7. JavaScript

INTRODUCTION

HTML defines the content of web pages. CSS specify the layout of web pages. JavaScript program the behavior of web pages.

Scripting Languages are of two types:

1. *Client-side Scripting*: Used for verifying simple validations at client-side. **Examples:** VBScript, JavaScript and Jscript
2. *Server-side Scripting*: Used for database verifications. **Examples:** ASP, JSP, Servlets

What is JavaScript?

JavaScript is the programming language of the Web. All modern HTML pages are using JavaScript. JavaScript is easy to learn. JavaScript is created by Netscape to add interactivity to HTML pages. It was renamed from “LiveScript”. JavaScript was standardized by ECMA (European Computer Manufacturer Association) and also called as ECMAScript.

JavaScript is a scripting language (lightweight programming language) used to execute code in client (browser). JavaScript is an object-based scripting language designed to enhance web pages that constructed with HTML documents. It is derived from C language syntax and case sensitive language.

What can we do with JavaScript?

JavaScript can

- Change HTML content (place dynamic content into HTML elements)
- Change HTML attributes
- Change CSS
- Validate Data entered by user
- Can react to events.

Benefits of JavaScript

JavaScript has a number of big benefits to anyone who wants to make their Web site dynamic:

- It is widely supported in web browsers;
- It gives easy access to the document objects and can manipulate most of them;
- Can give interesting animations without the long download times associated with many multimedia data types;
- Web surfers don’t need a special plug-in to use your scripts;
- It is relatively secure.
- You can’t get a virus infection directly from JavaScript.

Similarities between JavaScript & Java

1. Both have same kind of operators.
2. JavaScript uses similar control structures of Java.
3. Now a days, both are used as language for the use of internet.
4. Labeled break, continue are available in both.
5. Up to concept of objects and methods both are similar.

Difference between JavaScript and Java

1. Java is full featured programming language but JavaScript is not.
2. Inheritance, polymorphism, threads are not allowed in JavaScript.
3. Java source code is first compiled and the client interpreters the code, JavaScript code is not compiled but only interpreted.

Where to put JavaScript?

JavaScript is placed using <script> tag or script can be taken from the file, which typically contains extension .js.

JavaScript's in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, other times when a user triggers an event.

Scripts in the head section: Scripts to be executed when they are called, or when an event is triggered, go in the head section. When you place a script in the head section, you will ensure that the script is loaded before anyone uses it.

```
<html>
<head>
<script type="text/javascript" language="javascript">
....
</script>
</head>
```

Scripts in the body section: Scripts to be executed when the page loads go in the body section. When you place a script in the body section it generates the content of the page.

```
<html>
<head>
</head>
<body>
<script type="text/javascript" language="javascript">
....
</script>
</body>
```

Scripts in both the body and the head section: You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head section.

```
<html>
<head>
<script type="text/javascript" language="javascript">
....
</script>
</head>
<body>
<script type="text/javascript" language="javascript">
....
</script>
</body>
```

Using an External JavaScript

Sometimes you might want to run the same JavaScript on several pages, without having to write the same script on every page. To simplify this, you can write a JavaScript in an external file. Save the external JavaScript file with a .js file extension.

Note: *The external script cannot contain the <script> tag!*

To use the external script, point to the .js file in the "src" attribute of the <script> tag:

```
<html>
<head>
<script src="script.js" type="text/javascript" language="javascript">
</script>
</head>
<body>
</body>
</html>
```

Note: *Remember to place the script exactly where you normally would write the script!*

Variables in JavaScript

JavaScript variables are containers for storing data values. Variables need not be declared, it can be done using **var** keyword.

Example

```
var x = 5;
var y = 6;
var z = x + y; In this example, x, y, and z, are variables.
```

JavaScript is type-less language; it does not have any predefined data types.

JavaScript variables can hold numbers like 120, and text values like "Manoj". In programming, text values are called text strings. JavaScript can handle many types of data. Strings are written inside *double or single quotes*. Numbers are written without quotes. If you put quotes around a number, it will be treated as a text string.

JavaScript variables can hold many data types: numbers, strings, arrays, objects and more:

```
var length = 16;                               // Number
var lastName = "Johnson";                      // String
var cars = ["Saab", "Volvo", "BMW"];           // Array
var x = {firstName:"John", lastName:"Doe"};    // Object
```

JavaScript Strings

A string (or a text string) is a series of characters like "Manohar". Strings are written with quotes. You can use single or double quotes:

Example

```
var carName = "Alto K10";   // Using double quotes
var carName = 'Alto K10';   // Using single quotes
```

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example

```
var answer = "It's alright"; // Single quote inside double quotes
var answer = "He is called 'Superstar'";
                           // Single quotes inside double quotes
var answer = 'He is called "Superstar"';
                           // Double quotes inside single quotes
```

Variable life time begins with declaration (or assignment) and end when page is unloaded.

Operators

Arithmetic	**Relational**	**Logical**	**Miscellaneous**
+	==	&&	+ (Concatenation)
-	!=	||	instanceof
*	>	!	this
/	<		typeof
%	>=		new
++	<=		
--	===		
	!==		

JavaScript Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable. JavaScript comments can also be used to prevent execution, when testing alternative code.

Single line comments start with //. Any text between // and the end of a line, will be ignored by JavaScript (will not be executed).

Multi-line comments start with /* and end with */. Any text between /* and */ will be ignored by JavaScript.

Concatenation Operator:

The + operator can also be used to concatenate (add) strings. When used on strings, the + operator is called the concatenation operator.

```
	txt1 = "What a very";
	txt2 = "nice day";
	txt3 = txt1 + txt2;
```

The result of txt3 will be: What a verynice day

```
	txt1 = "What a very";
	txt2 = "nice day";
	txt3 = txt1 + " " + txt2;
```

The result of txt3 will be: What a very nice day

Adding two numbers will return the sum, but adding a number and a string will return a string:

Example

```
	x = 5 + 5;
	y = "5" + 5;
	z = "Hello" + 5;
```

The result of x, y, and z will be:

```
	10
	55
	Hello5
```

Conditional Operator:

```
	variablename = (condition)?value1:value2
```

If statement

Execute one of the code blocks depending upon the result of the condition. JavaScript allows all types of if statements i.e., Simple If, If..else, Nested if..else and else if ladder.

```
	if(condition) {
		// code to be executed if condition is true
	}
	else {
		// code to be executed if condition is false
	}
```

Switch Statement

You should use the switch statement if you want to select one of many blocks of code to be executed.

```
	switch(n) {
		case 1: code;	break;
		case 2: code;	break;
		case n: code;	break;
	}
```

For Loop

```
for(var=startvalue;var<=endvalue;var=var+increment) {
     code to be executed
}
for(variable in object) {
     code to be executed
}
```

While Loop

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

Do-while Loop

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

JavaScript Global

The JavaScript global properties and functions can be used with all the built-in JavaScript objects.

Global Properties

Property	Description
Infinity	A numeric value that represents positive/negative infinity
NaN	"Not-a-Number" value
undefined	Indicates that a variable has not been assigned a value

Global Functions

Function	Description	Examples

eval()	Evaluates a string and executes it as if it was script code	`var x = 10;`
		`var y = 20,a,b,c;`
		`a = eval("x * y");   //a=200`
		`b = eval("2 + 2");   //b=4`
		`c = eval("x + 17");  //c=27`

isFinite()	Determines whether a value is a finite, legal number	`var a = isFinite(123);     //a=true`
		`var b = isFinite(-1.23);   //b=true`
		`var c = isFinite(5-2);     //c=true`
		`var d = isFinite(0);       //d=true`
		`var e = isFinite("Hello");   //e=false`
		`var f = isFinite("2005/12/12"); //f=false`

isNaN()	Determines whether a value is an illegal number	`var  a = isNaN(123);    //a=false`
		`var  b = isNaN(-1.23);  //b=false`
		`var  c = isNaN(5-2);    //c=false`
		`var  d = isNaN(0);      //d=false`
		`var  e = isNaN("Hello"); //e=true`
		`var  f = isNaN("2005/12/12"); //f=true`

Number()	Converts an object's value to a number	`var x1 = Number(true);  //x1=1`
		`var x2 = Number(false); //x2=0`
		`var x3 = Number(new Date());`
		`//x3=1423715961429`
		`var x4 = Number("999"); //x4=999`
		`var x5 = Number("999 888"); //x5=NaN`

parseFloat()	Parses a string and returns a floating point number	
parseInt()	Parses a string and returns an integer	

String()	Converts an object's value to a string	`var x1 = String(Boolean(0));  //x1=false`
		`var x2 = String(Boolean(1));  //x2=true`
		`var x3 = String(new Date());`
		`//Wed Feb 11 2015 20:28:41 GMT-0800 (Pacific Standard Time)`
		`var x4 = String(12345);       //12345`

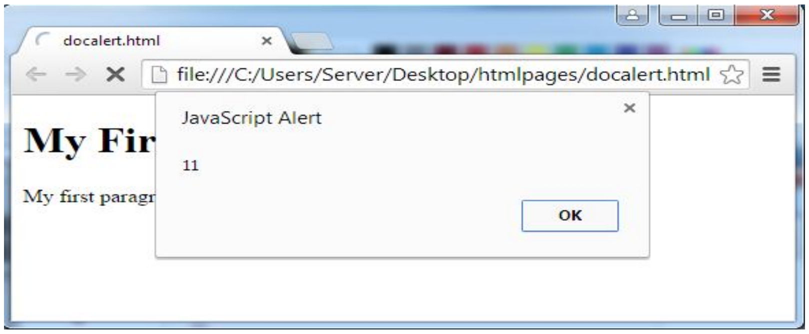
JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an alert box, using window.alert().
- Writing into the HTML output using document.write().
- Writing into an HTML element, using innerHTML.
- Writing into the browser console, using console.log().

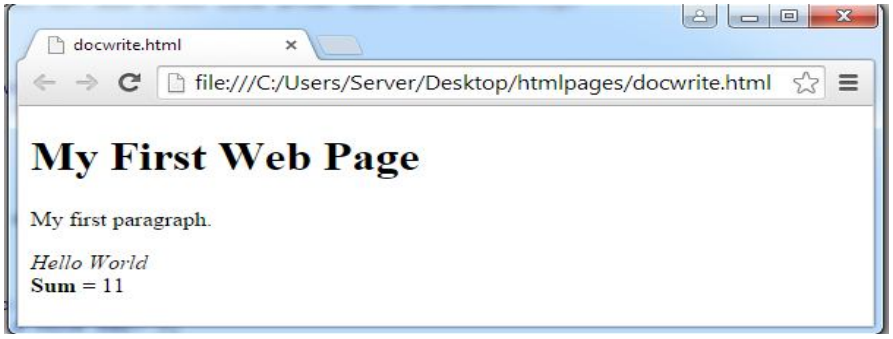
Using window.alert()

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
window.alert(5 + 6);
</script>
</body>
</html>
```



Using document.write(): To display some information on HTML body.

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
document.write("<i>Hello World</i><br/>");
document.write("<b>Sum = </b>" + (5 + 6));
</script>
</body>
</html>
```



My First Web Page

My first paragraph.

Hello World
Sum = 11

Using document.write() after an HTML document is fully loaded, will **delete all existing HTML**:

```
<!DOCTYPE html>
<html>
<body>
<button onclick="document.write(5 + 6)">Try it</button>
</body>
</html>
```

Using innerHTML:
To access an HTML element, JavaScript can use the document.getElementById(id) method. The id attribute defines the HTML element. The innerHTML property defines the HTML content:

```
<html>
<body>
Sum of 5 and 6: <h2 id="demo"></h2>
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
</body>
</html>
```

Using window.prompt():

The prompt() method displays a dialog box that prompts the visitor for input. A prompt box is often used if you want the user to input a value before entering a page.

Note: *When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value. Do not overuse this method, as it prevent the user from accessing other parts of the page until the box is closed.*

The prompt() method returns the input value if the user clicks "OK". If the user clicks "cancel" the method returns null.

Syntax

```
     prompt(text,defaultText);
```

```
<html>
<body>
<p id="demo"></p>
<script>
var person = prompt("Please enter your name", "");
if (person != null) {
      document.getElementById("demo").innerHTML =
        "Hello " + person + "! How are you today?";
}
</script>
</body>
</html>
```

Keep in mind that scripts are evaluated in the order they are listed in an HTML document, so your script gets run while the browser is processing the "head" section if script is written in head section. Since the browser hasn't yet parsed the "body" section it doesn't know about any element with id "demo".

However, by assigning a function to the "**window.onload**" event, you can delay the execution of your inner HTML assignment until the window has completely loaded all of the resources and safely manipulate the document.

Note that if your script was in the body section, after the element with id "demo" was listed, then the script would work without the need to wait for the window "load" event.

Program: Write a program to accept a number from user and display whether it is prime or not. Before testing for prime, check user has given number or not. If has not entered the number, display message that he must enter numeric.

```
<html>
<head>
      <title> Test for Prime</title>
      <script language = “JavaScript”>
```

```
          var n, i, p=1;
          do {
               n = parseInt(window.prompt("Enter Number to Test for
               Prime", "0"));
          }
          while(isNaN(n));
          for(i=2; i=(n/2); i++) {
               if(n%i == 0) {
                    p=0;
                    break;
               }
          }
          if(p == 0)
               window.alert("N is not Prime");
          else
               window.alert("N is Prime Number");
     </script>
</head>
</html>
```

User-Defined Functions

JavaScript allows users to define their functions. With the help of these functions programmers can divide a large program into small functions. Consider a small example you are writing a program to display prime Fibonacci numbers. Consider that you know very well about write a program to test whether a given number is prime or not and also you know how to display Fibonacci numbers.

Now you have to write a program combining these two logics. If you want to solve this problem easily, write a function to test a number is prime or not. Now in your Fibonacci program before displaying the number check a simple if condition like "if isPrime(f)" where f is Fibonacci number and isPrime() is a function that tests a given number is prime or not. It returns true if the given number is prime otherwise it returns false.

Functions in JavaScript

Divide and conquer can be achieved in JavaScript by using functions. A function can be defined as follows:

```
     function function-name (parameter-list)
     {
          declarations;
          statements;
          …
          return [expression];
     }
```

The function-name is any valid identifier. The parameter-list is list of arguments separated by commas that are supplied to function. Declarations are variables that are used in the function and they are called as local variables. The statements are meant for achieving the object of the function. Declarations and statements within the braces form the function body. This function body is also referred to as block. A block is a compound statement that includes declarations. The last statement return is used for result to be returned from the given function.

Program: Write a program to display square and cube of a given number by using user-defined functions.

```
<html>
<head>
     <title> Square and cube of given number</title>
     <script language = “JavaScript”>
          var a;
          a = parseInt(window.prompt(“Enter number:”, “0”));
          document.writeln(“Square of given number is <b>” +
          square(a) + “</b><br>”);
          document.writeln(“Cube of given number is <b>” + cube(a) +
          “</b><br>”);
          function square(k)
          {
               return k*k;
          }
          function cube(k)
          {
               return k*k*k;
          }
     </script>
</head>
</html>
```

Events

Events represent actions that JavaScript can react to. Event handlers are JavaScript functions that handle events. When JavaScript is used in HTML pages, JavaScript can "react" on these events.

HTML Events

An HTML event can be something the browser does, or something a user does. Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked
- Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

Mouse Events

Event	Description
onclick	The event occurs when the user clicks on an element
oncontextmenu	The event occurs when the user right-clicks on an element to open a context menu
ondblclick	The event occurs when the user double-clicks on an element
onmousedown	The event occurs when the user presses a mouse button over an element
onmouseenter	The event occurs when the pointer is moved onto an element
onmouseleave	The event occurs when the pointer is moved out of an element
onmousemove	The event occurs when the pointer is moving while it is over an element
onmouseover	The event occurs when the pointer is moved onto an element, or onto one of its children
onmouseout	The event occurs when a user moves the mouse pointer out of an element, or out of one of its children
onmouseup	The event occurs when a user releases a mouse button over an element

Keyboard Events

Event	Description
onkeydown	The event occurs when the user is pressing a key
onkeypress	The event occurs when the user presses a key
onkeyup	The event occurs when the user releases a key

Body/Frame/Object Events

Event	Description
onbeforeunload	The event occurs before the document is about to be unloaded
onload	The event occurs when an object has loaded
onresize	The event occurs when the document view is resized
onscroll	The event occurs when an element's scrollbar is being scrolled
onunload	The event occurs once a page has unloaded (for <body>)

Form Events

Event	Description
onblur	The event occurs when an element loses focus
onchange	The event occurs when the content of a form element, the selection, or the checked state have changed (for <input>, <keygen>, <select>, and <textarea>)
onfocus	The event occurs when an element gets focus
onfocusin	The event occurs when an element is about to get focus
onfocusout	The event occurs when an element is about to lose focus
oninput	The event occurs when an element gets user input

oninvalid	The event occurs when an element is invalid
onreset	The event occurs when a form is reset
onsearch	The event occurs when the user writes something in a search field (for <input="search">)
onselect	The event occurs after the user selects some text (for <input> and <textarea>)
onsubmit	The event occurs when a form is submitted

Accepting values from Text Box

In JavaScript we can refer the entered values from the text box which is defined in HTML forms. By taking the Name parameter we can perform this task. Simply we can give as follows

variable = document.formname.textboxname.value;

this means we can call any object on the form through document object. It just refers name of the form and name of the text box in which required value is stored. For example, form name is "Form1" and textbox name is "user" then you can assign its value in JavaScript variable as follows:

str = document.Form1.user.value;

You can also use **id** of the object to refer to entered values:

variable = document.getElementById(id).value;

Example: Program to accept a name and display the name on the screen.

```
<html>
      <head>
            <script type="text/javascript">
            function check() {
                  var txt = document.f1.name;
                  if(txt.value == "") {
                        alert("Please Enter your name");
                        txt.focus();
                  }
            }
            function fun() {
                  var txt = document.getElementById("name").value;
                  if(txt.value != "")
                  document.write("<h1>Hello, " + txt + " Welcome</h1>");
            }
            </script>
      </head>
      <body>
            <form name=f1>
            Enter your name:
            <input  type="text"  size=20  onblur="check()"  name="name"
            id="name"/>
            <input type="button" onclick="fun()" value="Click Here"/>
```

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

Program: Write a program to display the factorial of a given number.

```
<html>
<head><title> Recursion for Factorial </title>
     <script language = “JavaScript”>
          var a = window.prompt(“Enter Number for factorial:”, “0”);
          var num = parseInt(a);
          document.writeln(“Factorial  of  ”  +  num  +  “  is  ”  +
     factorial(num));
          function factorial (num) {
               if(num <= 1)
                    return 1;
               else
                    return num * factorial(num-1);
          }
     </script>
</head>
</html>
```

Objects in JavaScript

The following are the important objects of JavaScript:

String	Provides methods and properties related to string manipulation
Date	Provides methods related to date and time
Array	Provides methods related to array manipulations
Math	Provides properties and methods related to arithmetic
Boolean	Represents a Boolean
Document	Represents entire HTML document and can be used to access all elements in a page.
Window	Represents a browser window

String Object

JavaScript strings are used for storing and manipulating text.

String Properties

Property	Description
constructor	Returns the function that created the String object's prototype
length	Returns the length of a string
prototype	Allows you to add properties and methods to an object

String Methods

Method	Description
charAt()	Returns the character at the specified index (position)
charCodeAt()	Returns the Unicode of the character at the specified index

concat()	Joins two or more strings, and returns a copy of the joined strings
fromCharCode()	Converts Unicode values to characters
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string
localeCompare()	Compares two strings in the current locale
match()	Searches a string for a match against a regular expression, and returns the matches
replace()	Searches a string for a value and returns a new string with the value replaced
search()	Searches a string for a value and returns the position of the match
slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
substr()	Extracts a part of a string from a start position through a number of characters
substring()	Extracts a part of a string between two specified positions
toLocaleLowerCase()	Converts a string to lowercase letters, according to the host's locale
toLocaleUpperCase()	Converts a string to uppercase letters, according to the host's locale
toLowerCase()	Converts a string to lowercase letters
toString()	Returns the value of a String object
toUpperCase()	Converts a string to uppercase letters
trim()	Removes whitespace from both ends of a string
valueOf()	Returns the primitive value of a String object

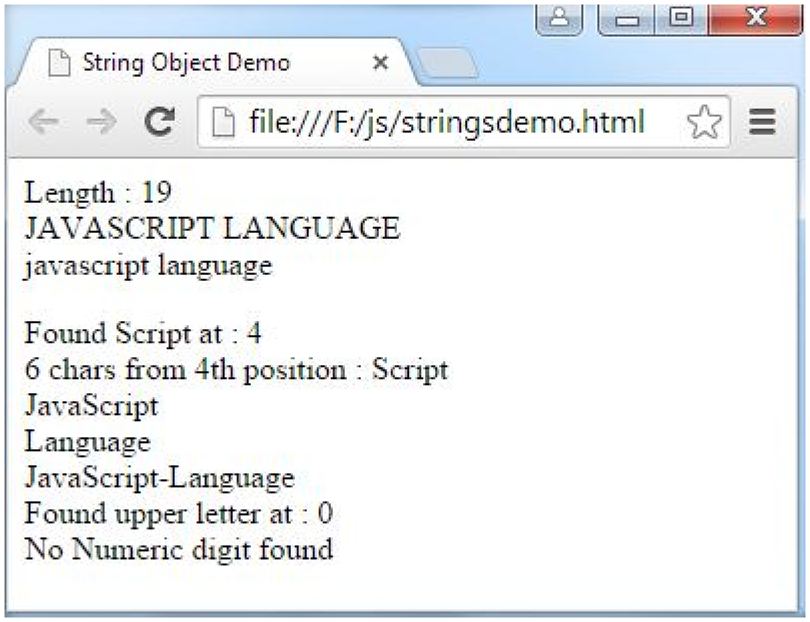
For example:

```
          var anystr = “30/8/2008”;
          var strings = anystr.split(“/”);
          var newstr = strings.join(“-“);
          document.writeln(“New Date ” + newstr);
Result as New Date 30-8-2008
```

Example: Program to illustrate String Object

```
<html>
     <title>String Object Demo</title>
     <script type="text/javascript">
          var st = "JavaScript Language";
          document.write("Length : " + st.length + "<br/>");
          document.write(st.toUpperCase()+"<br/>"+st.toLowerCase());
```

```
           document.write("<p>");
           document.write("Found Script at:"+st.indexOf("Script"));
           document.write("6 chars from 4th pos:"+st.substr(4,6));
           words = st.split(" ");
           for(i=0;i<words.length;i++)
                document.write(words[i] + "<br/>");
           s1 = words.join("-");
           document.write(s1 + "<br/>");
      document.write("Found upper letter at:"+st.search("[A-Z]"));
           if(st.match("[0-9]") == null)
                document.write("No Numeric digit found");
      </script>
</head>
</html>
```



```
Length : 19
JAVASCRIPT LANGUAGE
javascript language

Found Script at : 4
6 chars from 4th position : Script
JavaScript
Language
JavaScript-Language
Found upper letter at : 0
No Numeric digit found
```

Math Object

The Math object allows you to perform mathematical tasks on numbers. The Math object allows you to perform mathematical tasks. The Math object includes several mathematical methods.

Math Object Methods

Method	**Description**	**Examples**
abs(x)	Returns the absolute value of x	`Math.abs(-43) is 43`
acos(x)	Returns the arccosine of x, in radians	`Math.acos(0.5);` `1.0471975511965979`
asin(x)	Returns the arcsine of x, in radians	`Math.asin(0.5);` `0.5235987755982989`
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2	`Math.atan(2);` `1.1071487177940904`

	radians	
atan2(y,x)	Returns the arctangent of the quotient of its arguments	`Math.atan2(8, 4);` `1.1071487177940904`
ceil(x)	Returns x, rounded upwards to the nearest integer	`Math.ceil(4.4);` `//returns 5`
cos(x)	Returns the cosine of x (x is in radians)	`Math.cos(0.0) is 1.0`
exp(x)	Returns the value of E^x	`Math.exp(1);` `2.718281828459045`
floor(x)	Returns x, rounded downwards to the nearest integer	`Math.floor(4.7);` `//returns 4`
log(x)	Returns the natural logarithm (base E) of x	`Math.log(2);` `0.6931471805599453`
max(x,y,z,...,n)	Returns the number with the highest value	`Math.max(0,150,30,20,-8);` `// returns 150`
min(x,y,z,...,n)	Returns the number with the lowest value	`Math.min(0,150,30,20,-8);` `// returns -8`
pow(x,y)	Returns the value of x to the power of y	`Math.pow(2,3) is 8` `Math.pow(3,2) is 9`
random()	Returns a random number between 0 and 1	`Math.random()` `0.46184209338389337`
round(x)	Rounds x to the nearest integer	`Math.round(4.7);` `//returns 5` `Math.round(4.4);` `//returns 4`
sin(x)	Returns the sine of x (x is in radians)	`Math.sin(0.0) is 0.0`
sqrt(x)	Returns the square root of x	`Math.sqrt(9); 3`
tan(x)	Returns the tangent of an angle	`Math.tan(90);` `-1.995200412208242`

Math Constants

JavaScript provides 8 mathematical constants that can be accessed with the Math object:

Constant	**Description**	**Value**
Math.E	Euler's constant	2.718281828459045
Math.LN2	Natural Logarithm of 2	0.6931471805599453
Math.LN10	Natural Logarithm of 10	2.302585092994046
Math.LOG2E	Base 2 Logarithm of Euler's Constant	1.4426950408889634
Math.LOG10E	Base 10 Logarithm of Euler's Constant	0.4342944819032518
Math.PI	pi value that is result of 22/7	3.141592653589793
Math.SQRT1_2	Square root of 1/2(0.5)	0.7071067811865476
Math.SQRT2	Square root of 2	1.4142135623730951

Date Object

The Date object lets you work with dates (years, months, days, minutes, seconds, milliseconds)

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = Date();
</script>
```

Creating Date Objects

The Date object lets us work with dates. A date consists of a year, a month, a day, an hour, a minute, a second, and milliseconds. Date objects are created with the `new Date()` constructor. There are 4 ways of initiating a date:

1. **new Date()**
2. **new Date(milliseconds)**
   ```
   <script>
   var d = new Date(86400000);
   document.getElementById("demo").innerHTML = d;
   </script>
   Output: Fri Jan 02 1970 05:30:00 GMT+0530 (India Standard Time)
   ```
3. **new Date(dateString)**
   ```
   <script>
   var d = new Date("October 13, 2014 11:13:00");
   document.getElementById("demo").innerHTML = d;
   </script>
   Output: Mon Oct 13 2014 11:13:00 GMT+0530 (India Standard Time)
   ```
4. **new Date(year, month, day, hours, minutes, seconds, milliseconds)**
   ```
   <script>
   var d = new Date(99,5,24,11,33,30,0);
   document.getElementById("demo").innerHTML = d;
   </script>
   Output: Thu Jun 24 1999 11:33:30 GMT+0530 (India Standard Time)
   ```

Date Methods

When a Date object is created, a number of methods allow you to operate on it.

- When you display a date object in HTML, it is automatically converted to a string, with the toString() method.
- The toUTCString() method converts a date to a UTC string (a date display standard). [dateobject.toUTCString()]
  Example: Sun, 01 Mar 2015 13:34:18 GMT
- The toDateString() method converts a date to a more readable format:
  [dateobject.toDateString()] Example: Sun Mar 01 2015

Date Get Methods

Get methods are used for getting a part of a date.

Example: 01-03-2015 7.16 PM

Method	Description	Example
getDate()	Get the day as a number (1-31)	1
getDay()	Get the weekday as a number (0-6)	0 (Sun)
getFullYear()	Get the four digit year (yyyy)	2015
getHours()	Get the hour (0-23)	19
getMilliseconds()	Get the milliseconds (0-999)	870
getMinutes()	Get the minutes (0-59)	16
getMonth()	Get the month (0-11)	2
getSeconds()	Get the seconds (0-59)	29
getTime()	Get the time (milliseconds since January 1, 1970)	1425217724085

Date Set Methods

Set methods are used for setting a part of a date.

Method	Description	Example var d = new Date();
setDate()	Set the day as a number (1-31)	d.setDate(15);
setFullYear()	Set the year	d.setFullYear(2020, 0, 14);
setHours()	Set the hour (0-23)	d.setHours(22);
setMilliseconds()	Set the milliseconds (0-999)	d.setMilliseconds(566);
setMinutes()	Set the minutes (0-59)	d.setMinutes(24);
setMonth()	Set the month (0-11)	d.setMonth(6);
setSeconds()	Set the seconds (0-59)	d.setSeconds(45);
setTime()	Set the time (milliseconds since January 1, 1970)	d.setTime(1425217724085);

Array Object

JavaScript arrays are used to store multiple values in a single variable.

Displaying Arrays

```
<p id="demo"></p>
<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>
```

Syntax:

```
var var_name = new Array(size);    //bad
var var_name = ["value1", "value2", … "value-n"]; //good
var cars = new Array("value1", "value2", …); //bad
```

Arrays are Objects

Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays. But, JavaScript arrays are best described as arrays. Arrays use numbers to access its "elements". In this example, person[0] returns John:

```
Array: var person = ["John", "Doe", 46];
```

Objects use names to access its "members". In this example, person.firstName returns John:

```
Object: var person = {firstName:"John", lastName:"Doe", age:46};
```

In JavaScript, **arrays use numbered indexes**. In JavaScript, **objects use named indexes**.

```
var person = []
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
var x = person.length;         // person.length will return 3
var y = person[0];             // person[0] will return "John"
```

Converting Arrays to Strings

In JavaScript, all objects have the **valueOf()** and **toString()** methods.
The valueOf() method is the default behavior for an array. It returns an array as a string. For JavaScript arrays, valueOf() and toString() are equal.

The **join()** method also joins all array elements into a string. It behaves just like toString(), but you can specify the separator:

```
<p id="demo"></p>
<script>
var fruits = ["Banana", "Orange","Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
</script>
Output: Banana * Orange * Apple * Mango
```

Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements. This is what popping and pushing is: Popping items out of an array, or pushing items into an array.
The **pop()** method returns the string that was "popped out", removes the last element from an array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();   // Removes the last element ("Mango") from fruits
```

The **push()** method adds a new element to an array (at the end) and returns the new array length:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");  //  Adds a new element ("Kiwi") to fruits
```

Shifting Elements

Shifting is equivalent to popping, working on the first element instead of the last.
The **shift()** method removes the first element of an array, and "shifts" all other elements one place down. The shift() method returns the string that was "shifted out".

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift(); // Removes the first element "Banana" from fruits
```

The **unshift()** method adds a new element to an array (at the beginning), and "unshifts" older elements. The unshift() method returns the new array length.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon"); // Adds a new element "Lemon" to fruits
```

Changing Elements

Array elements are accessed using their index number:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi"; //Changes first element of fruits to "Kiwi"
```

The length property provides an easy way to append a new element to an array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Kiwi"; //Appends "Kiwi" to fruit
```

Deleting Elements

Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator delete:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0]; //Changes first element in fruits to undefined
```

Splicing an Array

The splice() method can be used to add new items to an array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

The first parameter (2) defines the position where new elements should be added (spliced in). The second parameter (0) defines how many elements should be removed. The rest of the parameters ("Lemon”, "Kiwi") define the new elements to be added.

Using splice() to Remove Elements

With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1); // Removes the first element of fruits
```

The first parameter (0) defines the position where new elements should be added (spliced in). The second parameter (1) defines how many elements should be removed. The rest of the parameters are omitted. No new elements will be added.

Sorting an Array

The sort() method sorts an array alphabetically:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();   //Sorts the elements of fruits
```

Numeric Sort

By default, the sort() function sorts values as strings. This works well for strings ("Apple" comes before "Banana"). However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1". Because of this, the sort() method will produce incorrect result when sorting numbers.

You can fix this by providing a compare function:

```
var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a-b});
```

Joining Arrays

The concat() method creates a new array by concatenating two arrays:

```
var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias","Linus"];
var myChildren = myGirls.concat(myBoys);
      // Concatenates (joins) myGirls and myBoys
```

The concat() method can take any number of array arguments:

```
var arr1 = ["Cecilie", "Lone"];
var arr2 = ["Emil", "Tobias","Linus"];
var arr3 = ["Robin", "Morgan"];
var myChildren = arr1.concat(arr2, arr3);
      // Concatenates arr1 with arr2 and arr3
```

Slicing an Array

The slice() method slices out a piece of an array:

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1, 3);
```

Reversing an Array

The reverse() method reverses the elements in an array. You can use it to sort an array in descending order:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();            // Sorts the elements of fruits
fruits.reverse();         // Reverses the order of the elements
```

Window Object

The window object represents an open window in a browser. If a document contain frames (<iframe> tags), the browser creates one window object for the HTML document, and one additional window object for each frame.

Window Object Properties

• closed	Returns true if window is closed
• history	Returns the History object for the window (See History object)
• innerHeight	Returns the inner height of a window's content area

- innerWidth Returns the inner width of a window's content area
- length Returns the number of <iframe> elements in the current window
- location Returns the Location object for the window (See Location object)
- name Sets or returns the name of a window
- navigator Returns the Navigator object for the window (See Navigator object)
- opener Returns a reference to the window that created the window
- outerHeight Returns the outer height of a window, including toolbars/scrollbars
- outerWidth Returns the outer width of a window, including toolbars/scrollbars
- pageXOffset/scrollX
  Returns the pixels the current document has been scrolled (horizontally) from the upper left corner of the window
- pageYOffset/scrollY
  Returns the pixels the current document has been scrolled (vertically) from the upper left corner of the window
- parent Returns the parent window of the current window
- screen Returns the Screen object for the window
- screenLeft Returns the horizontal coordinate of the window relative to the screen
- screenTop Returns the vertical coordinate of the window relative to the screen
- screenX Returns the horizontal coordinate of the window relative to the screen
- screenY Returns the vertical coordinate of the window relative to the screen
- self Returns the current window
- status Sets or returns the text in the statusbar of a window
- top Returns the topmost browser window

Window Object Methods

- alert() Displays an alert box with a message and an OK button
- clearInterval() Clears a timer set with setInterval()
- clearTimeout() Clears a timer set with setTimeout()
- close() Closes the current window
- confirm() Displays a dialog box with a message and an OK and a Cancel button
- focus() Sets focus to the current window
- moveBy() Moves a window relative to its current position
- moveTo() Moves a window to the specified position
- open() Opens a new browser window

print()	Prints the content of the current window
prompt()	Displays a dialog box that prompts the visitor for input
resizeBy()	Resizes the window by the specified pixels
resizeTo()	Resizes the window to the specified width and height
scrollBy()	Scrolls the document by the specified number of pixels
scrollTo()	Scrolls the document to the specified coordinates
setInterval()	Calls a function or evaluates an expression at specified intervals (in milliseconds)
setTimeout()	Calls a function or evaluates an expression after a specified number of milliseconds
stop()	Stops the window from loading

History Object

The history object contains the URLs visited by the user (within a browser window). The history object is part of the window object and is accessed through the window.history property.

History Object Properties & Methods

length	Returns the number of URLs in the history list [property]
back()	Loads the previous URL in the history list
forward()	Loads the next URL in the history list
go()	Loads a specific URL from the history list

Location Object

The location object contains information about the current URL. The location object is part of the window object and is accessed through the window.location property.

Location Object Properties

hash	Sets or returns the anchor part (#) of a URL
host	Sets or returns the hostname and port number of a URL
hostname	Sets or returns the hostname of a URL
href	Sets or returns the entire URL
origin	Returns the protocol, hostname and port number of a URL
pathname	Sets or returns the path name of a URL
port	Sets or returns the port number of a URL
protocol	Sets or returns the protocol of a URL
search	Sets or returns the querystring part of a URL

Location Object Methods

assign(url)	Loads a new document

• reload()	Reloads the current document
• replace(url)	Replaces the current document with a new one

JavaScript Window Navigator
The ***window.navigator*** object contains information about the visitor's browser.

Window Navigator: The window.navigator object can be written without the window prefix. Some examples:

- navigator.appName
- navigator.appCodeName
- navigator.platform

- ***Navigator Cookie Enabled***
    - The property **cookieEnabled** returns true if cookies are enabled, otherwise false.
- ***The Browser Names***
    - The properties **appName** and **appCodeName** return the name of the browser.
    - IE11, Chrome, Firefox, and Safari return appName "Netscape".
    - Chrome, Firefox, IE, Safari, and Opera all return appCodeName "Mozilla".
- ***The Browser Engine***
    - The property **product** returns the engine name of the browser.
- ***The Browser Version I***
    - The property **appVersion** returns version information about the browser:
- ***The Browser Version II***
    - The property **userAgent** also returns version information about the browser:
- ***The Browser Platform***
    - The property **platform** returns the browser platform (operating system).
- ***Is Java Enabled?***
    - The method **javaEnabled()** returns true if Java is enabled.

Program to display the Current Time updated using JavaScript

```
<!DOCTYPE html>
<html>
<head>
<script>
     function startTime() {
          var today=new Date();
          var h=today.getHours();
          var m=today.getMinutes();
          var s=today.getSeconds();
          m = checkTime(m);
          s = checkTime(s);
          document.getElementById('txt').innerHTML=
          h+":"+m+":"+s;
          var t = setTimeout(function(){startTime()},500);
     }
     function checkTime(i) {
          if (i<10) {i = "0" + i};
               // add zero in front of numbers < 10
          return i;
     }
</script>
</head>
<body onload="startTime()">
     Time is <div id="txt"></div>
</body>
</html>
```

JavaScript Quick Reference Card 1.0

Color key overleaf

ETKit Companion

An extended version of this cheatsheet is available in ETKit Companion – a 70 page all-color compendium of cheatsheets covering CSS, HTML, JavaScript, PHP and SQL. For more information please visit www.etkit.com.

Code Structure

```
var ...
//Global variable declarations
function funcA([param1,param2,...])
{
 var ...
 //Local variable declarations – visible in nested functions

 [function innerFuncA([iparam1,iparam2...])
 {
  var ...
  //Variables local to innerFuncA
 //your code here
 }]

aName='ExplainThat!';
//implicit global variable creation
//your code here
}
```

Nomenclature Rules

Function and variable names can consist of any alphanumeric character. $ and _ are allowed. The first character cannot be numeric. Many extended ASCII characters **are** allowed. There is no practical limit on name length. Names are case-sensitive.

If two or more variables or functions or a variable & a function are declared with the same name the last declaration obliterates all previous ones. Using a keyword as a variable or function name obliterates that keyword.

Visibility & Scope

Assignments without the use of the **var** keyword result in a new global variable of that name being created.

Variables declared with the **var** keyword outwith the body of a function are global. Variables declared with the **var** keyword inside the body of a function are local to that function. Local variables are visible to all nested functions.

Local entities hide globals bearing the same name.

Variable Types

string: var s = 'explainthat' or "explainthat"

number: var n = 3.14159, 100, 0...

boolean: var flag = false or true

object: var d = new Date();

function: var Greet = function sayHello() {alert('Hello')}

JavaScript is a weakly typed language – i.e. a simple assignment is sufficient to change the variable type. The **typeof** keyword can be used to check the current variable type.

Special Values

The special values **false**, **Infinity**, **NaN**, **null**, **true** & **undefined** are recognized. **null** is an object. **Infinity** and **NaN** are numbers.

Operators

Operator	Example	Result
+	3 + 2 'explain' + 'that'	5 explainthat
-	3 - 2	1
*	3*2	6
/	3/2	1.5
%	3%2	1
++	i = 2; i++[1], ++i[2]	3
--	i = 2; i--[1], --i[2]	1
== ==	3 = '3' 2 == 3	true false
===	3 === 3 3 === '3'	true false
<	2 < 3 'a' < 'A'	true false
<=	2 <= 3	true
>	2 > 3	false
>=	2 > 3	false
=	i = 2	i is assigned the value 2
+=	i+=1	3
-=	i-=1	2
i*=	i*=3	6
/=	i/=2	3
%=	i%=2	1
i = 2;j = 5;		
&& (AND)	(i <= 2) && (j < 7)	true
|| (OR)	(i%2 > 0) || (j%2 == 0)	false
! (NOT)	(i==2) && !(j%2 == 0)	true
i = 2;j = 7;		
& (bitwise)	i & j	2
| (bitwise)	i|j	7
^(XOR)	i^j	5
<<	2<<1	4
>>	2>>1	1
>>>	i=10 (binary 1010) i>>>2	2[3]

Internal Functions

decodeURI - reverses encodeURI

decodeURIComponent - reverses encodeURI...

encodeURI – encodes everything except **:/?&;,~@&=$+=_.*()#** and alphanumerics.

encodeURIComponent – encodes everything except **_.-!~*()** and alphaumerics.

escape – hexadecimal string encoding. Does not encode **+@/_-.*** and alphanumerics.

unescape – reverses **escape**

eval – evaluates JavaScript expressions

isNaN – true if the argument is not a number.

isFinite – isFinite(2/0) returns false

parseInt - parseInt(31.5°) returns 31

parseFloat - parseFloat(31.5°) returns 31.5

Array Object

length – number of elements in the array

concat – concatenates argument, returns new array.

join – returns elements as a string separated by argument (default is ,)

pop – suppress & return last element

push – adds new elements to end of array & returns new **length**.

reverse – inverts order of array elements

shift – suppress & return first element

slice – returns array slice. 1st arg is start position. 2nd arg is last position + 1

sort – alphanumeric sort if no argument. Pass sort function as argument for more specificity.

splice – discard and replace elements

unshift – append elements to start & return new **length**

Date Object

get#

getUTC#

set#

setUTC#

where # is one of Date, Day, FullYear, Hours, Milliseconds, Minutes, Month, Seconds, Time, TimezoneOffset

toDateString – the date in English.

toGMTString – the date & time in English.

toLocaleDateString – the date in the locale language.

toLocaleString – date & time in the locale language.

toLocaleTimeString – time in the locale language.

toTimeString – time in English.

toUTCString – date & time in UTC, English

valueOf – milliseconds since midnight 01 January 1970, UTC

Math Object

E, **LN10**, **LN2**, **LOG10E**, **LOG2E**, **PI**, **SQRT1_2**, **SQRT2**

abs – absolute value

#(n) - trigonometric functions

a#(n) - inverse trigonometric functions

where **#** is one of cos, sin or tan

ceil(n) – smallest whole number >= **n**

exp(n) – returns **e^n**

floor(n) – biggest whole number <= **n**

log(n) – logarithm of **n** to the base **e**

max(n_1,n_2) – bigger of n_1 and n_2

min(n_1,n_2) – smaller of n_1 and n_2

pow(a,b) - a^b

random – random number between 0 and 1

round(n) – **n** rounded down to closest integer

sqrt(n) – square root of n

Number Object

MAX_VALUE - ca 1.7977E+308

MIN_VALUE – ca 5E-324

NEGATIVE_INFINITY, **POSITIVE_INFINITY**

n.toExponential(m) – **n** in scientific notation with **m** decimal places.

n.toFixed() - **n** rounded to the **closest** whole number.

n.toPrecision(m) – **n** rounded to **m** figures.

Hexadecimal numbers are designated with the prefix **0x** or **0X.** e.g. 0xFF is the number 255.

String Object

length – number of characters in the string

s.charAt(n) – returns **s[n]**. **n** starts at 0

s.charCodeAt(n) – Unicode value of **s[n]**

s.fromCharCode(n_1,n_2..) - string built from Unicode values n_1, n_2...

s1.indexOf(s2,n) – location of **s2** in **s1** starting at position **n**

s1.lastIndexOf(s2) – location of **s2** in **s1** starting from

JavaScript Quick Reference Card 1.0

the end

s.substr(n_1,n_2) – returns substring starting from **n_1** upto character preceding n_2. No n_2 = extract till end. **n_1** < 0 = extract from end.

s.toLowerCase() - returns **s** in lower case characters

s.toUpperCase() - care to guess?

Escape Sequences

\n - new line, **\r** - carriage return, **\t** – tab character,

\ - \ character, \' - apostrophe, \" - quote

\uNNNN – Unicode character at NNNN

e.g. \u25BA gives the character ►

JavaScript in HTML

External JavaScript

```
<script type="text/javascript" defer="defer"
src="/scripts/explainthat.js"></script>
```

Inline JavaScript

```
<script type="text/javascript">
//your code here
</script>
```

Comments

/* Comments spanning multiple lines ***/**

// Simple, single line, comment

Conditional Execution

if (***Condition***) ***CodeIfTrue***;else ***CodeIfFalse***[4]

Multiline CodeIf# must be placed in braces, **{}**

```
switch (variable)
{
 case Value1:Code;
          break;
 case Value2:Code;
          break;
 .....
 default:Code;
}
```

variable can be boolean, number, string or even date.

(***condition***)**?**(***CodeIfTrue***)**:**(***CodeIfFalse***)

Parentheses are not necessary but advisable

Error Handling

Method 1: The onerror event

```
<script type="text/javascript">
function whenError(msg,url,lineNo){
 //use parameters to provide meaningful messages
}
window.onerror = whenError
</script>
```

Place this code in a **separate** **<script>..</script>** tag pair to trap errors occurring in other scripts. This technique blocks errors without taking corrective action.

Method 2: The **try..catch..finally** statement

```
function showLogValue(num){
 var s = 'No Error';
 try
 {if (num < 0) throw 'badnum';
  if (num == 0) throw 'zero'; }
 catch (err)
 { s = err;
  switch (err) {
   case 'badnum':num = -num;
              break;
  case 'zero':num = 1;
           break; }
 }
 [finally{ alert([s,Math.log(num)]);}]
}
```

The finally block is optional. The two techniques can be used in concert.

Looping

```
function whileLoop(num){
 while (num > 0)
 { alert(num);
  num--;}
}

function doLoop(num){
 do{
    alert(num);
    num--;
 }while (num > 0);
}

function forLoop(num){
 var i;
 for (i=0;i<num;i++){
    alert(num);
 }
}
```

break causes immediate termination of the loop.

loop statements after **continue** are skipped and the next execution of the loop is performed.

```
function forInLoop(){
 var s,x;
 for (x in document)
 {
  s=x + ' = ' + document[x];
  alert(s);
 }
}
```

This code is best tested in Opera which offers the option of stopping the script at each alert. In place of **document** any JavaScript object or an array can be used to loop through its properties/elements.

return

return causes immediate termination of the JavaScript function. If no value is returned, or if **return** is missing the function return type is **undefined.**

document Object

body - the body of the document

cookie - read/write the document cookies

domain – where was the document served from?

forms[] - array of all forms in the document

images[] - array of all images in the document

referrer – who pointed to this document?

URL – the URL for the document

getElementById(id) – element bearing ID of **id**

getElementsByName(n) – array of elements named **n**

getElementsByTagName(t) - array of **t** tagged elements

write – write plain or HTML text to the document

onload – occurs when the document is loaded

onunload – occurs when user browses away, tab is closed etc.

Element Object

By element we mean any HTML element retrieved using the **document**.**getElementBy#** methods.

attributes – all element attributes in an array

className – the CSS style assigned to the element

id – the **id** assigned to the element

innerHTML – HTML content of the element

innerText – content of the element shorn of all HTML tags. Does not work in Firefox

offset# – element dimensions (**# = Height/Width**) or location(**# = Left/Right**) in pixels

ownerDocument – take a guess

style – CSS style declaration

tagName – element tag type. Curiously, always in uppercase

textContent – the Firefox equivalent of **innerText**

location Object

host – URL of the site serving up the document

href – the entire URL to the document

pathname – the path to the document on the host

protocol – the protocol used, e.g. http

reload(p) - reload the document. From the cache if **p** is true.

replace(url) – replace the current document with the one at **url.** Discard document entry in browser history.

screen Object

height – screen height in pixels

width – screen width in pixels

window Object

alert(msg) – displays a dialog with **msg**

clearInterval(id) – clears interval **id** set by setInterval

clearTimeout(id) – clears timeout **id** set by setTimeout

confirm(msg) – shows a confirmation dialog

print() - prints the window contents

prompt(msg,[default]) – shows prompt dialog, optionally with default content. Returns content or **null**.

setInterval(expr,interval) – sets repeat at **interval** ms. The function **expr** is evaluated[5].

setTimeout(expr,time) Like **setInterval** but non-repeating. [5]

Notes

[1] Evaluates **after** use [2] Evaluates **before** use

[3] Zero-fill right shift [4] Note the semicolon!

[5] Passing arguments to function calls via **expr** is not well supported.

Color Coding

italics **–** user code **blue –** JavaScript Keywords

red – Option **object** – JavaScript DOM object

green – only numeric values **blue** **-** object properties

green – object methods **magenta** – object events

Tested with Internet Explorer 6+, Firefox 1.5+ & Opera 9.1+.