Javascript Tutorial

Tutorialspoint.com

***Javascript* is a scripting language produced by Netscape for use within HTML Web pages.**

JavaScript is loosely based on Java and it is built into all the major modern browsers. This tutorial gives an initial push to start you with Javascript. For more detail kindly check tutorialspoint.com/javascript

What is JavaScript ?

JavaScript is:

- 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

JavaScript Syntax:

A JavaScript consists of JavaScript statements that are placed within the <script>... </script> HTML tags in a web page.

You can place the <script> tag containing your JavaScript anywhere within you web page but it is preferred way to keep it within the <head> tags.

The <script> tag alert the browser program to begin interpreting all the text between these tags as a script. So simple syntax of your JavaScript will be as follows

```
<script ...>
  JavaScript code
</script>
```

The script tag takes two important attributes:

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

So your JavaScript segment will look like:

```
<script language="javascript" type="text/javascript">
  JavaScript code
</script>
```

Your First JavaScript Script:

Let us write our class example to print out "Hello World".

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

Above code will display following result:

```
Hello World!
```

Whitespace and Line Breaks:

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs.

Because you can use spaces, tabs, and newlines freely in your program so you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Semicolons are Optional:

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

```
<script language="javascript" type="text/javascript">
<!--
  var1 = 10
  var2 = 20
//-->
</script>
```

But when formatted in a single line as follows, the semicolons are required:

```
<script language="javascript" type="text/javascript">
<!--
  var1 = 10; var2 = 20;
//-->
</script>
```

Note: It is a good programming practice to use semicolons.

Case Sensitivity:

JavaScript is a case-sensitive language. This means that language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So identifiers *Time*, *TIme* and *TIME* will have different meanings in JavaScript.

NOTE: Care should be taken while writing your variable and function names in JavaScript.

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 //-->.

JavaScript Placement in HTML File:

There is a flexibility given to include JavaScript code anywhere in an HTML document. But there are following most preferred ways to include JavaScript in your HTML file.

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in and external file and then include in <head>...</head> section.

JavaScript DataTypes:

JavaScript allows you to work with three primitive data types:

- Numbers eg. 123, 120.50 etc.
- Strings of text e.g. "This text string" etc.
- Boolean e.g. true or false.

JavaScript also defines two trivial data types, *null* and *undefined*, each of which defines only a single value.

JavaScript Variables:

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows:

```
<script type="text/javascript">
<!--
var money;
var name;
//-->
</script>
```

JavaScript Variable Scope:

The scope of a variable is the region of your program in which it is defined. JavaScript variable will have only two scopes.

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

JavaScript Variable Names:

While naming your variables in JavaScript keep following rules in mind.

- You should not use any of the JavaScript reserved keyword as variable name. 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 the underscore character. For example, *123test* is an invalid variable name but *_123test* is a valid one.
- JavaScript variable names are case sensitive. For example, *Name* and *name* are two different variables.

JavaScript Reserved Words:

The following are reserved words in JavaScript. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

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

The Arithmatic Operators:

There are following arithmatic operators supported by JavaScript language:

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

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by denumerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0

++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

The Comparison Operators:

There are following comparison operators supported by JavaScript language

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

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

The Logical Operators:

There are following logical operators supported by JavaScript language

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

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false.

The Bitwise Operators:

There are following bitwise operators supported by JavaScript language

Assume variable A holds 2 and variable B holds 3 then:

Operator	Description	Example
&	Called Bitwise AND operator. It performs a Boolean AND operation on each bit of its integer arguments.	(A & B) is 2 .
|	Called Bitwise OR Operator. It performs a Boolean OR operation on each bit of its integer arguments.	(A | B) is 3.
^	Called Bitwise XOR Operator. It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.	(A ^ B) is 1.
~	Called Bitwise NOT Operator. It is a is a unary operator and operates by reversing all bits in the operand.	(~B) is -4 .
<<	Called Bitwise Shift Left Operator. It moves all bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying by 2, shifting two positions is equivalent to multiplying by 4, etc.	(A << 1) is 4.
>>	Called Bitwise Shift Right with Sign Operator. It moves all bits in its first operand to the right by the number of places specified in the second operand. The bits filled in on the left depend on the sign bit of the original operand, in order to preserve the sign of the result. If the first operand is positive, the result has zeros placed in the high bits; if the first operand is negative, the result has ones placed in the high bits. Shifting a value right one place is equivalent to dividing by 2 (discarding the remainder), shifting right two places is equivalent to integer division by 4, and so on.	(A >> 1) is 1.
>>>	Called Bitwise Shift Right with Zero Operator. This operator is just like the >> operator, except that the bits shifted in on the left are always zero,	(A >>> 1) is 1.

The Assignment Operators:

There are following assignment operators supported by JavaScript language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assigne value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies	C *= A is equivalent to C = C *

	right operand with the left operand and assign the result to left operand	A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Miscellaneous Operator

The Conditional Operator (? :)

There is an oprator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation. The conditioanl operator has this syntax:

Operator	Description	Example
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

The *typeof* Operator

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

The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

if statement:

The **if** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax:

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

if...else statement:

The **if...else** statement is the next form of control statement that allows JavaScript to execute statements in more controlled way.

Syntax:

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

if...else if... statement:

The **if...else if...** statement is the one level advance form of control statement that allows JavaScript to make correct decision out of several conditions.

Syntax:

```
if (expression 1){
   Statement(s) to be executed if expression 1 is true
}else if (expression 2){
   Statement(s) to be executed if expression 2 is true
}else if (expression 3){
   Statement(s) to be executed if expression 3 is true
}else{
   Statement(s) to be executed if no expression is true
}
```

switch statement:

The basic syntax of the **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each **case** against the value of the expression until a match is found. If nothing matches, a **default** condition will be used.

```
switch (expression)
{
  case condition 1: statement(s)
                    break;
  case condition 2: statement(s)
                    break;
   ...
  case condition n: statement(s)
                    break;
  default: statement(s)
}
```

The *while* Loop

The most basic loop in JavaScript is the **while** loop which would be discussed in this tutorial.

Syntax:

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

The *do...while* Loop:

The **do...while** loop is similar to the **while** loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is *false*.

Syntax:

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

The *for* Loop

The **for** loop is the most compact form of looping and includes the following three important parts:

- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if the given condition is true or not. If condition is true then code given inside the loop will be executed otherwise loop will come out.
- The iteration statement where you can increase or decrease your counter.

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

Syntax:

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

The *for...in* Loop

```
for (variablename in object){
  statement or block to execute
}
```

In each iteration one property from *object* is assigned to *variablename* and this loop continues till all the properties of the object are exhausted.

The *break* Statement:

The **break** statement, which was briefly introduced with the *switch* statement, is used to exit a loop early, breaking out of the enclosing curly braces.

The *continue* Statement:

The **continue** statement tells the interpreter to immediately start the next iteration of the loop and skip remaining code block.

When a **continue** statement is encountered, program flow will move to the loop check expression immediately and if condition remain true then it start next iteration otherwise control comes out of the loop.

Function Definition:

Before we use a function we need to define that function. The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces. The basic syntax is shown here:

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

```
<!--
function functionname(parameter-list)
{
  statements
}
//-->
</script>
```

Calling a Function:

To invoke a function somewhere later in the script, you would simple need to write the name of that function as follows:

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

<!--
sayHello();
//-->
</script>
```

Exceptions

Exceptions can be handled with the common try/catch/finally block structure.

```
<script type="text/javascript">
<!--
try {
   statementsToTry
} catch ( e ) {
      catchStatements
} finally {
      finallyStatements
}
//-->
</script>
```

The try block must be followed by either exactly one catch block or one finally block (or one of both). When an exception occurs in the catch block, the exception is placed in e and the catch block is executed. The finally block executes unconditionally after try/catch.

Alert Dialog Box:

An alert dialog box is mostly used to give a warning message to the users. Like if one input field requires to enter some text but user does not enter that field then as a part of validation you can use alert box to give warning message as follows:

```
<head>
<script type="text/javascript">
<!--
   alert("Warning Message");
//-->

</script>
</head>
```

Confirmation Dialog Box:

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

You can use confirmation dialog box as follows:

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

Prompt Dialog Box:

You can use prompt dialog box as follows:

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

Page Re-direction

This is very 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:

```
<head>
<script type="text/javascript">
<!--
   window.location="http://www.newlocation.com";
//-->
</script>
</head>
```

The void Keyword:

The **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. Its syntax could be one of the following:

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

or:

void(func())
javascript:void(func())
//-->
</script>
</head>
```

The Page Printing:

JavaScript helps you to implement this functionality using **print** function of *window* object.

The JavaScript print function **window.print()** will print the current web page when executed. You can call this function directly using *onclick* event as follows:

```
<head>
<script type="text/javascript">
<!--
//-->
</script>
</head>
<body>
<form>
<input type="button" value="Print" onclick="window.print()" />
</form>
</body>
```

Storing Cookies:

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

Syntax:

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

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.

JavaScript - Page Redirection

What is page redirection ?

When you click a URL to reach to a page X but internally you are directed to another page Y that simply happens because of page re-direction. This concept is different from JavaScript Page Refresh.

There could be various reasons why you would like to redirect from original page. I'm listing down few of the reasons:

- You did not like the name of your domain and you are moving to a new one. Same time you want to direct your all visitors to new site. In such case you can maintain your old domain but put a single page with a page re-direction so that your all old domain visitors can come to your new domain.
- You have build-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 appropriate page.
- The Search Engines may have already indexed your pages. But while moving to another domain then 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 make search engine a fool otherwise this could get your web site banned.

How Page Re-direction works ?

Example 1:

This is very 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:

```
<head>
<script type="text/javascript">
<!--
   window.location="http://www.newlocation.com";
//-->
</script>
</head>
```

To understand it in better way you can Try it yourself.

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. Following is the simple example to implement the same:

```
<head>
<script type="text/javascript">
<!--
function Redirect()
{
    window.location="http://www.newlocation.com";
}

document.write("You will be redirected to main page in 10 sec.");
setTimeout('Redirect()', 10000);
//-->
</script>
</head>
```

Here *setTimeout()* is a built-in JavaScript function which can be used to execute another function after a given time interval.

To understand it in better way you can Try it yourself.

Example 3:

Following is the example to redirect site visitors on different pages based on their browsers :

```
<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>
```

JavaScript - Errors & Exceptions Handling

There are three types of errors in programming: (a) Syntax Errors and (b) Runtime Errors (c) Logical Errors:

Syntax errors:

Syntax errors, also called parsing errors, occur at compile time for traditional programming languages and at interpret time for JavaScript.

For example, the following line causes a syntax error because it is missing a closing parenthesis:

```
<script type="text/javascript">
<!--
window.print(;
//-->
</script>
```

When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected and code in other threads gets executed assuming nothing in them depends on the code containing the error.

Runtime errors:

Runtime errors, also called exceptions, occur during execution (after compilation/interpretation).