result="C Grade";
13. break;
14. default:
15. result="No Grade";
16. }
17. document.write(result);
18. </script>
```

Output of the above example

```
B Grade
```

Let's understand the behaviour of switch statement in JavaScript.

```
1.  <script>
2.  var grade='B';
3.  var result;
4.  switch(grade){
5.  case 'A':
6.  result+=" A Grade";
7.  case 'B':
8.  result+=" B Grade";
9.  case 'C':
10. result+=" C Grade";
11. default:
12. result+=" No Grade";
13. }
14. document.write(result);
15. </script>
```

Output of the above example

```
undefined B Grade C Grade No Grade
```

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.

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

Output:

```
1
2
3
4
5
```

2) JavaScript while loop

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

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

Let's see the simple example of while loop in javascript.

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

Output:

```
11
12
13
14
15
```

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);
```

Let's see the simple example of do while loop in javascript.

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

Output:

```
21
22
23
24
25
```

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.

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

JavaScript Functions can have 0 or more arguments.

JavaScript Function Example

Let's see the simple example of function in JavaScript that does not has arguments.

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

Output of the above example

JavaScript Function Arguments

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

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

Output of the above example

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.

```
<script>
function getInfo(){
return "hello javatpoint! How r u?";
```

```
4. }
5. </script>
6. <script>
7. document.write(getInfo());
8. </script>
```

Output of the above example

```
hello javatpoint! How r u?
```

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.

Method	Description
apply()	It is used to call a function contains this value and a single array of arguments.
bind()	It is used to create a new function.
call()	It is used to call a function contains this value and an argument list.

toString()	It returns the result in a form of a string.

JavaScript Function Object Examples

Example 1

Let's see an example to display the sum of given numbers.

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

Output:

```
7
```

Example 2

Let's see an example to display the power of provided value.

```
<script>
var pow=new Function("num1","num2","return Math.pow(num1,num2)");
document.writeln(pow(2,3));
</script>
```

Output:

```
8
```

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:

```
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.

```
<script>
emp={id:102,name:"Shyam Kumar",salary:40000}
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

Output of the above example

```
102 Shyam Kumar 40000
```

2) By creating instance of Object

The syntax of creating object directly is given below:

```
var objectname=new Object();
```

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

Let's see the example of creating object directly.

```
<script>
var emp=new Object();
```

```
3. emp.id=101;
4. emp.name="Ravi Malik";
5. emp.salary=50000;
6. document.write(emp.id+" "+emp.name+" "+emp.salary);
7. </script>
```

Output of the above example

```
101 Ravi 50000
```

---

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.

```
1. <script>
2. function emp(id,name,salary){
3. this.id=id;
4. this.name=name;
5. this.salary=salary;
6. }
7. e=new emp(103,"Vimal Jaiswal",30000);
8.
9. document.write(e.id+" "+e.name+" "+e.salary);
10. </script>
```

Output of the above example

```
103 Vimal Jaiswal 30000
```

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.

```
1. <script>
```

```
2. function emp(id,name,salary){
3. this.id=id;
4. this.name=name;
5. this.salary=salary;
6.
7. this.changeSalary=changeSalary;
8. function changeSalary(otherSalary){
9. this.salary=otherSalary;
10. }
11. }
12. e=new emp(103,"Sonoo Jaiswal",30000);
13. document.write(e.id+" "+e.name+" "+e.salary);
14. e.changeSalary(45000);
15. document.write("<br>"+e.id+" "+e.name+" "+e.salary);
16. </script>
```

Output of the above example

```
103 Sonoo Jaiswal 30000
103 Sonoo Jaiswal 45000
```

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:

```
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.

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

The .length property returns the length of an array.

Output of the above example

```
Sonoo
Vimal
Ratan
```

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.

```
1. <script>
2. var i;
3. var emp = new Array();
4. emp[0] = "Arun";
5. emp[1] = "Varun";
6. emp[2] = "John";
7.
8. for (i=0;i<emp.length;i++){
9. document.write(emp[i] + "<br>");
10. }
11. </script>
```

Output of the above example

```
Arun
Varun
John
```

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.

```
1. <script>
2. var emp=new Array("Jai","Vijay","Smith");
3. for (i=0;i<emp.length;i++){
4. document.write(emp[i] + "<br>");
5. }
6. </script>
```

Output of the above example

```
Jai
Vijay
Smith
```

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.

```
1. <script>
2. var str="This is string literal";
3. document.write(str);
4. </script>
```

Output:

```
This is string literal
```

---

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.

```
1. <script>
2. var stringname=new String("hello javascript string");
3. document.write(stringname);
4. </script>
```

Output:

```
hello javascript string
```

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.

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

Output:

```
v
```

2) JavaScript String concat(str) Method

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

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

Output:

```
javascript concat example
```

3) JavaScript String indexOf(str) Method

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

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

Output:

```
11
```

4) JavaScript String lastIndexOf(str) Method

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

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

Output:

```
16
```

5) JavaScript String toLowerCase() Method

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

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

Output:

```
javascript tolowercase example
```

6) JavaScript String toUpperCase() Method

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

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

Output:

```
JAVASCRIPT TOUPPERCASE EXAMPLE
```

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.

```
<script>
var s1="abcdefgh";
var s2=s1.slice(2,5);
document.write(s2);
</script>
```

Output:

```
cde
```

8) JavaScript String trim() Method

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

```
<script>
var s1="     javascript trim    ";
var s2=s1.trim();
document.write(s2);
</script>
```

Output:

```
javascript trim
```

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.

```
1. Current Date and Time: <span id="txt"></span>
2. <script>
3. var today=new Date();
4. document.getElementById('txt').innerHTML=today;
5. </script>
```

Output:

```
Current Date and Time: Fri Mar 03 2023 19:41:43 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>
```

Output:

```
Date is: 3/3/2023
```

JavaScript Current Time Example

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

```
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>
```

Output:

```
Current Time: 19:41:43
```

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.

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

```
3. window.onload=function(){getTime();}
4. function getTime(){
5. var today=new Date();
6. var h=today.getHours();
7. var m=today.getMinutes();
8. var s=today.getSeconds();
9. // add a zero in front of numbers<10
10. m=checkTime(m);
11. s=checkTime(s);
12. document.getElementById('txt').innerHTML=h+":"+m+":"+s;
13. setTimeout(function(){getTime()},1000);
14. }
15. //setInterval("getTime()",1000);//another way
16. function checkTime(i){
17. if (i<10){
18.  i="0" + i;
19.  }
20. return i;
21. }
22. </script>
```

Output:

```
Current Time: 19:41:57
```

JavaScript Math

The **JavaScript math** object provides several constants and methods to perform mathematical operation. Unlike date object, it doesn't have constructors.

JavaScript Math Methods

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

Methods	Description
abs()	It returns the absolute value of the given number.

acos()	It returns the arccosine of the given number in radians.
asin()	It returns the arcsine of the given number in radians.
atan()	It returns the arc-tangent of the given number in radians.
cbrt()	It returns the cube root of the given number.
ceil()	It returns a smallest integer value, greater than or equal to the given number.
cos()	It returns the cosine of the given number.
cosh()	It returns the hyperbolic cosine of the given number.
exp()	It returns the exponential form of the given number.
floor()	It returns largest integer value, lower than or equal to the given number.
hypot()	It returns square root of sum of the squares of given numbers.
log()	It returns natural logarithm of a number.
max()	It returns maximum value of the given numbers.
min()	It returns minimum value of the given numbers.
pow()	It returns value of base to the power of exponent.
random()	It returns random number between 0 (inclusive) and 1 (exclusive).
round()	It returns closest integer value of the given number.
sign()	It returns the sign of the given number
sin()	It returns the sine of the given number.
sinh()	It returns the hyperbolic sine of the given number.
sqrt()	It returns the square root of the given number
tan()	It returns the tangent of the given number.
tanh()	It returns the hyperbolic tangent of the given number.
trunc()	It returns an integer part of the given number.

Math.sqrt(n)

The JavaScript math.sqrt(n) method returns the square root of the given number.

```
1. Square Root of 17 is: <span id="p1"></span>
2. <script>
3. document.getElementById('p1').innerHTML=Math.sqrt(17);
4. </script>
```

Output:

```
Square Root of 17 is: 4.123105625617661
```

Math.random()

The JavaScript math.random() method returns the random number between 0 to 1.

```
1. Random Number is: <span id="p2"></span>
2. <script>
3. document.getElementById('p2').innerHTML=Math.random();
4. </script>
```

Output:

```
Random Number is: 0.4612602832393742
```

Math.pow(m,n)

The JavaScript math.pow(m,n) method returns the m to the power of n that is m^n.

```
1. 3 to the power of 4 is: <span id="p3"></span>
2. <script>
3. document.getElementById('p3').innerHTML=Math.pow(3,4);
4. </script>
```

Output:

```
3 to the power of 4 is: 81
```

Math.floor(n)

The JavaScript math.floor(n) method returns the lowest integer for the given number. For example 3 for 3.7, 5 for 5.9 etc.

```
1. Floor of 4.6 is: <span id="p4"></span>
2. <script>
3. document.getElementById('p4').innerHTML=Math.floor(4.6);
4. </script>
```

Output:

```
Floor of 4.6 is: 4
```

Math.ceil(n)

The JavaScript math.ceil(n) method returns the largest integer for the given number. For example 4 for 3.7, 6 for 5.9 etc.

```
1. Ceil of 4.6 is: <span id="p5"></span>
2. <script>
3. document.getElementById('p5').innerHTML=Math.ceil(4.6);
4. </script>
```

Output:

```
Ceil of 4.6 is: 5
```

Math.round(n)

The JavaScript math.round(n) method returns the rounded integer nearest for the given number. If fractional part is equal or greater than 0.5, it goes to upper value 1 otherwise lower value 0. For example 4 for 3.7, 3 for 3.3, 6 for 5.9 etc.

```
1. Round of 4.3 is: <span id="p6"></span><br>
2. Round of 4.7 is: <span id="p7"></span>
3. <script>
4. document.getElementById('p6').innerHTML=Math.round(4.3);
5. document.getElementById('p7').innerHTML=Math.round(4.7);
6. </script>
```

Output:

```
Round of 4.3 is: 4
Round of 4.7 is: 5
```

Math.abs(n)

The JavaScript math.abs(n) method returns the absolute value for the given number. For example 4 for -4, 6.6 for -6.6 etc.

```
1. Absolute value of -4 is: <span id="p8"></span>
2. <script>
3. document.getElementById('p8').innerHTML=Math.abs(-4);
4. </script>
```

Output:

```
Absolute value of -4 is: 4
```

JavaScript Number Object

The **JavaScript number** object *enables you to represent a numeric value*. It may be integer or floating-point. JavaScript number object follows IEEE standard to represent the floating-point numbers.

By the help of Number() constructor, you can create number object in JavaScript. For example:

```
1. var n=new Number(value);
```

If value can't be converted to number, it returns NaN(Not a Number) that can be checked by isNaN() method.

You can direct assign a number to a variable also. For example:

```
1. var x=102;//integer value
2. var y=102.7;//floating point value
3. var z=13e4;//exponent value, output: 130000
4. var n=new Number(16);//integer value by number object
```

Output:

```
102 102.7 130000 16
```

JavaScript Number Constants

Let's see the list of JavaScript number constants with description.

Constant	Description
MIN_VALUE	returns the largest minimum value.
MAX_VALUE	returns the largest maximum value.
POSITIVE_INFINITY	returns positive infinity, overflow value.
NEGATIVE_INFINITY	returns negative infinity, overflow value.
NaN	represents "Not a Number" value.

JavaScript Number Methods

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

Methods	Description
isFinite()	It determines whether the given value is a finite number.
isInteger()	It determines whether the given value is an integer.
parseFloat()	It converts the given string into a floating point number.
parseInt()	It converts the given string into an integer number.
toExponential()	It returns the string that represents exponential notation of the given number.
toFixed()	It returns the string that represents a number with exact digits after a decimal point.
toPrecision()	It returns the string representing a number of specified precision.
toString()	It returns the given number in the form of string.

JavaScript Boolean

JavaScript Boolean is an object that represents value in two states: *true* or *false*. You can create the JavaScript Boolean object by Boolean() constructor as given below.

```
1. Boolean b=new Boolean(value);
```

The default value of JavaScript Boolean object is *false*.

JavaScript Boolean Example

```
1. <script>
2. document.write(10<20);//true
3. document.write(10<5);//false
4. </script>
```

JavaScript Boolean Properties

Property	Description
constructor	returns the reference of Boolean function that created Boolean object.
prototype	enables you to add properties and methods in Boolean prototype.

JavaScript Boolean Methods

Method	Description
toSource()	returns the source of Boolean object as a string.
toString()	converts Boolean into String.
valueOf()	converts other type into Boolean.

Browser Object Model

1. Browser Object Model (BOM)

The **Browser Object Model** (BOM) is used to interact with the browser.

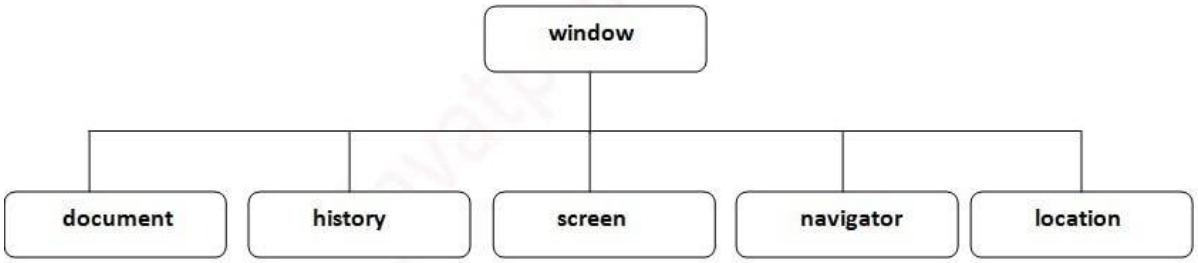
The default object of browser is window means you can call all the functions of window by specifying window or directly. For example:

```
1. window.alert("hello javatpoint");
```

is same as:

```
1. alert("hello javatpoint");
```

You can use a lot of properties (other objects) defined underneath the window object like document, history, screen, navigator, location, innerHeight, innerWidth,



Window Object

1. Window Object
2. Properties of Window Object
3. Methods of Window Object
4. Example of Window Object

The **window object** represents a window in browser. An object of window is created automatically by the browser.

Window is the object of browser, **it is not the object of javascript**. The javascript objects are string, array, date etc.

Methods of window object

The important methods of window object are as follows:

Method	Description
alert()	displays the alert box containing message with ok button.
confirm()	displays the confirm dialog box containing message with ok and cancel button.
prompt()	displays a dialog box to get input from the user.
open()	opens the new window.
close()	closes the current window.
setTimeout()	performs action after specified time like calling function, evaluating expressions etc.

Example of alert() in javascript

It displays alert dialog box. It has message and ok button.

```
<script type="text/javascript">
function msg(){
 alert("Hello Alert Box");
}
</script>
<input type="button" value="click" onclick="msg()"/>
```

Output of the above example

Example of confirm() in javascript

It displays the confirm dialog box. It has message with ok and cancel buttons.

```
<script type="text/javascript">
function msg(){
var v= confirm("Are u sure?");
if(v==true){
```

```
5. alert("ok");
6. }
7. else{
8. alert("cancel");
9. }
10.
11. }
12. </script>
13.
14. <input type="button" value="delete record" onclick="msg()"/>
```

Output of the above example

Example of prompt() in javascript

It displays prompt dialog box for input. It has message and textfield.

```
1. <script type="text/javascript">
2. function msg(){
3. var v= prompt("Who are you?");
4. alert("I am "+v);
5.
6. }
7. </script>
8.
9. <input type="button" value="click" onclick="msg()"/>
```

Output of the above example

Example of open() in javascript

It displays the content in a new window.

```
1. <script type="text/javascript">
2. function msg(){
3. open("http://www.javatpoint.com");
4. }
5. </script>
```

```
6. <input type="button" value="javatpoint" onclick="msg()"/>
```

Output of the above example

Example of setTimeout() in javascript

It performs its task after the given milliseconds.

```
1. <script type="text/javascript">
2. function msg(){
3. setTimeout(
4. function(){
5. alert("Welcome to Javatpoint after 2 seconds")
6. },2000);
7.
8. }
9. </script>
10.
11. <input type="button" value="click" onclick="msg()"/>
```

JavaScript History Object

The **JavaScript history object** represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.

The history object is the window property, so it can be accessed by:

```
1. window.history
```

Or,

```
1. history
```

Property of JavaScript history object

There are only 1 property of history object.

No.	Property	Description
1	length	returns the length of the history URLs.

Methods of JavaScript history object

There are only 3 methods of history object.

No.	Method	Description
1	forward()	loads the next page.
2	back()	loads the previous page.
3	go()	loads the given page number.

Example of history object

Let's see the different usage of history object.

```
1. history.back();//for previous page
2. history.forward();//for next page
3. history.go(2);//for next 2nd page
4. history.go(-2);//for previous 2nd page
```

JavaScript Navigator Object

The **JavaScript navigator object** is used for browser detection. It can be used to get browser information such as appName, appCodeName, userAgent etc.

The navigator object is the window property, so it can be accessed by:

1. window.navigator

Or,

1. navigator

Property of JavaScript navigator object

There are many properties of navigator object that returns information of the browser.

No.	Property	Description
1	appName	returns the name
2	appVersion	returns the version
3	appCodeName	returns the code name
4	cookieEnabled	returns true if cookie is enabled otherwise false
5	userAgent	returns the user agent
6	language	returns the language. It is supported in Netscape and Firefox only.
7	userLanguage	returns the user language. It is supported in IE only.
8	plugins	returns the plugins. It is supported in Netscape and Firefox only.
9	systemLanguage	returns the system language. It is supported in IE only.
10	mimeTypes[]	returns the array of mime type. It is supported in Netscape and Firefox only.

11	platform	returns the platform e.g. Win32.
12	online	returns true if browser is online otherwise false.

Methods of JavaScript navigator object

The methods of navigator object are given below.

No.	Method	Description
1	javaEnabled()	checks if java is enabled.
2	taintEnabled()	checks if taint is enabled. It is deprecated since JavaScript 1.2.

Example of navigator object

Let's see the different usage of history object.

```
<script>
document.writeln("<br/>navigator.appCodeName: "+navigator.appCodeName);
document.writeln("<br/>navigator.appName: "+navigator.appName);
document.writeln("<br/>navigator.appVersion: "+navigator.appVersion);
document.writeln("<br/>navigator.cookieEnabled: "+navigator.cookieEnabled
);
document.writeln("<br/>navigator.language: "+navigator.language);
document.writeln("<br/>navigator.userAgent: "+navigator.userAgent);
document.writeln("<br/>navigator.platform: "+navigator.platform);
document.writeln("<br/>navigator.onLine: "+navigator.onLine);
</script>
```

```
navigator.appCodeName: Mozilla
navigator.appName: Netscape
navigator.appVersion: 5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/37.0.2062.124 Safari/537.36
navigator.cookieEnabled: true
navigator.language: en-US
navigator.userAgent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36
```

```
(KHTML, like Gecko) Chrome/37.0.2062.124 Safari/537.36
navigator.platform: Win32
navigator.onLine: true
```

JavaScript Screen Object

The **JavaScript screen object** holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc.

The navigator object is the window property, so it can be accessed by:

1. window.screen

Or,

1. screen

Property of JavaScript Screen Object

There are many properties of screen object that returns information of the browser.

No.	Property	Description
1	width	returns the width of the screen
2	height	returns the height of the screen
3	availWidth	returns the available width
4	availHeight	returns the available height
5	colorDepth	returns the color depth

6	pixelDepth	returns the pixel depth.

Example of JavaScript Screen Object

Let's see the different usage of screen object.

```
<script>
document.writeln("<br/>screen.width: "+screen.width);
document.writeln("<br/>screen.height: "+screen.height);
document.writeln("<br/>screen.availWidth: "+screen.availWidth);
document.writeln("<br/>screen.availHeight: "+screen.availHeight);
document.writeln("<br/>screen.colorDepth: "+screen.colorDepth);
document.writeln("<br/>screen.pixelDepth: "+screen.pixelDepth);
</script>
```

```
screen.width: 1366
screen.height: 768
screen.availWidth: 1366
screen.availHeight: 728
screen.colorDepth: 24
screen.pixelDepth: 24
```

Document Object Model

1. Document Object
2. Properties of document object
3. Methods of document object
4. Example of document object

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

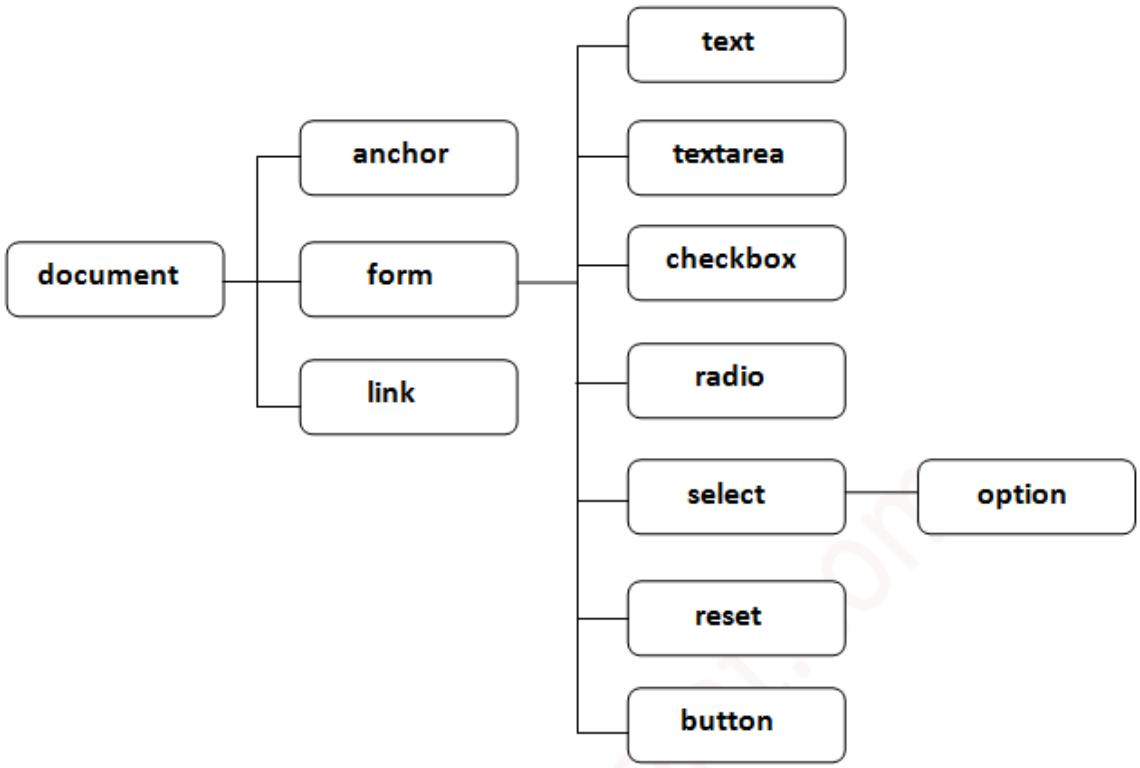
Play Video

1. 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">
```

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

Javascript - document.getElementById() method

1. getElementById() method
2. Example of getElementById()

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.

```
1. <script type="text/javascript">
2. function getcube(){
3. var number=document.getElementById("number").value;
4. alert(number*number*number);
5. }
6. </script>
7. <form>
8. Enter No:<input type="text" id="number" name="number"/><br/>
9. <input type="button" value="cube" onclick="getcube()"/>
10. </form>
```

GetElementsByClassName()

The getElementsByClassName() method is used for selecting or getting the elements through their class name value. This DOM method returns an array-like object that consists of all the elements having the specified classname. On calling the getElementsByClassName() method on any particular element, it will search the whole document and will return only those elements which match the specified or given class name.

Syntax

```
1. var ele=document.getELementsByClassName('name');
```

Here, name is the mandatory argument to be passed. It is the string that specifies either a single classname or multiple class names to match.

Example of getElementsByClassName() Method

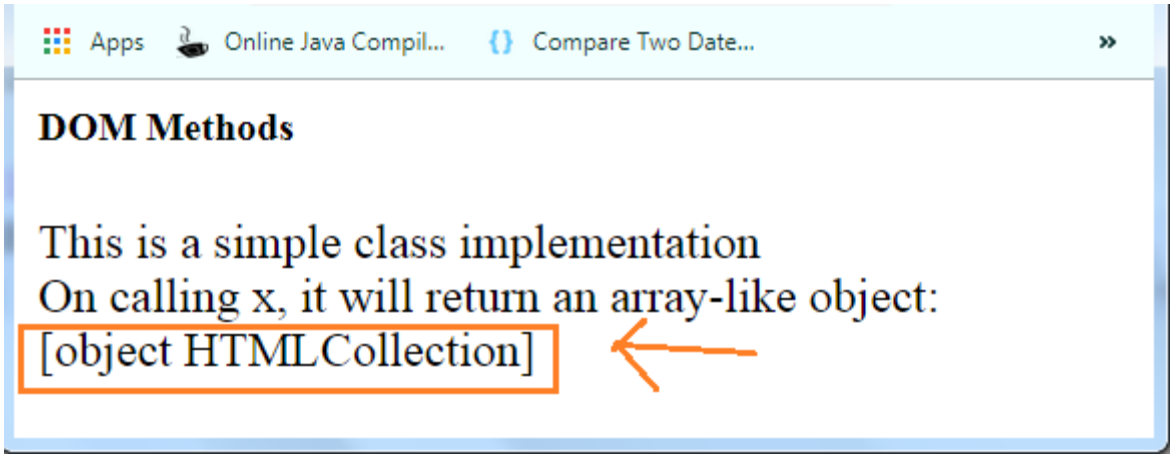
Let's look at some examples to know and understand the practical implementation of the method.

Example

It is a simple class implementation that returns an array-like object on invoking the variable x.

```
1. <html>
2. <head> <h5>DOM Methods </h5> </head>
3. <body>
4. <div class="Class">
5. This is a simple class implementation
6. </div>
7. <script type="text/javascript">
8. var x=document.getElementsByClassName('Class');
9. document.write("On calling x, it will return an arrsy-like object: <br>"+x);
10. </script>
11. </body>
12. </html>
```

Output:



Similarly, we can implement the getElementsByClassName() method for returning collections of elements for multiple classes.

Difference between getElementsByClassName(), querySelector() and querySelectorAll() Methods

getElementsByClassName(): It matches the elements with the specified class name, and returns a set of the matched elements. The returned elements are live HTML collection of elements. These live elements can be further updated if any changes are made in the Document Object Model.

querySelector(): It returns only a single element that matches the specified classname. If it does not find any matching element, it returns null.

The main point to understand is that all the above-described methods return either one element or a list, but the getELementsByClassName() method serves the **dynamic** updation, and the other two methods serve for the **static**.

Javascript - document.getElementsByName() method

1. getElementsByName() method
2. Example of getElementsByName()

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:

```
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 getElementBy TagName() method.</p>
<p>Let's see the simple example</p>
<button onclick="countpara()">count paragraph</button>
```

Output of the above example

This is a pragraph

Here we are going to count total number of paragraphs by getElementByTagName() method.

Let's see the simple example

count paragraph

Another example of document.getElementsByTagName() method

In this example, we going to count total number of h2 and h3 tags used in the document.

```
<script type="text/javascript">
function counth2(){
var totalh2=document.getElementsByTagName("h2");
alert("total h2 tags are: "+totalh2.length);
}
function counth3(){
```

```
7. var totalh3=document.getElementsByTagName("h3");
8. alert("total h3 tags are: "+totalh3.length);
9. }
10. </script>
11. <h2>This is h2 tag</h2>
12. <h2>This is h2 tag</h2>
13. <h3>This is h3 tag</h3>
14. <h3>This is h3 tag</h3>
15. <h3>This is h3 tag</h3>
16. <button onclick="counth2()">count h2</button>
17. <button onclick="counth3()">count h3</button>
```

Output of the above example

Javascript - innerHTML

1. javascript innerHTML
2. Example of innerHTML property

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><tex
tarea 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

1. javascript innerText
2. Example of innerText property

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){
```

```
5.  msg="good";
6.  }
7.  else{
8.  msg="poor";
9.  }
10. document.getElementById('mylocation').innerText=msg;
11. }
12.
13. </script>
14. <form name="myForm">
15. <input type="password" value="" name="userPass" onkeyup="validate()">
16. Strength:<span id="mylocation">no strength</span>
17. </form>
```

JavaScript Form Validation

1. JavaScript form validation
2. Example of JavaScript validation
3. JavaScript email 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.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="abc.jsp" onsubmit="return validateform()" >
Name: <input type="text" name="name"><br/>
Password: <input type="password" name="password"><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;
}
```

```
13. }
14. </script>
15.
16. <form name="f1" action="register.jsp" onsubmit="return matchpass()">
17. Password:<input type="password" name="password" /><br/>
18. Re-enter Password:<input type="password" name="password2"/><br/>
19. <input type="submit">
20. </form>
```

JavaScript Number Validation

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

```
1.  <script>
2.  function validate(){
3.  var num=document.myform.num.value;
4.  if (isNaN(num)){
5.    document.getElementById("numloc").innerHTML="Enter Numeric value only"
    ;
6.    return false;
7.  }else{
8.    return true;
9.    }
10. }
11. </script>
12. <form name="myform" onsubmit="return validate()" >
13. Number: <input type="text" name="num"><span id="numloc"></span><b
    r/>
14. <input type="submit" value="submit">
15. </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.

```
1. <script>
2. function validate(){
3. var name=document.f1.name.value;
4. var password=document.f1.password.value;
5. var status=false;
6.
7. if(name.length<1){
8. document.getElementById("nameloc").innerHTML=
9. " <img src='unchecked.gif'/> Please enter your name";
10. status=false;
11. }else{
12. document.getElementById("nameloc").innerHTML=" <img src='checked.gif'/>
    ";
13. status=true;
14. }
15. if(password.length<6){
16. document.getElementById("passwordloc").innerHTML=
17. " <img src='unchecked.gif'/> Password must be at least 6 char long";
18. status=false;
19. }else{
20. document.getElementById("passwordloc").innerHTML=" <img src='checked.gi
    f'/>";
21. }
22. return status;
23. }
24. </script>
25.
26. <form name="f1" action="#" onsubmit="return validate()">
27. <table>
28. <tr><td>Enter Name:</td><td><input type="text" name="name"/>
29. <span id="nameloc"></span></td></tr>
30. <tr><td>Enter Password:</td><td><input type="password" name="passw
    ord"/>
31. <span id="passwordloc"></span></td></tr>
32. <tr><td colspan="2"><input type="submit" value="register"/></td></tr>

33. </table>
```

```
34. </form>
```

Output:

Enter Name:

Enter Password:

register

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.

```
1. <script>
2. function validateemail()
3. {
4. var x=document.myform.email.value;
5. var atposition=x.indexOf("@");
6. var dotposition=x.lastIndexOf(".");
7. if (atposition<1 || dotposition<atposition+2 || dotposition+2>=x.length){
8.   alert("Please enter a valid e-
   mail address \n atpostion:"+atposition+"\n dotposition:"+dotposition);
9.   return false;
10.  }
11.}
12. </script>
13. <body>
14. <form name="myform"  method="post" action="#" onsubmit="return validat
   eemail();">
15. Email: <input type="text" name="email"><br/>
16. <input type="submit" value="register">
```

```
17. </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.

```
1. <script>
2. //Declaring class
3. class Employee
4.  {
5. //Initializing an object
6.    constructor(id,name)
7.    {
8.      this.id=id;
9.      this.name=name;
10.   }
11. //Declaring method
12.   detail()
13.   {
```

```
14.  document.writeln(this.id+" "+this.name+"<br>")
15.   }
16.  }
17. //passing object to a variable
18. var e1=new Employee(101,"Martin Roy");
19. var e2=new Employee(102,"Duke William");
20. e1.detail(); //calling method
21. e2.detail();
22. </script>
```

Output:

```
101 Martin Roy
102 Duke William
```

Class Declarations Example: Hoisting

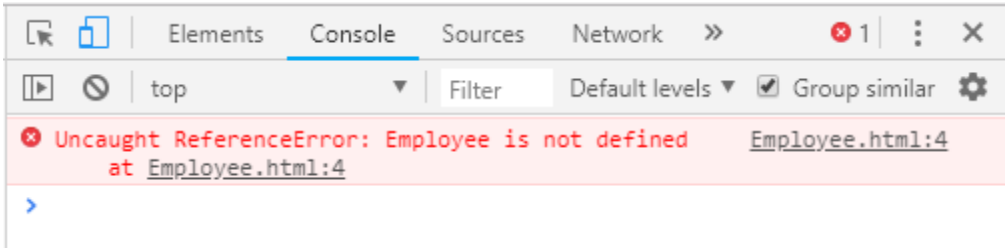
Unlike function declaration, the class declaration is not a part of JavaScript hoisting. So, it is required to declare the class before invoking it.

Let's see an example.

```
1.  <script>
2.  //Here, we are invoking the class before declaring it.
3.  var e1=new Employee(101,"Martin Roy");
4.  var e2=new Employee(102,"Duke William");
5.  e1.detail(); //calling method
6.  e2.detail();
7.
8.  //Declaring class
9.  class Employee
10.  {
11. //Initializing an object
12.    constructor(id,name)
13.    {
14.      this.id=id;
15.      this.name=name;
16.    }
17.    detail()
18.    {
```

```
19.  document.writeln(this.id+" "+this.name+"<br>")
20.    }
21.  }
22. </script>
```

Output:



Class Declarations Example: Re-declaring Class

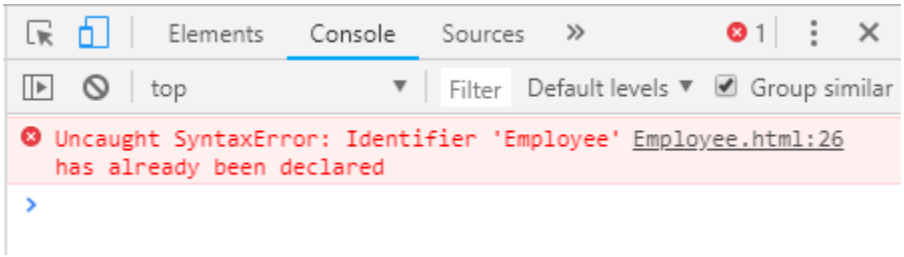
A class can be declared once only. If we try to declare class more than one time, it throws an error.

Let's see an example.

```
1.  <script>
2.  //Declaring class
3.  class Employee
4.    {
5.  //Initializing an object
6.      constructor(id,name)
7.      {
8.        this.id=id;
9.        this.name=name;
10.     }
11.     detail()
12.     {
13.  document.writeln(this.id+" "+this.name+"<br>")
14.     }
15.  }
16. //passing object to a variable
17. var e1=new Employee(101,"Martin Roy");
18. var e2=new Employee(102,"Duke William");
19. e1.detail(); //calling method
```

```
20. e2.detail();
21. //Re-declaring class
22. class Employee
23.  {
24.  }
25. </script>
```

Output:



Class expressions

Another way to define a class is by using a class expression. Here, it is not mandatory to assign the name of the class. So, the class expression can be named or unnamed. The class expression allows us to fetch the class name. However, this will not be possible with class declaration.

Unnamed Class Expression

The class can be expressed without assigning any name to it.

Let's see an example.

```
1. <script>
2. var emp = class {
3.   constructor(id, name) {
4.     this.id = id;
5.     this.name = name;
6.   }
7. };
8. document.writeln(emp.name);
9. </script>
```

Output:

emp

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.

```
<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;
    }
```

```
32.    detail()
33.    {
34.   document.writeln(this.id+" "+this.name+"<br>")
35.    }
36.  }
37. //passing object to a variable
38. var e1=new emp(103,"James Bella");
39. var e2=new emp(104,"Nick Johnson");
40. e1.detail(); //calling method
41. e2.detail();
42. </script>
```

Output:

```
101 Martin Roy
102 Duke William
103 James Bella
104 Nick Johnson
```

Named Class Expression Example

We can express the class with the particular name. Here, the scope of the class name is up to the class body. The class is retrieved using class.name property.

```
1.  <script>
2.  var emp = class Employee {
3.    constructor(id, name) {
4.      this.id = id;
5.      this.name = name;
6.    }
7.  };
8.  document.writeln(emp.name);
9.  /*document.writeln(Employee.name);
10. Error occurs on console:
11. "ReferenceError: Employee is not defined
12. */
13. </script>
```

Output:

```
Employee
```

JavaScript Prototype Object

JavaScript is a prototype-based language that facilitates the objects to acquire properties and features from one another. Here, each object contains a prototype object.

In JavaScript, whenever a function is created the prototype property is added to that function automatically. This property is a prototype object that holds a constructor property.

Syntax:

1. ClassName.prototype.methodName

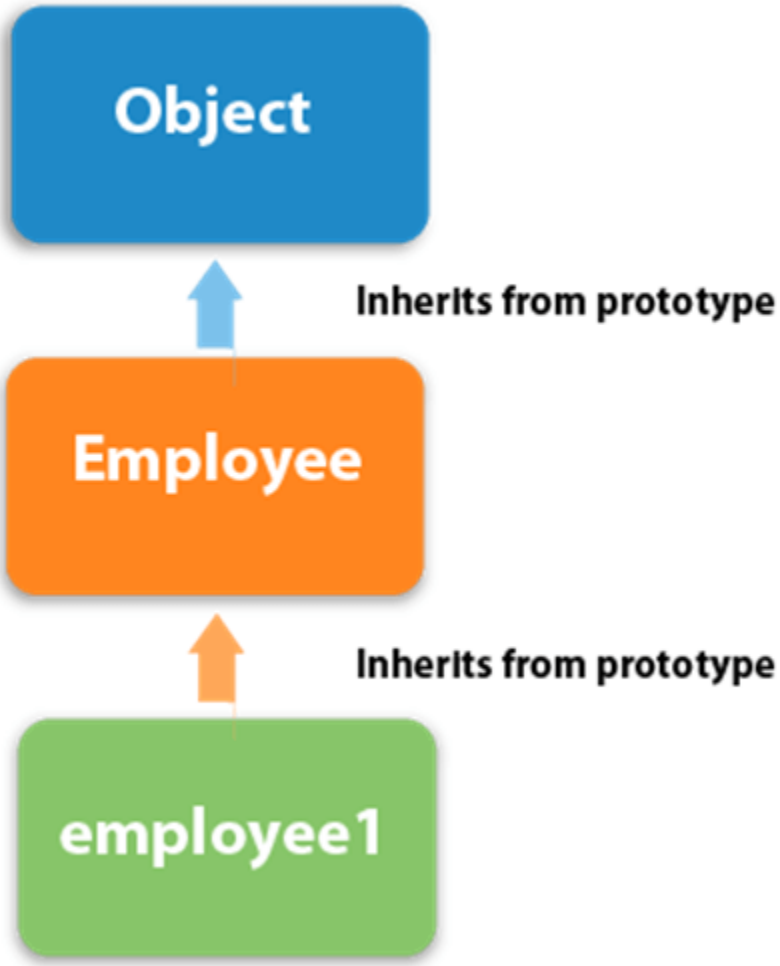
What is the requirement of a prototype object?

Whenever an object is created in JavaScript, its corresponding functions are loaded into memory. So, a new copy of the function is created on each object creation.

In a prototype-based approach, all the objects share the same function. This ignores the requirement of creating a new copy of function for each object. Thus, the functions are loaded once into the memory.

Prototype Chaining

In JavaScript, each object contains a prototype object that acquires properties and methods from it. Again an object's prototype object may contain a prototype object that also acquires properties and methods, and so on. It can be seen as prototype chaining.



JavaScript Prototype Object Example 1

Let's see an example to add a new method to the constructor function.

```
<script>
function Employee(firstName,lastName)
{
  this.firstName=firstName;
  this.lastName=lastName;
}

Employee.prototype.fullName=function()
 {
   return this.firstName+" "+this.lastName;
```

```
11.  }
12.
13. var employee1=new Employee("Martin","Roy");
14. var employee2=new Employee("Duke", "William");
15. document.writeln(employee1.fullName()+"<br>");
16. document.writeln(employee2.fullName());
17. </script>
```

Output:

```
Martin Roy
Duke William
```

Example 2

Let's see an example to add a new property to the constructor function.

```
1.  <script>
2.  function Employee(firstName,lastName)
3.  {
4.    this.firstName=firstName;
5.    this.lastName=lastName;
6.  }
7.
8.  Employee.prototype.company="Javatpoint"
9.
10. var employee1=new Employee("Martin","Roy");
11. var employee2=new Employee("Duke", "William");
12. document.writeln(employee1.firstName+" "+employee1.lastName+" "+emplo
    yee1.company+"<br>");
13. document.writeln(employee2.firstName+" "+employee2.lastName+" "+emplo
    yee2.company);
14. </script>
```

Output:

```
Martin Roy Javatpoint
Duke William Javatpoint
```

JavaScript Constructor Method

A JavaScript constructor method is a special type of method which is used to initialize and create an object. It is called when memory is allocated for an object.

Points to remember

- The constructor keyword is used to declare a constructor method.
- The class can contain one constructor method only.
- JavaScript allows us to use parent class constructor through super keyword.

Constructor Method Example

Let's see a simple example of a constructor method.

```
<script>
class Employee {
  constructor() {
    this.id=101;
    this.name = "Martin Roy";
  }
}
var emp = new Employee();
document.writeln(emp.id+" "+emp.name);
</script>
```

Output:

```
101 Martin Roy
```

Constructor Method Example: super keyword

The super keyword is used to call the parent class constructor. Let's see an example.

```
<script>
class CompanyName
{
  constructor()
  {
    this.company="Javatpoint";
```

```
7.    }
8.  }
9.  class Employee extends CompanyName {
10.  constructor(id,name) {
11.   super();
12.    this.id=id;
13.    this.name=name;
14.  }
15. }
16. var emp = new Employee(1,"John");
17. document.writeln(emp.id+" "+emp.name+" "+emp.company);
18. </script>
```

Output:

```
1 John Javatpoint
```

JavaScript static Method

The JavaScript provides static methods that belong to the class instead of an instance of that class. So, an instance is not required to call the static method. These methods are called directly on the class itself.

Points to remember

- The static keyword is used to declare a static method.
- The static method can be of any name.
- A class can contain more than one static method.
- If we declare more than one static method with a similar name, the JavaScript always invokes the last one.
- The static method can be used to create utility functions.
- We can use this keyword to call a static method within another static method.
- We cannot use this keyword directly to call a static method within the non-static method. In such case, we can call the static method either using the class name or as the property of the constructor.

JavaScript static Method Example 1

Let's see a simple example of a static method.

```
<script>
class Test
{
  static display()
  {
    return "static method is invoked"
  }
}
document.writeln(Test.display());
</script>
```

Output:

```
static method is invoked
```

Example 2

Le's see an example to invoke more than one static method.

```
<script>
class Test
{
  static display1()
  {
    return "static method is invoked"
  }
  static display2()
  {
    return "static method is invoked again"
  }
}
document.writeln(Test.display1()+"<br>");
document.writeln(Test.display2());
</script>
```

Output:

```
static method is invoked
static method is invoked again
```

Example 3

Let's see an example to invoke more than one static method with similar names.

```
<script>
class Test
{
  static display()
  {
    return "static method is invoked"
  }
  static display()
  {
    return "static method is invoked again"
  }
}
document.writeln(Test.display());
</script>
```

Output:

```
static method is invoked again
```

Example 4

Let's see an example to invoke a static method within the constructor.

```
<script>
class Test {
  constructor() {
  document.writeln(Test.display()+"<br>");
  document.writeln(this.constructor.display());
  }

  static display() {
      return "static method is invoked"
  }
}
var t=new Test();
```

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

Output:

```
static method is invoked
static method is invoked
```

Example 5

Let's see an example to invoke a static method within the non-static method.

```
1. <script>
2. class Test {
3.   static display() {
4.       return "static method is invoked"
5.   }
6.
7.   show() {
8.    document.writeln(Test.display()+"<br>");
9.    }
10.}
11.var t=new Test();
12.t.show();
13.</script>
```

Output:

```
static method is invoked
```

JavaScript Encapsulation

The JavaScript Encapsulation is a process of binding the data (i.e. variables) with the functions acting on that data. It allows us to control the data and validate it. To achieve an encapsulation in JavaScript: -

- Use var keyword to make data members private.
- Use setter methods to set the data and getter methods to get that data.

The encapsulation allows us to handle an object using the following properties:

Read/Write - Here, we use setter methods to write the data and getter methods read that data.

Read Only - In this case, we use getter methods only.

Write Only - In this case, we use setter methods only.

JavaScript Encapsulation Example

Let's see a simple example of encapsulation that contains two data members with its setter and getter methods.

```
<script>
class Student
 {
   constructor()
   {
     var name;
     var marks;
   }
     getName()
     {
       return this.name;
     }
   setName(name)
   {
     this.name=name;
   }

   getMarks()
   {
     return this.marks;
   }
  setMarks(marks)
  {
    this.marks=marks;
  }

  }
```

```
28.    var stud=new Student();
29.     stud.setName("John");
30.     stud.setMarks(80);
31.     document.writeln(stud.getName()+" "+stud.getMarks());
32. </script>
```

Output:

```
John 80
```

JavaScript Encapsulation Example: Validate

In this example, we validate the marks of the student.

```
1.  <script>
2.  class Student
3.   {
4.     constructor()
5.     {
6.        var name;
7.        var marks;
8.     }
9.        getName()
10.       {
11.         return this.name;
12.       }
13.     setName(name)
14.     {
15.       this.name=name;
16.     }
17.
18.     getMarks()
19.     {
20.       return this.marks;
21.     }
22.    setMarks(marks)
23.    {
24.       if(marks<0||marks>100)
25.       {
```

```
26.          alert("Invalid Marks");
27.        }
28.       else
29.        {
30.          this.marks=marks;
31.        }
32.    }
33.    }
34.    var stud=new Student();
35.     stud.setName("John");
36.     stud.setMarks(110);//alert() invokes
37.     document.writeln(stud.getName()+" "+stud.getMarks());
38. </script>
```

Output:

```
John undefined
```

JavaScript Encapsulation Example: Prototype-based approach

Here, we perform prototype-based encapsulation.

```
1.  <script>
2.  function Student(name,marks)
3.  {
4.    var s_name=name;
5.    var s_marks=marks;
6.    Object.defineProperty(this,"name",{
7.      get:function()
8.      {
9.        return s_name;
10.     },
11.   set:function(s_name)
12.   {
13.     this.s_name=s_name;
14.   }
15.
16. });
17.
```

```
18.    Object.defineProperty(this,"marks",{
19.    get:function()
20.    {
21.      return s_marks;
22.    },
23.   set:function(s_marks)
24.   {
25.     this.s_marks=s_marks;
26.   }
27.
28. });
29.
30. }
31.   var stud=new Student("John",80);
32.   document.writeln(stud.name+" "+stud.marks);
33. </script>
```

Output:

```
John 80
```

JavaScript Inheritance

The JavaScript inheritance is a mechanism that allows us to create new classes on the basis of already existing classes. It provides flexibility to the child class to reuse the methods and variables of a parent class.

The JavaScript **extends** keyword is used to create a child class on the basis of a parent class. It facilitates child class to acquire all the properties and behavior of its parent class.

Points to remember

- It maintains an IS-A relationship.
- The extends keyword is used in class expressions or class declarations.
- Using extends keyword, we can acquire all the properties and behavior of the inbuilt object as well as custom classes.
- We can also use a prototype-based approach to achieve inheritance.

JavaScript extends Example: inbuilt object

In this example, we extends **Date** object to display today's date.

```
<script>
class Moment extends Date {
  constructor() {
    super();
  }}
var m=new Moment();
document.writeln("Current date:")
document.writeln(m.getDate()+"-"+(m.getMonth()+1)+"-"+m.getFullYear());
</script>
```

Output:

```
Current date: 31-8-2018
```

Let's see one more example to display the year value from the given date.

```
<script>
class Moment extends Date {
  constructor(year) {
    super(year);
  }}
var m=new Moment("August 15, 1947 20:22:10");
document.writeln("Year value:")
document.writeln(m.getFullYear());
</script>
```

Output:

```
Year value: 1947
```

JavaScript extends Example: Custom class

In this example, we declare sub-class that extends the properties of its parent class.

```
<script>
class Bike
{
```

```
4.    constructor()
5.    {
6.      this.company="Honda";
7.    }
8.  }
9.  class Vehicle extends Bike {
10.  constructor(name,price) {
11.   super();
12.    this.name=name;
13.    this.price=price;
14.  }
15. }
16. var v = new Vehicle("Shine","70000");
17. document.writeln(v.company+" "+v.name+" "+v.price);
18. </script>
```

Output:

```
Honda Shine 70000
```

JavaScript extends Example: a Prototype-based approach

Here, we perform prototype-based inheritance. In this approach, there is no need to use class and extends keywords.

```
1.  <script>
2.  //Constructor function
3.  function Bike(company)
4.  {
5.     this.company=company;
6.  }
7.
8.  Bike.prototype.getCompany=function()
9.  {
10.  return this.company;
11. }
12. //Another constructor function
13. function Vehicle(name,price) {
14.  this.name=name;
```

```
15.   this.price=price;
16.   }
17. var bike = new Bike("Honda");
18. Vehicle.prototype=bike; //Now Bike treats as a parent of Vehicle.
19. var vehicle=new Vehicle("Shine",70000);
20. document.writeln(vehicle.getCompany()+" "+vehicle.name+" "+vehicle.price);
21. </script>
```

Output:

```
Honda Shine 70000
```

JavaScript Polymorphism

The polymorphism is a core concept of an object-oriented paradigm that provides a way to perform a single action in different forms. It provides an ability to call the same method on different JavaScript objects. As JavaScript is not a type-safe language, we can pass any type of data members with the methods.

JavaScript Polymorphism Example 1

Let's see an example where a child class object invokes the parent class method.

```
1.   <script>
2.   class A
3.    {
4.       display()
5.     {
6.        document.writeln("A is invoked");
7.     }
8.    }
9.   class B extends A
10.  {
11.  }
12. var b=new B();
13. b.display();
14. </script>
```

Output:

```
A is invoked
```

Example 2

Let's see an example where a child and parent class contains the same method. Here, the object of child class invokes both classes method.

```
<script>
class A
 {
   display()
  {
    document.writeln("A is invoked<br>");
  }
 }
class B extends A
 {
  display()
  {
    document.writeln("B is invoked");
  }
 }

var a=[new A(), new B()]
a.forEach(function(msg)
{
msg.display();
});
</script>
```

Output:

```
A is invoked
B is invoked
```

Example 3

Let's see the same example with prototype-based approach.

```
<script>
function A()
```

```
3. {
4. }
5. A.prototype.display=function()
6. {
7.   return "A is invoked";
8. }
9. function B()
10. {
11.
12. }
13.
14. B.prototype=Object.create(A.prototype);
15.
16. var a=[new A(), new B()]
17.
18. a.forEach(function(msg)
19. {
20.   document.writeln(msg.display()+"<br>");
21. });
22. <script>
```

Output:

```
A is invoked
B is invoked
```

JavaScript Abstraction

An abstraction is a way of hiding the implementation details and showing only the functionality to the users. In other words, it ignores the irrelevant details and shows only the required one.

Points to remember

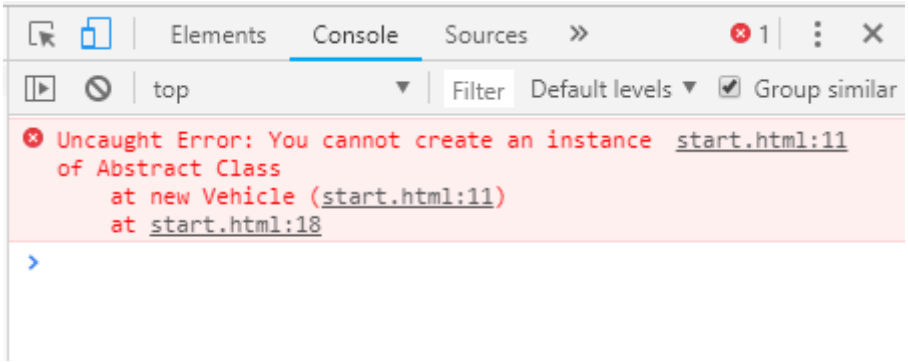
- We cannot create an instance of Abstract Class.
- It reduces the duplication of code.

JavaScript Abstraction Example

Example 1

Let's check whether we can create an instance of Abstract class or not.

```
1. <script>
2. //Creating a constructor function
3. function Vehicle()
4. {
5.     this.vehicleName= vehicleName;
6.     throw new Error("You cannot create an instance of Abstract class");
7.
8. }
9. Vehicle.prototype.display=function()
10. {
11.     return this.vehicleName;
12. }
13. var vehicle=new Vehicle();
14. </script>
```



Example 2

Let's see an example to achieve abstraction.

```
1. <script>
2. //Creating a constructor function
3.  function Vehicle()
4. {
5.     this.vehicleName="vehicleName";
6.     throw new Error("You cannot create an instance of Abstract Class");
7. }
8. Vehicle.prototype.display=function()
```

```
{
    return "Vehicle is: "+this.vehicleName;
}
//Creating a constructor function
function Bike(vehicleName)
{
    this.vehicleName=vehicleName;
}
//Creating object without using the function constructor
Bike.prototype=Object.create(Vehicle.prototype);
var bike=new Bike("Honda");
document.writeln(bike.display());


</script>
```

Output:

```
Vehicle is: Honda
```

Example 3

In this example, we use instanceof operator to test whether the object refers to the corresponding class.

```
<script>
//Creating a constructor function
 function Vehicle()
{
    this.vehicleName=vehicleName;
    throw new Error("You cannot create an instance of Abstract class");
}
//Creating a constructor function
function Bike(vehicleName)
{
    this.vehicleName=vehicleName;
}
Bike.prototype=Object.create(Vehicle.prototype);
var bike=new Bike("Honda");
```

```
15. document.writeln(bike instanceof Vehicle);
16. document.writeln(bike instanceof Bike);
17.
18. </script>
```

Output:

```
true true
```

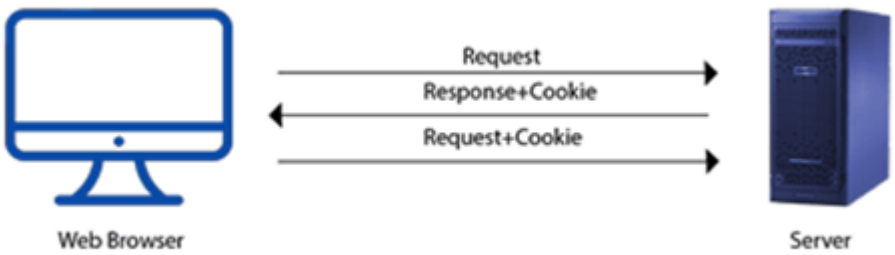
JavaScript Cookies

A cookie is an amount of information that persists between a server-side and a client-side. A web browser stores this information at the time of browsing.

A cookie contains the information as a string generally in the form of a name-value pair separated by semi-colons. It maintains the state of a user and remembers the user's information among all the web pages.

How Cookies Works?

- When a user sends a request to the server, then each of that request is treated as a new request sent by the different user.
- So, to recognize the old user, we need to add the cookie with the response from the server.
- browser at the client-side.
- Now, whenever a user sends a request to the server, the cookie is added with that request automatically. Due to the cookie, the server recognizes the users.



How to create a Cookie in JavaScript?

In JavaScript, we can create, read, update and delete a cookie by using **document.cookie** property.

The following syntax is used to create a cookie:

```
1. document.cookie="name=value";
```

JavaScript Cookie Example

Example 1

Let's see an example to set and get a cookie.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. </head>
5. <body>
6. <input type="button" value="setCookie" onclick="setCookie()">
7. <input type="button" value="getCookie" onclick="getCookie()">
8.    <script>
9.    function setCookie()
10.   {
11.      document.cookie="username=Duke Martin";
12.   }
13.   function getCookie()
14.   {
15.      if(document.cookie.length!=0)
16.      {
17.      alert(document.cookie);
18.      }
19.      else
20.      {
21.      alert("Cookie not available");
22.      }
23.   }
24.   </script>
25.
26. </body>
27. </html>
```

Example 2

Here, we display the cookie's name-value pair separately.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<input type="button" value="setCookie" onclick="setCookie()">
<input type="button" value="getCookie" onclick="getCookie()">
    <script>
    function setCookie()
    {
        document.cookie="username=Duke Martin";
    }
    function getCookie()
    {
        if(document.cookie.length!=0)
        {
            var array=document.cookie.split("=");
        alert("Name="+array[0]+" "+"Value="+array[1]);
        }
        else
        {
        alert("Cookie not available");
        }
    }
    </script>

</body>
</html>
```

Example 3

In this example, we provide choices of color and pass the selected color value to the cookie. Now, cookie stores the last choice of a user in a browser. So, on reloading the web page, the user's last choice will be shown on the screen.

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.  </head>
5.  <body>
6.      <select id="color" onchange="display()">
7.          <option value="Select Color">Select Color</option>
8.          <option value="yellow">Yellow</option>
9.          <option value="green">Green</option>
10.         <option value="red">Red</option>
11.     </select>
12.     <script type="text/javascript">
13.         function display()
14.         {
15.             var value = document.getElementById("color").value;
16.             if (value != "Select Color")
17.             {
18.                 document.bgColor = value;
19.                 document.cookie = "color=" + value;
20.             }
21.         }
22.         window.onload = function ()
23.         {
24.             if (document.cookie.length != 0)
25.             {
26.                 var array = document.cookie.split("=");
27.                 document.getElementById("color").value = array[1];
28.                 document.bgColor = array[1];
29.             }
30.         }
31.
32.
33.     </script>
34. </body>
35. </html>
```

Cookie Attributes

JavaScript provides some optional attributes that enhance the functionality of cookies. Here, is the list of some attributes with their description.

Attributes	Description
expires	It maintains the state of a cookie up to the specified date and time.
max-age	It maintains the state of a cookie up to the specified time. Here, time is given in seconds.
path	It expands the scope of the cookie to all the pages of a website.
domain	It is used to specify the domain for which the cookie is valid.

Cookie expires attribute

The cookie expires attribute provides one of the ways to create a persistent cookie. Here, a date and time are declared that represents the active period of a cookie. Once the declared time is passed, a cookie is deleted automatically.

Let's see an example of cookie expires attribute.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<input type="button" value="setCookie" onclick="setCookie()">
<input type="button" value="getCookie" onclick="getCookie()">
    <script>
    function setCookie()
    {
        document.cookie="username=Duke Martin;expires=Sun, 20 Aug 2030 12
    :00:00 UTC";
    }
    function getCookie()
    {
        if(document.cookie.length!=0)
```

```
16.       {
17.           var array=document.cookie.split("=");
18.       alert("Name="+array[0]+" "+"Value="+array[1]);
19.       }
20.       else
21.       {
22.       alert("Cookie not available");
23.       }
24.   }
25.    </script>
26. </body>
27. </html>
```

Cookie max-age attribute

The cookie max-age attribute provides another way to create a persistent cookie. Here, time is declared in seconds. A cookie is valid up to the declared time only.

Let's see an example of cookie max-age attribute.

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.  </head>
5.  <body>
6.  <input type="button" value="setCookie" onclick="setCookie()">
7.  <input type="button" value="getCookie" onclick="getCookie()">
8.     <script>
9.     function setCookie()
10.    {
11.       document.cookie="username=Duke Martin;max-
    age=" + (60 * 60 * 24 * 365) + ";"
12.    }
13.    function getCookie()
14.    {
15.       if(document.cookie.length!=0)
16.       {
17.           var array=document.cookie.split("=");
```

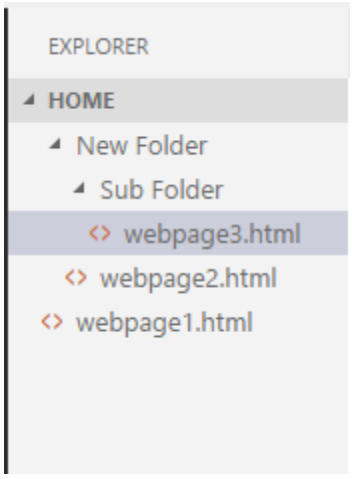
```
18.         alert("Name="+array[0]+" "+"Value="+array[1]);
19.         }
20.         else
21.         {
22.         alert("Cookie not available");
23.         }
24.     }
25.     </script>
26. </body>
27. </html>
```

Cookie path attribute

If a cookie is created for a webpage, by default, it is valid only for the current directory and sub-directory. JavaScript provides a path attribute to expand the scope of cookie up to all the pages of a website.

Cookie path attribute Example

Let's understand the path attribute with the help of an example.



Here, if we create a cookie for webpage2.html, it is valid only for itself and its sub-directory (i.e., webpage3.html). It is not valid for webpage1.html file.

In this example, we use path attribute to enhance the visibility of cookies up to all the pages. Here, you all just need to do is to maintain the above directory structure and put the below program in all three web pages. Now, the cookie is valid for each web page.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<input type="button" value="setCookie" onclick="setCookie()">
<input type="button" value="getCookie" onclick="getCookie()">
    <script>
    function setCookie()
    {
        document.cookie="username=Duke Martin;max-
    age=" + (60 * 60 * 24 * 365) + ";path=/;"
    }
    function getCookie()
    {
        if(document.cookie.length!=0)
        {
            var array=document.cookie.split("=");
        alert("Name="+array[0]+" "+"Value="+array[1]);
        }
        else
        {
        alert("Cookie not available");
        }
    }
    </script>
</body>
</html>
```

Cookie domain attribute

A JavaScript domain attribute specifies the domain for which the cookie is valid. Let's suppose if we provide any domain name to the attribute such like:

```
domain=javatpoint.com
```

Here, the cookie is valid for the given domain and all its sub-domains.

However, if we provide any sub-domain to the attribute such like:

```
1. omain=training.javatpoint.com
```

Here, the cookie is valid only for the given sub-domain. So, it's a better approach to provide domain name instead of sub-domain.

JavaScript Events

The change in the state of an object is known as an **Event**. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called **Event Handling**. Thus, js handles the HTML events via **Event Handlers**.

For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Let's discuss some examples over events and their handlers.

Click Event

1. **<html>**
2. **<head>** Javascript Events **</head>**
3. **<body>**

```
<script language="Javascript" type="text/Javascript">
    <!--
    function clickevent()
    {
        document.write("This is JavaTpoint");
    }
    //-->
</script>
<form>
<input type="button" onclick="clickevent()" value="Who's this?"/>
</form>
</body>
</html>
```

MouseOver Event

```
<html>
<head>
<h1> Javascript Events </h1>
</head>
<body>
<script language="Javascript" type="text/Javascript">
    <!--
    function mouseoverevent()
    {
        alert("This is JavaTpoint");
    }
    //-->
</script>
<p onmouseover="mouseoverevent()"> Keep cursor over me</p>
</body>
</html>
```

Focus Event

```
<html>
<head> Javascript Events</head>
<body>
```

```
<h2> Enter something here</h2>
<input type="text" id="input1" onfocus="focusevent()"/>
<script>
<!--
    function focusevent()
    {
        document.getElementById("input1").style.background=" aqua";
    }
//-->
</script>
</body>
</html>
```

Keydown Event

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onkeydown="keydownevent()"/>
<script>
<!--
    function keydownevent()
    {
        document.getElementById("input1");
        alert("Pressed a key");
    }
//-->
</script>
</body>
</html>
```

Load event

```
<html>
<head>Javascript Events</head>
</br>
<body onload="window.alert('Page successfully loaded');">
```

```
<script>
<!--
document.write("The page is loaded successfully");
//-->
</script>
</body>
</html>
```

JavaScript addEventListener()

The **addEventListener()** method is used to attach an event handler to a particular element. It does not override the existing event handlers. Events are said to be an essential part of the JavaScript. A web page responds according to the event that occurred. Events can be user-generated or generated by API's. An event listener is a JavaScript's procedure that waits for the occurrence of an event.

The addEventListener() method is an inbuilt function of JavaScript. We can add multiple event handlers to a particular element without overwriting the existing event handlers.

Syntax

```
element.addEventListener(event, function, useCapture);
```

Although it has three parameters, the parameters ***event*** and ***function*** are widely used. The third parameter is optional to define. The values of this function are defined as follows.

Parameter Values

event: It is a required parameter. It can be defined as a string that specifies the event's name.

function: It is also a required parameter. It is a JavaScript function which responds to the event occur.

useCapture: It is an optional parameter. It is a Boolean type value that specifies whether the event is executed in the bubbling or capturing phase. Its possible values are **true** and **false**. When it is set to true, the event handler executes in the capturing phase. When it is set to false, the handler executes in the bubbling phase. Its default value is **false**.

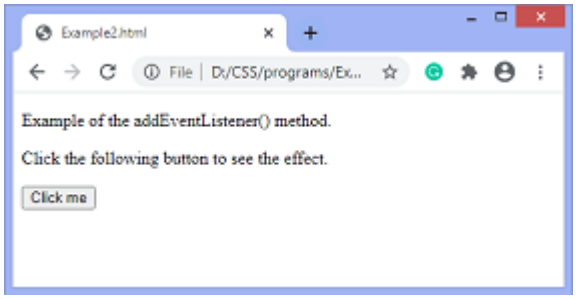
Let's see some of the illustrations of using the addEventListener() method.

Example

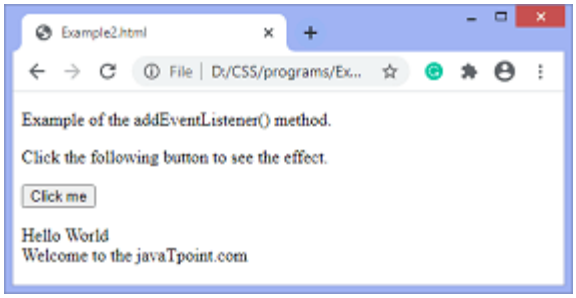
It is a simple example of using the addEventListener() method. We have to click the given HTML button to see the effect.

```
<!DOCTYPE html>
<html>
<body>
<p> Example of the addEventListener() method. </p>
<p> Click the following button to see the effect. </p>
<button id = "btn"> Click me </button>
<p id = "para"></p>
<script>
document.getElementById("btn").addEventListener("click", fun);
function fun() {
document.getElementById("para").innerHTML = "Hello World" + "<br>" + "W
elcome to the  javaTpoint.com";
}
</script>
</body>
</html>
```

Output



After clicking the given HTML button, the output will be -



Now, in the next example we will see how to add many events to the same element without overwriting the existing events.

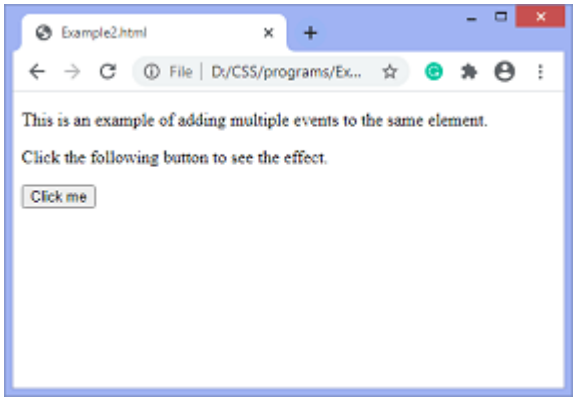
Example

In this example, we are adding multiple events to the same element.

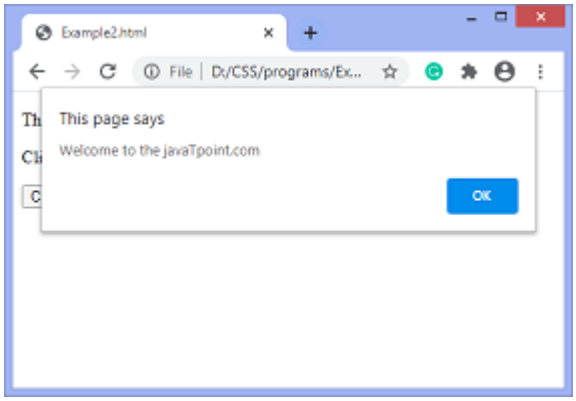
```
<!DOCTYPE html>
<html>
<body>
<p> This is an example of adding multiple events to the same element. </p>
<p> Click the following button to see the effect. </p>
<button id = "btn"> Click me </button>
<p id = "para"></p>
<p id = "para1"></p>
<script>
function fun() {
    alert("Welcome to the javaTpoint.com");
}

function fun1() {
   document.getElementById("para").innerHTML =  "This is second function";

}
function fun2() {
   document.getElementById("para1").innerHTML =  "This is third function";
}
var mybtn = document.getElementById("btn");
mybtn.addEventListener("click", fun);
mybtn.addEventListener("click", fun1);
mybtn.addEventListener("click", fun2);
```

25. </script>
26. </body>
27. </html>

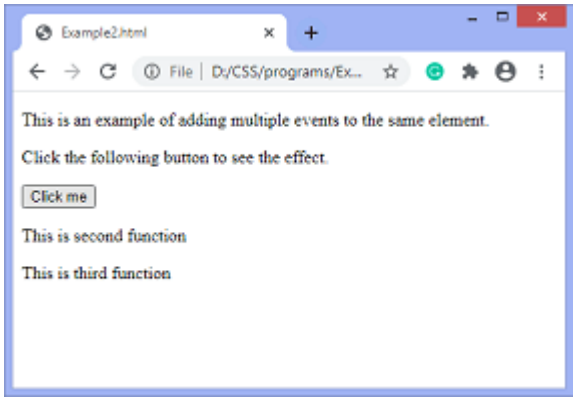
Output



Now, when we click the button, an alert will be displayed. After clicking the given HTML button, the output will be -



When we exit the alert, the output is -

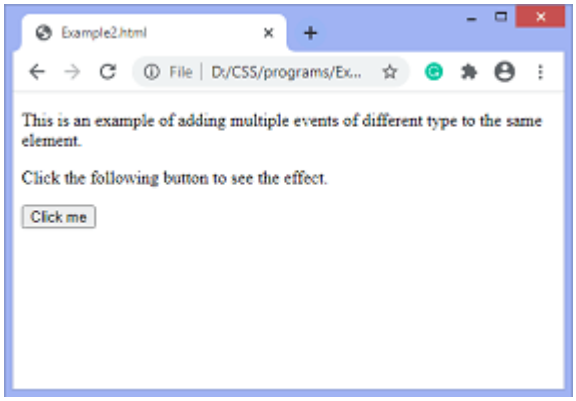


Example

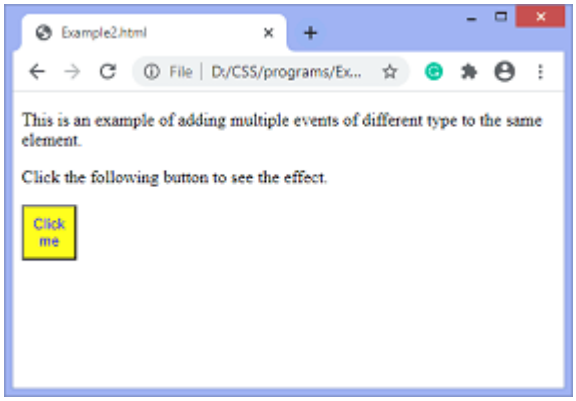
In this example, we are adding multiple events of a different type to the same element.

```
<!DOCTYPE html>
<html>
<body>
<p> This is an example of adding multiple events of different type to the same eleme
nt. </p>
<p> Click the following button to see the effect. </p>
<button id = "btn"> Click me </button>
<p id = "para"></p>
<script>
function fun() {
    btn.style.width = "50px";
    btn.style.height = "50px";
    btn.style.background = "yellow";
    btn.style.color = "blue";
}

function fun1() {
   document.getElementById("para").innerHTML =  "This is second function";

}
function fun2() {
    btn.style.width = "";
    btn.style.height = "";
    btn.style.background = "";
    btn.style.color = "";
}
var mybtn = document.getElementById("btn");
mybtn.addEventListener("mouseover", fun);
mybtn.addEventListener("click", fun1);
mybtn.addEventListener("mouseout", fun2);
</script>
</body>
</html>
```

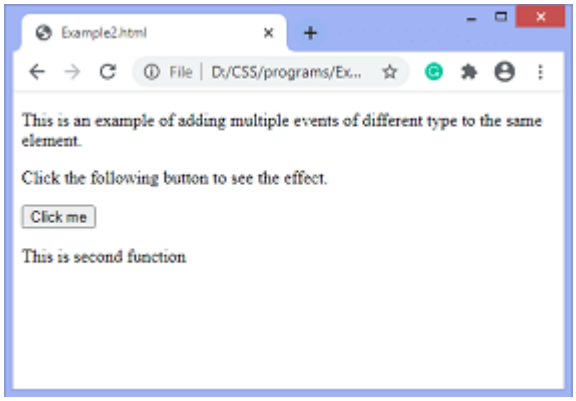
Output



When we move the cursor over the button, the output will be -



After clicking the button and leave the cursor, the output will be -



Event Bubbling or Event Capturing

Now, we understand the use of the third parameter of JavaScript's addEventListener(), i.e., ***useCapture.***

In HTML DOM, **Bubbling** and **Capturing** are the two ways of event propagation. We can understand these ways by taking an example.

Suppose we have a div element and a paragraph element inside it, and we are applying the **"click"** event to both of them using the **addEventListener()** method. Now the question is on clicking the paragraph element, which element's click event is handled first.

So, in **Bubbling,** the event of paragraph element is handled first, and then the div element's event is handled. It means that in bubbling, the inner element's event is handled first, and then the outermost element's event will be handled.

In **Capturing** the event of div element is handled first, and then the paragraph element's event is handled. It means that in capturing the outer element's event is handled first, and then the innermost element's event will be handled.

```
addEventListener(event, function, useCapture);
```

We can specify the propagation using the ***useCapture*** parameter. When it is set to false (which is its default value), then the event uses bubbling propagation, and when it is set to true, there is the capturing propagation.

We can understand the *bubbling* and *capturing* using an illustration.

Example

In this example, there are two div elements. We can see the bubbling effect on the first div element and the capturing effect on the second div element.

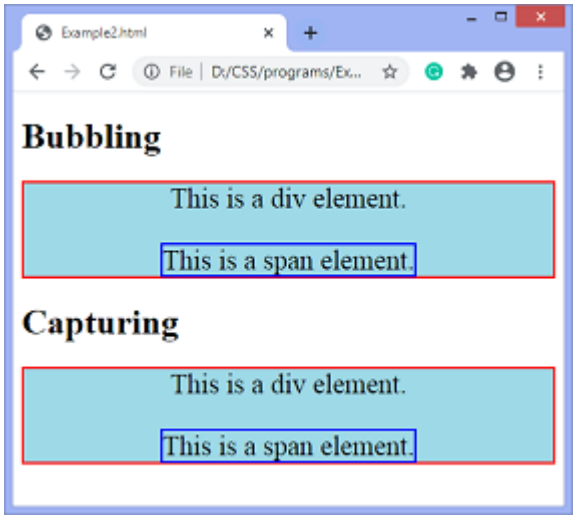
When we double click the span element of the first div element, then the span element's event is handled first than the div element. It is called *bubbling*.

But when we double click the span element of the second div element, then the div element's event is handled first than the span element. It is called *capturing*.

```
<!DOCTYPE html>
<html>
<head>
<style>
div{
background-color: lightblue;
border: 2px solid red;
font-size: 25px;
text-align: center;
}
```

```
span{
border: 2px solid blue;
}
</style>
</head>
<body>
<h1> Bubbling </h1>
<div id = "d1">
This is a div element.
<br><br>
<span id = "s1"> This is a span element. </span>
</div>
<h1> Capturing </h1>
<div id = "d2"> This is a div element.
<br><br>
<span id = "s2"> This is a span element. </span>
</div>

<script>
document.getElementById("d1").addEventListener("dblclick", function() {alert('You have double clicked on div element')}, false);
document.getElementById("s1").addEventListener("dblclick", function() {alert('You have double clicked on span element')}, false);
document.getElementById("d2").addEventListener("dblclick", function() {alert('You have double clicked on div element')}, true);
document.getElementById("s2").addEventListener("dblclick", function() {alert('You have double clicked on span element')}, true);
</script>
</body>
</html>
```

Output



JavaScript onclick event

The **onclick** event generally occurs when the user clicks on an element. It allows the programmer to execute a JavaScript's function when an element gets clicked. This event can be used for validating a form, warning messages and many more.

Using JavaScript, this event can be dynamically added to any element. It supports all HTML elements except **<html>**, **<head>**, **<title>**, **<style>**, **<script>**, **<base>**, **<iframe>**, **<bdo>**, **
, **<meta>, and **<param>**. It means we cannot apply the **onclick** event on the given tags.

In HTML, we can use the **onclick** attribute and assign a JavaScript function to it. We can also use the JavaScript's **addEventListener()** method and pass a **click** event to it for greater flexibility.

Syntax

Now, we see the syntax of using the **onclick** event in HTML and in javascript (without **addEventListener()** method or by using the **addEventListener()** method).

In HTML

```
<element onclick = "fun()">
```

In JavaScript

```
object.onclick = function() { myScript };
```

In JavaScript by using the addEventListener() method

1. object.addEventListener("click", myScript);

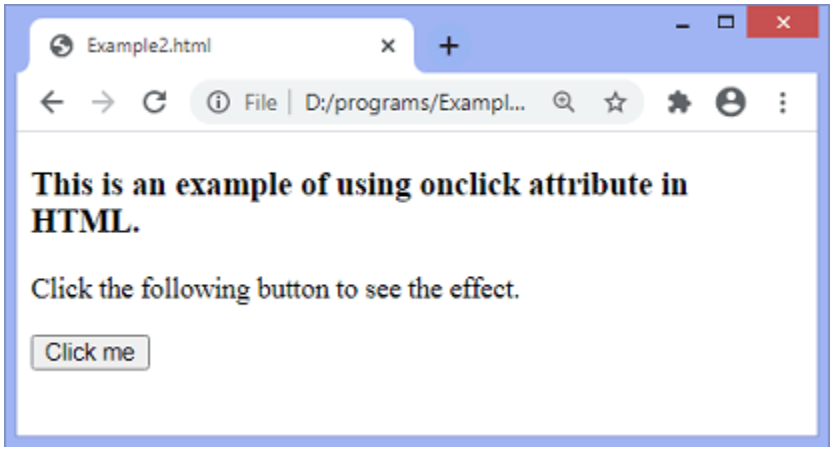
Let's see how to use **onclick** event by using some illustrations. Now, we will see the examples of using the **onclick** event in HTML, and in JavaScript.

Example1 - Using onclick attribute in HTML

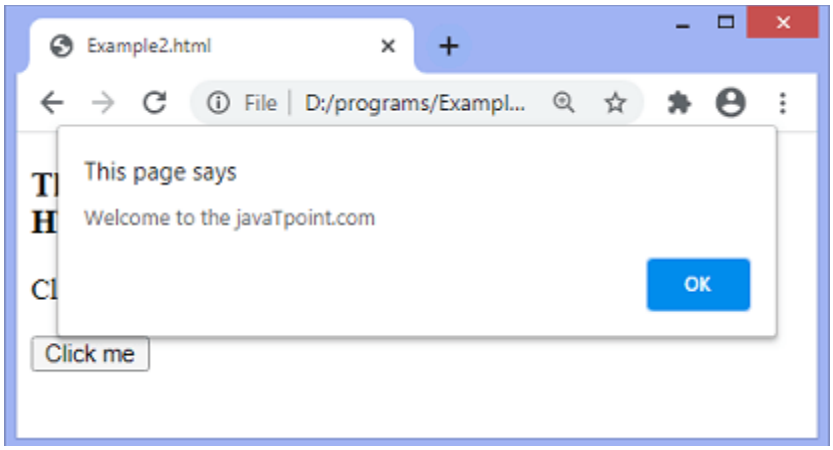
In this example, we are using the HTML **onclick** attribute and assigning a JavaScript's function to it. When the user clicks the given button, the corresponding function will get executed, and an alert dialog box will be displayed on the screen.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <script>
5. function fun() {
6. alert("Welcome to the javaTpoint.com");
7. }
8. </script>
9. </head>
10. <body>
11. <h3> This is an example of using onclick attribute in HTML. </h3>
12. <p> Click the following button to see the effect. </p>
13. <button onclick = "fun()">Click me</button>
14. </body>
15. </html>
```

Output



After clicking the given button, the output will be -



Example2 - Using JavaScript

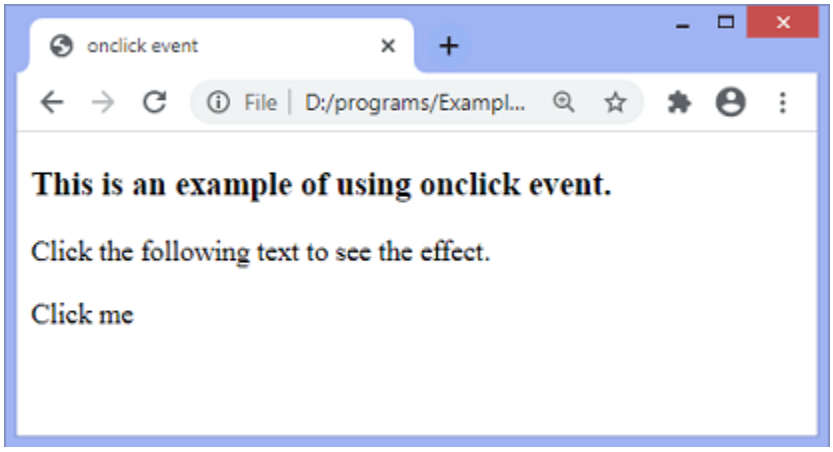
In this example, we are using JavaScript's **onclick** event. Here we are using the **onclick** event with the paragraph element.

When the user clicks on the paragraph element, the corresponding function will get executed, and the text of the paragraph gets changed. On clicking the **<p>** element, the background color, size, border, and color of the text will also get change.

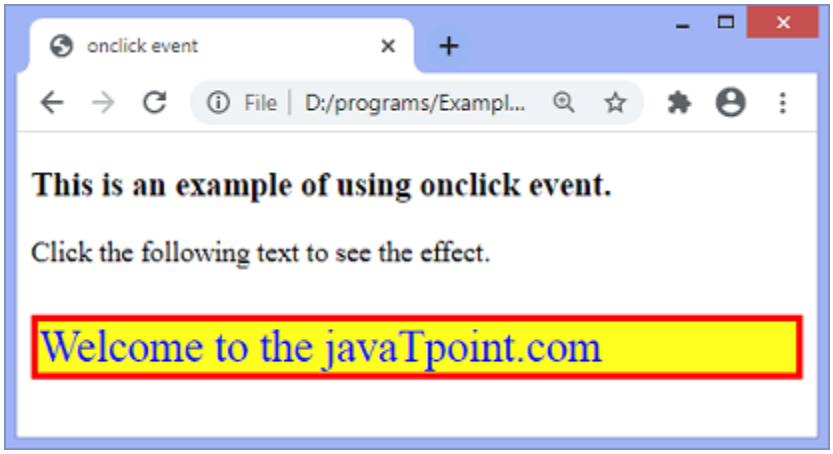
```
<!DOCTYPE html>
<html>
<head>
<title> onclick event </title>
</head>
<body>
<h3> This is an example of using onclick event. </h3>
<p> Click the following text to see the effect. </p>
<p id = "para">Click me</p>
<script>
document.getElementById("para").onclick = function() {
fun()
};
function fun() {
document.getElementById("para").innerHTML = "Welcome to the javaTpoint.com";
document.getElementById("para").style.color = "blue";
document.getElementById("para").style.backgroundColor = "yellow";
```

```
18. document.getElementById("para").style.fontSize = "25px";
19. document.getElementById("para").style.border = "4px solid red";
20. }
21. </script>
22.
23. </body>
24. </html>
```

Output



After clicking the text ***Click me,*** the output will be -



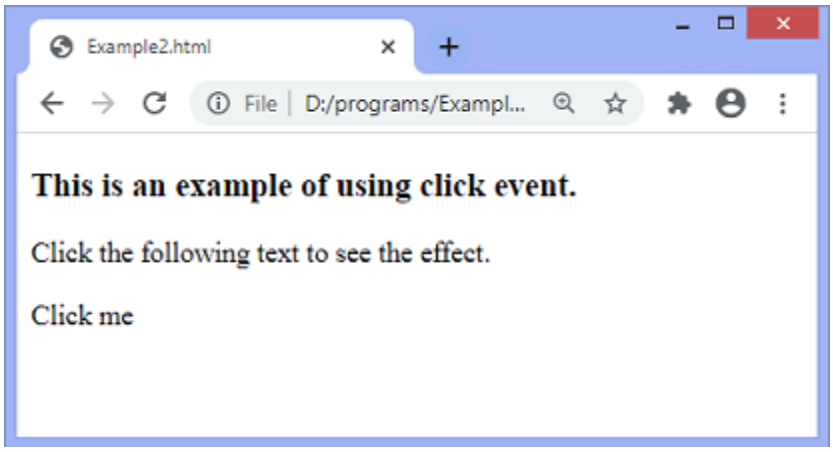
Example3 - Using addEventListener() method

In this example, we are using JavaScript's **addEventListener()** method to attach a **click** event to the paragraph element. When the user clicks the paragraph element, the text of the paragraph gets changed.

On clicking the paragraph, the background color and font-size of elements will also change.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. </head>
5. <body>
6. <h3> This is an example of using click event. </h3>
7. <p> Click the following text to see the effect. </p>
8. <p id = "para">Click me</p>
9. <script>
10. document.getElementById("para").onclick = function() {
11. fun()
12. };
13. function fun() {
14. document.getElementById("para").innerHTML = "Welcome to the javaTpoint.com";
15. document.getElementsByTagName("body")[0].style.color = "blue";
16. document.getElementsByTagName("body")[0].style.backgroundColor = "lightgreen";
17. document.getElementsByTagName("body")[0].style.fontSize = "25px";
18. document.getElementById("para").style.border = "4px solid red";
19. }
20. </script>
21.
22. </body>
23. </html>
```

Output



On clicking the text ***Click me***, the output will be -

JavaScript dblclick event

The **dblclick** event generates an event on double click the element. The event fires when an element is clicked twice in a very short span of time. We can also use the JavaScript's **addEventListener()** method to fire the double click event.

In HTML, we can use the **ondblclick** attribute to create a double click event.

Syntax

Now, we see the syntax of creating double click event in HTML and in javascript (without using **addEventListener()** method or by using the **addEventListener()** method).

In HTML

```
<element ondblclick = "fun()">
```

In JavaScript

```
object.ondblclick = function() { myScript };
```

In JavaScript by using the addEventListener() method

```
object.addEventListener("dblclick", myScript);
```

Let's see some of the illustrations to understand the double click event.

Example - Using ondblclick attribute in HTML

In this example, we are creating the double click event using the HTML **ondblclick** attribute.

```
<!DOCTYPE html>
<html>
<head>
</head>

<body>
<h1 id = "heading" ondblclick = "fun()"> Hello world :):) </h1>
<h2> Double Click the text "Hello world" to see the effect. </h2>
<p> This is an example of using the <b> ondblclick </b> attribute. </p>
<script>
function fun() {
document.getElementById("heading").innerHTML = " Welcome to the javaTpoint.com ";
}
</script>
</body>
</html>
```

Output

After the execution of the above code, the output will be -

After double-clicking the text **"Hello world"**, the output will be -

Now, we will see how to create double click event using JavaScript.

Example - Using JavaScript

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. </head>
5.
6. <body>
7. <h1 id = "heading"> Hello world :):) </h1>
8. <h2> Double Click the text "Hello world" to see the effect. </h2>
```

```
9.  <p> This is an example of creating the double click event using JavaScript. </
    p>
10. <script>
11. document.getElementById("heading").ondblclick = function() { fun() };
12. function fun() {
13. document.getElementById("heading").innerHTML = " Welcome to the javaTpoi
    nt.com ";
14. }
15. </script>
16. </body>
17.
18. </html>
```

Output

After double-clicking the text **"Hello world"**, the output will be -

Example - Using JavaScript's addEventListener() method

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. </head>
5. 
6. <body>
7. <h1 id = "heading"> Hello world :):) </h1>
8. <h2> Double Click the text "Hello world" to see the effect. </h2>
9. <p> This is an example of creating the double click event using the <b> addE
   ventListener() method </b>. </p>
10. <script>
11. document.getElementById("heading").addEventListener("dblclick", fun);
12. function fun() {
13. document.getElementById("heading").innerHTML = " Welcome to the javaTpoi
    nt.com ";
14. }
15. </script>
16. </body>
17. 
18. </html>
```

Output

After double-clicking the text **"Hello world"**, the output will be -

JavaScript onload

In JavaScript, this event can apply to launch a particular function when the page is fully displayed. It can also be used to verify the type and version of the visitor's browser. We can check what cookies a page uses by using the **onload** attribute.

In HTML, the onload attribute fires when an object has been loaded. The purpose of this attribute is to execute a script when the associated element loads.

In HTML, the **onload** attribute is generally used with the **<body>** element to execute a script once the content (including CSS files, images, scripts, etc.) of the webpage is completely loaded. It is not necessary to use it only with <body> tag, as it can be used with other HTML elements.

The difference between the **document.onload** and **window.onload** is: **document.onload** triggers before the loading of images and other external content. It is fired before the **window.onload**. While the **window.onload** triggers when the entire page loads, including CSS files, script files, images, etc.

Syntax

```
1. window.onload = fun()
```

Let's understand this event by using some examples.

Example1

In this example, there is a div element with a height of 200px and a width of 200px. Here, we are using the **window.onload()** to change the background color, width, and height of the **div** element after loading the web page.

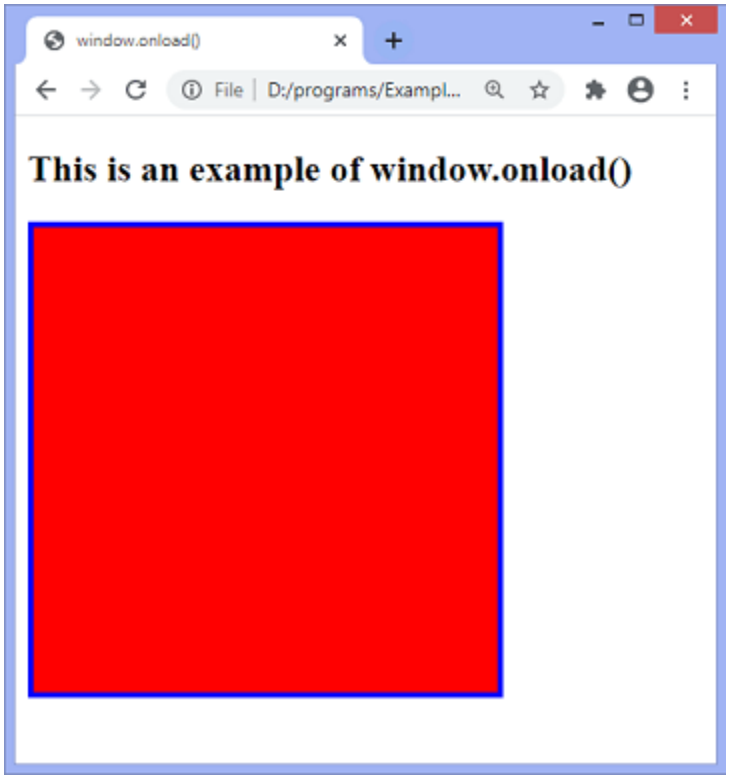
The background color is set to **'red'**, and width and height are set to **300px** each.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset = " utf-8">
5. <title> window.onload() </title>
6. <style type = "text/css">
7. #bg{
8. width: 200px;
9. height: 200px;
10. border: 4px solid blue;
11. }
12. </style>
13. <script type = "text/javascript">
14. window.onload = function(){
15. document.getElementById("bg").style.backgroundColor = "red";
16. document.getElementById("bg").style.width = "300px";
17. document.getElementById("bg").style.height = "300px";
18. }
19. </script>
20. </head>
```

```
21. <body>
22. <h2> This is an example of window.onload() </h2>
23. <div id = "bg"></div>
24. </body>
25. </html>
```

Output

After the execution of the code and loading of the page, the output will be -



Example2

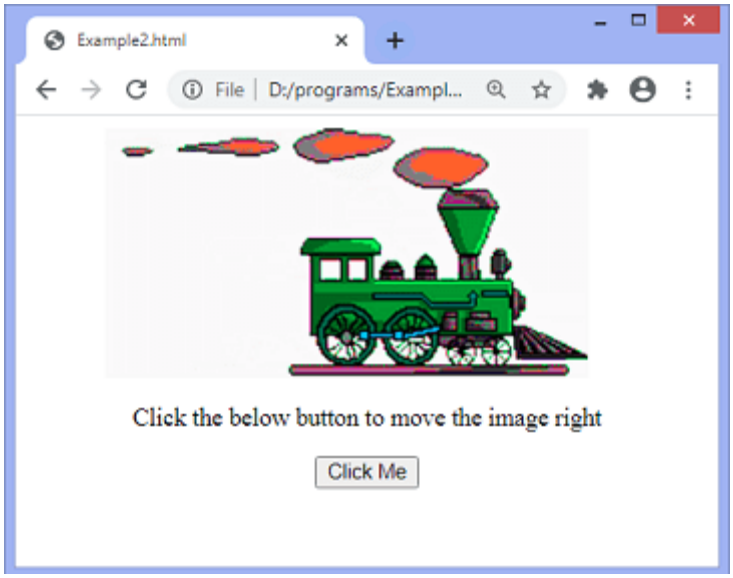
In this example, we are implementing a simple animation by using the properties of the DOM object and functions of javascript. We use the JavaScript function getElementById() for getting the DOM object and then assign that object into a global variable.

```
1. <html>
2.   <head>
3.     <script type = "text/javascript">
4.
5.         var img = null;
6.         function init(){
```

```
7.              img = document.getElementById('myimg');
8.              img.style.position = 'relative';
9.              img.style.left = '50px';
10.          }
11.          function moveRight(){
12.             img.style.left = parseInt(
13.             img.style.left) + 100 + 'px';
14.          }
15.          window.onload = init;
16.
17.     </script>
18.  </head>
19.
20.  <body>
21.     <form>
22.        <img id = "myimg" src = "train1.png" />
23.        <center>
24.           <p>Click the below button to move the image right</p>
25.        <input type = "button" value = "Click Me" onclick = "moveRight();" />

26.     </center>
27.     </form>
28.  </body>
29.
30. </html>
```

Output

After the successful execution of the above code, the output will be -



Now, there is an example in which we will use the HTML **onload** attribute and the JavaScript functions.

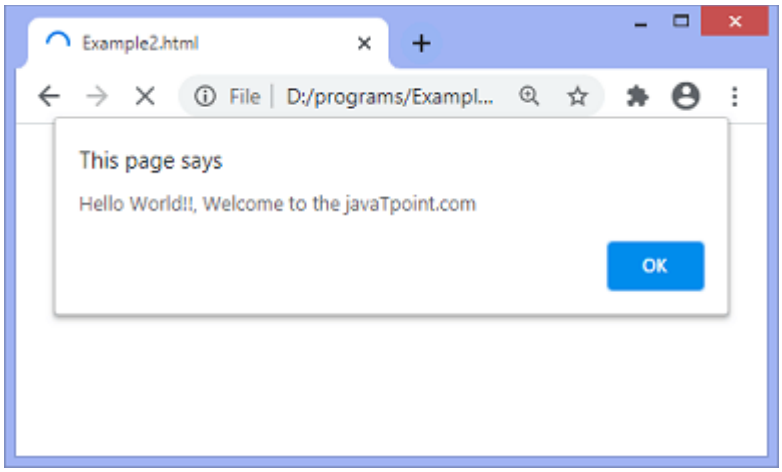
Example3

It is a simple example of using the HTML **onload** attribute with the function defined in JavaScript. In this example, the **alert()** function gets called whenever the document refresh.

```
<!DOCTYPE html>
<html>
<head>
<style>
</style>
<script>
   function fun() {
      alert("Hello World!!, Welcome to the javaTpoint.com");
   }
</script>
</head>
<body onload = "fun()">
<h1> Example of the HTML onload attribute </h1>
<p> Try to refresh the document to see the effect. </p>
</body>
</html>
```

Output

After the execution of the above code, the output will be -



JavaScript onresize event

The **onresize** event in JavaScript generally occurs when the window has been resized. To get the size of the window, we can use the JavaScript's ***window.outerWidth*** and ***window.outerHeight*** events. We can also use the JavaScript's properties such as ***innerWidth, innerHeight, clientWidth, ClientHeight, offsetWidth, offsetHeight*** to get the size of an element.

In HTML, we can use the **onresize** attribute and assign a JavaScript function to it. We can also use the JavaScript's **addEventListener()** method and pass a **resize** event to it for greater flexibility.

Syntax

Now, we see the syntax of using the **onresize** event in HTML and in javascript (without **addEventListener()** method or by using the **addEventListener()** method).

In HTML

```
<element onresize = "fun()">
```

In JavaScript

```
object.onresize = function() { myScript };
```

In JavaScript by using the addEventListener() method

```
1. object.addEventListener("resize", myScript);
```

Let's see some of the illustrations to understand the **onresize** event.

Example

In this example, we are using the HTML **onresize** attribute. Here, we are using the ***window.outerWidth*** and ***window.outerHeight*** events of JavaScript to get the height and width of the window.

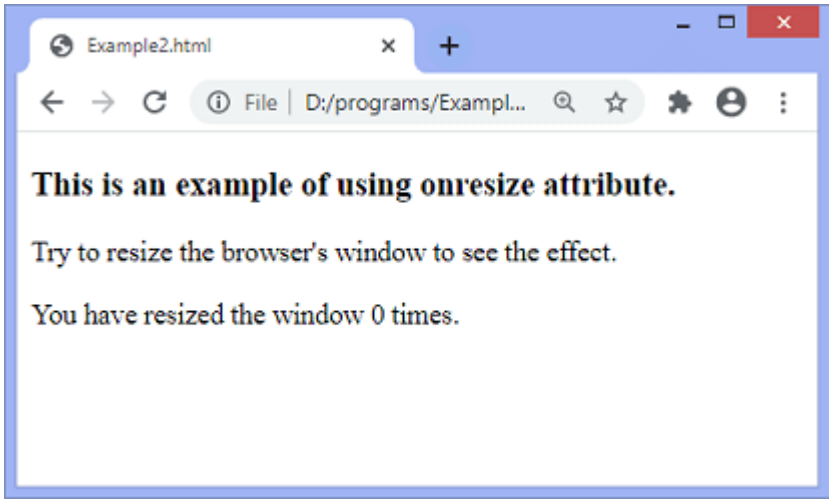
When the user resizes the window, the updated width and height of the window will be displayed on the screen. It will also display how many times the user tried to resize the window. When we change the height of the window, the updated height will change accordingly. Similarly, when we change the width of the window, the updated width will change accordingly.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <script>
5. var i = 0;
6.
7. function fun() {
8. var res = "Width = " + window.outerWidth + "<br>" + "Height = " + window.
   outerHeight;
9. document.getElementById("para").innerHTML = res;
10.
11. var res1 = i += 1;
12. document.getElementById("s1").innerHTML = res1;
13. }
14. </script>
15. </head>
16. <body onresize = "fun()">
17. <h3> This is an example of using onresize attribute. </h3>
18. <p> Try to resize the browser's window to see the effect. </p>
19.
20. <p id = "para"> </p>
21. <p> You have resized the window <span id = "s1"> 0 </span> times.</p>
22. </body>
```

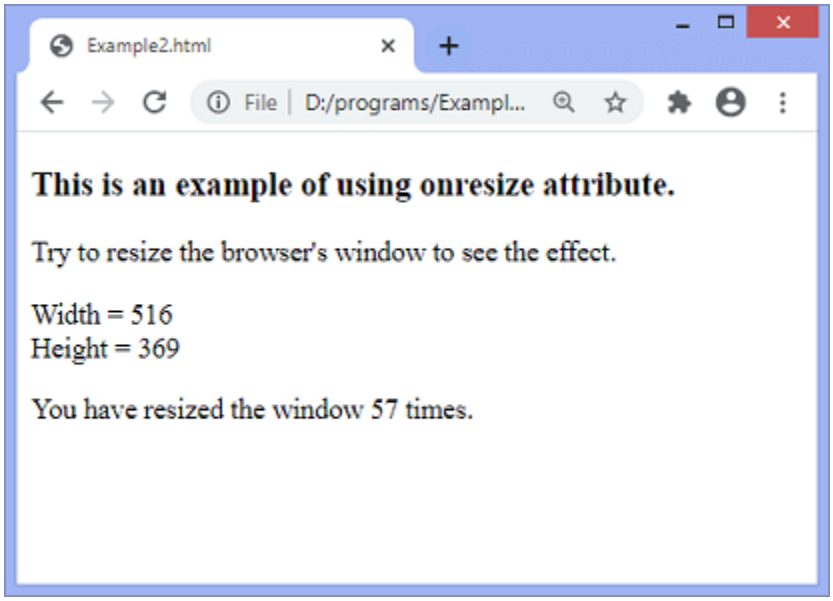
23. </html>

Output

After the execution of the above code, the output will be -



When we try to resize the window, the output will be -



Example - Using JavaScript

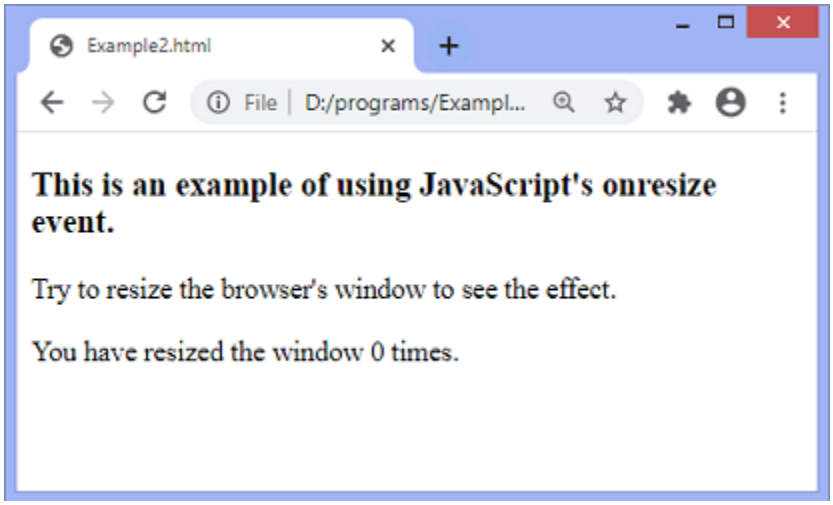
In this example, we are using JavaScript's **onresize** event.

1. <!DOCTYPE html>
2. **<html>**
3. **<head>**
4. **</head>**

```
<body>
<h3> This is an example of using JavaScript's onresize event. </h3>
<p> Try to resize the browser's window to see the effect. </p>

<p id = "para"> </p>
<p> You have resized the window <span id = "s1"> 0 </span> times.</p>
<script>
document.getElementsByTagName("BODY")[0].onresize = function() {fun()};
var i = 0;

function fun() {
var res = "Width = " + window.outerWidth + "<br>" + "Height = " + window.
outerHeight;
document.getElementById("para").innerHTML = res;

var res1 = i += 1;
document.getElementById("s1").innerHTML = res1;
}
</script>
</body>
</html>
```

Output

After the execution of the above code, the output will be -



When we try to resize the window, the output will be -

Example - Using addEventListener() method

In this example, we are using JavaScript's **addEventListener()** method.

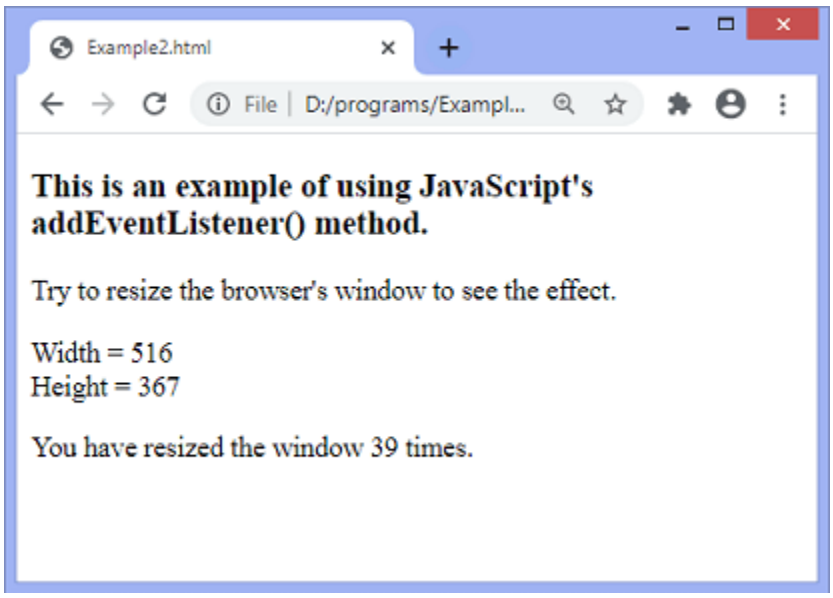
```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. </head>
5. <body>
6. <h3> This is an example of using JavaScript's addEventListener() method. </h
   3>
7. <p> Try to resize the browser's window to see the effect. </p>
8.
9. <p id = "para"> </p>
10. <p> You have resized the window <span id = "s1"> 0 </span> times.</p>
11. <script>
12. window.addEventListener("resize", fun);
13. var i = 0;
14.
15. function fun() {
16. var res = "Width = " + window.outerWidth + "<br>" + "Height = " + window.
    outerHeight;
17. document.getElementById("para").innerHTML = res;
18.
19. var res1 = i += 1;
```

```
20. document.getElementById("s1").innerHTML = res1;
21. }
22. </script>
23. </body>
24. </html>
```

Output

After the execution of the above code, the output will be -

When we try to resize the window, the output will be -



Exception Handling in JavaScript

An exception signifies the presence of an abnormal condition which requires special operable techniques. In programming terms, an exception is the anomalous code that breaks the normal flow of the code. Such exceptions require specialized programming constructs for its execution.

What is Exception Handling

In programming, exception handling is a process or method used for handling the abnormal statements in the code and executing them. It also enables to handle the flow control of the code/program. For handling the code, various handlers are used that process the exception and execute the code. **For example**, the Division of a non-zero value with zero will result into infinity always, and it is an exception. Thus, with the help of exception handling, it can be executed and handled.

In exception handling:

A throw statement is used to raise an exception. It means when an abnormal condition occurs, an exception is thrown using throw.

The thrown exception is handled by wrapping the code into the try...catch block. If an error is present, the catch block will execute, else only the try block statements will get executed.

Thus, in a programming language, there can be different types of errors which may disturb the proper execution of the program.

Types of Errors

While coding, there can be three types of errors in the code:

1. **Syntax Error:** When a user makes a mistake in the pre-defined syntax of a programming language, a syntax error may appear.
2. **Runtime Error:** When an error occurs during the execution of the program, such an error is known as Runtime error. The codes which create runtime errors are known as Exceptions. Thus, exception handlers are used for handling runtime errors.
3. **Logical Error:** An error which occurs when there is any logical mistake in the program that may not produce the desired output, and may terminate abnormally. Such an error is known as Logical error.

Error Object

When a runtime error occurs, it creates and throws an Error object. Such an object can be used as a base for the user-defined exceptions too. An error object has two properties:

1. name: This is an object property that sets or returns an error name.
2. **message:** This property returns an error message in the string form.

Although Error is a generic constructor, there are following standard built-in error types or error constructors beside it:

1. **EvalError:** It creates an instance for the error that occurred in the eval(), which is a global function used for evaluating the js string code.
2. **InternalError:** It creates an instance when the js engine throws an internal error.
3. **RangeError:** It creates an instance for the error that occurs when a numeric variable or parameter is out of its valid range.
4. **ReferenceError:** It creates an instance for the error that occurs when an invalid reference is de-referenced.
5. **SyntaxError:** An instance is created for the syntax error that may occur while parsing the eval().
6. **TypeError:** When a variable is not a valid type, an instance is created for such an error.
7. **URIError:** An instance is created for the error that occurs when invalid parameters are passed in **encodeURI()** or **decodeURI()**.

Exception Handling Statements

There are following statements that handle if any exception occurs:

- throw statements
- try...catch statements
- try...catch...finally statements.

JavaScript try…catch

A try...catch is a commonly used statement in various programming languages. Basically, it is used to handle the error-prone part of the code. It initially tests the code for all possible errors it may contain, then it implements actions to tackle those errors (if occur). A good programming approach is to keep the complex code within the try...catch statements.

Let's discuss each block of statement individually:

try{} statement: Here, the code which needs possible error testing is kept within the try block. In case any error occur, it passes to the catch{} block for taking suitable actions and handle the error. Otherwise, it executes the code written within.

catch{} statement: This block handles the error of the code by executing the set of statements written within the block. This block contains either the user-defined exception handler or the built-in handler. This block executes only when any error-prone code needs to be handled in the try block. Otherwise, the catch block is skipped.

Syntax:

```
1. try{
2. expression; } //code to be written.
3. catch(error){
4. expression; } // code for handling the error.
```

try…catch example

```
1. <html>
2. <head> Exception Handling</br></head>
3. <body>
4. <script>
5. try{
6. var a= ["34","32","5","31","24","44","67"]; //a is an array
7. document.write(a);    // displays elements of a
8. document.write(b); //b is undefined but still trying to fetch its value. Thus catch block
   will be invoked
9. }catch(e){
```

```
10. alert("There is error which shows "+e.message); //Handling error
11. }
12. </script>
13. </body>
14. </html>
```

Throw Statement

Throw statements are used for throwing user-defined errors. User can define and throw their own custom errors. When throw statement is executed, the statements present after it will not execute. The control will directly pass to the catch block.

Syntax:

```
1. throw exception;
```

try…catch…throw syntax

```
1. try{
2. throw exception; // user can define their own exception
3. }
4. catch(error){
5. expression; }  // code for handling exception.
```

The exception can be a string, number, object, or boolean value.

throw example with try…catch

```
1. <html>
2. <head>Exception Handling</head>
3. <body>
4. <script>
5. try {
6.    throw new Error('This is the throw keyword'); //user-defined throw statement.
7. }
8. catch (e) {
9.   document.write(e.message); // This will generate an error message
10. }
11. </script>
```

```
12. </body>
13. </html>
```

With the help of throw statement, users can create their own errors.

try…catch…finally statements

Finally is an optional block of statements which is executed after the execution of try and catch statements. Finally block does not hold for the exception to be thrown. Any exception is thrown or not, finally block code, if present, will definitely execute. It does not care for the output too.

Syntax:

```
1. try{
2. expression;
3. }
4. catch(error){
5. expression;
6. }
7. finally{
8. expression; } //Executable code
```

try…catch…finally example

```
1. <html>
2. <head>Exception Handling</head>
3. <body>
4. <script>
5. try{
6. var a=2;
7. if(a==2)
8. document.write("ok");
9. }
10. catch(Error){
11. document.write("Error found"+e.message);
12. }
13. finally{
14. document.write("Value of a is 2 ");
```

```
15. }
16. </script>
17. </body>
18. </html>
```

JavaScript this keyword

The this keyword is a reference variable that refers to the current object. Here, we will learn about this keyword with help of different examples.

JavaScript this Keyword Example

Let's see a simple example of this keyword.

```
1.  <script>
2.  var address=
3.  {
4.  company:"Javatpoint",
5.  city:"Noida",
6.  state:"UP",
7.  fullAddress:function()
8.  {
9.  return this.company+" "+this.city+" "+this.state;
10. }
11. };
12.
13.
14. var fetch=address.fullAddress();
15. document.writeln(fetch);
16.
17. </script>
```

Output:

```
Javatpoint Noida UP
```

The following ways can be used to know which object is referred by this keyword.

Global Context

In global context, variables are declared outside the function. Here, this keyword refers to the window object.

```
<script>
var website="Javatpoint";
function web()
{
document.write(this.website);
}
web();
</script>
```

The call() and apply() method

The call() and apply() method allows us to write a method that can be used on different objects.

```
<script>
var emp_address = {
    fullAddress: function() {
        return this.company + " " + this.city+" "+this.state;
    }
}
var address = {
    company:"Javatpoint",
    city:"Noida",
    state:"UP",

}

document.writeln(emp_address.fullAddress.call(address));
document.writeln(emp_address.fullAddress.apply(address));</script>
```

The bind() Method

The bind() method was introduced in **ECMAScript 5**. It creates a new function whose this keyword refers to the provided value, with a given sequence of arguments.

```
<script>
```

```
2. var lang="Java";
3.
4. function lang_name(call)
5. {
6.
7.     call();
8. };
9.
10. var obj={
11.
12.  lang:"JavaScript",
13.  language:function()
14.  {
15.    document.writeln(this.lang+ " is a popular programming language.");
16.  }
17. };
18. lang_name(obj.language);
19. lang_name(obj.language.bind(obj));
20. </script>
```