Are there constants in JavaScript?

Asked 11 years ago Active 1 year, 3 months ago Viewed 479k times

1116

150

Is there a way to use constants in JavaScript?

If not, what's the common practice for specifying variables that are used as constants?

javascript constants

edited Aug 21 '17 at 15:12
Stevoisiak
7,641 12 55 121

asked Sep 24 '08 at 22:45
fuentesjr
21.7k 26 70 78

34 `Chrome` allows you to use the keyword `const` to use constants. eg `const ASDF = "asdf"` . However, since `const` isn't multi browser compatible, I usually stick with a `var` declaration. – Jacksonkr Dec 8 '11 at 22:22

20 `try{const thing=1091;}catch(e){var thing=1091;}` works. – Derek 朕會功夫 Jan 22 '12 at 3:39

13 Derek: wouldn't your try/catch limit the scope of the thing you're declaring to the try/catch block? If you're not scoping properly then what's the point of specifying `const` or `var` at all? – Coderer Mar 26 '13 at 10:13

8 @Coderer in the current implementations, this will work, as `const` has the same scope as `var` , and that's function-level, not block-level. If you follow the upcoming ECMAScript standard instead, `const` has the same scope as `let` , which means it won't work. – Jasper Jul 22 '13 at 19:27

3 @Coderer Wrong language. Variables in javascript are function scope. This isn't C. – doug65536 Sep 11 '13 at 18:34

33 Answers

1 2 next

Since ES2015, JavaScript has a notion of `const` :

This will work in pretty much all browsers except IE 8, 9 and 10. Some may also need strict mode enabled.

You can use `var` with conventions like ALL_CAPS to show that certain values should not be modified if you need to support older browsers or are working with legacy code:

```
var MY_CONSTANT = "some-value";
```

edited Mar 4 '18 at 6:33
railgun
559 1 7 20

answered Sep 24 '08 at 22:46
John Millikin
164k 35 197 213

60 How about const x = 24; – Zachary Scott Mar 23 '10 at 2:34

93 Note that if you don't need cross-browser compatibility (or you're server-side programming in Rhino or Node.js) you can use the `const` keyword. It's currently supported by all modern browsers except for IE. – Husky Aug 1 '11 at 19:16

17 These days (3.5 years later) you can use `Object.defineProperty` to create read-only properties that also can't be deleted. This works in the current version of all major browsers (but incorrectly in IE8). See the answer by @NotAName – Phrogz Jun 1 '12 at 3:50

12 @Amyth That's not really helpful. It's a style **guide** proposed by a sole vendor. As per Husky's point above, IE compatibility is totally irrelevant when writing server-side JS. – aendrew Nov 6 '13 at 11:40

32 Since this answer is still highly ranked by Google in 2015 it should be said that it is now obsolete. The `const` keyword is now officially part of the language and is supported by every browser. According to statcounter.com only a few percent of internet users still use old browser versions that didn't support `const` . – twhb Sep 6 '15 at 16:50

307

Are you trying to protect the variables against modification? If so, then you can use a module pattern:

```
var CONFIG = (function() {
    var private = {
        'MY_CONST': '1',
        'ANOTHER_CONST': '2'
    };

    return {
       get: function(name) { return private[name]; }
    };
})();
```

```
CONFIG.MY_CONST = '2';
alert('MY_CONST: ' + CONFIG.get('MY_CONST'));  // 1

CONFIG.private.MY_CONST = '2';                 // error
alert('MY_CONST: ' + CONFIG.get('MY_CONST'));  // 1
```

Using this approach, the values cannot be modified. But, you have to use the get() method on CONFIG :(.

If you don't need to strictly protect the variables value, then just do as suggested and use a convention of ALL CAPS.

answered Sep 25 '08 at 3:14

Burke
3,229 1 12 8

13 Note that you could just return a function for the value of CONFIG. That would save you calling CONFIG.get() all the time. – Mathew Byrne Sep 25 '08 at 6:58

4 Pretty solution. But such things should be wrapped as a library to not reinvent them in any new project. – andrii Mar 8 '09 at 9:53

82 `CONFIG.get = someNewFunctionThatBreaksTheCode` ... All in all, you absolutely cannot enforce constants in JS (w/o const keyword). Just about the only thing you can do is limit visibility. – Thomas Eding Jan 11 '10 at 23:20

28 I do believe that `private` is a future reserved word in JavaScript, I wouldn't use that if I were you. – Zaq Aug 5 '12 at 16:49

This is also the registry pattern. – user656925 Aug 15 '12 at 21:14

120

The `const` keyword is in [the ECMAScript 6 draft] but it thus far only enjoys a smattering of browser support: http://kangax.github.io/compat-table/es6/. The syntax is:

```
const CONSTANT_NAME = 0;
```

edited Aug 8 '15 at 10:43

answered Mar 26 '09 at 20:29

Bill the Lizard
306k 159 506 799

13 If you try to assign a value to a `const` , it doesn't throw any errors. The assignment just fails and the constant still has its original value. This is a major

6 Keep an eye on browser support here: kangax.github.io/es5-compat-table/es6/#const – Mark McDonald May 4 '13 at 5:18

6 @MatrixFrog assignment will raise an error with `'use strict'` . – sam Mar 13 '15 at 17:19

Should I define constants in `ALL CAPS` ? – Lewis Aug 8 '15 at 6:53

1 @Tresdin It is a common naming convention to define constants in all caps. Nothing in the language spec forces it, but it's not a bad idea. It makes it clearer what your intention is, so it improves code readability. – Bill the Lizard Aug 8 '15 at 10:42

68

```
"use strict";

var constants = Object.freeze({
    "π": 3.141592653589793 ,
    "e": 2.718281828459045 ,
    "i": Math.sqrt(-1)
});

constants.π;        // -> 3.141592653589793
constants.π = 3;    // -> TypeError: Cannot assign to read only property 'π' …
constants.π;        // -> 3.141592653589793

delete constants.π; // -> TypeError: Unable to delete property.
constants.π;        // -> 3.141592653589793
```

See Object.freeze. You can use `const` if you want to make the `constants` reference read-only as well.

edited Oct 24 '17 at 19:58

answered May 4 '14 at 23:58

sam

26.7k 2 35 35

2 Should mention this only works on IE9+ kangax.github.io/compat-table/es5 . – Cordle Jun 3 '14 at 14:27

I would, if it didn't have the broken implementation of `i` – Alnitak Jan 16 '15 at 12:43

Note: this behaves similar to the ES6 `const` declaration, e.g. properties cannot be re-declared or re-assigned, but if they are of datatype `object` , they can be mutated. – le_m Jun 13 '16 at 2:56

Exactly what I was looking for. You can also use `const constants` instead of `var constants` to prevent the variable from being reassigned. – Jarett Millard Apr 26 '17 at 21:04

64

IE does support constants, sort of, e.g.:

```
<script language="VBScript">
 Const IE_CONST = True
</script>
<script type="text/javascript">
 if (typeof TEST_CONST == 'undefined') {
    const IE_CONST = false;
 }
 alert(IE_CONST);
</script>
```

answered Oct 26 '09 at 19:22

C Nagle
681 5 2

50 Boy, talk about something that isn't cross browser . . . Still +1 for thinking a bit outside the box. – Tom Oct 26 '09 at 19:36

14 VBScript? What's that? ;) – tybro0103 Sep 27 '13 at 4:06

2 I usually Vote down for general related cross browsers question, with an IE specific answer. Cause I hate people who think that IE javascript implementation is 'the One' , and others are just to be ignored. Who is using other borwsers than IE, btw? – Ant Oct 2 '13 at 20:54

Likely in ES6: wiki.ecmascript.org/doku.php?id=harmony:const – Ciro Santilli 新疆改造中心法轮功六四事件 Jun 20 '14 at 16:22

@Cooluhuru this script appears to handle both IE browsers (using VBScript) and non-IE browsers (using JavaScript `const`). Can you explain what's wrong with it? – Andrew Grimm Jul 13 '16 at 7:47

58

ECMAScript 5 does introduce `Object.defineProperty`:

```
Object.defineProperty (window,'CONSTANT',{ value : 5, writable: false });
```

It's supported in every modern browser (as well as IE ≥ 9).

See also: Object.defineProperty in ES5?


sam
26.7k 2 35 35


Not a Name
858 6 17

1 It's worth noting that this is not like a traditional constant. This will only allow you to define a constant property (of a non constant object). Also this does not generate an error and even returns the value you attempt to set. It simply doesn't write the value. – Cory Gross Jul 7 '13 at 14:03

3 I recently read that attempting to assign to a property with `writable: false` *will* actually throw an error if the code that does the assignment is being interpreted under ECMAScript 5's strict mode. Just another reason to be writing `'use strict'` into your code. – Cory Gross Jul 21 '13 at 23:10

6 You can actually omit `writable: false` since that's the default. – sam May 4 '14 at 23:39

24

No, not in general. Firefox implements `const` but I know IE doesn't.

@John points to a common naming practice for consts that has been used for years in other languages, I see no reason why you couldn't use that. Of course that doesn't mean someone will not write over the variable's value anyway. :)

edited May 23 '17 at 12:18


Community ♦
1 1

answered Sep 24 '08 at 22:45


Jason Bunting
50.7k 12 93 92

11 As everyone knows, if IE doesn't implement it, it might as well not exist. – Josh Hinman Sep 24 '08 at 22:47

3 Unfortunately, and practically speaking - it is true. IE does own a huge share of the market. If I owned a business and had web applications used internally, I would standardize on FF. I don't know why so many people care about IE, it blows. – Jason Bunting Sep 24 '08 at 22:50

@Rich: Who said my opinion was fact? You made quite the assumption. Besides, as far as I am concerned, the fact that IE sucks is a fact. You can have your own facts, I didn't say you had to believe mine. :P Take a Xanax or something... – Jason Bunting Sep 24 '08 at 23:44

@Rich B, yea that was just a dumb comment to make, and trust me, I would know, I make plenty of dumb comments. @Jason B. - interesting, I ran into this very problem last night.. const worked in FF but not IE. Thanks for clarification – theman_on_vista Apr 8 '09 at 13:14

Who cares about IE? I don't! FF or Chrome or Opera etc... can be installed almost in every OS platform. Also computer retailers usually know the old IE version sucks so they often (or even every time) install alternative browsers before selling a computer. So I've decided for my developed app to not care at all about incompatible browsers: if browser developers care about respecting standards their product can use my app, if not, users will use a different browser... I can live with it ;-) But can Microsoft live loosing part of the market? No they can't so "They" will change their dev politic! – willy wonka Jan 4 '17 at 8:04

20

Mozillas MDN Web Docs contain good examples and explanations about `const`. Excerpt:

```
// define MY_FAV as a constant and give it the value 7
const MY_FAV = 7;

// this will throw an error - Uncaught TypeError: Assignment to constant variable.
MY_FAV = 20;
```

But it is sad that IE9/10 still does not support `const`. And the reason it's absurd:

> So, what is IE9 doing with const? So far, our decision has been to not support it. It isn't yet a consensus feature as it has never been available on all browsers.
>
> ...
>
> In the end, it seems like the best long term solution for the web is to leave it out and to wait for standardization processes to run their course.

They don't implement it because other browsers didn't implement it correctly?! Too afraid of making it better? Standards definitions or not, a constant is a constant: set once, never changed.

And to all the ideas: Every function can be overwritten (XSS etc.). So there is no difference in `var` or `function(){return}`. `const` is the only real constant.

Update: IE11 supports `const`:

> IE11 includes support for the well-defined and commonly used features of the emerging ECMAScript 6 standard including let, **`const`**, `Map`, `Set`, and `WeakMap`, as well as `__proto__` for improved interoperability.

edited Oct 15 '17 at 10:04

answered Apr 30 '11 at 9:57

mgutt
3,479 1 34 52

24 "it has never been available on all browsers". If you don't make it available in IE then it will never be in all browsers. – km1 Aug 15 '12 at 14:11

driving standardization is not for everyone ;) - companies come and they go again - thank you for quoting the odds in the wood – Quicker Mar 10 '16 at 21:22

19

In JavaScript, my preference is to use functions to return constant values.

```
function MY_CONSTANT() {
   return "some-value";
}


alert(MY_CONSTANT());
```

answered Nov 24 '10 at 19:46

MTS
247 2 2

6 Worth pointing out that this falls into the same problem mentioned in @Burkes answer (@trinithis' comment). `MY_CONSTANT = function() { return "some-other-value"; } breaks it. +1 though, decent and quick solution. – Patrick M Jul 13 '12 at 12:21

13 -1. This has no benefit over var SOME_NAME = value (it's still mutable), is more code, and requires explaining. – mikemaccana Mar 1 '13 at 15:39

@PatrickM while it's true that you can modify that kind of pseudo-constants, in other languages like e.g. C, on which you *shouldn't be able to* modify constants, you can still do it via e.g. pointers. So as long as you use some approach which at least *suggests* that it's a constant, it's fine imo. – rev Jul 28 '14 at 11:43

17

If you don't mind using functions:

```
var constant = function(val) {
   return function() {
        return val;
    }
}
```

This approach gives you functions instead of regular variables, but it guarantees* that no one can alter the value once it's set.

```
a = constant(10);
```

```
b = constant(20);

b(); // 20
```

I personally find this rather pleasant, specially after having gotten used to this pattern from knockout observables.

*Unless someone redefined the function `constant` before you called it

answered Dec 30 '13 at 19:42

hasen
87.4k 60 171 213

1 underscore.js implements a constant function identical to this code. – Upperstage Jun 18 '14 at 17:13

Simple, concise and answers the spirit of OP's question. This should have received more upVotes. – Mac Nov 11 '14 at 19:41

3 This never really worked for me. Even though the closure makes it immutable, the var you assign it to can still be overwritten. Ex: `a = constant(10); a(10); // 10` followed by `a = constant(25); a(); //25` , no errors or warnings given, no indication that your constant has been broken. – Patrick M Sep 21 '15 at 19:46

If I reassign value to `a` then it's changes to new value – Saurabh Sharma Oct 5 '16 at 6:05

17

with the "new" Object api you can do something like this:

```
var obj = {};
Object.defineProperty(obj, 'CONSTANT', {
  configurable: false
  enumerable: true,
  writable: false,
  value: "your constant value"
});
```

take a look at this on the Mozilla MDN for more specifics. It's not a first level variable, as it is attached to an object, but if you have a scope, anything, you can attach it to that. `this` should work as well. So for example doing this in the global scope will declare a pseudo constant value on the window (which is a really bad idea, you shouldn't declare global vars carelessly)

```
Object.defineProperty(this, 'constant', {
```

```
  configurable: false
});

> constant
=> 7
> constant = 5
=> 7
```

note: assignment will give you back the assigned value in the console, but the variable's value will not change

edited Jul 28 '14 at 8:37

99 Problems - Syntax ain't one

476 7 15

answered Nov 2 '12 at 11:45

tenshou

362 3 12

Not working in safari, and in mozilla if you execute define statement again - with a different value - it will reassign the value. – Akshay Dec 11 '13 at 8:32

2 Not 'not working in safari', not *supported* in safari. Not the same. And it should throw a 'Uncaught TypeError: Cannot redefine property: <property name here>' if you try that. either you're doing it wrong, or your ff implemented it incorrectly. I guess it's a mix of both. – tenshou Dec 17 '13 at 21:48

14

Group constants into structures where possible:

Example, in my current game project, I have used below:

```
var CONST_WILD_TYPES = {
    REGULAR: 'REGULAR',
    EXPANDING: 'EXPANDING',
    STICKY: 'STICKY',
    SHIFTING: 'SHIFTING'
};
```

Assignment:

```
var wildType = CONST_WILD_TYPES.REGULAR;
```

Comparision:

```
if (wildType === CONST_WILD_TYPES.REGULAR) {
    // do something here
}
```

More recently I am using, for comparision:

```
switch (wildType) {
    case CONST_WILD_TYPES.REGULAR:
        // do something here
        break;
    case CONST_WILD_TYPES.EXPANDING:
        // do something here
        break;
}
```

IE11 is with new ES6 standard that has 'const' declaration.
Above works in earlier browsers like IE8, IE9 & IE10.

edited Apr 10 '16 at 2:06

answered Jul 2 '15 at 13:12

Manohar Reddy Poreddy

7,774 1 64 64

12

You can easily equip your script with a mechanism for constants that can be set but not altered. An attempt to alter them will generate an error.

```
/* author Keith Evetts 2009 License: LGPL
anonymous function sets up:
global function SETCONST (String name, mixed value)
global function CONST (String name)
constants once set may not be altered - console error is generated
they are retrieved as CONST(name)
the object holding the constants is private and cannot be accessed from the outer script
directly, only through the setter and getter provided
*/

(function(){
```

```
}
      if (!value) { throw new Error(' no value supplied for constant ' + name); }
      else if ((name in constants) ) { throw new Error('constant ' + name + ' is already
defined'); }
      else {
          constants[name] = value;
          return true;
    }
  };
  self.CONST = function(name) {
      if (typeof name !== 'string') { throw new Error('constant name is not a string');
}
      if ( name in constants ) { return constants[name]; }
      else { throw new Error('constant ' + name + ' has not been defined'); }
  };
}())


// -------------  demo --------------------------
SETCONST( 'VAT', 0.175 );
alert( CONST('VAT') );


//try to alter the value of VAT
try{
  SETCONST( 'VAT', 0.22 );
} catch ( exc )  {
   alert (exc.message);
}
//check old value of VAT remains
alert( CONST('VAT') );


// try to get at constants object directly
constants['DODO'] = "dead bird";  // error
```

edited May 1 '11 at 15:46

p.campbell
70.9k 56 226 303

answered Apr 8 '11 at 7:15

Keith
121 1 2

Forget IE and use the `const` keyword.

65.4k 38 145 200

9 works for me! but then I'm writing a chrome extension, so I know I'm on a sane browser ... – yoyo Nov 11 '11 at 16:50

1 @yoyo best part about writing extensions and addons -- no cross-browser support! – Ian Aug 17 '14 at 16:10

1 @Ian Welcome to 2019, the cross-browser inconsistency has almost disappeared :) – Fusseldieb Jun 3 at 12:21

9

Yet there is no exact cross browser predefined way to do it , you can achieve it by controlling the scope of variables as showed on other answers.

But i will suggest to use name space to distinguish from other variables. this will reduce the chance of collision to minimum from other variables.

Proper namespacing like

```
var iw_constant={
     name:'sudhanshu',
     age:'23'
     //all varibale come like this
}
```

so while using it will be `iw_constant.name` or `iw_constant.age`

You can also block adding any new key or changing any key inside iw_constant using Object.freeze method. However its not supported on legacy browser.

ex:

```
Object.freeze(iw_constant);
```

For older browser you can use polyfill for freeze method.

If you are ok with calling function following is best cross browser way to define constant. Scoping your object within a self executing function and returning a get function for your constants ex:

```
var iw_constant= (function(){
       var allConstant={
             name:'sudhanshu',
             age:'23'
             //all varibale come like this

       };

       return function(key){
          allConstant[key];
       }
    };
```

//to get the value use `iw_constant('name')` or `iw_constant('age')`

** In both example you have to be very careful on name spacing so that your object or function shouldn't be replaced through other library.(If object or function itself wil be replaced your whole constant will go)

edited May 23 '17 at 12:10
Community ♦
1 1

answered Feb 6 '13 at 7:06

Sudhanshu Yadav
2,043 13 18

7

For a while, I specified "constants" (which still weren't actually constants) in object literals passed through to `with()` statements. I thought it was so clever. Here's an example:

```
with ({
    MY_CONST : 'some really important value'
}) {
    alert(MY_CONST);
}
```

In the past, I also have created a `CONST` namespace where I would put all of my constants. Again, with the overhead. Sheesh.

Now, I just do `var MY_CONST = 'whatever';` to KISS.

answered Sep 24 '08 at 23:29

15 If theres something more evil than eval, it's definitely `with` . – NikiC May 1 '11 at 14:51

3 eval is very evil! It burnt down my house one time! – W3Geek Jan 2 '14 at 20:14

6

My opinion (works only with objects).

```
var constants = (function(){
  var a = 9;

  //GLOBAL CONSTANT (through "return")
  window.__defineGetter__("GCONST", function(){
    return a;
  });

  //LOCAL CONSTANT
  return {
    get CONST(){
      return a;
    }
  }
})();

constants.CONST = 8; //9
alert(constants.CONST); //9
```

Try! But understand - this is object, but not simple variable.

Try also just:

```
const a = 9;
```

edited Oct 29 '12 at 9:22

Alessandro Minoccheri

27.7k 19 94 139

answered Oct 29 '12 at 9:01

user1635543

97 1 3

5

It seems that most of the answers that I've come across is using functions to hold the constants. As many of the users of the MANY forums post about, the functions can be easily over written by users on the client side. I was intrigued by Keith Evetts' answer that the constants object can not be accessed by the outside, but only from the functions on the inside.

So I came up with this solution:

Put everything inside an anonymous function so that way, the variables, objects, etc. cannot be changed by the client side. Also hide the 'real' functions by having other functions call the 'real' functions from the inside. I also thought of using functions to check if a function has been changed by a user on the client side. If the functions have been changed, change them back using variables that are 'protected' on the inside and cannot be changed.

```
/*Tested in: IE 9.0.8; Firefox 14.0.1; Chrome 20.0.1180.60 m; Not Tested in Safari*/

(function(){
  /*The two functions _define and _access are from Keith Evetts 2009 License: LGPL
(SETCONST and CONST).
    They're the same just as he did them, the only things I changed are the variable
names and the text
    of the error messages.
  */

  //object literal to hold the constants
  var j = {};

  /*Global function _define(String h, mixed m). I named it define to mimic the way PHP
'defines' constants.
    The argument 'h' is the name of the const and has to be a string, 'm' is the value
of the const and has
    to exist. If there is already a property with the same name in the object holder,
then we throw an error.
    If not, we add the property and set the value to it. This is a 'hidden' function and
the user doesn't
    see any of your coding call this function. You call the _makeDef() in your code and
that function calls
    this function.    -    You can change the error messages to whatever you want them
to say.
  */
  self._define = function(h,m) {
      if (typeof h !== 'string') { throw new Error('I don\'t know what to do.'); }
      if (!m) { throw new Error('I don\'t know what to do.'); }
      else if ((h in j) ) { throw new Error('We have a problem!'); }
      else {
          j[h] = m;
```

```
  /*Global function _makeDef(String t, mixed y). I named it makeDef because we 'make the
define' with this
    function. The argument 't' is the name of the const and doesn't need to be all caps
because I set it
    to upper case within the function, 'y' is the value of the value of the const and
has to exist. I
    make different variables to make it harder for a user to figure out whats going on.
We then call the
    _define function with the two new variables. You call this function in your code to
set the constant.
    You can change the error message to whatever you want it to say.
  */
  self._makeDef = function(t, y) {
      if(!y) { throw new Error('I don\'t know what to do.'); return false; }
      q = t.toUpperCase();
      w = y;
      _define(q, w);
  };

  /*Global function _getDef(String s). I named it getDef because we 'get the define'
with this function. The
    argument 's' is the name of the const and doesn't need to be all capse because I set
it to upper case
    within the function. I make a different variable to make it harder for a user to
figure out whats going
    on. The function returns the _access function call. I pass the new variable and the
original string
    along to the _access function. I do this because if a user is trying to get the
value of something, if
    there is an error the argument doesn't get displayed with upper case in the error
message. You call this
    function in your code to get the constant.
  */
  self._getDef = function(s) {
      z = s.toUpperCase();
      return _access(z, s);
  };

  /*Global function _access(String g, String f). I named it access because we 'access'
the constant through
    this function. The argument 'g' is the name of the const and its all upper case, 'f'
is also the name
    of the const, but its the original string that was passed to the _getDef() function.
If there is an
    error, the original string, 'f', is displayed. This makes it harder for a user to
```

```
    the constant value. If not, we check if the 'f' variable exists, if not, set it to
the value of 'g' and
    throw an error. This is a 'hidden' function and the user doesn't see any of your
coding call this
    function. You call the _getDef() function in your code and that function calls this
function.
    You can change the error messages to whatever you want them to say.
  */
  self._access = function(g, f) {
      if (typeof g !== 'string') { throw new Error('I don\'t know what to do.'); }
      if ( g in j ) { return j[g]; }
      else { if(!f) { f = g; } throw new Error('I don\'t know what to do. I have no idea
what \''+f+'\' is.'); }
  };

  /*The four variables below are private and cannot be accessed from the outside script
except for the
    functions inside this anonymous function. These variables are strings of the four
above functions and
    will be used by the all-dreaded eval() function to set them back to their original
if any of them should
    be changed by a user trying to hack your code.
  */
  var _define_func_string = "function(h,m) {"+"      if (typeof h !== 'string') { throw
new Error('I don\\'t know what to do.'); }"+"      if (!m) { throw new Error('I don\\'t
know what to do.'); }"+"      else if ((h in j) ) { throw new Error('We have a
problem!'); }"+"      else {"+"          j[h] = m;"+"          return true;"+"    }"+"
}";
  var _makeDef_func_string = "function(t, y) {"+"      if(!y) { throw new Error('I
don\\'t know what to do.'); return false; }"+"      q = t.toUpperCase();"+"      w =
y;"+"      _define(q, w);"+"  }";
  var _getDef_func_string = "function(s) {"+"      z = s.toUpperCase();"+"      return
_access(z, s);"+"  }";
  var _access_func_string = "function(g, f) {"+"      if (typeof g !== 'string') { throw
new Error('I don\\'t know what to do.'); }"+"      if ( g in j ) { return j[g]; }"+"
else { if(!f) { f = g; } throw new Error('I don\\'t know what to do. I have no idea what
\\''+f+'\\' is.'); }"+"  }";

  /*Global function _doFunctionCheck(String u). I named it doFunctionCheck because we're
'checking the functions'
    The argument 'u' is the name of any of the four above function names you want to
check. This function will
    check if a specific line of code is inside a given function. If it is, then we do
nothing, if not, then
    we use the eval() function to set the function back to its original coding using the
function string
```

```
function. You call the
    doCodeCheck() function and that function calls this function.    -    You can change
the error messages to
    whatever you want them to say.
  */
  self._doFunctionCheck = function(u) {
      var errMsg = 'We have a BIG problem! You\'ve changed my code.';
      var doError = true;
      d = u;
      switch(d.toLowerCase())
      {
           case "_getdef":
               if(_getDef.toString().indexOf("z = s.toUpperCase();") != -1) { /*do
nothing*/ }
               else { eval("_getDef = "+_getDef_func_string); if(doError === true) {
throw new Error(errMsg); } }
               break;
           case "_makedef":
               if(_makeDef.toString().indexOf("q = t.toUpperCase();") != -1) { /*do
nothing*/ }
               else { eval("_makeDef = "+_makeDef_func_string); if(doError === true) {
throw new Error(errMsg); } }
               break;
           case "_define":
               if(_define.toString().indexOf("else if((h in j) ) {") != -1) { /*do
nothing*/ }
               else { eval("_define = "+_define_func_string); if(doError === true) {
throw new Error(errMsg); } }
               break;
           case "_access":
               if(_access.toString().indexOf("else { if(!f) { f = g; }") != -1) { /*do
nothing*/ }
               else { eval("_access = "+_access_func_string); if(doError === true) {
throw new Error(errMsg); } }
               break;
           default:
                if(doError === true) { throw new Error('I don\'t know what to do.'); }
      }
  };

  /*Global function _doCodeCheck(String v). I named it doCodeCheck because we're 'doing
a code check'. The argument
    'v' is the name of one of the first four functions in this script that you want to
check. I make a different
    variable to make it harder for a user to figure out whats going on. You call this
function in your code to check
```

```
        l = v;
        _doFunctionCheck(l);
    };
}())
```

It also seems that security is really a problem and there is not way to 'hide' you programming from the client side. A good idea for me is to compress your code so that it is really hard for anyone, including you, the programmer, to read and understand it. There is a site you can go to: http://javascriptcompressor.com/. (This is not my site, don't worry I'm not advertising.) This is a site that will let you compress and obfuscate Javascript code for free.

1. Copy all the code in the above script and paste it into the top textarea on the javascriptcompressor.com page.
2. Check the Base62 encode checkbox, check the Shrink Variables checkbox.
3. Press the Compress button.
4. Paste and save it all in a .js file and add it to your page in the head of your page.

edited Aug 3 '12 at 5:31

answered Aug 3 '12 at 4:23

Steven Kapaun
51 1 2

This is a good solution that could be nicely wrapped up as a library to include. But I dislike the naming of your variables in this code. Why drop the descriptive names like "name" and "value" as used in Keith's code? Minor issue, but still. – Cordle Jun 3 '14 at 14:24

5

Clearly this shows the need for a standardized cross-browser const keyword.

But for now:

```
var myconst = value;
```

or

```
Object['myconst'] = value;
```

Both seem sufficient and anything else is like shooting a fly with a bazooka.


Devin Burke
11.3k 11 45 77


codemuncher
59 1 1

take the good old var myconst = value; and for debugging use extra debugging code... - works like crazy as long as not all browsers support const – Quicker Mar 10 '16 at 21:28

4

I use `const` instead of `var` in my Greasemonkey scripts, but it is because they will run only on Firefox...
Name convention can be indeed the way to go, too (I do both!).

answered Sep 24 '08 at 22:49


PhiLho
36.1k 4 84 124

4

In JavaScript my practice has been to avoid constants as much as I can and use strings instead. Problems with constants appear when you want to expose your constants to the outside world:

For example one could implement the following Date API:

```
date.add(5, MyModule.Date.DAY).add(12, MyModule.Date.HOUR)
```

But it's much shorter and more natural to simply write:

```
date.add(5, "days").add(12, "hours")
```

This way "days" and "hours" really act like constants, because you can't change from the outside how many seconds "hours" represents. But it's easy to overwrite `MyModule.Date.HOUR` .

This kind of approach will also aid in debugging. If Firebug tells you `action === 18` it's pretty hard to figure out what it means, but when you see `action === "save"` then it's immediately clear.

answered Jan 17 '11 at 14:07


Rene Saarsoo

It is unfortunately pretty easy to make spelling mistakes - e.g. `"Hours"` instead of `"hours"` - but an IDE might let you know early on that `Date.Hours` is not defined. – le_m Jun 13 '16 at 3:09

4

Okay, this is ugly, but it gives me a constant in Firefox and Chromium, an inconstant constant (WTF?) in Safari and Opera, and a variable in IE.

Of course eval() is evil, but without it, IE throws an error, preventing scripts from running.

Safari and Opera support the const keyword, but *you can change the const's value*.

In this example, server-side code is writing JavaScript to the page, replacing {0} with a value.

```
try{
    // i can haz const?
    eval("const FOO='{0}';");
    // for reals?
    var original=FOO;
    try{
        FOO='?NO!';
    }catch(err1){
        // no err from Firefox/Chrome - fails silently
        alert('err1 '+err1);
    }
    alert('const '+FOO);
    if(FOO=='?NO!'){
        // changed in Sf/Op - set back to original value
        FOO=original;
    }
}catch(err2){
    // IE fail
    alert('err2 '+err2);
    // set var (no var keyword - Chrome/Firefox complain about redefining const)
    FOO='{0}';
    alert('var '+FOO);
}
alert('FOO '+FOO);
```

What is this good for? Not much, since it's not cross-browser. At best, maybe a little peace of mind that at least *some* browsers won't let bookmarklets or third-party script modify the value.

answered Jun 28 '11 at 4:39

Webveloper
894 9 15

3 `inconstant constant` ? WTF? – Derek 朕會功夫 Dec 8 '11 at 4:59

1 Love that code! Ugly as heck, but a good test for const support. =) – Stein G. Strindhaug Dec 13 '11 at 8:23

1 somewhat funny, ey - how many lines can you type to declare a const? – Quicker Mar 10 '16 at 21:27

4

If it is worth mentioning, you can define constants in angular using `$provide.constant()`

```
angularApp.constant('YOUR_CONSTANT', 'value');
```

edited Jan 16 '15 at 12:35

answered Dec 25 '14 at 11:16

Muhammad Reda
19.6k 10 77 97

...and you can use const in VBA... xbrowser?... ups... ;) – Quicker Mar 10 '16 at 21:30

OP asks about javascript, answer deals with specific heavily opinionated JS framework. Practically off-topic. – rounce May 10 '16 at 12:43

2 @rounce: Off-topic answers are still answers, don't *flag* them as **Not an Answer** but *downvote* and *vote to delete* instead. See How do I properly use the "Not an Answer" flag? – Kevin Guan May 10 '16 at 14:08

@KevinGuan Noted, will do in future. – rounce May 10 '16 at 17:18

4

An improved version of Burke's answer that lets you do `CONFIG.MY_CONST` instead of `CONFIG.get('MY_CONST')`.

It requires IE9+ or a real web browser.

```
var CONFIG = (function() {
    var constants = {
        'MY_CONST': 1
```

```
    var result = {};
    for (var n in constants)
        if (constants.hasOwnProperty(n))
            Object.defineProperty(result, n, { value: constants[n] });

    return result;
}());
```

** The properties are read-only, only if the initial values are immutable.*

edited May 23 '17 at 10:31
Community ♦
1 1

answered Dec 25 '14 at 10:52

Şafak Gür
3,727 3 42 78

4

JavaScript ES6 (re-)introduced the `const` keyword which is supported in all major browsers.

> Variables declared via `const` cannot be re-declared or re-assigned.

Apart from that, `const` behaves similar to `let` .

It behaves as expected for primitive datatypes (Boolean, Null, Undefined, Number, String, Symbol):

```
const x = 1;
x = 2;
console.log(x); // 1 ...as expected, re-assigning fails
```

Attention: Be aware of the pitfalls regarding objects:

```
const o = {x: 1};
o = {x: 2};
console.log(o); // {x: 1} ...as expected, re-assigning fails

o.x = 2;
console.log(o); // {x: 2} !!! const does not make objects immutable!

const a = [];
a = [1];
```

```
a.push(1);
console.log(a); // [1] !!! const does not make objects immutable
```

If you really need an immutable and absolutely constant object: Just use `const ALL_CAPS` to make your intention clear. It is a good convention to follow for all `const` declarations anyway, so just rely on it.

answered Jun 13 '16 at 2:44

le_m
12.7k 4 42 50

From **IE11** only :-(– Mo. Jun 15 '17 at 8:39

3

Another alternative is something like:

```
var constants = {
      MY_CONSTANT : "myconstant",
      SOMETHING_ELSE : 123
    }
  , constantMap = new function ConstantMap() {};

for(var c in constants) {
  !function(cKey) {
    Object.defineProperty(constantMap, cKey, {
      enumerable : true,
      get : function(name) { return constants[cKey]; }
    })
  }(c);
}
```

Then simply: `var foo = constantMap.MY_CONSTANT`

If you were to `constantMap.MY_CONSTANT = "bar"` it would have no effect as we're trying to use an assignment operator with a getter, hence `constantMap.MY_CONSTANT === "myconstant"` would remain true.

edited Nov 26 '14 at 15:27

answered Aug 28 '14 at 11:42

rounce

in Javascript already exists constants. You define a constant like this:

3

```
const name1 = value;
```

This cannot change through reassignment.

answered May 4 '15 at 15:50

Erik Lucio
653 5 8

Per the link in the answer, this is an experimental feature and should be used with caution. – Johnie Karr Jun 13 '15 at 2:02

Of course, I agree with you. But in the last versions of browsers It works. – Erik Lucio Jun 15 '15 at 7:22

3

The keyword 'const' was proposed earlier and now it has been officially included in ES6. By using the const keyword, you can pass a value/string that will act as an immutable string.

answered Jul 28 '15 at 10:52

Ritumoni Sharma
96 6

2

Introducing constants into JavaScript is at best a hack.

A nice way of making persistent and globally accessible values in JavaScript would be declaring an object literal with some "read-only" properties like this:

```
my={get constant1(){return "constant 1"},
    get constant2(){return "constant 2"},
    get constant3(){return "constant 3"},
    get constantN(){return "constant N"}
    }
```

```
my.constant1; >> "constant 1"
my.constant1 = "new constant 1";
my.constant1; >> "constant 1"
```

As we can see, the "my.constant1" property has preserved its original value. You've made yourself some nice 'green' temporary constants...

But of course this will only guard you from accidentally modifying, altering, nullifying, or emptying your property constant value with a direct access as in the given example.

Otherwise I still think that constants are for dummies. And I still think that exchanging your great freedom for a small corner of deceptive security is the worst trade possible.

edited Mar 20 '12 at 2:45

Bill the Lizard
306k 159 506 799

answered Feb 10 '12 at 6:12
user1170379

2

`Rhino.js` implements `const` in addition to what was mentioned above.

answered Nov 5 '12 at 15:51

isomorphismes
5,316 8 43 65

---

1 2 next

protected by Mr. Alien Jul 22 '13 at 8:02

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?