Example 1:

<html>

<body>

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

<!--

document.write("Hello World!")

//-->

</script>

</body>

</html>

# What is JavaScript ?

Javascript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

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

# Advantages of JavaScript

The merits of using JavaScript are −

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

# Limitations of JavaScript

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

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

**Syntax**

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

You can place the **<script>** tags, containing your JavaScript, anywhere within you web page, but it is normally recommended that you should keep it within the **<head>** tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

<script ...>

JavaScript code

</script>

The script tag takes two important attributes −

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

So your JavaScript segment will look like −

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

JavaScript code

</script>

**Semicolons are Optional**

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

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

<!--

var1 = 10

var2 = 20

//-->

</script>

But when formatted in a single line as follows, you must use semicolons −

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

<!--

var1 = 10; var2 = 20;

//-->

</script>

**JavaScript is a case-sensitive language.**

**Comments in JavaScript**

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

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

**Example**

The following example shows how to use comments in JavaScript.

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

<!--

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

/\*

\* This is a multiline comment in JavaScript

\* It is very similar to comments in C Programming

\*/

//-->

</script>

The most preferred ways to include JavaScript in an HTML file is Script in <head>...</head> section.

<html>

<head>

<script type="text/javascript">

<!--

function sayHello() {

alert("Hello World")

}

//-->

</script>

</head>

<body>

<input type="button" onclick="sayHello()" value="Say Hello" />

</body>

</html>

# JavaScript in External File

<html>

<head>

<script type="text/javascript" src="filename.js" ></script>

</head>

<body>

.......

</body>

</html>

you can keep the following content in **filename.js** file and then you can use **sayHello** function in your HTML file after including the filename.js file.

function sayHello() {

alert("Hello World")

}

# JavaScript Variables

<script type="text/javascript">

<!--

var money;

var name;

//-->

</script>

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

<script type="text/javascript">

<!--

var money, name;

//-->

</script>

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

# JavaScript Variable Scope

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

<html>

<body onload = checkscope();>

<script type = "text/javascript">

<!--

var myVar = "global"; // Declare a global variable

function checkscope( ) {

var myVar = "local"; // Declare a local variable

document.write(myVar);

}

//-->

</script>

</body>

</html>

# Variable Names

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

**Reserved Words**

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

# Operators

## Arithmetic Operators

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

|  |  |
| --- | --- |
| Sr.No | Operator and Description |
| 1 | + (Addition)  Adds two operands  Ex: A + B will give 30 |
| 2 | - (Subtraction)  Subtracts the second operand from the first  Ex: A - B will give -10 |
| 3 | \* (Multiplication)  Multiply both operands  Ex: A \* B will give 200 |
| 4 | / (Division)  Divide the numerator by the denominator  Ex: B / A will give 2 |
| 5 | % (Modulus)  Outputs the remainder of an integer division  Ex: B % A will give 0 |
| 6 | ++ (Increment)  Increases an integer value by one  Ex: A++ will give 11 |
| 7 | -- (Decrement)  Decreases an integer value by one  Ex: A-- will give 9 |

## Comparison Operators

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

|  |  |
| --- | --- |
| Sr.No | Operator and Description |
| 1 | = = (Equal)  Checks if the value of two operands are equal or not, if yes, then the condition becomes true.  Ex: (A == B) is not true. |
| 2 | != (Not Equal)  Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true.  Ex: (A != B) is true. |
| 3 | > (Greater than)  Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true.  Ex: (A > B) is not true. |
| 4 | < (Less than)  Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true.  Ex: (A < B) is true. |
| 5 | >= (Greater than or Equal to)  Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.  Ex: (A >= B) is not true. |
| 6 | <= (Less than or Equal to)  Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.  Ex: (A <= B) is true. |

## typeof Operator

The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

Here is a list of the return values for the **typeof** Operator.

|  |  |
| --- | --- |
| **Type** | **String Returned by typeof** |
| Number | "number" |
| String | "string" |
| Boolean | "boolean" |
| Object | "object" |
| Function | "function" |
| Undefined | "undefined" |
| Null | "object" |

## if...else Statement

syntax:

if (expression){

Statement(s) to be executed if expression is true

}

else{

Statement(s) to be executed if expression is false

}

Example:

<html>

<body>

<script type="text/javascript">

<!--

var age = 15;

if( age > 18 ){

document.write("<b>Qualifies for driving</b>");

}

else{

document.write("<b>Does not qualify for driving</b>");

}

//-->

</script>

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

</body>

</html>

### if...else if... statement

if (expression 1){

Statement(s) to be executed if expression 1 is true

}

else if (expression 2){

Statement(s) to be executed if expression 2 is true

}

else if (expression 3){

Statement(s) to be executed if expression 3 is true

}

else{

Statement(s) to be executed if no expression is true

}

## Switch case

Example:

<html>

<body>

<script type="text/javascript">

<!--

var grade='A';

document.write("Entering switch block<br />");

switch (grade)

{

case 'A': document.write("Good job<br />");

break;

case 'B': document.write("Pretty good<br />");

break;

case 'C': document.write("Passed<br />");

break;

case 'D': document.write("Not so good<br />");

break;

case 'F': document.write("Failed<br />");

break;

default: document.write("Unknown grade<br />")

}

document.write("Exiting switch block");

//-->

</script>

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

</body>

</html>

## While loop

Same as java

## For loop

Example:

<html>

<body>

<script type="text/javascript">

<!--

var count;

document.write("Starting Loop" + "<br />");

for(count = 0; count < 10; count++){

document.write("Current Count : " + count );

document.write("<br />");

}

document.write("Loop stopped!");

//-->

</script>

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

</body>

</html>

## for...in loop

## Functions

**Syntax**

The basic syntax is shown here.

<script type="text/javascript">

<!--

function functionname(parameter-list)

{

statements

}

//-->

</script>

Example: function with arguments

<html>

<head>

<script type="text/javascript">

function sayHello(name, age)

{

document.write (name + " is " + age + " years old.");

}

</script>

</head>

<body>

<p>Click the following button to call the function</p>

<form>

<input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">

</form>

<p>Use different parameters inside the function and then try...</p>

</body>

</html>

Functions with return:

<html>

<head>

<script type="text/javascript">

function concatenate(first, last)

{

var full;

full = first + last;

return full;

}

function secondFunction()

{

var result;

result = concatenate('Zara', 'Ali');

document.write (result );

}

</script>

</head>

<body>

<p>Click the following button to call the function</p>

<form>

<input type="button" onclick="secondFunction()" value="Call Function">

</form>

<p>Use different parameters inside the function and then try...</p>

</body>

</html>

**Nested Functions**

<html>

<head>

<script type="text/javascript">

<!--

function hypotenuse(a, b) {

function square(x) { return x\*x; }

return Math.sqrt(square(a) + square(b));

}

function secondFunction(){

var result;

result = hypotenuse(1,2);

document.write ( result );

}

//-->

</script>

</head>

<body>

<p>Click the following button to call the function</p>

<form>

<input type="button" onclick="secondFunction()" value="Call Function">

</form>

<p>Use different parameters inside the function and then try...</p>

</body>

</html>

## The Function() Constructor

Syntax:

<script type="text/javascript">

<!--

var variablename = new Function(Arg1, Arg2..., "Function Body");

//-->

</script>

Example:

<html>

<head>

<script type="text/javascript">

<!--

var func = new Function("x", "y", "return x\*y;");

function secondFunction(){

var result;

result = func(10,20);

document.write ( result );

}

//-->

</script>

</head>

<body>

<p>Click the following button to call the function</p>

<form>

<input type="button" onclick="secondFunction()" value="Call Function">

</form>

<p>Use different parameters inside the function and then try...</p>

</body>

</html>

## Function Literals

Syntax1:

<script type="text/javascript">

<!--

var variablename = function(Argument List){

Function Body

};

//-->

</script>

Syntax2:

<script type="text/javascript">

<!--

var variablename = function FunctionName(Argument List){

Function Body

};

//-->

</script>

Example:

<html>

<head>

<script type="text/javascript">

<!--

var func = function(x,y){ return x\*y };

function secondFunction(){

var result;

result = func(10,20);

document.write ( result );

}

//-->

</script>

</head>

<body>

<p>Click the following button to call the function</p>

<form>

<input type="button" onclick="secondFunction()" value="Call Function">

</form>

<p>Use different parameters inside the function and then try...</p>

</body>

</html>

## Events

**<body> and <frameset> Level Events:**

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

**<form> Level Events:**

|  |  |  |
| --- | --- | --- |
| Attribute | Value | Description |
| onchange | script | Script runs when the element changes |
| onsubmit | script | Script runs when the form is submitted |
| onreset | script | Script runs when the form is reset |
| onselect | script | Script runs when the element is selected |
| onblur | script | Script runs when the element loses focus |
| onfocus | script | Script runs when the element gets focus |

**Keyboard Events**

|  |  |  |
| --- | --- | --- |
| Attribute | Value | Description |
| onkeydown | script | Script runs when key is pressed |
| onkeypress | script | Script runs when key is pressed and released |
| onkeyup | script | Script runs when key is released |

**Other Events:**

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

Example:

<html>

<head>

<script type="text/javascript">

<!--

function sayHello() {

alert("Hello World")

}

//-->

</script>

</head>

<body>

<p>Click the following button and see result</p>

<form>

<input type="button" onclick="sayHello()" value="Say Hello" />

</form>

</body>

</html>

## Cookies

**What are Cookies ?**

Web Browsers and Servers use HTTP protocol to communicate and HTTP is a stateless protocol. But for a commercial website, it is required to maintain session information among different pages. For example, one user registration ends after completing many pages. But how to maintain users' session information across all the web pages.

In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

**How It Works ?**

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the browser sends the same cookie to the server for retrieval. Once retrieved, your server knows/remembers what was stored earlier.

Cookies are a plain text data record of 5 variable-length fields −

* **Expires** − The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
* **Domain** − The domain name of your site.
* **Path** − The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
* **Secure** − If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
* **Name=Value** − Cookies are set and retrieved in the form of key-value pairs

Cookies were originally designed for CGI programming. The data contained in a cookie is automatically transmitted between the web browser and the web server, so CGI scripts on the server can read and write cookie values that are stored on the client.

JavaScript can also manipulate cookies using the **cookie** property of the**Document** object. JavaScript can read, create, modify, and delete the cookies that apply to the current web page.

**Storing Cookies**

The simplest way to create a cookie is to assign a string value to the document.cookie object, which looks like this.

document.cookie = "key1=value1;key2=value2;expires=date";

Here the **expires** attribute is optional. If you provide this attribute with a valid date or time, then the cookie will expire on a given date or time and thereafter, the cookies' value will not be accessible.

**Note** − Cookie values may not include semicolons, commas, or whitespace. For this reason, you may want to use the JavaScript **escape()** function to encode the value before storing it in the cookie. If you do this, you will also have to use the corresponding **unescape()** function when you read the cookie value.

**Example**

Try the following. It sets a customer name in an input cookie.

<html>

<head>

<script type="text/javascript">

<!--

function WriteCookie()

{

if( document.myform.customer.value == "" ){

alert("Enter some value!");

return;

}

cookievalue= escape(document.myform.customer.value) + ";";

document.cookie="name=" + cookievalue;

document.write ("Setting Cookies : " + "name=" + cookievalue );

}

//-->

</script>

</head>

<body>

<form name="myform" action="">

Enter name: <input type="text" name="customer"/>

<input type="button" value="Set Cookie" onclick="WriteCookie();"/>

</form>

</body>

</html>

**Reading Cookies**

Reading a cookie is just as simple as writing one, because the value of the document.cookie object is the cookie. So you can use this string whenever you want to access the cookie. The document.cookie string will keep a list of name=value pairs separated by semicolons, where name is the name of a cookie and value is its string value.

You can use strings' split() function to break a string into key and values as follows −

Example

Try the following example to get all the cookies.

<html>

<head>

<script type="text/javascript">

<!--

function ReadCookie()

{

var allcookies = document.cookie;

document.write ("All Cookies : " + allcookies );

// Get all the cookies pairs in an array

cookiearray = allcookies.split(';');

// Now take key value pair out of this array

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

name = cookiearray[i].split('=')[0];

value = cookiearray[i].split('=')[1];

document.write ("Key is : " + name + " and Value is : " + value);

}

}

//-->

</script>

</head>

<body>

<form name="myform" action="">

<p> click the following button and see the result:</p>

<input type="button" value="Get Cookie" onclick="ReadCookie()"/>

</form>

</body>

</html>

**Note** − Here **length** is a method of **Array** class which returns the length of an array. We will discuss Arrays in a separate chapter. By that time, please try to digest it.

**Note** − There may be some other cookies already set on your machine. The above code will display all the cookies set on your machine.

**Setting Cookies Expiry Date**

You can extend the life of a cookie beyond the current browser session by setting an expiration date and saving the expiry date within the cookie. This can be done by setting the ‘expires’ attribute to a date and time.

Example

Try the following example. It illustrates how to extend the expiry date of a cookie by 1 Month.

<html>

<head>

<script type="text/javascript">

<!--

function WriteCookie()

{

var now = new Date();

now.setMonth( now.getMonth() + 1 );

cookievalue = escape(document.myform.customer.value) + ";"

document.cookie="name=" + cookievalue;

document.cookie = "expires=" + now.toUTCString() + ";"

document.write ("Setting Cookies : " + "name=" + cookievalue );

}

//-->

</script>

</head>

<body>

<form name="formname" action="">

Enter name: <input type="text" name="customer"/>

<input type="button" value="Set Cookie" onclick="WriteCookie()"/>

</form>

</body>

</html>

**Deleting a Cookie**

Sometimes you will want to delete a cookie so that subsequent attempts to read the cookie return nothing. To do this, you just need to set the expiry date to a time in the past.

**Example**

Try the following example. It illustrates how to delete a cookie by setting its expiry date to one month behind the current date.

<html>

<head>

<script type="text/javascript">

<!--

function WriteCookie()

{

var now = new Date();

now.setMonth( now.getMonth() - 1 );

cookievalue = escape(document.myform.customer.value) + ";"

document.cookie="name=" + cookievalue;

document.cookie = "expires=" + now.toUTCString() + ";"

document.write("Setting Cookies : " + "name=" + cookievalue );

}

//-->

</script>

</head>

<body>

<form name="formname" action="">

Enter name: <input type="text" name="customer"/>

<input type="button" value="Set Cookie" onclick="WriteCookie()"/>

</form>

</body>

</html>

# Page Refresh

You can refresh a web page using JavaScript **location.reload** method. This code can be called automatically upon an event or simply when the user clicks on a link. If you want to refresh a web page using a mouse click, then you can use the following code −

<a href="javascript:location.reload(true)">Refresh Page</a>

## Auto Refresh

You can also use JavaScript to refresh the page automatically after a given time period. Here **setTimeout()** is a built-in JavaScript function which can be used to execute another function after a given time interval.

**Example**

Try the following example. It shows how to refresh a page after every 5 seconds. You can change this time as per your requirement.

<html>

<head>

<script type="text/JavaScript">

<!--

function AutoRefresh( t ) {

setTimeout("location.reload(true);", t);

}

//-->

</script>

</head>

<body onload="JavaScript:AutoRefresh(5000);">

<p>This page will refresh every 5 seconds.</p>

</body>

</html>

# Page Redirection

## What is Page Redirection ?

You might have encountered a situation where you clicked a URL to reach a page X but internally you were directed to another page Y. It happens due to **page redirection**.

There could be various reasons why you would like to redirect a user from the original page. We are listing down a few of the reasons −

* You did not like the name of your domain and you are moving to a new one. In such a scenario, you may want to direct all your visitors to the new site. Here you can maintain your old domain but put a single page with a page redirection such that all your old domain visitors can come to your new domain.
* You have built-up various pages based on browser versions or their names or may be based on different countries, then instead of using your server-side page redirection, you can use client-side page redirection to land your users on the appropriate page.
* The Search Engines may have already indexed your pages. But while moving to another domain, you would not like to lose your visitors coming through search engines. So you can use client-side page redirection. But keep in mind this should not be done to fool the search engine, it could lead your site to get banned.

**How Page Re-direction Works ?**

The implementations of Page-Redirection are as follows.

**Example 1**

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

<html>

<head>

<script type="text/javascript">

<!--

function Redirect() {

window.location="http://www.tutorialspoint.com";

}

//-->

</script>

</head>

<body>

<p>Click the following button, you will be redirected to home page.</p>

<form>

<input type="button" value="Redirect Me" onclick="Redirect();" />

</form>

</body>

</html>

**Example 2**

You can show an appropriate message to your site visitors before redirecting them to a new page. This would need a bit time delay to load a new page. The following example shows how to implement the same. Here setTimeout() is a built-in JavaScript function which can be used to execute another function after a given time interval.

<html>

<head>

<script type="text/javascript">

<!--

function Redirect() {

window.location="http://www.tutorialspoint.com";

}

document.write("You will be redirected to main page in 10 sec.");

setTimeout('Redirect()', 10000);

//-->

</script>

</head>

<body>

</body>

</html>

Example 3

The following example shows how to redirect your site visitors onto a different page based on their browsers.

<html>

<head>

<script type="text/javascript">

<!--

var browsername=navigator.appName;

if( browsername == "Netscape" )

{

window.location="http://www.location.com/ns.htm";

}

else if ( browsername =="Microsoft Internet Explorer")

{

window.location="http://www.location.com/ie.htm";

}

else

{

window.location="http://www.location.com/other.htm";

}

//-->

</script>

</head>

<body>

</body>

</html>

# Dialog Boxes

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

## Alert Dialog Box

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

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

**Example**

<html>

<head>

<script type="text/javascript">

<!--

function Warn() {

alert ("This is a warning message!");

document.write ("This is a warning message!");

}

//-->

</script>

</head>

<body>

<p>Click the following button to see the result: </p>

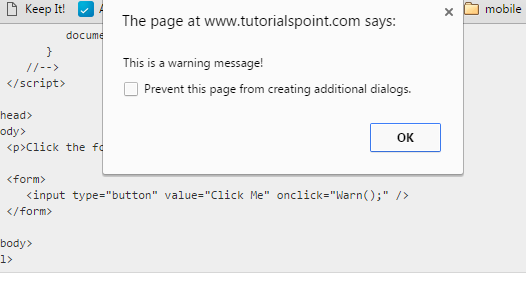
<form>

<input type="button" value="Click Me" onclick="Warn();" />

</form>

</body>

</html>



## Confirmation Dialog Box

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

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

### **Example**

<html>

<head>

<script type="text/javascript">

<!--

function getConfirmation(){

var retVal = confirm("Do you want to continue ?");

if( retVal == true ){

document.write ("User wants to continue!");

return true;

}

else{

document.write ("User does not want to continue!");

return false;

}

}

//-->

</script>

</head>

<body>

<p>Click the following button to see the result: </p>

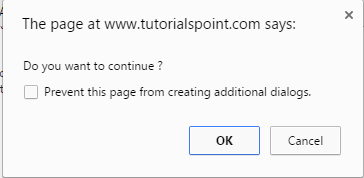
<form>

<input type="button" value="Click Me" onclick="getConfirmation();" />

</form>

</body>

</html>



## Prompt Dialog Box

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

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

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

**Example**

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

<html>

<head>

<script type="text/javascript">

<!--

function getValue(){

var retVal = prompt("Enter your name : ", "your name here");

document.write("You have entered : " + retVal);

}

//-->

</script>

</head>

<body>

<p>Click the following button to see the result: </p>

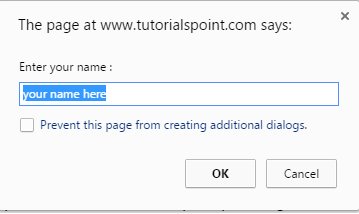
<form>

<input type="button" value="Click Me" onclick="getValue();" />

</form>

</body>

</html>



# Void Keyword

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

**Syntax**

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

<head>

<script type="text/javascript">

<!--

void func()

javascript:void func()

or:

void(func())

javascript:void(func())

//-->

</script>

</head>

Example 1

The most common use of this operator is in a client-side *javascript:* URL, where it allows you to evaluate an expression for its side-effects without the browser displaying the value of the evaluated expression.

Here the expression alert ('Warning!!!') is evaluated but it is not loaded back into the current document −

<html>

<head>

<script type="text/javascript">

<!--

//-->

</script>

</head>

<body>

<p>Click the following, This won't react at all...</p>

<a href="javascript:void(alert('Warning!!!'))">Click me!</a>

</body>

</html>

Example 2

Take a look at the following example. The following link does nothing because the expression "0" has no effect in JavaScript. Here the expression "0" is evaluated, but it is not loaded back into the current document.

<html>

<head>

<script type="text/javascript">

<!--

//-->

</script>

</head>

<body>

<p>Click the following, This won't react at all...</p>

<a href="javascript:void(0))">Click me!</a>

</body>

</html>

Example 3

Another use of void is to purposely generate the undefined value as follows.

<html>

<head>

<script type="text/javascript">

<!--

function getValue(){

var a,b,c;

a = void ( b = 5, c = 7 );

document.write('a = ' + a + ' b = ' + b +' c = ' + c );

}

//-->

</script>

</head>

<body>

<p>Click the following to see the result:</p>

<form>

<input type="button" value="Click Me" onclick="getValue();" />

</form>

</body>

</html>

# Page Printing

Many times you would like to place a button on your webpage to print the content of that web page via an actual printer. JavaScript helps you to implement this functionality using the **print** function of **window** object.

The JavaScript print function **window.print()** prints the current web page when executed. You can call this function directly using the **onclick** event as shown in the following example.

Example

Try the following example.

<html>

<head>

<script type="text/javascript">

<!--

//-->

</script>

</head>

<body>

<form>

<input type="button" value="Print" onclick="window.print()" />

</form>

</body>

<html>

# Objects Overview

## User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called**Object**.

**The new Operator**

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

var employee = new Object();

var books = new Array("C++", "Perl", "Java");

var day = new Date("August 15, 1947");

**The Object() Constructor**

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called Object() to build the object. The return value of the Object() constructor is assigned to a variable.

The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the var keyword.

**Example 1**

Try the following example; it demonstrates how to create an Object.

<html>

<head>

<title>User-defined objects</title>

<script type="text/javascript">

var book = new Object(); // Create the object

book.subject = "Perl"; // Assign properties to the object

book.author = "Mohtashim";

</script>

</head>

<body>

<script type="text/javascript">

document.write("Book name is : " + book.subject + "<br>");

document.write("Book author is : " + book.author + "<br>");

</script>

</body>

</html>

**Example 2**

This example demonstrates how to create an object with a User-Defined Function. Here this keyword is used to refer to the object that has been passed to a function.

<html>

<head>

<title>User-defined objects</title>

<script type="text/javascript">

function book(title, author){

this.title = title;

this.author = author;

}

</script>

</head>

<body>

<script type="text/javascript">

var myBook = new book("Perl", "Mohtashim");

document.write("Book title is : " + myBook.title + "<br>");

document.write("Book author is : " + myBook.author + "<br>");

</script>

</body>

</html>

Example 3:

<html>

<head>

<title>**User-defined objects**</title>

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

myBook**={**myTitle**:** "Kak"**,** myAuthor**:**"Auth"**};**

</script>

</head>

<body>

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

document.write**(**"Book title is : " **+** myBook.myTitle **+** "<br>"**);**

document.write**(**"Book author is : " **+** myBook.myAuthor **+** "<br>"**);**

</script>

</body>

</html>

## Defining Methods for an Object

**Example**

Try the following example; it shows how to add a function along with an object.

<html>

<head>

<title>User-defined objects</title>

<script type="text/javascript">

// Define a function which will work as a method

function addPrice(amount){

this.price = amount;

}

function book(title, author){

this.title = title;

this.author = author;

this.addPrice = addPrice; // Assign that method as property.

}

</script>

</head>

<body>

<script type="text/javascript">

var myBook = new book("Perl", "Mohtashim");

myBook.addPrice(100);

document.write("Book title is : " + myBook.title + "<br>");

document.write("Book author is : " + myBook.author + "<br>");

document.write("Book price is : " + myBook.price + "<br>");

</script>

</body>

</html>

## The 'with' Keyword

The **‘with’** keyword is used as a kind of shorthand for referencing an object's properties or methods.

The object specified as an argument to **with** becomes the default object for the duration of the block that follows. The properties and methods for the object can be used without naming the object.

**Syntax**

The syntax for with object is as follows −

with (object){

properties used without the object name and dot

}

Example

Try the following example.

<html>

<head>

<title>User-defined objects</title>

<script type="text/javascript">

// Define a function which will work as a method

function addPrice(amount){

with(this){

price = amount;

}

}

function book(title, author){

this.title = title;

this.author = author;

this.price = 0;

this.addPrice = addPrice; // Assign that method as property.

}

</script>

</head>

<body>

<script type="text/javascript">

var myBook = new book("Perl", "Mohtashim");

myBook.addPrice(100);

document.write("Book title is : " + myBook.title + "<br>");

document.write("Book author is : " + myBook.author + "<br>");

document.write("Book price is : " + myBook.price + "<br>");

</script>

</body>

</html>

## JavaScript Native Objects

JavaScript has several built-in or native objects. These objects are accessible anywhere in your program and will work the same way in any browser running in any operating system.

Here is the list of all important JavaScript Native Objects −

* [JavaScript Number Object](http://www.tutorialspoint.com/javascript/javascript_number_object.htm)
* [JavaScript Boolean Object](http://www.tutorialspoint.com/javascript/javascript_boolean_object.htm)
* [JavaScript String Object](http://www.tutorialspoint.com/javascript/javascript_strings_object.htm)
* [JavaScript Array Object](http://www.tutorialspoint.com/javascript/javascript_arrays_object.htm)
* [JavaScript Date Object](http://www.tutorialspoint.com/javascript/javascript_date_object.htm)
* [JavaScript Math Object](http://www.tutorialspoint.com/javascript/javascript_math_object.htm)
* [JavaScript RegExp Object](http://www.tutorialspoint.com/javascript/javascript_regexp_object.htm)

# Reference

Define and access Object method: <https://www.youtube.com/watch?v=6lQEtgFnZTY&list=PL46F0A159EC02DF82&index=27>