* [**Skip to main content**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval#content)
* [**Select language**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval#language)
* [**Skip to search**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval#main-q)
* **Technologies▼**
* **References & Guides▼**
* **Feedback▼**

Top of Form

Search MDN

Bottom of Form

[**Sign in**](https://developer.mozilla.org/users/github/login/?next=/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval)

**eval()**

1. [Web technology for developers](https://developer.mozilla.org/en-US/docs/Web)
2. [JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript)
3. [JavaScript reference](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference)
4. [Standard built-in objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects)
5. [eval()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval)

English▼

On this Page

* [Syntax](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval#Syntax)
* [Description](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval#Description)
* [Never use eval()!](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval#Never_use_eval!)
* [Examples](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval#Examples)
* [Specifications](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval#Specifications)
* [Browser compatibility](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval#Browser_compatibility)
* [Firefox-specific notes](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval#Firefox-specific_notes)
* [See also](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval#See_also)

The **eval()** function evaluates JavaScript code represented as a string.

**Warning:** Executing JavaScript from a string is an enormous security risk. It is far too easy for a bad actor to run arbitrary code when you use eval(). See [Never use eval()!](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval#Never_use_eval!), below.

**Syntax**

eval(*string*)

Parameters

**string**

A string representing a JavaScript expression, statement, or sequence of statements. The expression can include variables and properties of existing objects.

Return value

The completion value of evaluating the given code. If the completion value is empty, [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) is returned.

**Description**

eval() is a function property of the global object.

The argument of the eval() function is a string. If the string represents an expression, eval() evaluates the expression. If the argument represents one or more JavaScript statements, eval() evaluates the statements. Do not call eval() to evaluate an arithmetic expression; JavaScript evaluates arithmetic expressions automatically.

If you construct an arithmetic expression as a string, you can use eval() to evaluate it at a later time. For example, suppose you have a variable x. You can postpone evaluation of an expression involving x by assigning the string value of the expression, say "3 \* x + 2", to a variable, and then calling eval() at a later point in your script.

If the argument of eval() is not a string, eval() returns the argument unchanged. In the following example, the String constructor is specified and eval() returns a String object rather than evaluating the string.

eval(new String('2 + 2')); // returns a String object containing "2 + 2"

eval('2 + 2'); // returns 4

You can work around this limitation in a generic fashion by using toString().

var expression = new String('2 + 2');

eval(expression.toString()); // returns 4

If you use the eval function *indirectly,* by invoking it via a reference other than eval, [as of ECMAScript 5](http://www.ecma-international.org/ecma-262/5.1/#sec-10.4.2) it works in the global scope rather than the local scope. This means, for instance, that function declarations create global functions, and that the code being evaluated doesn't have access to local variables within the scope where it's being called.

function test() {

var x = 2, y = 4;

console.log(eval('x + y')); // Direct call, uses local scope, result is 6

var geval = eval; // equivalent to calling eval in the global scope

console.log(geval('x + y')); // Indirect call, uses global scope, throws ReferenceError because `x` is undefined

(0, eval)('x + y'); // another example of Indirect call

}

**Never use eval()!**

eval() is a dangerous function, which executes the code it's passed with the privileges of the caller. If you run eval() with a string that could be affected by a malicious party, you may end up running malicious code on the user's machine with the permissions of your webpage / extension. More importantly, a third-party code can see the scope in which eval() was invoked, which can lead to possible attacks in ways to which the similar [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function) is not susceptible.

eval() is also slower than the alternatives, since it has to invoke the JavaScript interpreter, while many other constructs are optimized by modern JS engines.

Additionally, modern javascript interpreters convert javascript to machine code. This means that any concept of variable naming gets obliterated. Thus, any use of eval() will force the browser to do long expensive variable name lookups to figure out where the variable exists in the machine code and set its value. Additonally, new things can be introduced to that variable through eval() such as changing the type of that variable, forcing the browser to re-evaluate all of the generated machine code to compensate.

Fortunately, there's a very good alternative to eval(): simply using [window.Function()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function). See this example of how to convert code using a dangerous eval() to using Function(), see below.

Bad code with eval():

function looseJsonParse(obj){

return eval("(" + obj + ")");

}

console.log(looseJsonParse(

"{a:(4-1), b:function(){}, c:new Date()}"

))

Better code without eval():

function looseJsonParse(obj){

return Function('"use strict";return (' + obj + ')')();

}

console.log(looseJsonParse(

"{a:(4-1), b:function(){}, c:new Date()}"

))

Comparing the two code snippets above, the two code snippets might seem to work the same way, but think again: the eval() one is a great deal slower. Notice c: new Date() in the evaluated object. In the function without the eval(), the object is being evaluated in the global scope, so it is safe for the browser to assume that Date refers to window.Date() instead of a local variable called Date. But, in the code using eval(), the browser cannot assume this since what if your code looked like the following:

function Date(n){

return ["Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"][n%7 || 0];

}

function looseJsonParse(obj){

return eval("(" + obj + ")");

}

console.log(looseJsonParse(

"{a:(4-1), b:function(){}, c:new Date()}"

))

Thus, in the eval() version of the code, the browser is forced to make the expensive lookup call to check to see if there are any local variables called Date(). This is incredibly inefficient compared to Function().

In a related circumstance, what if you actually wanted your Date() function to be able to be called from the code inside Function(). Should you just take the easy way out and fall back to eval()? No! Never. Instead try the approach below.

function Date(n){

return ["Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"][n%7 || 0];

}

function runCodeWithDateFunction(obj){

return Function('"use strict";return (' + obj + ')')()(

Date

);

}

console.log(runCodeWithDateFunction(

"function(Date){ return Date(5) }"

))

The code above may seem inefficiently slow because of the triple nested function, but let's analyze the benefits of the above efficient method:

* It allows the code in the string passed to runCodeWithDateFunction() to be minified.
* Function call overhead is minimal, making the far smaller code size well worth the benefit
* Function() more easily allows your code to utilize the performance buttering "use strict";
* The code does not use eval(), making it orders of magnitude faster than otherwise.

Lastly, let's examine minification. With using  Function() as shown above, you can minify the code string passed to runCodeWithDateFunction() far more efficiently because the function arguments names can be minified too as seen in the minified code below.

console.log(Function('"use strict";return(function(a){return a(5)})')()(function(a){

return"Monday Tuesday Wednesday Thursday Friday Saturday Sunday".split(" ")[a%7||0]}));

There are also additional safer (and faster!) alternatives to eval() or Function() for common use-cases.

Accessing member properties

You should not use eval() to convert property names into properties. Consider the following example where the property of the object to be accessed is not known until the code is executed. This can be done with eval():

var obj = { a: 20, b: 30 };

var propName = getPropName(); // returns "a" or "b"

eval( 'var result = obj.' + propName );

However, eval() is not necessary here. In fact, its use here is discouraged. Instead, use the [property accessors](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_Accessors), which are much faster and safer:

var obj = { a: 20, b: 30 };

var propName = getPropName(); // returns "a" or "b"

var result = obj[ propName ]; // obj[ "a" ] is the same as obj.a

You can even use this method to access descendant properties. Using eval() this would look like:

var obj = {a: {b: {c: 0}}};

var propPath = getPropPath(); // returns e.g. "a.b.c"

eval( 'var result = obj.' + propPath );

Avoiding eval() here could be done by splitting the property path and looping through the different properties:

function getDescendantProp(obj, desc) {

var arr = desc.split('.');

while (arr.length) {

obj = obj[arr.shift()];

}

return obj;

}

var obj = {a: {b: {c: 0}}};

var propPath = getPropPath(); // returns e.g. "a.b.c"

var result = getDescendantProp(obj, propPath);

Setting a property that way works similarly:

function setDescendantProp(obj, desc, value) {

var arr = desc.split('.');

while (arr.length > 1) {

obj = obj[arr.shift()];

}

return obj[arr[0]] = value;

}

var obj = {a: {b: {c: 0}}};

var propPath = getPropPath(); // returns e.g. "a.b.c"

var result = setDescendantProp(obj, propPath, 1); // obj.a.b.c will now be 1

Use functions instead of evaluating snippets of code

JavaScript has [first-class functions](http://en.wikipedia.org/wiki/First-class_function), which means you can pass functions as arguments to other APIs, store them in variables and objects' properties, and so on. Many DOM APIs are designed with this in mind, so you can (and should) write:

// instead of setTimeout(" ... ", 1000) use:

setTimeout(function() { ... }, 1000);

// instead of elt.setAttribute("onclick", "...") use:

elt.addEventListener('click', function() { ... } , false);

[Closures](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures) are also helpful as a way to create parameterized functions without concatenating strings.

Parsing JSON (converting strings to JavaScript objects)

If the string you're calling eval() on contains data (for example, an array: "[1, 2, 3]"), as opposed to code, you should consider switching to [JSON](https://developer.mozilla.org/en-US/docs/Glossary/JSON), which allows the string to use a subset of JavaScript syntax to represent data. See also [Downloading JSON and JavaScript in extensions](https://developer.mozilla.org/en-US/docs/Downloading_JSON_and_JavaScript_in_extensions).

Note that since JSON syntax is limited compared to JavaScript syntax, many valid JavaScript literals will not parse as JSON. For example, trailing commas are not allowed in JSON, and property names (keys) in object literals must be enclosed in quotes. Be sure to use a JSON serializer to generate strings that will be later parsed as JSON.

Pass data instead of code

For example, an extension designed to scrape contents of web-pages could have the scraping rules defined in [XPath](https://developer.mozilla.org/en-US/docs/XPath) instead of JavaScript code.

Run code with limited privileges

If you must run the code, consider running it with reduced privileges. This advice applies mainly to extensions and XUL applications, which can use [Components.utils.evalInSandbox](https://developer.mozilla.org/en-US/docs/Components.utils.evalInSandbox) for this.

**Examples**

Using eval

In the following code, both of the statements containing eval() return 42. The first evaluates the string "x + y + 1"; the second evaluates the string "42".

var x = 2;

var y = 39;

var z = '42';

eval('x + y + 1'); // returns 42

eval(z); // returns 42

Using eval to evaluate a string of JavaScript statements

The following example uses eval() to evaluate the string str. This string consists of JavaScript statements that open an alert dialog box and assign z a value of 42 if x is five, and assigns 0 to z otherwise. When the second statement is executed, eval() will cause these statements to be performed, and it will also evaluate the set of statements and return the value that is assigned to z.

var x = 5;

var str = "if (x == 5) {console.log('z is 42'); z = 42;} else z = 0;";

console.log('z is ', eval(str));

If you define multiple values then the last value is returned.

var x = 5;

var str = "if (x == 5) {console.log('z is 42'); z = 42; x = 420; } else z = 0;";

console.log('x is ', eval(str)); // z is 42 x is 420

Last expression is evaluated

eval() returns the value of the last expression evaluated.

var str = 'if ( a ) { 1 + 1; } else { 1 + 2; }';

var a = true;

var b = eval(str); // returns 2

console.log('b is : ' + b);

a = false;

b = eval(str); // returns 3

console.log('b is : ' + b);

eval as a string defining function requires "(" and ")" as prefix and suffix

var fctStr1 = 'function a() {}'

var fctStr2 = '(function a() {})'

var fct1 = eval(fctStr1) // return undefined

var fct2 = eval(fctStr2) // return a function

**Specifications**

|  |  |  |
| --- | --- | --- |
| **Specification** | **Status** | **Comment** |
| [ECMAScript 1st Edition (ECMA-262)](https://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%201st%20edition,%20June%201997.pdf) | Standard | Initial definition. |
| [ECMAScript 5.1 (ECMA-262) The definition of 'eval' in that specification.](https://www.ecma-international.org/ecma-262/5.1/#sec-15.1.2.1) | Standard |  |
| [ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'eval' in that specification.](https://www.ecma-international.org/ecma-262/6.0/#sec-eval-x) | Standard |  |
| [ECMAScript Latest Draft (ECMA-262) The definition of 'eval' in that specification.](https://tc39.es/ecma262/#sec-eval-x) | Draft |  |

**Browser compatibility**

[Update compatibility data on GitHub](https://github.com/mdn/browser-compat-data)

|  | **Desktop** | | | | | | **Mobile** | | | | | | **Server** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Chrome** | **Edge** | **Firefox** | **Internet Explorer** | **Opera** | **Safari** | **Android webview** | **Chrome for Android** | **Firefox for Android** | **Opera for Android** | **Safari on iOS** | **Samsung Internet** | **Node.js** |
| **eval** | Full supportYes | Full support12 | Full support1 | Full support3 | Full supportYes | Full supportYes | Full supportYes | Full supportYes | Full support4 | Full supportYes | Full supportYes | Full supportYes | Full supportYes |

[**What are we missing?**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval)

Legend

**Full support**

Full support

**Firefox-specific notes**

* Historically eval() had an optional second argument, specifying an object in whose context the evaluation was to be performed. This argument was non-standard, and was definitely removed from Firefox 4. See [bug 531675](https://bugzilla.mozilla.org/show_bug.cgi?id=531675).

**See also**

* [uneval()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/uneval)
* [Property accessors](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_Accessors)
* [WebExtensions: Using eval in content scripts](https://developer.mozilla.org/en-US/Add-ons/WebExtensions/Content_scripts#Using_eval()_in_content_scripts)

**Metadata**

* **Last modified:** Oct 22, 2019, [by MDN contributors](https://wiki.developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval$history)

Related Topics

1. [***JavaScript***](https://developer.mozilla.org/en-US/docs/Web/JavaScript)
2. [**Tutorials:**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Tutorials)
3. Complete beginners
4. JavaScript Guide
5. Intermediate
6. Advanced
7. [**References:**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference)
8. Built-in objects
   1. [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)
   2. [ArrayBuffer](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer)
   3. [AsyncFunction](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/AsyncFunction)
   4. [Atomics](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Atomics)
   5. [BigInt](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/BigInt)
   6. [BigInt64Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/BigInt64Array)
   7. [BigUint64Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/BigUint64Array)
   8. [Boolean](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Boolean)
   9. [DataView](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/DataView)
   10. [Date](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date)
   11. [Error](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Error)
   12. [EvalError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/EvalError)
   13. [Float32Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Float32Array)
   14. [Float64Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Float64Array)
   15. [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function)
   16. [Generator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Generator)
   17. [GeneratorFunction](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/GeneratorFunction)
   18. [Infinity](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Infinity)
   19. [Int16Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Int16Array)
   20. [Int32Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Int32Array)
   21. [Int8Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Int8Array)
   22. [InternalError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/InternalError)
   23. [Intl](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Intl)
   24. [Intl.Collator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Collator)
   25. [Intl.DateTimeFormat](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/DateTimeFormat)
   26. [Intl.ListFormat](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ListFormat)
   27. [Intl.Locale](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Locale)
   28. [Intl.NumberFormat](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/NumberFormat)
   29. [Intl.PluralRules](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/PluralRules)
   30. [Intl.RelativeTimeFormat](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RelativeTimeFormat)
   31. [JSON](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON)
   32. [Map](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map)
   33. [Math](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math)
   34. [NaN](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/NaN)
   35. [Number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number)
   36. [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object)
   37. [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)
   38. [Proxy](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy)
   39. [RangeError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RangeError)
   40. [ReferenceError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ReferenceError)
   41. [Reflect](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Reflect)
   42. [RegExp](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp)
   43. [Set](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set)
   44. [SharedArrayBuffer](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SharedArrayBuffer)
   45. [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)
   46. [Symbol](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Symbol)
   47. [SyntaxError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SyntaxError)
   48. [TypeError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypeError)
   49. [TypedArray](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypedArray)
   50. [URIError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/URIError)
   51. [Uint16Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Uint16Array)
   52. [Uint32Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Uint32Array)
   53. [Uint8Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Uint8Array)
   54. [Uint8ClampedArray](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Uint8ClampedArray)
   55. [WeakMap](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/WeakMap)
   56. [WeakSet](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/WeakSet)
   57. [WebAssembly](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/WebAssembly)
   58. [decodeURI()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/decodeURI)
   59. [decodeURIComponent()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/decodeURIComponent)
   60. [encodeURI()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encodeURI)
   61. [encodeURIComponent()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encodeURIComponent)
   62. [escape()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/escape)
   63. [eval()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval)
   64. [globalThis](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/globalThis)
   65. [isFinite()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/isFinite)
   66. [isNaN()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/isNaN)
   67. [null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/null)
   68. [parseFloat()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseFloat)
   69. [parseInt()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseInt)
   70. [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined)
   71. [unescape()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/unescape)
   72. [uneval()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/uneval)
9. Expressions & operators
10. Statements & declarations
11. Functions
12. Classes
13. Errors
14. Misc
15. New in JavaScript
16. [**Documentation:**](https://developer.mozilla.org/en-US/docs/MDN)
17. Useful lists
18. Contribute

Top of Form

**Learn the best of web development**

Get the latest and greatest from MDN delivered straight to your inbox.

E-mail

Sign up now

Bottom of Form

Hide Newsletter Sign-up

[MDN Web Docs](https://developer.mozilla.org/en-US/)

**MDN**

* [Web Technologies](https://developer.mozilla.org/en-US/docs/Web)
* [Learn Web Development](https://developer.mozilla.org/en-US/docs/Learn)
* [About MDN](https://developer.mozilla.org/en-US/docs/MDN/About)
* [Feedback](https://developer.mozilla.org/en-US/docs/MDN/Feedback)

[Mozilla](https://mozilla.org/)

**Mozilla**

* [About](https://www.mozilla.org/about/)
* [Contact Us](https://www.mozilla.org/contact/)
* [Firefox](https://www.mozilla.org/firefox/?utm_source=developer.mozilla.org&utm_campaign=footer&utm_medium=referral)
* [Terms](https://www.mozilla.org/about/legal/terms/mozilla)
* [Privacy](https://www.mozilla.org/privacy/websites/)
* [Cookies](https://www.mozilla.org/privacy/websites/#cookies)

© 2005-2019 Mozilla and individual contributors.

Content is available under [these licenses](https://developer.mozilla.org/docs/MDN/About#Copyrights_and_licenses).

Source: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval#Never_use_eval!>