Beginning JavaScript®, Fifth Edition
By Jeremy McPeak and Paul Wilton

B

JavaScript Core Reference

This appendix outlines the syntax of all the JavaScript core language functions and objects with their properties and methods. If changes have occurred between versions, they have been noted.

BROWSER REFERENCE

The following table outlines which JavaScript version is in use and in which browser it is used. Note that earlier versions of Internet Explorer implemented Jscript, Microsoft's version of JavaScript. However, Jscript's features are relatively the same as JavaScript.

JAVASCRIPT VERSION	MOZILLA FIREFOX	INTERNET EXPLORER	CHROME	SAFARI	OPERA
1.0		3.0			
1.1					
1.2					
1.3		4.0			
1.4					
1.5	1.0	5.5, 6, 7, 8	1–10	3-5	6, 7, 8, 9
1.6	1.5				
1.7	2.0		28		
1.8	3.0				11.5
1.8.1	3.5				
1.8.2	3.6				
1.8.5	4	9	32	6	11.6

RESERVED WORDS

Various words and symbols are reserved by JavaScript. These words cannot be used as variable names, nor can the symbols be used within them. They are listed in the following table.

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

Other Identifiers to Avoid

It is best to avoid the use of the following identifiers as variable names.

JavaScript 1.0

```
abs acos anchor asin atan atan2 big blink bold ceil charAt comment cos Date E
escape eval exp fixed floor fontcolor fontsize getDate getDay getHours getMinutes
getMonth getSeconds getTime getTimezoneOffset getYear indexOf isNaN italics
lastIndexOf link log LOG10E LOG2E LN10 LN2 Math max min Object parse parseFloat
parseInt PI pow random round, setDate setHours setMinutes setMonth setSeconds
setTime setYear sin slice small sqrt SQRT1_2 SQRT2 strike String sub substr
substring sup tan toGMTString toLocaleString toLowerCase toUpperCase unescape UTC
```

JavaScript 1.1

```
caller className constructor java JavaArray JavaClass JavaObject JavaPackage
join length MAX_VALUE MIN_VALUE NaN NEGATIVE_INFINITY netscape Number POSITIVE_
INFINITY prototype reverse sort split sun toString valueOf
```

JavaScript 1.2

```
arity callee charCodeAt compile concat exec fromCharCode global ignoreCase index
input label lastIndex lastMatch lastParen leftContext match multiline Number
Packages pop push RegExp replace rightContext search shift slice splice source
String test unshift unwatch watch
```

JavaScript 1.3

```
apply call getFullYear getMilliseconds getUTCDate getUTCDay getUTCFullYear
getUTCHours getUTCMilliseconds getUTCMinutes getUTCMonth getUTCSeconds Infinity
isFinite NaN setFullYear setMilliseconds setUTCDate setUTCFullYear setUTCHours
setUTCMilliseconds setUTCMinutes setUTCMonth setUTCSeconds toSource toUTCString
undefined
```

JAVASCRIPT OPERATORS

The following sections list the various operators available to you in JavaScript.

Assignment Operators

Assignment operators allow you to assign a value to a variable. The following table lists the different assignment operators you can use.

NAME	INTRODUCED	MEANING
Assignment	JavaScript 1.0	Sets variable `v1` to the value of variable `v2`. `var v1 = v2;`
Shorthand addition *or* Shorthand concatenation same as `v1 = v1 + v2`	JavaScript 1.0	`v1 += v2`
Shorthand subtraction same as `v1 = v1 – v2`	JavaScript 1.0	`v1 –= v2`
Shorthand multiplication same as `v1 = v1 * v2`	JavaScript 1.0	`v1 *= v2`
Shorthand division same as `v1 = v1 / v2`	JavaScript 1.0	`v1 /= v2`
Shorthand modulus same as `v1 = v1 % v2`	JavaScript 1.0	`v1 %= v2`
Shorthand left-shift same as `v1 = v1 << v2`	JavaScript 1.0	`v1 <<= v2`
Shorthand right-shift same as `v1 = v1 >> v2`	JavaScript 1.0	`v1 >>= v2`
Shorthand zero-fill right-shift same as `v1 = v1 >>> v2`	JavaScript 1.0	`v1 >>>= v2`
Shorthand AND same as `v1 = v1 & v2`	JavaScript 1.0	`v1 &= v2`
Shorthand XOR same as `v1 = v1 ^ v2`	JavaScript 1.0	`v1 ^= v2`
Shorthand OR same as `v1 = v1 | v2`	JavaScript 1.0	`v1 |= v2`

Comparison Operators

Comparison operators allow you to compare one variable or value with another. Any comparison statement returns a boolean value.

NAME	INTRODUCED	MEANING
Equal	JavaScript 1.0	`v1 == v2` True if two operands are strictly equal or equal once cast to the same type.
Not equal	JavaScript 1.0	`v1 != v2` True if two operands are not strictly equal or not equal once cast to the same type.

Greater than	JavaScript 1.0	`v1 > v2` True if left-hand side (LHS) operand is greater than right-hand side (RHS) operand.
Greater than or equal to	JavaScript 1.0	`v1 >= v2` True if LHS operand is greater than or equal to RHS operand.
Less than	JavaScript 1.0	`v1 < v2` True if LHS operand is less than RHS operand.
Less than or equal to	JavaScript 1.0	`v1 <= v2` True if LHS operand is less than or equal to RHS operand.
Strictly equal	JavaScript 1.3	`v1 === v2` True if operands are equal and of the same type.
Not strictly equal	JavaScript 1.3	`v1 !== v2` True if operands are not strictly equal.

Arithmetic Operators

Arithmetic operators allow you to perform arithmetic operations between variables or values.

NAME	INTRODUCED	MEANING
Addition	JavaScript 1.0	`v1 + v2` Sum of `v1` and `v2`. (Concatenation of `v1` and `v2`, if either operand is a string.)
Subtraction	JavaScript 1.0	`v1 – v2` Difference between `v1` and `v2`.
Multiplication	JavaScript 1.0	`v1 * v2` Product of `v1` and `v2`.
Division	JavaScript 1.0	`v1 / v2` Quotient of `v2` into `v1`.
Modulus	JavaScript 1.0	`v1 % v2` Integer remainder of dividing `v1` by `v2`.

continues

(continued)

NAME	INTRODUCED	MEANING
Prefix increment	JavaScript 1.0	`++v1 * v2(v1 + 1) * v2` Note: `v1` will be left as `v1 + 1`.
Postfix increment	JavaScript 1.0	`v1++ * v2(v1 * v2)` `v1` is then incremented by `1`.
Prefix decrement	JavaScript 1.0	`-- v1 * v2(v1 – 1) * v2` Note: `v1` is left as `v1 - 1`.
Postfix decrement	JavaScript 1.0	`v1 --  * v2(v1 * v2)` `v1` is then decremented by `1`.

Bitwise Operators

Bitwise operators work by converting values in `v1` and `v2` to 32-bit binary numbers and then comparing the individual bits of these two binary numbers. The result is returned as a normal decimal number.

NAME	INTRODUCED	MEANING
Bitwise AND	JavaScript 1.0	`v1 & v2` The bitwise AND lines up the bits in each operand and performs an AND operation between the two bits in the same position. If both bits are `1`, the resulting bit in this position of the returned number is `1`. If either bit is `0`, the resulting bit in this position of the returned number is `0`.
Bitwise OR	JavaScript 1.0	`v1 | v2` The bitwise OR lines up the bits in each operand and performs an OR operation between the two bits in the same position. If either bit is `1`, the resulting bit in this position of the returned number is `1`. If both bits are `0`, the resulting bit in this position of the returned number is `0`.
Bitwise XOR	JavaScript 1.0	`v1 ^ v2` The bitwise XOR lines up the bits in each operand and performs an XOR operation between the two bits in the same position. The resulting bit in this position is `1` only if one bit from both operands is `1`. Otherwise, the resulting bit in this position of the returned number is `0`.
Bitwise NOT	JavaScript 1.0	`v1 ~ v2` Inverts all the bits in the number.

Bitwise Shift Operators

These work by converting values in `v1` to 32-bit binary numbers and then moving the bits in the number to the left or the right by the specified number of places.

NAME	INTRODUCED	MEANING
Left-shift	JavaScript 1.0	`v1 << v2` Shifts `v1` to the left by `v2` places, filling the new gaps in with zeros.
Sign-propagating right-shift	JavaScript 1.4	`v1 >> v2` Shifts `v1` to the right by `v2` places, ignoring the bits shifted off the number.
Zero-fill right-shift	JavaScript 1.0	`v1 >>> v2` Shifts `v1` to the right by `v2` places, ignoring the bits shifted off the number and adding `v2` zeros to the left of the number.

Logical Operators

These should return one of the boolean literals, `true` or `false`. However, this may not happen if `v1` or `v2` is neither a boolean value nor a value that easily converts to a boolean value, such as `0`, `1`, `null`, the empty string, or `undefined`.

NAME	INTRODUCED	MEANING
Logical AND	JavaScript 1.0	`v1 && v2` Returns `true` if both `v1` and `v2` are `true`, or `false` otherwise. Will not evaluate `v2` if `v1` is `false`.
Logical OR	JavaScript 1.0	`v1 || v2` Returns `false` if both `v1` and `v2` are `false`, or `true` if one operand is `true`. Will not evaluate `v2` if `v1` is `true`.
Logical NOT	JavaScript 1.0	`!v1` Returns `false` if `v1` is `true`, or `true` otherwise.

Object Operators

JavaScript provides a number of operators to work with objects. The following table lists them.

NAME	INTRODUCED	MEANING
`delete`	JavaScript 1.2	`delete obj` Deletes an object, one of its properties, or the element of an array at the specified index. Also deletes variables *not* declared with the `var` keyword.
`in`	JavaScript 1.4	`for (prop in somObj)` Returns `true` if `someObj` has the named property.
`instanceof`	JavaScript 1.4	`someObj instanceof ObjType` Returns `true` if `someObj` is of type `ObjType`; otherwise, returns `false`.
`new`	JavaScript 1.0	`new ObjType()` Creates a new instance of an object with type `ObjType`.
`this`	JavaScript 1.0	`this.property` Refers to the current object.

Miscellaneous Operators

The following table lists miscellaneous operators.

NAME	INTRODUCED	MEANING
Conditional operator	JavaScript 1.0	`(evalquery) ? v1 : v2` If `evalquery` is `true`, the operator returns `v1`; otherwise it returns `v2`.
Comma operator	JavaScript 1.0	`var v3 = (v1 + 2, v2 * 2)` Evaluates both operands while treating the two as one expression. Returns the value of the second operand. In this example, `v3` holds the resulting value of `v2 * 2`.
`typeof`	JavaScript 1.1	`typeof v1` Returns a string holding the type of `v1`, which is not evaluated.
`void`	JavaScript 1.1	`void(eva1)` Evaluates `eval1` but does not return a value.

Operator Precedence

Does `1 + 2 * 3 = 1 + (2 * 3) = 7` or does it equal `(1 + 2) * 3 = 9`?

Operator precedence determines the order in which operators are evaluated. For example, the multiplicative operator (`*`) has a higher precedence than the additive operator (`+`). Therefore, the correct answer to the previous question is:

```
1 + (2 * 3)
```

The following table lists the operator precedence in JavaScript from highest to lowest. The third column explains whether to read `1+2+3+4` as `((1+2)+3)+4` (left to right) or `1+(2+(3+(4)))` (right to left).

OPERATOR TYPE	OPERATORS	EVALUATION ORDER FOR LIKE ELEMENTS
Member	`.` or `[]`	Left to right
Create instance	`new`	Right to left
Function call	`()`	Left to right
Increment	`++`	N/a
Decrement	`--`	N/a
Logical not	`!`	Right to left
Bitwise not	`~`	Right to left
Unary +	`+`	Right to left
Unary –	`–`	Right to left
Type of	`typeof`	Right to left
Void	`void`	Right to left
Delete	`delete`	Right to left
Multiplication	`*`	Left to right
Division	`/`	Left to right
Modulus	`%`	Left to right
Addition	`+`	Left to right
Subtraction	`–`	Left to right
Bitwise shift	`<<, >>, >>>`	Left to right
Relational	`<, <=, >, >=`	Left to right
In	`in`	Left to right
Instance of	`instanceof`	Left to right

continues

(continued)

OPERATOR TYPE	OPERATORS	EVALUATION ORDER FOR LIKE ELEMENTS
Equality	`==, !=, ===, !===`	Left to right
Bitwise AND	`&`	Left to right
Bitwise XOR	`^`	Left to right
Bitwise OR	`|`	Left to right
Logical AND	`&&`	Left to right
Logical OR	`||`	Left to right
Conditional	`?:`	Right to left
Assignment	`=, +=, -=, *=, /=, %=, <<=, >>=, >>>=, &=, ^=, |=`	Right to left
Comma	`,`	Left to right

JAVASCRIPT STATEMENTS

The following tables describe core JavaScript statements.

Block

JavaScript blocks start with an opening curly brace ({) and end with a closing curly brace (}). Block statements are meant to make the contained single statements execute together, such as the body of a function or a condition.

STATEMENT	INTRODUCED	DESCRIPTION
`{ }`	JavaScript 1.5	Used to group statements as delimited by the curly brackets.

Conditional

The following table lists conditional statements for JavaScript as well as the version in which they were introduced.

STATEMENT	INTRODUCED	DESCRIPTION
`if`	JavaScript 1.2	Executes a block of code if a specified condition is `true`.
`else`	JavaScript 1.2	The second half of an `if` statement. Executes a block of code if the result of the `if` statement is `false`.
`switch`	JavaScript 1.2	Specifies various blocks of statements to be executed depending on the value of the expression passed in as the argument.

Declarations

These keywords declare variables or functions in JavaScript code.

STATEMENT	INTRODUCED	DESCRIPTION
`var`	JavaScript 1.0	Used to declare a variable. Initializing it to a value is optional at the time of declaration.
`function`	JavaScript 1.0	Used to declare a function with the specified parameters, which can be strings, numbers, or objects. To return a value, the function must use the `return` statement.

Loop

Loops execute a block of code while a specified condition is `true`.

STATEMENT	INTRODUCED	DESCRIPTION
`do...while`	JavaScript 1.2	Executes the statements specified until the test condition after the `while` evaluates to `false`. The statements are executed at least once because the test condition is evaluated last.
`for`	JavaScript 1.0	Creates a loop controlled according to the three optional expressions enclosed in the parentheses after the `for` and separated by semicolons. The first of these three expressions is the initial-expression, the second is the test condition, and the third is the increment-expression.
`for...in`	JavaScript 1.0	Used to iterate over all the properties of an object using a variable. For each property the specified statements within the loop are executed.
`while`	JavaScript 1.0	Executes a block of statements if a test condition evaluates to `true`. The loop then repeats, testing the condition with each repeat, ceasing if the condition evaluates to `false`.
`break`	JavaScript 1.0	Used within a `while` or `for` loop to terminate the loop and transfer program control to the statement following the loop. Can also be used with a `label` to `break` to a particular program position outside of the loop.
`label`	JavaScript 1.2	An identifier that can be used with `break` or `continue` statements to indicate where the program should continue execution after the loop execution is stopped.

Execution Control Statements

Code execution is controlled in a variety of ways. In addition to the conditional and loop statements, the following statements also contribute to execution control.

STATEMENT	INTRODUCED	DESCRIPTION
`continue`	JavaScript 1.0	Used to stop execution of the block of statements in the current iteration of a `while` or `for` loop; execution of the loop continues with the next iteration.
`return`	JavaScript 1.0	Used to specify the value to be returned by a function.
`with`	JavaScript 1.0	Specifies the default object for a block of code.

Exception Handling Statements

Errors are a natural part of programming, and JavaScript provides you with the means to catch errors and handle them gracefully.

STATEMENT	INTRODUCED	DESCRIPTION
`Throw`	JavaScript 1.4	Throws a custom exception defined by the user.
`try... catch... finally`	JavaScript 1.4	Executes the statements in the `try` block; if any exceptions occur, these are handled in the `catch` block. The `finally` block allows you to stipulate statements that will be executed after both the `try` and `catch` statements.

Other Statements

The following table lists other JavaScript statements and when they were introduced.

STATEMENT	INTRODUCED	DESCRIPTION
`// single line comment`	JavaScript 1.0	Single lines of notes that are ignored by the script engine and that can be used to explain the code.
`/* multi-line comment */`	JavaScript 1.0	Multiple lines of notes that are ignored by the script engine and that can be used to explain the code.

TOP-LEVEL PROPERTIES AND FUNCTIONS

These are core properties and functions, which are not associated with any lower-level object, although in the terminology used by ECMAScript and by Jscript, they are described as properties and methods of the global object.

The top-level properties were introduced in JavaScript 1.3, but in previous versions, `Infinity` and `NaN` existed as properties of the `Number` object.

Top-Level Properties

PROPERTY	INTRODUCED	DESCRIPTION
`Infinity`	JavaScript 1.3	Returns infinity.
`NaN`	JavaScript 1.3	Returns a value that is not a number.
`undefined`	JavaScript 1.3	Indicates that a value has not been assigned to a variable.

Top-Level Functions

FUNCTION	INTRODUCED	DESCRIPTION
`decodeURI()`	JavaScript 1.5	Used to decode a URI encoded with `encodeURI()`.
`decodeURIcomponent()`	JavaScript 1.5	Used to decode a URI encoded with `encodeURIComponent()`.
`encodeURI()`	JavaScript 1.5	Used to compose a new version of a complete URI, replacing each instance of certain characters. It is based on the UTF-8 encoding of the characters.
`encodeURIComponent()`	JavaScript 1.5	Used to compose a new version of a complete URI by replacing each instance of the specified character with escape sequences. Representation is via the UTF encoding of the characters.
`escape()`	JavaScript 1.0	Used to encode a string in the ISO Latin-1 character set; for example, to add to a URL.
`eval()`	JavaScript 1.0	Returns the result of the JavaScript code, which is passed in as a string parameter.
`isFinite()`	JavaScript 1.3	Indicates whether the argument is a finite number.
`isNaN()`	JavaScript 1.1	Indicates if the argument is not a number.
`Number()`	JavaScript 1.2	Converts an object to a number.
`parseFloat()`	JavaScript 1.0	Parses a string and returns it as a floating-point number.

continues

(continued)

FUNCTION	INTRODUCED	DESCRIPTION
`parseInt()`	JavaScript 1.0	Parses a string and returns it as an integer. An optional second parameter specifies the base of the number to be converted.
`String()`	JavaScript 1.2	Converts an object to a string.
`unescape()`	JavaScript 1.0	Returns the ASCII string for the specified hexadecimal encoding value.

JAVASCRIPT CORE OBJECTS

This section describes the objects available in the JavaScript core language and their methods and properties.

Array

The `Array` object represents an array of variables. It was introduced in JavaScript 1.1. An `Array` object can be created with the `Array` constructor:

```
var objArray = new Array(10);            // an array of 11 elements
var objArray = new Array("1", "2", "4"); // an array of 3 elements
```

Arrays can also be created using array literal syntax:

```
var objArray = [];
```

Literal syntax is the preferred method of creating an array.

Properties

PROPERTY	INTRODUCED	DESCRIPTION
`constructor`	JavaScript 1.1	Used to reference the constructor function for the object.
`length`	JavaScript 1.1	Returns the number of elements in the array.
`prototype`	JavaScript 1.1	Returns the prototype for the object, which can be used to extend the object's interface.

> ***NOTE*** *Square brackets (`[]`) surrounding a parameter means that parameter is optional.*

Methods

METHOD	INTRODUCED	DESCRIPTION
`concat(value1 [, value2,...])`	JavaScript 1.2	Concatenates two arrays and returns the new array thus formed.
`every(testFn(element, index, array))`	JavaScript 1.6	Iterates over the array, executing `testFn()` on every element. Returns `true` if all iterations return `true`. Otherwise, it returns `false`.
`filter(testFn(element, index, array))`	JavaScript 1.6	Iterates over the array, executing `testFn()` on every element. Returns a new array of elements that pass `testFn()`.
`foreach(fn(element, index, array))`	JavaScript 1.6	Iterates over the array, executing `fn()` on every element.
`indexOf(element [, startIndex])`	JavaScript 1.6	Returns an index of the specified element if found, or `-1` if not found. Starts at `startIndex` if specified.
`join([separator])`	JavaScript 1.1	Joins all the elements of an array into a single string delimited by a separator if specified.
`lastIndexOf(element[, startIndex])`	JavaScript 1.6	Searches an array starting at the last element and moves backwards. Returns an index of the specified element if found, or `-1` if not found. Starts at `startIndex` if specified.
`map(fn(element, index, array))`	JavaScript 1.6	Iterates over the array, executing `fn()` on every element. Returns a new array based on the outcome of `fn()`.
`pop()`	JavaScript 1.2	Pops the last element from the end of the array and returns that element.
`push(value1 [, value2, ...])`	JavaScript 1.2	Pushes one or more elements onto the end of the array and returns the new length of the array. The array's new `length` is returned.
`reverse()`	JavaScript 1.1	Reverses the order of the elements in the array, so the first element becomes the last and the last becomes the first.
`shift()`	JavaScript 1.2	Removes the first element from the beginning of the array and returns that element.
`slice(startIndex [, endIndex])`	JavaScript 1.2	Returns a slice of the array starting at the start index and ending at the element before the end index.

continues

(continued)

METHOD	INTRODUCED	DESCRIPTION
`some(testFn(element, index, array))`	JavaScript 1.6	Iterates over the array, executing `testFn()` on every element. Returns `true` if at least one result of `testFn()` is `true`.
`sort([sortFn(a,b)])`	JavaScript 1.1	Sorts the elements of the array. Executes `sortFn()` for sorting if it is provided.
`splice(startIndex [, length, value1, ...)`	JavaScript 1.2	Removes the amount of elements denoted by `length` starting at `startIndex`. Provided values replace the deleted elements. Returns the deleted elements.
`toString()`	JavaScript 1.1	Converts the `Array` object into a string.
`unshift(value1 [, value2, ...])`	JavaScript 1.2	Adds elements to the beginning of the array and returns the new length.
`valueOf()`	JavaScript 1.1	Returns the primitive value of the array.

Boolean

The `Boolean` object is used as a wrapper for a boolean value. It was introduced in JavaScript 1.1. It is created with the `Boolean` constructor, which takes as a parameter the initial value for the object (if this is not a boolean value, it will be converted into one).

Falsey values are `null`, `undefined`, `""`, and `0`. All other values are considered truthy.

Properties

PROPERTY	INTRODUCED	DESCRIPTION
`constructor`	JavaScript 1.1	Specifies the function that creates an object's prototype.
`prototype`	JavaScript 1.1	Returns the prototype for the object, which can be used to extend the object's interface.

Methods

METHOD	INTRODUCED	DESCRIPTION
`toString()`	JavaScript 1.1	Converts the `Boolean` object into a string.
`valueOf()`	JavaScript 1.1	Returns the primitive value of the `Boolean` object.

Date

The `Date` object is used to represent a given date-time. It was introduced in JavaScript 1.0.

Properties

PROPERTY	INTRODUCED	DESCRIPTION
`constructor`	JavaScript 1.1	Used to reference the constructor function for the object.
`prototype`	JavaScript 1.1	Returns the prototype for the object, which can be used to extend the object's interface.

Methods

METHOD	INTRODUCED	DESCRIPTION
`getDate()`	JavaScript 1.0	Retrieves the date in the month from the `Date` object.
`getDay()`	JavaScript 1.0	Retrieves the day of the week from the `Date` object.
`getFullYear()`	JavaScript 1.3	Retrieves the full year from the `Date` object.
`getHours()`	JavaScript 1.0	Retrieves the hour of the day from the `Date` object.
`getMilliseconds()`	JavaScript 1.3	Retrieves the number of milliseconds from the `Date` object.
`getMinutes()`	JavaScript 1.0	Retrieves the number of minutes from the `Date` object.
`getMonth()`	JavaScript 1.0	Retrieves the month from the `Date` object.
`getSeconds()`	JavaScript 1.0	Retrieves the number of seconds from the `Date` object.
`getTime()`	JavaScript 1.0	Retrieves the number of milliseconds since January 1, 1970 00:00:00 from the `Date` object.
`getTimezoneOffset()`	JavaScript 1.0	Retrieves the difference in minutes between the local time zone and universal time (UTC).
`getUTCDate()`	JavaScript 1.3	Retrieves the date in the month from the `Date` object adjusted to universal time.
`getUTCDay()`	JavaScript 1.3	Retrieves the day of the week from the `Date` object adjusted to universal time.
`getUTCFullYear()`	JavaScript 1.3	Retrieves the year from the `Date` object adjusted to universal time.

continues

(continued)

METHOD	INTRODUCED	DESCRIPTION
`getUTCHours()`	JavaScript 1.3	Retrieves the hour of the day from the `Date` object adjusted to universal time.
`getUTCMilliseconds()`	JavaScript 1.3	Retrieves the number of milliseconds from the `Date` object adjusted to universal time.
`getUTCMinutes()`	JavaScript 1.3	Retrieves the number of minutes from the `Date` object adjusted to universal time.
`getUTCMonth()`	JavaScript 1.3	Retrieves the month from the `Date` object adjusted to universal time.
`getUTCSeconds()`	JavaScript 1.3	Retrieves the number of seconds from the `Date` object adjusted to universal time.
`getYear()`	JavaScript 1.0	Retrieves the year from the `Date` object.
`parse(dateString)`	JavaScript 1.0	Retrieves the number of milliseconds in a date since January 1, 1970 00:00:00, local time.
`setDate(dayOfMonth)`	JavaScript 1.0	Sets the date in the month for the `Date` object.
`setFullYear(year [, month, day])`	JavaScript 1.3	Sets the full year for the `Date` object.
`setHours(hours [, minutes, seconds, milliseconds])`	JavaScript 1.0	Sets the hour of the day for the `Date` object.
`setMilliseconds (milliseconds)`	JavaScript 1.3	Sets the number of milliseconds for the `Date` object.
`setMinutes(minutes [, seconds, milliseconds])`	JavaScript 1.0	Sets the number of minutes for the `Date` object.
`setMonth(month [, day])`	JavaScript 1.0	Sets the month for the `Date` object.
`setSeconds(seconds [, milliseconds])`	JavaScript 1.0	Sets the number of seconds for the `Date` object.
`setTime(milliseconds)`	JavaScript 1.0	Sets the time for the `Date` object according to the number of milliseconds since January 1, 1970 00:00:00.
`setUTCDate(dayOfMonth)`	JavaScript 1.3	Sets the date in the month for the `Date` object according to universal time.
`setUTCFullYear(year [, month, day])`	JavaScript 1.3	Sets the full year for the `Date` object according to universal time.

`setUTCHours(hours [, minutes, seconds, milliseconds])`	JavaScript 1.3	Sets the hour of the day for the `Date` object according to universal time.
`setUTCMilliseconds (milliseconds)`	JavaScript 1.3	Sets the number of milliseconds for the `Date` object according to universal time.
`setUTCMinutes(mintes [, seconds, milliseconds])`	JavaScript 1.3	Sets the number of minutes for the `Date` object according to universal time.
`setUTCMonth(month [, day])`	JavaScript 1.3	Sets the month for the `Date` object according to universal time.
`setUTCSeconds()`	JavaScript 1.3	Sets the number of seconds for the `Date` object according to universal time.
`setYear(year)`	JavaScript 1.0	Sets the year for the `Date` object. Deprecated in favor of `setFullYear()`.
`toGMTString()`	JavaScript 1.0	Converts the `Date` object to a string according to Greenwich Mean Time. Replaced by `toUTCString`.
`toLocaleString()`	JavaScript 1.0	Converts the `Date` object to a string according to the local time zone.
`toString()`	JavaScript 1.1	Converts the `Date` object into a string.
`toUTCString()`	JavaScript 1.3	Converts the `Date` object to a string according to universal time.
`UTC(year, month [, day, hours, minutes, seconds, milliseconds])`	JavaScript 1.0	Retrieves the number of milliseconds in a date since January 1, 1970 00:00:00, universal time.
`valueOf()`	JavaScript 1.1	Returns the primitive value of the `Date` object.

Function

Introduced in JavaScript 1.1, a `Function` object is created with the `Function` constructor.

Functions can be defined in a variety of ways. You can create a function using the following standard function statement:

```
function functionName() {
    // code here
}
```

You can also create an anonymous function and assign it to a variable. The following code demonstrates this approach:

```
var functionName = function() {
    // code here
};
```

The trailing semicolon is not a typo because this statement is an assignment operation, and all assignment operations should end with a semicolon.

Functions are objects, and thus they have a constructor. It's possible to create a function using the `Function` object's constructor as shown in the following code:

```
var functionName = new Function("arg1", "arg2", "return arg1 + arg2");
```

The first arguments to the constructor are the names of the function's parameters—you can add as many parameters as you need. The last parameter you pass to the constructor is the function's body. The previous code creates a function that accepts two arguments and returns their sum.

There are very few instances where you will use the `Function` constructor. It is preferred to define a function using the standard function statement or by creating an anonymous function and assigning it to a variable.

Properties

PROPERTY	INTRODUCED	DESCRIPTION
`arguments`	JavaScript 1.1	An array containing the parameters passed into the function.
`arguments.length`	JavaScript 1.1	Returns the number of parameters passed into the function.
`constructor`	JavaScript 1.1	Used to reference the constructor function for the object.
`length`	JavaScript 1.1	Returns the number of parameters expected by the function. This differs from `arguments.length`, which returns the number of parameters actually passed into the function.
`prototype`	JavaScript 1.1	Returns the prototype for the object, which can be used to extend the object's interface.

Methods

METHOD	INTRODUCED	DESCRIPTION
`apply(thisObj, arguments)`	JavaScript 1.3	Calls a function or method as if it belonged to `thisObj` and passes `arguments` to the function or method. `arguments` must be an array.

`call(thisObj, arg1, ...)`	JavaScript 1.3	Identical to `apply()`, except arguments are passed individually instead of in an array.
`toString()`	JavaScript 1.1	Converts the `Function` object into a string.
`valueOf()`	JavaScript 1.1	Returns the primitive value of the `Function` object.

JSON

The `JSON` object contains methods for parsing JavaScript Object Notation (JSON) into objects and serializing JavaScript objects into JSON. Introduced in JavaScript 1.8.5, the `JSON` object is a top-level object, which can be accessed without a constructor.

Methods

METHOD	INTRODUCED	DESCRIPTION
`parse(json)`	JavaScript 1.8.5	Transforms JSON into a JavaScript object or value.
`stringify(obj)`	JavaScript 1.8.5	Transforms a JavaScript object or value into JSON.

Math

The `Math` object provides methods and properties used for mathematical calculations. Introduced in JavaScript 1.0, the `Math` object is a top-level object, which can be accessed without a constructor.

Properties

PROPERTY	INTRODUCED	DESCRIPTION
`E`	JavaScript 1.0	Returns Euler's constant (the base of natural logarithms; approximately 2.718).
`LN10`	JavaScript 1.0	Returns the natural logarithm of 10 (approximately 2.302).
`LN2`	JavaScript 1.0	Returns the natural logarithm of 2 (approximately 0.693).
`LOG10E`	JavaScript 1.0	Returns the Base 10 logarithm of E (approximately 0.434).
`LOG2E`	JavaScript 1.0	Returns the Base 2 logarithm of E (approximately 1.442).
`PI`	JavaScript 1.0	Returns pi, the ratio of the circumference of a circle to its diameter (approximately 3.142).
`SQRT1_2`	JavaScript 1.0	Returns the square root of 1/2 (approximately 0.707).
`SQRT2`	JavaScript 1.0	Returns the square root of 2 (approximately 1.414).

Methods

METHOD	INTRODUCED	DESCRIPTION
`abs(x)`	JavaScript 1.0	Returns the absolute (positive) value of a number.
`acos(x)`	JavaScript 1.0	Returns the arccosine of a number (in radians).
`asin(x)`	JavaScript 1.0	Returns the arcsine of a number (in radians).
`atan(x)`	JavaScript 1.0	Returns the arctangent of a number (in radians).
`atan2(y, x)`	JavaScript 1.0	Returns the angle (in radians) between the x-axis and the position represented by the y and x coordinates passed in as parameters.
`ceil(x)`	JavaScript 1.0	Returns the value of a number rounded up to the nearest integer.
`cos(x)`	JavaScript 1.0	Returns the cosine of a number.
`exp(x)`	JavaScript 1.0	Returns E to the power of the argument passed in.
`floor(x)`	JavaScript 1.0	Returns the value of a number rounded down to the nearest integer.
`log(x)`	JavaScript 1.0	Returns the natural logarithm (base E) of a number.
`max(a, b)`	JavaScript 1.0	Returns the greater of two numbers passed in as parameters.
`min(a, b)`	JavaScript 1.0	Returns the lesser of two numbers passed in as parameters.
`pow(x, y)`	JavaScript 1.0	Returns the first parameter raised to the power of the second.
`random()`	JavaScript 1.1	Returns a pseudo-random number between 0 and 1.
`round(x)`	JavaScript 1.0	Returns the value of a number rounded up or down to the nearest integer.
`sin(x)`	JavaScript 1.0	Returns the sine of a number.
`sqrt(x)`	JavaScript 1.0	Returns the square root of a number.
`tan(x)`	JavaScript 1.0	Returns the tangent of a number.

Number

The `Number` object acts as a wrapper for primitive numeric values. Introduced in JavaScript 1.1, a `Number` object is created using the `Number` constructor with the initial value for the number passed in as a parameter.

Properties

PROPERTY	INTRODUCED	DESCRIPTION
`constructor`	JavaScript 1.1	Used to reference the constructor function for the object.
`MAX_VALUE`	JavaScript 1.1	Returns the largest number that can be represented in JavaScript (approximately 1.79E+308).
`MIN_VALUE`	JavaScript 1.1	Returns the smallest number that can be represented in JavaScript (5E–324).
`NaN`	JavaScript 1.1	Returns a value that is "not a number."
`NEGATIVE_INFINITY`	JavaScript 1.1	Returns a value representing negative infinity.
`POSITIVE_INFINITY`	JavaScript 1.1	Returns a value representing (positive) infinity.
`prototype`	JavaScript 1.1	Returns the prototype for the object, which can be used to extend the object's interface.

Methods

METHOD	INTRODUCED	DESCRIPTION
`toExponential(fractionDigits)`	JavaScript 1.5	Returns a string containing the exponent notation of a number. The parameter should be between 0 and 20 and determines the number of digits after the decimal.
`toFixed([digits])`	JavaScript 1.5	The format number for `digits` number of digits. The number is rounded up, and `0`s are added after the decimal point to achieve the desired decimal length.
`toPrecision([precision])`	JavaScript 1.5	Returns a string representing the `Number` object to the specified precision.
`toString()`	JavaScript 1.1	Converts the `Number` object into a string.
`valueOf()`	JavaScript 1.1	Returns the primitive value of the `Number` object.

Object

`Object` is the primitive type for JavaScript objects, from which all other objects are descended (that is, all other objects inherit the methods and properties of the `Object` object). Introduced in JavaScript 1.0, you can create an `Object` object using the `Object` constructor as follows:

```
var obj = new Object();
```

You can also create an object using object literal notation like this:

```
var obj = {};
```

Literal notation is the preferred method of creating an object.

Properties

PROPERTY	INTRODUCED	DESCRIPTION
`constructor`	JavaScript 1.1	Used to reference the constructor function for the object.
`prototype`	JavaScript 1.1	Returns the prototype for the object, which can be used to extend the object's interface.

Methods

METHOD	INTRODUCED	DESCRIPTION
`hasOwnProperty(propertyName)`	JavaScript 1.5	Checks whether the specified property is inherited. Returns `true` if not inherited; `false` if inherited.
`isPrototypeOf(obj)`	JavaScript 1.5	Determines if the specified object is the prototype of another object.
`propertyIsEnumerable(propertyName)`	JavaScript 1.5	Determines if the specified property can be seen by a `for in` loop.
`toString()`	JavaScript 1.0	Converts the `Object` object into a string.
`valueOf()`	JavaScript 1.1	Returns the primitive value of the `Object` object.

RegExp

The `RegExp` object is used to find patterns within string values. `RegExp` objects can be created in two ways: using the `RegExp` constructor or a text literal. It was introduced in JavaScript 1.2.

Some of the properties in the following table have both long and short names. The short names are derived from the Perl programming language.

Properties

PROPERTY	INTRODUCED	DESCRIPTION
`constructor`	JavaScript 1.2	Used to reference the constructor function for the object.
`global`	JavaScript 1.2	Indicates whether all possible matches in the string are to be made, or only the first. Corresponds to the `g` flag.
`ignoreCase`	JavaScript 1.2	Indicates whether the match is to be case-insensitive. Corresponds to the `i` flag.
`input`	JavaScript 1.2	The string against which the regular expression is matched.
`lastIndex`	JavaScript 1.2	The position in the string from which the next match is to be started.
`multiline`	JavaScript 1.2	Indicates whether strings are to be searched across multiple lines. Corresponds with the `m` flag.
`prototype`	JavaScript 1.2	Returns the prototype for the object, which can be used to extend the object's interface.
`source`	JavaScript 1.2	The text of the pattern for the regular expression.

Methods

METHOD	INTRODUCED	DESCRIPTION
`exec(stringToSearch)`	JavaScript 1.2	Executes a search for a match in the string parameter passed in.
`test(stringToMatch)`	JavaScript 1.2	Tests for a match in the string parameter passed in.
`toString()`	JavaScript 1.2	Converts the `RegExp` object into a string.
`valueOf()`	JavaScript 1.2	Returns the primitive value of the `RegExp` object.

Special Characters Used in Regular Expressions

CHARACTER	EXAMPLES	FUNCTION
\	/n/ matches n;/\n/ matches a linefeed character;/^/ matches the start of a line; and/\^/ matches ^.	For characters that are by default treated as normal characters, the backslash indicates that the next character is to be interpreted with a special value. For characters that are usually treated as special characters, the backslash indicates that the next character is to be interpreted as a normal character.
^	/^A/ matches the first but not the second A in "A man called Adam."	Matches the start of a line or of the input.
$	/r$/ matches only the last r in "horror."	Matches the end of a line or of the input.
*	/ro*/ matches r in "right," ro in "wrong," and "roo" in "room."	Matches the preceding character zero or more times.
+	/l+/ matches l in "life," ll in "still," and lll in "stilllife."	Matches the preceding character one or more times. For example, /a+/ matches the a in "candy" and all the as in "caaaaaaandy."
?	/Smythe?/ matches "Smyth" and "Smythe."	Matches the preceding character once or zero times.
.	/.b/ matches the second but not the first ob in "blob."	Matches any character apart from the newline character.
(x)	/(Smythe?)/ matches "Smyth" and "Smythe" in "John Smyth and Rob Smythe" and allows the substrings to be retrieved as RegExp.$1 and RegExp.$2, respectively.	Matches x and remembers the match. The matched substring can be retrieved from the elements of the array that results from the match, or from the RegExp object's properties $1, $2 ... $9, or lastParen.
x|y	/Smith|Smythe/ matches "Smith" and "Smythe."	Matches either x or y (where x and y are blocks of characters).
{n}	/l{2}/ matches ll in "still" and the first two ls in "stilllife."	Matches exactly n instances of the preceding character (where n is a positive integer).

{n,}	/l{2,}/ matches ll in "still" and lll in "stilllife."	Matches n or more instances of the preceding character (where n is a positive integer).
{n,m}	/l{1,2}/ matches l in "life," ll in "still," and the first two ls in "stilllife."	Matches between n and m instances of the preceding character (where n and m are positive integers).
[xyz]	[ab] matches a and b; [a-c] matches a, b and c.	Matches any one of the characters in the square brackets. A range of characters in the alphabet can be matched using a hyphen.
[^xyz]	[^aeiouy] matches s in "easy"; [^a-y] matches z in "lazy."	Matches any character except those enclosed in the square brackets. A range of characters in the alphabet can be specified using a hyphen.
[\b]		Matches a backspace.
\b	/t\b/ matches the first t in "about time."	Matches a word boundary (for example, a space or the end of a line).
\B	/t\Bi/ matches ti in "it is time."	Matches when there is no word boundary in this position.
\cX	/\cA/ matches Ctrl+A.	Matches a control character.
\d	/IE\d/ matches IE4, IE5, etc.	Matches a digit character. This is identical to [0-9].
\D	/\D/ matches the decimal point in "3.142."	Matches any character that is not a digit. This is identical to [^0-9].
\f		Matches a form-feed character.
\n		Matches a line-feed character.
\r		Matches a carriage return character.
\s	/\s/ matches the space in "not now."	Matches any white space character, including space, tab, line-feed, etc. This is identical to [\f\n\r\t\v].
\S	/\S/ matches a in "a."	Matches any character other than a white space character. This is identical to [^ \f\n\r\t\v].

continues

(continued)

CHARACTER	EXAMPLES	FUNCTION
`\t`		Matches a tab character.
`\v`		Matches a vertical tab character.
`\w`	`/\w/` matches `O` in "`O?!`" and `1` in "`$1`."	Matches any alphanumeric character or the underscore. This is identical to `[A-Za-z0-9_]`.
`\W`	`/\W/` matches `$` in "`$10million`" and `@` in "`j_smith@wrox`."	Matches any non-alphanumeric character (excluding the underscore). This is identical to `[^A-Za-z0-9_]`.
`()\n`	`/(Joh?n)` and `\1/` matches `John and John` in "`John and John's friend`" but does not match "`John and Jon`."	Matches the last substring that matched the `n`th match placed in parentheses and remembered (where `n` is a positive integer).
`\octal\xhex`	`/\x25/` matches `%`.	Matches the character corresponding to the specified octal or hexadecimal escape value.

String

The `String` object is used to contain a string of characters. It was introduced in JavaScript 1.0. This must be distinguished from a string literal, but the methods and properties of the `String` object can also be accessed by a string literal, because a temporary object will be created when they are called.

The HTML methods in the last table are not part of any ECMAScript standard, but they have been part of the JavaScript language since version 1.0. They can be useful because they dynamically generate HTML.

Properties

Property	Introduced	Description
`constructor`	JavaScript 1.1	Used to reference the constructor function for the object.
`length`	JavaScript 1.0	Returns the number of characters in the string.
`prototype`	JavaScript 1.1	Returns the prototype for the object, which can be used to extend the object's interface.

Methods

METHOD	INTRODUCED	DESCRIPTION
`charAt(index)`	JavaScript 1.0	Returns the character at the specified position in the string.
`charCodeAt(index)`	JavaScript 1.2	Returns the Unicode value of the character at the specified position in the string.
`concat(value1, value2, ...)`	JavaScript 1.2	Concatenates the strings supplied as arguments and returns the string thus formed.
`fromCharCode(value1, value2, ...)`	JavaScript 1.2	Returns the string formed from the concatenation of the characters represented by the supplied Unicode values.
`indexOf(substr [, startIndex])`	JavaScript 1.0	Returns the position within the `String` object of the first match for the supplied substring. Returns `-1` if the substring is not found. Starts the search at `startIndex` if specified.
`lastIndexOf(substr [, startIndex])`	JavaScript 1.0	Returns the position within the `String` object of the last match for the supplied substring. Returns `-1` if the substring is not found. Starts the search at `startIndex` if specified.
`match(regexp)`	JavaScript 1.2	Searches the string for a match to the supplied pattern. Returns an array or `null` if not found.
`replace(regexp, newValue)`	JavaScript 1.2	Used to replace a substring that matches a regular expression with a new value.
`search(regexp)`	JavaScript 1.2	Searches for a match between a regular expression and the string. Returns the index of the match, or `-1` if not found.
`slice(startIndex [, endIndex])`	JavaScript 1.0	Returns a substring of the `String` object.
`split(delimiter)`	JavaScript 1.1	Splits a `String` object into an array of strings by separating the string into substrings.
`substr(startIndex [, length])`	JavaScript 1.0	Returns a substring of the characters from the given starting position and containing the specified number of characters.
`substring(startIndex [, endIndex])`	JavaScript 1.0	Returns a substring of the characters between two positions in the string. The character at `endIndex` is not included in the substring.
`toLowerCase()`	JavaScript 1.0	Returns the string converted to lowercase.
`toUpperCase()`	JavaScript 1.0	Returns the string converted to uppercase.

HTML Methods

METHOD	INTRODUCED	DESCRIPTION
`anchor(name)`	JavaScript 1.0	Returns the string surrounded by `<a>...</a>` tags with the `name` attribute assigned the passed parameter.
`big()`	JavaScript 1.0	Encloses the string in `<big>...</big>` tags.
`blink()`	JavaScript 1.0	Encloses the string in `<blink>...</blink>` tags.
`bold()`	JavaScript 1.0	Encloses the string in `<b>...</b>` tags.
`fixed()`	JavaScript 1.0	Encloses the string in `<tt>...</tt>` tags.
`fontcolor(color)`	JavaScript 1.0	Encloses the string in `<font>...</font>` tags with the `color` attribute assigned a parameter value.
`fontsize(size)`	JavaScript 1.0	Encloses the string in `<font>...</font>` tags with the `size` attribute assigned a parameter value.
`italics()`	JavaScript 1.0	Encloses the string in `<i>...</i>` tags.
`link(url)`	JavaScript 1.0	Encloses the string in `<a>...</a>` tags with the `href` attribute assigned a parameter value.
`small()`	JavaScript 1.0	Encloses the string in `<small>...</small>` tags.
`strike()`	JavaScript 1.0	Encloses the string in `<strike>...</strike>` tags.
`sub()`	JavaScript 1.0	Encloses the string in `<sub>...</sub>` tags.
`sup()`	JavaScript 1.0	Encloses the string in `<sup>...</sup>` tags and causes a string to be displayed as superscript.