25th September 2019

Eval: run a code string

The built-in `eval` function allows to execute a string of code.

The syntax is:

```
let result = eval(code);
```

For example:

```
let code = 'alert("Hello")';
eval(code); // Hello
```

A string of code may be long, contain line breaks, function declarations, variables and so on.

The result of `eval` is the result of the last statement.

For example:

```
let value = eval('1+1');
alert(value); // 2
```

```
let value = eval('let i = 0; ++i');
alert(value); // 1
```

The eval'ed code is executed in the current lexical environment, so it can see outer variables:

```
let a = 1;

function f() {
  let a = 2;

  eval('alert(a)'); // 2
}

f();
```

It can change outer variables as well:

```
let x = 5;
eval("x = 10");
alert(x); // 10, value modified
```

In strict mode, `eval` has its own lexical environment. So functions and variables, declared inside eval, are not visible outside:

```
// reminder: 'use strict' is enabled in runnable examples by default

eval("let x = 5; function f() {}");

alert(typeof x); // undefined (no such variable)
// function f is also not visible
```

Without `use strict`, `eval` doesn't have its own lexical environment, so we would see `x` and `f` outside.

Using "eval"

In modern programming `eval` is used very sparingly. It's often said that "eval is evil".

The reason is simple: long, long time ago JavaScript was a much weaker language, many things could only be done with `eval`. But that time passed a decade ago.

Right now, there's almost no reason to use `eval`. If someone is using it, there's a good chance they can replace it with a modern language construct or a JavaScript Module.

Please note that its ability to access outer variables has side-effects.

Code minifiers (tools used before JS gets to production, to compress it) rename local variables into shorter ones (like `a`, `b` etc) to make the code smaller. That's usually safe, but not if `eval` is used, as local variables may be accessed from eval'ed code string. So minifiers don't do that renaming for all variables potentially visible from `eval`. That negatively affects code compression ratio.

Using outer local variables inside `eval` is also considered a bad programming practice, as it makes maintaining the code more difficult.

There are two ways how to be totally safe from such problems.

If eval'ed code doesn't use outer variables, please call `eval` as `window.eval(...)`:

This way the code is executed in the global scope:

```
let x = 1;
{
  let x = 5;
  window.eval('alert(x)'); // 1 (global variable)
}
```

If eval'ed code needs local variables, change `eval` to `new Function` and pass them as arguments:

```
let f = new Function('a', 'alert(a)');

```

```
f(5); // 5
```

The `new Function` construct is explained in the chapter The "new Function" syntax. It creates a function from a string, also in the global scope. So it can't see local variables. But it's so much clearer to pass them explicitly as arguments, like in the example above.

Summary

A call to `eval(code)` runs the string of code and returns the result of the last statement.

- Rarely used in modern JavaScript, as there's usually no need.
- Can access outer local variables. That's considered bad practice.
- Instead, to `eval` the code in the global scope, use `window.eval(code)` .
- Or, if your code needs some data from the outer scope, use `new Function` and pass it as arguments.

Tasks

Eval-calculator

importance: 4

Create a calculator that prompts for an arithmetic expression and returns its result.

There's no need to check the expression for correctness in this task. Just evaluate and return the result.

Run the demo

solution

Previous lesson Next lesson

Share Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>` , for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)

© 2007—2020 Ilya Kantorabout the projectcontact usterms of usageprivacy policy