Addison-Wesley's

JavaScript Reference Card

Kathleen M. Goelz and Carol J. Schwartz, Rutgers University

Javascript: A scripting language designed to be integrated into HTML code to produce enhanced, dynamic, interactive web pages.

DATA TYPES

Definition: The classification of values based on the specific categories in which they are stored.

Primitive Types: String, Boolean, Integer, Floating Point, Null, Void

Composite Types: Object, Array, Function. Composite data types are in separate sections of the code.

NUMERIC

Integer: Positive or negative numbers with no fractional parts or decimal places.

Floating Point: Positive or negative numbers that contain a decimal point or exponential notations.

String: A sequence of readable characters or text, surrounded by single or double quotes.

Boolean: The logical values True/False, etc. used to compare data or make decisions.

Null: The variable does not have a value; nothing to report. Null is not the same as zero, which is a numeric value.

Casting: Moving the contents of a variable of one type to a variable of a different type. You don't move the contents to a different variable; it stays in the same variable but the data type is changed or "re-cast".

VARIABLES

Definition: A placeholder for storing data. In JavaScript, a declaration statement consists of the reserved word **var** and the name (identifier) of one or more variables.

Format:

```
var variable_name
```

[**var** command is used to *declare* (create) variables]

Examples:

```
var myHouseColor
var myAddress
var vacation_house, condominium,
    primaryResidence
```

Rules for Naming Variables:

1. Variables cannot be reserved words.
2. Variables must begin with a letter or underscore and cannot begin with symbols, numbers, or arithmetic notations.
3. Spaces cannot be included in a variable name.

Hints:

1. Although variables in JavaScript can be used without being declared, it is good programming practice to declare (initialize), all variables.
2. Variable names are case sensitive; for example *X* does not equal *x*.

PEARSON
Addison Wesley

9 780321 320759 90000
ISBN 0-321-32075-1

INITIALIZING VARIABLES

Use the declaration statement to assign a value to the variable. The value is on the right of the equal sign; the variable is on the left.

Format:

```
var  variable_name = value
```

Examples:

```
var  myHouseColor = "yellow"
```
[literal string value yellow assigned to variable `myHouseColor`]

```
var  myAddress = 473
```
[numeric value 473 assigned to variable `myAddress`]

```
var  bookTitle = "Time Capsule", cost =
     28.95, publisher = "Tucker Bay"
```
[multiple variables can be assigned in one statement]

DECISION MAKING AND CONTROL STRUCTURES

Definition: Statements and structures used to change the order in which computer operations will occur.

Types:

Conditional Branching	IF, IF-ELSE, IF-ELSE IF, SWITCH, WHILE, DO, FOR

CONDITIONALS

IF Statement: A conditional branching statement used to determine whether a stated condition is TRUE.

Format:

```
if  (condition)   {
        statements if condition is TRUE
    }
```

Example:

```
if  (score >= 65")  {
        grade = "Pass";
        message = "Congratulations";
}
```

IF-ELSE Statement: A conditional branching statement that includes a path to follow if the condition is TRUE and a path to follow if the condition is FALSE.

Format:

```
if   (condition)   {
   statements if condition is TRUE;
}
else    {
   statements if condition is FALSE;
}
```

Example:

```
if  (score >= 65) {
   grade = "Pass";
   message = "Congratulations";
}
else     {
   grade = "Fail"
   message = "Try again";
}
```

IF-ELSE IF Statement: A conditional branching statement that allows for more than two possible paths. The first time a true condition is encountered, the statement is executed and the remaining conditions will not be tested.

Format:

```
if  (condition) {
   Statements if condition is TRUE;
}
else if (condition)  {
   Statements if condition is TRUE;
}
else  {
   Statements if no prior condition is
     true;
}
```

Example:

```
if        (score>=90)  {
   grade="A";
}
else if  (score>=80)  {
   grade="B";
}
else if  (score>=70)  {
   grade="C";
}
else if  (score>=65)  {
   grade="D";
}
else                  {
   grade="F";
}
```

SWITCH Statement: An alternative to the IF-ELSE IF statement for handling multiple options. Compares the expression to the test values to find a match.

Format:

```
switch (expression or variable name)  {
  case label:
    statements if expression matches
     this label;
    break;
  case label:
    statements if expression matches
     this label;
    break;
  default:
    statements if expression does not
     match any label;
    break;
  }
```

Example:

```
switch (colorchoice)       {
  case "red":
    document.bgColor="red";
    break;
  case "blue":
    document.bgColor="blue";
    break;
  default:
    document.bgColor="white";
    break;
  }
```

LOOPS

Loops cause a segment of code to repeat until a stated condition is met. You can use any loop format for any type of code

FOR LOOP:

Format:

```
For (intialize; conditional test;
     increment/decrement)         {
    Statements to execute;
}
```

Example:

```
For (var i=0; i<=10; i++)         {
   document.write ("This is line " + i);
}
```

DO/WHILE LOOP:

Format:

```
do    {
   Statements to execute;
}
while (condition);
```

Example:

```
var i=0;
do    {
   document.write ("This is line " + i);
   i++;
}
while (i <=10);
```

WHILE LOOP:

Format:

```
while (condition) {
    Statements;
    Increment/decrement;
}
```

Example:

```
var i = 0;
while (i<=10)     {
    document.write ("This is line " + i);
    i++;
}
```

Hint: Watch out for infinite loops, which do not have a stopping condition or have a stopping condition that will never be reached.

OBJECTS

Definition: Objects are a composite data type which contain properties and methods. JavaScript contains built-in objects and allows the user to create custom objects.

Creating Objects: Use the **new** constructor

```
var X = new Array()
```

Examples:

date, time, math, strings, arrays

ARRAY OBJECT

Definition: Array object is a variable that stores multiple values. Each value is given an index number in the array and each value is referred to by the array name and the index number. Arrays, like simple variables, can hold any kind of data. You can leave the size blank when you create an array. The size of the array will be determined by the number of items placed in it.

Format:

```
var arrayname = new Array(size)
```

Hint: When you create an array, you create a new instance of the array object. All properties and methods of the array object are available to your new array.

Example:

```
var days = new Array (7)
```

This creates an array of seven elements using the array constructor.
The first item is `days[0]`, the last item is `days[6]`.

Initializing Arrays:

Array items can be treated as simple variables:

```
days[0] = "Sunday";
days[1] = "Monday";
```
etc.

STRING OBJECT

Definition: String object is created by assigning a string to a variable, or by using the new object constructor.

Example:

```
var name = "Carol";
var name = new String("Carol");
```

Properties:

`Length:`	returns the number of characters in the string
`Prototype:`	allows the user to add methods and properties to the string

Methods:

String formatting methods (similar to HTML formatting tags)

```
String.big
String.blink
String.italics
```

Substring methods (allow user to find, match, or change patterns of characters in the string)

```
indexOf()
charAt()
replace()
```

MATH OBJECT

Definition: Math object allows arithmetic calculations not supported by the basic math operators. Math is a built-in object that the user does not need to define.

Examples:

`Math.abs(number)`	returns absolute value of the numeric argument
`Math.cos(number)`	returns the cosine of the argument, in radians
`Math.round(number)`	rounds number to the nearest integer

DATE/TIME OBJECTS

Date object provides methods for getting or setting information about the date and time.

Note: Dates before January 1, 1970 are not supported.

FUNCTIONS

Definition: A pre-written block of code that performs a specific task. Some functions return values; others perform a task like sorting, but return no value. Function names follow the same rules as variables names. There may or may not be an argument or parameter in the parenthesis, but the parenthesis has to be there.

User-defined Functions:

Example:

`ParseInt()` or `ParseFloat()` convert a string to a number.

To create a function:

Format:

```
function name_of_function (arguments) {
     statements to execute when
      function is called;
}
```

Example:

```
function kilosToPounds (){
     pounds=kilos*2.2046;
}
```

This new function takes the value of the variable kilos, multiplies it by 2.2046, and assigns the result to the variable pounds.

To call a function: Give the name of the function followed by its arguments, if any

`ParseInt(X);` converts the data stored in the variable X into a numeric value.

`kilosToPounds(17);` converts 17 kilos to the same mass in pounds, returning the value 37.4782.

METHODS

Definition: A special kind of function used to describe or instruct the way the object behaves. Each object type in JavaScript has associated methods available.

Examples:

```
array.sort();
document.write();
string.length();
```

Calling: To call or use a method, state the method name followed by its parameters in parentheses.

Example:

```
document.write(“Hello, world!”);
```

PUTTING IT TOGETHER: JAVASCRIPT AND HTML ON THE WEB

Cookies: Text-file messages stored by the browser on the user’s computer

Purpose: To identify the user, store preferences, and present customized information each time the user visits the page

Types:

Temporary (transient, session) — stored in temporary memory and available only during active browser session

Persistent (permanent, stored) — remain on user’s computer until deleted or expired

Browser Detection: A script written to determine which browser is running; determine if the browser has the capabilities to load the webpage and support the javascript code; and, if needed, load alternate javascript code to match the browser and platform.

Sniffing: A script written to determine whether a specific browser feature is present; i.e., detecting the presence of Flash before loading a webpage.

Event Handling: Use HTML event attributes (mouseover, mouse click, etc.) and connect event to a JavaScript function called an event handler

OPERATORS

ARITHMETIC

+	addition	adds two numbers
–	subtraction	subtracts one number from another
*	multiplication	multiplies two numbers
/	division	divides one number by another
%	modulus	returns the integer remainder after dividing two numbers
++	increment	adds one to a numeric variable
—	decrement	subtracts one from a numeric variable

STRING

+	concatenation	concatenates or joins two strings or other elements
+=	concatenation/ assignment	concatenates two string variables and assigns the result to the first variable

LOGICAL

&&	logical AND	Compares two operands; returns true if both are true, otherwise returns false
||	logical OR	Compares two operands; returns true if either operand is true, otherwise returns false
!	logical NOT	Returns false if its operand can be converted to true, otherwise returns false

COMPARISON

==	Returns true if the operands are equal
!=	Returns true if the operands are not equal
===	Returns true if the operands are equal and the same data type
!==	Returns true if the operands are not equal and/or not the same data type
>	Returns true if the first operand is greater than the second
>=	Returns true if the first operand is greater than or equal to the second
<	Returns true if the first operand is less than the second
<=	Returns true if the first operand is less than or equal to the second

ASSIGNMENT

=	Assigns the value of the seond operand to the first operand
+=	Adds two numeric operands and assigns the result to the first operand
–=	Subtracts the second operand from the first, and assigns the result to the first
*=	Multiplies two operands, assigns the result to the first
/=	Divides the first operand by the second, assigns the result to the first
%=	Finds the modulus of two numeric operands, and assigns the result to the first

RESERVED WORDS

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	shor	while
do	import	static	with
double	in	super	

JavaScript Quick Reference Card 1.03

ExplainThat!™

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++[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(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

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

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

s1.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?

JavaScript Quick Reference Card 1.03

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,[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.