



Copyright©, 2007-2008 BrandsPatch LLC

<http://www.explainth.at>

Color key overleaf

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++, ++i ²	3
--	i = 2; i--, --i ²	1
==	3 == '3'	true
===	3 === 3 3 === '3'	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 encodeURIComponent...
encodeURI - encodes everything except :/?&;~@&=\$+~()*# and alphanumerics.
encodeURIComponent - encodes everything except _!~()* and alphanumerics.
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 eⁿ

floor(n) - biggest whole number <= n

log(n) - logarithm of n to the base e

max(n₁,n₂) - bigger of n₁ and n₂

min(n₁,n₂) - smaller of n₁ and n₂

pow(a,b) - a^b

random - random number between 0 and 1

round(n) - n rounded down to closest integer

sqr(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(n₁,n₂,...) - string built from Unicode values n₁, n₂,...

s.indexOf(s2,n) - location of s2 in s1 starting at position n

s.lastIndexOf(s2) - location of s2 in s1 starting from the end

s.substr(n₁,n₂) - returns substring starting from n₁ upto character preceding n₂. No n₂ = extract till end. n₁ < 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.getElementById** 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,[args]) - sets repeat at **interval** ms. The function **expr** is evaluated, optionally with **args** passed as parameters.

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

Notes

¹ Evaluates **after** use ² Evaluates **before** use

³ Zero-fill right shift ⁴ Note the semicolon!

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+.

If you find this reference card useful please help us by creating links to our site <http://www.explainth.at> where you will find other quick reference cards and many other free programming resources.