Detecting an undefined object property

Asked 16 years, 4 months ago Modified 1 year, 4 months ago

Viewed 1.5m times

How do I check if an object property in JavaScript is undefined?

3242

javascript object undefined object-property

Share

Improve this question

Follow

edited Aug 16, 2022 at 15:34

TylerH

21.2k ● 76 ● 79 ● 110

asked Aug 26, 2008 at 7:25

Matt Sheppard

118k ● 46 ● 113 ● 134

Look for recent answers in this thread. In 'modern' javascript, consider using the `in` operator: `'key' in obj ? 'obj has key property' : 'obj does not have key property'` ` – AlejandroVD Aug 23, 2021 at 16:19

I'm open to picking a new 'correct' answer if you have one in mind that covers the old way as completely as the current one and also addresses more modern options.
– Matt Sheppard Aug 24, 2021 at 20:59

49 Answers

Sorted by: Highest score (default)

1 2 Next

2950

The usual way to check if the value of a property is the special value `undefined`, is:

```
if(o.myProperty === undefined) {
  alert("myProperty value is the special value `undefi
}
```

To check if an object does not actually have such a property, and will therefore return `undefined` by default when you try to access it:

```
if(!o.hasOwnProperty('myProperty')) {
  alert("myProperty does not exist");
}
```

To check if the value associated with an identifier is the special value `undefined`, *or* if that identifier has not been declared:

```
if(typeof myVariable === 'undefined') {
  alert('myVariable is either the special value `undef
declared');
}
```

Note: this last method is the only way to refer to an **undeclared** identifier without an early error, which is different from having a value of `undefined`.

In versions of JavaScript prior to ECMAScript 5, the property named "undefined" on the global object was writeable, and therefore a simple check `foo === undefined` might behave unexpectedly if it had accidentally been redefined. In modern JavaScript, the property is read-only.

However, in modern JavaScript, "undefined" is not a keyword, and so variables inside functions can be named "undefined" and shadow the global property.

If you are worried about this (unlikely) edge case, you can use the void operator to get at the special `undefined` value itself:

```
if(myVariable === void 0) {
  alert("myVariable is the special value `undefined`")
}
```

Share Improve this answer

Follow

edited Feb 17, 2022 at 9:13

MWO

2,810 ● 2 ● 11 ● 27

answered Jan 6, 2009 at 12:27

Erwin

---

10 if something is null the it is defined (as null), but you can conjugate the too checks. The annoying detail of the above code is that you can't define a function to check it, well you can define the function... but try to use it. – neu-rah Jun 25, 2012 at 19:20

6 @neu-rah why can't you write a function? why wouldn't something like this work? It seems to work for me. Is there a case I'm not considering? jsfiddle.net/djH9N/6 – Zack Sep 24, 2012 at 19:01

8 @Zack Your tests for isNullorUndefined did not consider the case where you call isNullOrUndefined(f) and f is undeclared (i.e. where there is no "var f" declaration). – pnkfelix Feb 15, 2013 at 15:08

146 Blah, thousands of votes now. This is the worst possible way to do it. I hope passers-by see this comment and decide to check… ahem… *other* answers. – Ry- ♦ May 14, 2014 at 3:05

22 You can just use `obj !== undefined` now. `undefined` used to be mutable, like `undefined = 1234` what would cause interesting results. But after Ecmascript 5, it's not writable anymore, so we can use the simpler version. codereadability.com/how-to-check-for-undefined-in-javascript – Bruno Buccolo Mar 15, 2016 at 20:50

991

I believe there are a number of incorrect answers to this topic. Contrary to common belief, "undefined" is **not** a keyword in JavaScript and can in fact have a value assigned to it.

Correct Code

The most robust way to perform this test is:

```
if (typeof myVar === "undefined")
```

This will always return the correct result, and even handles the situation where `myVar` is not declared.

Degenerate code. DO NOT USE.

```
var undefined = false;  // Shockingly, this is complet
if (myVar === undefined) {
    alert("You have been misled. Run away!");
}
```

Additionally, `myVar === undefined` will raise an error in the situation where myVar is undeclared.

Share Improve this answer

Follow

edited Aug 15, 2017 at 20:52

Dave Jarvis
31.2k ● 43 ● 181 ● 323

answered Aug 23, 2010 at 18:03

MarkPflug
29.5k ● 8 ● 50 ● 58

25 in addition to Marks comments, I don't get this: "myVar === undefined will raise an error in the situation where myVar is undeclared." - why is this bad? Why would I *not* want to have an error if I'm referencing undeclared variables? – eis Aug 20, 2013 at 15:12

6 Also keep in mind you can always do `void 0` to get the value that `undefined` points to. So you can do `if (myVar === void 0)`. the `0` isn't special, you can literally put any expression there. – Claudiu Oct 3, 2013 at 17:45

35 In modern browsers (FF4+, IE9+, Chrome unknown), it's no longer possible to modify `undefined`. [MDN: undefined](#)
– user247702 Feb 7, 2014 at 14:09

13 This answer is incorrect as well. The question was about undefined object properties, not undefined variables. There is a significant difference. It is, for example, perfectly reasonable to do `if (obj.field === undefined)`. I think the risk of someone doing `var undefined = false;` is overrated. You will have to program unreasonably defensive if you want to protect against all such kinds of side effects caused by poor programming. – Zero3 Dec 15, 2015 at 15:38

10 Funny that people suggest these silly and error-prone hacks to avoid a shadowed `undefined` (which could only be done by a terrible developer), yet they blithely use other global identifiers that could also have been shadowed. Bizarre. Just bizarre. – user8897421 Dec 18, 2017 at 13:39

289

Many answers here are vehement in recommending `typeof`, but `typeof` *is a bad choice*. It should *never* be used for checking whether variables have the value `undefined`, because it acts as a combined check for the value `undefined` and for whether a variable exists. In the vast majority of cases, you know when a variable exists, and `typeof` will just introduce the potential for a silent failure if you make a typo in the variable name or in the string literal `'undefined'`.

```
var snapshot = …;

if (typeof snaposhot === 'undefined') {
    //         ^
    // misspelled¹ – this will never run, but it won’t
}
```

```
var foo = …;

if (typeof foo === 'undefned') {
    //                     ^
    // misspelled – this will never run, but it won’t
}
```

So unless you’re doing feature detection², where there’s uncertainty whether a given name will be in scope (like checking `typeof module !== 'undefined'` as a step in code specific to a CommonJS environment), `typeof` is a harmful choice when used on a variable, and the correct option is to compare the value directly:

```
var foo = …;

if (foo === undefined) {
    ⋮
}
```

Some common misconceptions about this include:

- that reading an “uninitialized” variable (`var foo`) or parameter (`function bar(foo) { … }`, called as `bar()`) will fail. This is simply not true – variables without explicit initialization and parameters that weren’t given values always become `undefined`, and are always in scope.
- that `undefined` can be overwritten. It’s true that `undefined` isn’t a keyword, but it *is* read-only and

> non-configurable. There are other built-ins you probably don’t avoid despite their non-keyword status (`Object`, `Math`, `NaN`…) and practical code usually isn’t written in an actively malicious environment, so this isn’t a good reason to be worried about `undefined`. (But if you are writing a code generator, feel free to use `void 0`.)

With how variables work out of the way, it’s time to address the actual question: object properties. There is no reason to ever use `typeof` for object properties. The earlier exception regarding feature detection doesn’t apply here – `typeof` only has special behaviour on variables, and expressions that reference object properties are not variables.

This:

```
if (typeof foo.bar === 'undefined') {
    ⋮
}
```

is *always exactly equivalent* to this[3]:

```
if (foo.bar === undefined) {
    ⋮
}
```

and taking into account the advice above, to avoid confusing readers as to why you’re using `typeof`, because it makes the most sense to use `===` to check for equality, because it could be refactored to checking a variable's value later, and because it just plain looks better, **you should always use `=== undefined`[3] here as well**.

Something else to consider when it comes to object properties is whether you really want to check for `undefined` at all. A given property name can be absent on an object (producing the value `undefined` when read), present on the object itself with the value `undefined`, present on the object's prototype with the value `undefined`, or present on either of those with a non-`undefined` value. `'key' in obj` will tell you whether a key is anywhere on an object's prototype chain, and `Object.prototype.hasOwnProperty.call(obj, 'key')` will tell you whether it's directly on the object. I won't go into detail in this answer about prototypes and using objects as string-keyed maps, though, because it's mostly intended to counter all the bad advice in other answers irrespective of the possible interpretations of the original question. Read up on object prototypes on MDN for more!

[1] unusual choice of example variable name? this is real dead code from the NoScript extension for Firefox.

[2] don't assume that not knowing what's in scope is okay in general, though. bonus vulnerability caused by abuse of dynamic scope: Project Zero 1225

[3] once again assuming an ES5+ environment and that `undefined` refers to the `undefined` property of the global object.

Share Improve this answer edited Jan 30, 2023 at 22:12

Follow

answered Feb 26, 2014 at 21:17

Ry- ♦
225k ● 56 ● 487 ● 496

1 Any non-default context can also overwrite, say, `Math`, or `Object`, or `setTimeout`, or literally anything that you expect to find in the global scope by default. – CherryDT Mar 17, 2020 at 21:21

ohw kay foo !== void 0 works and is shorter guess i can use that :3 and i knew you were referring to yer own answer when ya commented that >:D – Fuseteam Jul 29, 2020 at 23:54

now it doesn't document what it does tho and might be confusing to read so instead i'll make a function isDefined return true if is not void 0 and false if it is and return that instead >:3 – Fuseteam Jul 30, 2020 at 0:02

@Fuseteam: Use `!== undefined`. Actually, I should probably remove the advice about `void 0`. – Ry- ♦ Jul 30, 2020 at 0:18

@Fuseteam: It's not as readable and has no practical advantages over `undefined`. – Ry- ♦ Jul 30, 2020 at 18:41

175

In JavaScript there is **null** and there is **undefined**. They have different meanings.

- **undefined** means that the variable value has not been defined; it is not known what the value is.
- **null** means that the variable value is defined and set to null (has no value).

Marijn Haverbeke states, in his free, online book "Eloquent JavaScript" (emphasis mine):

> There is also a similar value, null, whose meaning is 'this value is defined, but it does not have a value'. The difference in meaning between undefined and null is mostly academic, and usually not very interesting. **In practical programs, it is often necessary to check whether something 'has a value'. In these cases, the expression something == undefined may be used, because, even though they are not exactly the same value, null == undefined will produce true.**

So, I guess the best way to check if something was undefined would be:

```
if (something == undefined)
```

Object properties should work the same way.

```
var person = {
    name: "John",
    age: 28,
    sex: "male"
};

alert(person.name); // "John"
alert(person.fakeVariable); // undefined
```

Share Improve this answer

Follow

edited Aug 21, 2020 at 15:26

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Aug 26, 2008 at 7:36

codes_occasionally
9,566 ● 9 ● 45 ● 61

46 if (something == undefined) is better written as if (something === undefined) – Sebastian Rittau Nov 30, 2009 at 9:47

61 It should be pointed out that this is not entirely safe. `undefined` is just a variable that can be re-assigned by the user: writing `undefined = 'a';` will cause your code to no longer do what you think it does. Using `typeof` is better and also works for variables (not just properties) that haven't been declared. – Gabe Moothart Apr 14, 2010 at 15:18

7 if something is an undefined global variable, (something == undefined) brings up javascript error. – Morgan Cheng Apr 21, 2010 at 3:04

8 The problem with this is that if var a = null then a == undefined evaluates to true, even though a is most certainly defined. – Andrew May 19, 2011 at 18:50

7 This interpretation of the "Eloquent Javascript" comment is *backward*. If you really do just want to check for undefined, the suggested code will not work (it will also detect the condition defined but no value has been assined yet [i.e.null]).a null value. The suggested code "if (something == undefined) ..." checks for *both* undefined and null (no value set), i.e. it's interpreted as "if ((something is undefined) OR (something is null)) ..." What the author is saying is that often what you *really* want is to check for *both* undefined and null. – Chuck Kollars May 17, 2012 at 22:35

137

What does this mean: **"undefined object property"**?

Actually it can mean two quite different things! First, it can mean *the property that has never been defined* in the object and, second, it can mean the *property that has an undefined value*. Let's look at this code:

```
var o = { a: undefined }
```

Is `o.a` undefined? Yes! Its value is undefined. Is `o.b` undefined? Sure! There is no property 'b' at all! OK, see now how different approaches behave in both situations:

```
typeof o.a == 'undefined' // true
typeof o.b == 'undefined' // true
o.a === undefined // true
o.b === undefined // true
'a' in o // true
'b' in o // false
```

We can clearly see that `typeof obj.prop == 'undefined'` and `obj.prop === undefined` are equivalent, and they do not distinguish those different situations. And `'prop' in obj` can detect the situation when a property hasn't been defined at all and doesn't pay attention to the property value which may be undefined.

So what to do?

1) You want to know if a property is undefined by either the first or second meaning (the most typical situation).

```
obj.prop === undefined // IMHO, see "final fight" belo
```

2) You want to just know if object has some property and don't care about its value.

```
'prop' in obj
```

Notes:

- You can't check an object and its property at the same time. For example, this `x.a === undefined` or this `typeof x.a == 'undefined'` raises `ReferenceError: x is not defined` if x is not defined.
- Variable `undefined` is a global variable (so actually it is `window.undefined` in browsers). It has been supported since ECMAScript 1st Edition and since ECMAScript 5 it is **read only**. So in modern browsers it can't be *redefined to true* as many authors love to frighten us with, but this is still a true for older browsers.

Final fight: `obj.prop === undefined` vs `typeof obj.prop == 'undefined'`

Pluses of `obj.prop === undefined`:

- It's a bit shorter and looks a bit prettier

- The JavaScript engine will give you an error if you have misspelled `undefined`

Minuses of `obj.prop === undefined`:

- `undefined` can be overridden in old browsers

Pluses of `typeof obj.prop == 'undefined'`:

- It is really universal! It works in new and old browsers.

Minuses of `typeof obj.prop == 'undefined'`:

- `'undefned'` (*misspelled*) here is just a string constant, so the JavaScript engine can't help you if you have misspelled it like I just did.

Update (for server-side JavaScript):

Node.js supports the global variable `undefined` as `global.undefined` (it can also be used without the 'global' prefix). I don't know about other implementations of server-side JavaScript.

Share Improve this answer Follow

edited Sep 26, 2015 at 19:01

answered Aug 8, 2013 at 20:28

Konstantin Smolyanin
19k ● 13 ● 64 ● 58

@Bergi thank you for your comment. I have corrected my answer. In my defense I can say that currently (as of v.0.10.18) official Node.js documentation says nothing about `undefined` as a member of `global`. Also neither `console.log(global);` nor `for (var key in global) { ... }` doesn't show *undefined* as a member of *global*. But test like `'undefined' in global` show the opposite. – Konstantin Smolyanin Sep 11, 2013 at 10:53

4 It didn't need extra documentation since it's in the EcmaScript spec, which also says that `[[Enumerable]]` is false :-) – Bergi Sep 11, 2013 at 11:00

5 Regarding `Minuses of typeof obj.prop == 'undefined'`, this can be avoided by writing as `typeof obj.prop == typeof undefined`. This also gives a very nice symmetry. – hlovdal Oct 24, 2014 at 11:01

3 @hlovdal: That's totally pointless vs. `obj.prop === undefined`. – Ry- ♦ Apr 11, 2018 at 21:38

2 When we are true to the question headline **„*Detecting* an undefined property“**, not true to the (different and much easier) question in the first sentence („check if undefined...“), you answer `if ('foo' in o`)… your answer is truly the first correct answer here. Pretty much everybody else just answers that sentence. – Frank N Jun 11, 2018 at 8:46

75

The issue boils down to three cases:

1. The object has the property and its value is not `undefined`.
2. The object has the property and its value is `undefined`.

3. The object does not have the property.

This tells us something I consider important:

There is a difference between an undefined member and a defined member with an undefined value.

But unhappily `typeof obj.foo` does not tell us which of the three cases we have. However we can combine this with `"foo" in obj` to distinguish the cases.

```
                               |  typeof obj.x === 'un
1.                     { x:1 } |  false
2.    { x : (function(){})() } |  true
3.                          {} |  true
```

Its worth noting that these tests are the same for `null` entries too

```
                               |  typeof obj.x === 'un
                    { x:null } |  false
```

I'd argue that in some cases it makes more sense (and is clearer) to check whether the property is there, than checking whether it is undefined, and the only case where this check will be different is case 2, the rare case of an actual entry in the object with an undefined value.

For example: I've just been refactoring a bunch of code that had a bunch of checks whether an object had a given property.

```
if( typeof blob.x != 'undefined' ) {  fn(blob.x); }
```

Which was clearer when written without a check for undefined.

```
if( "x" in blob ) { fn(blob.x); }
```

But as has been mentioned these are not exactly the same (but are more than good enough for my needs).

Share Improve this answer Follow

edited Jan 30, 2014 at 2:49

answered Jun 8, 2011 at 4:04

Michael Anderson
73.3k ● 8 ● 146 ● 192

10 Hi Michael. Great suggestion, and I think it does make things cleaner. One gotcha that I found, however, is when using the ! operator with "in". You have to say `if (!("x" in blob)) {}` with brackets around the in, because the ! operator has precedence over 'in'. Hope that helps someone. – Simon E. Jun 15, 2011 at 0:28

Sorry Michael, but this is incorrect, or at least misleading, in light of the original question. 'in' is not a sufficient way to test whether an object property has typeof undefined. For proof, please see this fiddle: jsfiddle.net/CsLKJ/4 – tex Feb 25, 2012 at 12:04

2 Those two code parts do a different thing! Consider and object given by `a = {b: undefined}` ; then `typeof a.b`

`=== typeof a.c === 'undefined'` but `'b' in a` and `!('c' in a)`. – mgol Sep 27, 2012 at 14:07

3 +1. The OP doesn't make it clear whether the property exists and has the value *undefined*, or whether the property itself is undefined (i.e. doesn't exist). – RobG Apr 1, 2014 at 1:12

I would suggest changing point (2.) in your first table to `{ x : undefined }` or at least add it as another alternative to (2.) in the table - I had to think for a moment to realize that point (2.) evaluates to `undefined` (although you mention that later on). – mucaho May 14, 2015 at 16:32

48

```
if ( typeof( something ) == "undefined")
```

This worked for me while the others didn't.

Share Improve this answer Follow

edited Sep 1, 2010 at 21:49

Peter Ajtai
57.6k ● 13 ● 123 ● 141

answered Jul 27, 2010 at 16:03

Kevin
589 ● 4 ● 2

50 parens are unnecessary since typeof is an operator – aehlke Aug 10, 2010 at 11:22

13 But they make it clearer what is being checked. Otherwise it might be read as `typeof (something == "undefined")`. – Abhi Beckert Sep 6, 2012 at 0:28

If you need the parentheses, then you should learn operator precedence in JS: developer.mozilla.org/en-

US/docs/Web/JavaScript/Reference/… – Ian Mar 7, 2014 at 17:17

12 Parenthesis are useful precisely because you do NOT need to learn operator precedence in JS, nor do you need to speculate whether future maintenance programmers will need to learn operator precedence in JS. – DaveWalley Apr 11, 2014 at 14:48

31 Parenthesis are useful to clarify things. But in this case they just make the operator look like a function. No doubt this clarifies the intent of the programmer. But if you're unsure about operator precedence you should rather write it as `(typeof something) === "undefined"`. – Robert May 14, 2014 at 18:31

44

I'm not sure where the origin of using `===` with `typeof` came from, and as a convention I see it used in many libraries, but the typeof operator returns a string literal, and we know that up front, so why would you also want to type check it too?

```
typeof x;                          // some string literal
"undefined"
if (typeof x === "string") {    // === is redundant bec
typeof returns a string literal
if (typeof x == "string") {     // sufficient
```

Share Improve this answer Follow

edited Jun 29, 2011 at 7:21

Simon E.
58.4k ● 18 ● 144 ● 137

answered Sep 22, 2010 at 14:20

Eric
473 ● 4 ● 2

Great point Eric. Is there a performance hit from checking type also? – Simon E. Jun 29, 2011 at 7:16

6 @Simon: quite the contrary - one could expect slight performance hit from avoiding coercion in '===' case. Quick and dirty test has shown '===' is 5% faster than '==' under FF5.0.1 – Antony Hatchkins Dec 18, 2011 at 8:24

6 More thorough test has shown that under FF,IE and Chrome '==' is more or less faster than '===' (5-10%) and Opera doesn't make any difference at all: jsperf.com/triple-equals-vs-twice-equals/6 – Antony Hatchkins Dec 18, 2011 at 9:55

4 Using `==` still requires *at least* a type check - the interpreter can't compare the two operands without knowing their type first. – Alnitak Aug 13, 2012 at 15:12

8 `==` is one less character than `===` :) – svidgen Jun 28, 2013 at 14:54

32

I didn't see (hope I didn't miss it) anyone checking the object before the property. So, this is the shortest and most effective (though not necessarily the most clear):

```
if (obj && obj.prop) {
  // Do something;
}
```

If the obj or obj.prop is undefined, null, or "falsy", the if statement will not execute the code block. This is *usually* the desired behavior in most code block statements (in JavaScript).

UPDATE: (7/2/2021)

The latest version of JavaScript introduces a new operator for optional chaining: `?.`

This is probably going to be the most explicit and efficient method of checking for the existence of object properties, moving forward.

Ref: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Optional_chaining

Share Improve this answer Follow

edited Jul 2, 2021 at 12:49

answered Sep 25, 2012 at 18:41

Joe Johnson
1,824 ● 16 ● 20

2 If you want to know why this works: Javascript: Logical Operators and truthy / falsy – mb21 Feb 4, 2013 at 16:57

if you want to assign the property to a variable if it's defined, not null and not falsey, else use some default value, you can use: `var x = obj && obj.prop || 'default';` – Stijn de Witt Nov 1, 2015 at 2:27

I believe the question is for checking against undefined explicitly. Your condition check against all false values of JS.

– NikoKyriakid Jul 20, 2018 at 9:49

Don't do this, it fails if `obj.prop` is `false` , `0` , `""` and various other falsey values. This is exactly what we should not do and also applies to optional chaining, which should not be used in this manner. – Seth Apr 29, 2022 at 0:11

27

Crossposting my answer from related question *How can I check for "undefined" in JavaScript?*.

Specific to this question, see test cases with `someObject.<whatever>`.

Some scenarios illustrating the results of the various answers: http://jsfiddle.net/drzaus/UVjM4/

(Note that the use of `var` for `in` tests make a difference when in a scoped wrapper)

Code for reference:

```
(function(undefined) {
    var definedButNotInitialized;
    definedAndInitialized = 3;
    someObject = {
        firstProp: "1"
        , secondProp: false
        // , undefinedProp not defined
    }
    // var notDefined;

    var tests = [
        'definedButNotInitialized in window',
        'definedAndInitialized in window',
        'someObject.firstProp in window',
        'someObject.secondProp in window',
        'someObject.undefinedProp in window',
        'notDefined in window',

        '"definedButNotInitialized" in window',
        '"definedAndInitialized" in window',
        '"someObject.firstProp" in window',
        '"someObject.secondProp" in window',
        '"someObject.undefinedProp" in window',
        '"notDefined" in window',

        'typeof definedButNotInitialized == "undefined
        'typeof definedButNotInitialized === typeof un
        'definedButNotInitialized === undefined',
        '! definedButNotInitialized',
        '!! definedButNotInitialized',

        'typeof definedAndInitialized == "undefined"',
        'typeof definedAndInitialized === typeof undef
        'definedAndInitialized === undefined',
        '! definedAndInitialized',
        '!! definedAndInitialized',

        'typeof someObject.firstProp == "undefined"',
        'typeof someObject.firstProp === typeof undefi
        'someObject.firstProp === undefined',
        '! someObject.firstProp',
        '!! someObject.firstProp',

        'typeof someObject.secondProp == "undefined"',
        'typeof someObject.secondProp === typeof undef
        'someObject.secondProp === undefined',
        '! someObject.secondProp',
        '!! someObject.secondProp',

        'typeof someObject.undefinedProp == "undefined
        'typeof someObject.undefinedProp === typeof un
        'someObject.undefinedProp === undefined',
        '! someObject.undefinedProp',
        '!! someObject.undefinedProp',

        'typeof notDefined == "undefined"',
        'typeof notDefined === typeof undefined',
        'notDefined === undefined',
        '! notDefined',
        '!! notDefined'
    ];

    var output = document.getElementById('results');
    var result = '';
    for(var t in tests) {
        if( !tests.hasOwnProperty(t) ) continue; // bl

        try {
            result = eval(tests[t]);
        } catch(ex) {
            result = 'Exception--' + ex;
        }
        console.log(tests[t], result);
        output.innerHTML += "\n" + tests[t] + ": " + r
    }
})();
```

And results:

```
definedButNotInitialized in window: true
definedAndInitialized in window: false
someObject.firstProp in window: false
someObject.secondProp in window: false
someObject.undefinedProp in window: true
notDefined in window: Exception--ReferenceError: notDe
"definedButNotInitialized" in window: false
"definedAndInitialized" in window: true
"someObject.firstProp" in window: false
"someObject.secondProp" in window: false
"someObject.undefinedProp" in window: false
"notDefined" in window: false
typeof definedButNotInitialized == "undefined": true
typeof definedButNotInitialized === typeof undefined:
definedButNotInitialized === undefined: true
! definedButNotInitialized: true
!! definedButNotInitialized: false
typeof definedAndInitialized == "undefined": false
typeof definedAndInitialized === typeof undefined: fal
definedAndInitialized === undefined: false
! definedAndInitialized: false
```

```
!! definedAndInitialized: true
typeof someObject.firstProp == "undefined": false
typeof someObject.firstProp === typeof undefined: fals
someObject.firstProp === undefined: false
! someObject.firstProp: false
!! someObject.firstProp: true
typeof someObject.secondProp == "undefined": false
typeof someObject.secondProp === typeof undefined: fal
someObject.secondProp === undefined: false
! someObject.secondProp: true
!! someObject.secondProp: false
typeof someObject.undefinedProp == "undefined": true
typeof someObject.undefinedProp === typeof undefined:
someObject.undefinedProp === undefined: true
! someObject.undefinedProp: true
!! someObject.undefinedProp: false
typeof notDefined == "undefined": true
typeof notDefined === typeof undefined: true
notDefined === undefined: Exception--ReferenceError: n
! notDefined: Exception--ReferenceError: notDefined is
!! notDefined: Exception--ReferenceError: notDefined i
```

Share Improve this answer

Follow

edited Jul 25, 2020 at 0:29

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Jan 13, 2013 at 17:43

drzaus
24.9k ● 17 ● 145 ● 205

21

If you do

```
if (myvar == undefined )
{
    alert('var does not exists or is not initialized')
}
```

it will fail when the variable `myvar` does not exists, because myvar is not defined, so the script is broken and the test has no effect.

Because the window object has a global scope (default object) outside a function, a declaration will be 'attached' to the window object.

For example:

```
var myvar = 'test';
```

The global variable *myvar* is the same as *window.myvar* or *window['myvar']*

To avoid errors to test when a global variable exists, you better use:

```
if(window.myvar == undefined )
{
    alert('var does not exists or is not initialized')
}
```

The question if a variable really exists doesn't matter, its value is incorrect. Otherwise, it is silly to initialize variables with undefined, and it is better use the value false to initialize. When you know that all variables that you declare are initialized with false, you can simply check its type or rely on `!window.myvar` to check if it has a proper/valid value. So even when the variable is not defined then `!window.myvar` is the same for `myvar = undefined` or `myvar = false` or `myvar = 0`.

When you expect a specific type, test the type of the variable. To speed up testing a condition you better do:

```
if( !window.myvar || typeof window.myvar != 'string' )
{
    alert('var does not exists or is not type of strin
}
```

When the first and simple condition is true, the interpreter skips the next tests.

It is always better to use the instance/object of the variable to check if it got a valid value. It is more stable and is a better way of programming.

(y)

Share Improve this answer Follow

edited May 20, 2016 at 18:11

answered Aug 12, 2011 at 14:40

Codebeat

6,580 ● 6 ● 61 ● 104

16

[ECMAScript 10](https://) introduced a new feature - **optional chaining** which you can use to use a property of an object only when an object is defined like this:

```
const userPhone = user?.contactDetails?.phone;
```

It will reference to the phone property only when user and contactDetails are defined.

Ref. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Optional_chaining

Share Improve this answer Follow

edited Jul 25, 2020 at 0:15
Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Sep 27, 2019 at 6:56
Przemek Struciński
5,198 ● 1 ● 31 ● 20

I used to use a lot the function get from lodash, very convenient for accessing this kind of objects, but the new optional chaining covers up most of the uses of _.get – Al Hill Jun 5, 2020 at 16:17

This is the answer to the question. – Ben Racicot Mar 23, 2022 at 15:42

15

In the article *Exploring the Abyss of Null and Undefined in JavaScript* I read that frameworks like Underscore.js use this function:

```
function isUndefined(obj){
    return obj === void 0;
}
```

Share Improve this answer

edited Nov 9, 2014 at 12:22

Follow

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Dec 19, 2013 at 10:44

Marthijn
3,392 ● 2 ● 33 ● 49

3 `void 0` is just a short way of writing `undefined` (since that's what *void* followed by any expression returns), it saves 3 charcters. It could also do `var a; return obj === a;`, but that's one more character. :-) – RobG Apr 1, 2014 at 1:16

2 `void` is a reserved word, whereas `undefined` is not i.e. while `undefined` is equal to `void 0` by default, you can assign a value to `undefined` e.g. `undefined = 1234`. – Brian M. Hunt Sep 14, 2015 at 13:08

`isUndefined(obj)`: 16 chars. `obj === void 0`: 14 chars. 'nough said. – Stijn de Witt Nov 1, 2015 at 2:41

15

Simply anything is not defined in JavaScript, is **undefined**, doesn't matter if it's a property inside an **Object/Array** or as just a simple variable...

JavaScript has `typeof` which make it very easy to detect an undefined variable.

Simply check if `typeof whatever === 'undefined'` and it will return a boolean.

That's how the famous function `isUndefined()` in AngularJs v.1x is written:

```
function isUndefined(value) {return typeof value === '
```

So as you see the function receive a value, if that value is defined, it will return `false`, otherwise for undefined values, return `true`.

So let's have a look what gonna be the results when we passing values, including object properties like below, this is the list of variables we have:

```
var stackoverflow = {};
stackoverflow.javascipt = 'javascript';
var today;
var self = this;
var num = 8;
var list = [1, 2, 3, 4, 5];
var y = null;
```

and we check them as below, you can see the results in front of them as a comment:

```
isUndefined(stackoverflow); //false
isUndefined(stackoverflow.javascipt); //false
isUndefined(today); //true
isUndefined(self); //false
isUndefined(num); //false
isUndefined(list); //false
isUndefined(y); //false
isUndefined(stackoverflow.java); //true
isUndefined(stackoverflow.php); //true
isUndefined(stackoverflow && stackoverflow.css); //tru
```

As you see we can check anything with using something like this in our code, as mentioned you can simply use `typeof` in your code, but if you are using it over and over, create a function like the angular sample which I share and keep reusing as following DRY code pattern.

Also one more thing, for checking property on an object in a real application which you not sure even the object exists or not, check if the object exists first.

If you check a property on an object and the object doesn't exist, will throw an error and stop the whole application running.

```
isUndefined(x.css);
VM808:2 Uncaught ReferenceError: x is not defined(…)
```

So simple you can wrap inside an if statement like below:

```
if(typeof x !== 'undefined') {
  //do something
}
```

Which also equal to isDefined in Angular 1.x...

```
function isDefined(value) {return typeof value !== 'un
```

Also other javascript frameworks like underscore has similar defining check, but I recommend you use `typeof` if you already not using any frameworks.

I also add this section from MDN which has got useful information about typeof, undefined and void(0).

> **Strict equality and undefined**
> You can use undefined and the strict equality and inequality operators to determine whether a variable has a value. In the following code, the variable x is not defined, and the if statement evaluates to true.

```
var x;
if (x === undefined) {
   // these statements execute
}
else {
   // these statements do not execute
}
```

> Note: The strict equality operator rather than the standard equality operator must be used here, because x == undefined also checks whether x is null, while strict equality doesn't. null is not equivalent to undefined. See comparison operators for details.

> **Typeof operator and undefined**
> Alternatively, typeof can be used:

```
var x;
if (typeof x === 'undefined') {
   // these statements execute
}
```

> One reason to use typeof is that it does not throw an error if the variable has not been declared.

```
// x has not been declared before
if (typeof x === 'undefined') { // evaluates to true w
   // these statements execute
}

if (x === undefined) { // throws a ReferenceError

}
```

> However, this kind of technique should be avoided. JavaScript is a statically scoped language, so knowing if a variable is declared can be read by seeing whether it is declared in an enclosing context. The only exception is the global scope, but the global scope is bound to the global object, so checking the existence of a variable in the global context can be done by checking the existence of a property on the global object (using the in operator, for instance).

> **Void operator and undefined**
>
> The void operator is a third alternative.

```
var x;
if (x === void 0) {
   // these statements execute
```

```
}

// y has not been declared before
if (y === void 0) {
  // throws a ReferenceError (in contrast to `typeof`
}
```

more > here

Share Improve this answer Follow

edited Oct 2, 2019 at 2:51

answered May 24, 2017 at 14:15

Alireza
105k ● 27 ● 277 ● 173

13

```
"propertyName" in obj //-> true | false
```

Share Improve this answer Follow

answered May 5, 2014 at 0:13

sam
40.7k ● 2 ● 42 ● 37

13

The solution is incorrect. In JavaScript,

```
null == undefined
```

will return true, because they both are "casted" to a boolean and are false. The correct way would be to check

```
if (something === undefined)
```

which is the identity operator...

Share Improve this answer Follow

edited Nov 9, 2014 at 12:07
Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Aug 26, 2008 at 12:38
Ricky
5,377 ● 2 ● 33 ● 30

3 To be clear, `===` is type equality + (primitive equality | object identity), where primitives include strings. I think most people consider `'abab'.slice(0,2) === 'abab'.slice(2)` unintuitive if one considers `===` as the identity operator. – clacke Jul 30, 2010 at 8:49

1 Wrong. This throws an error if the variable has not been created. Should not be voted up. Use typeof instead. – Simon E. Jun 15, 2011 at 0:22

What solution? Can you link directly to it? – Peter Mortensen Jul 24, 2020 at 23:34

13

'*if (window.x) { }*' is error safe

Most likely you want `if (window.x)`. This check is safe even if x hasn't been declared (`var x;`) - browser doesn't throw an error.

Example: I want to know if my browser supports History API

```
if (window.history) {
    history.call_some_function();
}
```

How this works:

window is an object which holds all global variables as its members, and it is legal to try to access a non-existing member. If **x** hasn't been declared or hasn't been set then `window.x` returns **undefined**. **undefined** leads to **false** when **if()** evaluates it.

Share Improve this answer Follow

edited Nov 23, 2014 at 13:19

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Feb 10, 2014 at 16:26

DenisS
1,677 ● 19 ● 15

But what if you run in Node? `typeof history != 'undefined'` actually works in both systems. – Stijn de Witt Nov 1, 2015 at 2:28

Reading through this, I'm amazed I didn't see this. I have found multiple algorithms that would work for this.

13

Never Defined

If the value of an object was never defined, this will prevent from returning `true` if it is defined as `null` or `undefined`. This is helpful if you want true to be returned for values set as `undefined`

```
if(obj.prop === void 0) console.log("The value has nev
```

Defined as undefined Or never Defined

If you want it to result as `true` for values defined with the value of `undefined`, or never defined, you can simply use `=== undefined`

```
if(obj.prop === undefined) console.log("The value is d
never defined");
```

Defined as a falsy value, undefined,null, or never defined.

Commonly, people have asked me for an algorithm to figure out if a value is either falsy, `undefined`, or `null`. The following works.

```
if(obj.prop == false || obj.prop === null || obj.prop
    console.log("The value is falsy, null, or undefine
}
```

Share Improve this answer

Follow

answered Feb 15, 2015 at 3:10

Travis
1,284 ● 1 ● 16 ● 35

4 I think you can replace the last example with `if (!obj.prop)` – Stijn de Witt Nov 1, 2015 at 2:24

@StijndeWitt, you can, I was pretty inexperienced when I wrote this, and my English seems to have been equally bad, nevertheless, there isn't anything *incorrect* in the answer – Travis Apr 7, 2017 at 14:31

3 `var obj = {foo: undefined}; obj.foo === void 0` -> `true`. How is that "never defined as `undefined`"? This is wrong. – Patrick Roberts Jun 15, 2017 at 18:40

@PatrickRoberts You're right. When I wrote this answer in February 2015 (before ES6) the first option I outlined did indeed work, but it is now outdated. – Travis May 14, 2020 at 22:31

10

Compare with `void 0`, for terseness.

```
if (foo !== void 0)
```

It's not as verbose as `if (typeof foo !== 'undefined')`

Share Improve this answer Follow

answered Jan 2, 2014 at 12:59

bevacqua
48.4k ● 56 ● 174 ● 286

3 But it will throw a ReferenceError if `foo` is undeclared.
– daniel1426 Mar 7, 2014 at 22:46

1 @daniel1426: So if there's an error in your code, you want to hide it instead of fixing it? Not a great approach, IMO.
– user8897421 Dec 18, 2017 at 13:47

This is not used to hide errors. It's the common way to detect the properties of the environment to define polyfills. For instance: if(typeof Promise === 'undefined'){ /* define Promise */ } – gaperton Oct 4, 2018 at 3:05

10

You can get an array all undefined with path using the following code.

```
function getAllUndefined(object) {

       function convertPath(arr, key) {
           var path = "";
           for (var i = 1; i < arr.length; i++) {

               path += arr[i] + "->";
           }
           path += key;
           return path;
       }


       var stack = [];
       var saveUndefined= [];
       function getUndefiend(obj, key) {

           var t = typeof obj;
```

```
            switch (t) {
                case "object":
                    if (t === null) {
                        return false;
                    }
                    break;
                case "string":
                case "number":
                case "boolean":
                case "null":
                    return false;
                default:
                    return true;
            }
            stack.push(key);
            for (k in obj) {
                if (obj.hasOwnProperty(k)) {
                    v = getUndefiend(obj[k], k);
                    if (v) {
                        saveUndefined.push(convertPath
                    }
                }
            }
            stack.pop();

        }

        getUndefiend({
            "": object
        }, "");
        return saveUndefined;
    }
```

jsFiddle link

Share Improve this answer

Follow

edited Nov 9, 2014 at 12:16

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Oct 17, 2011 at 11:22

Anoop
23.2k ● 10 ● 64 ● 79

While it won't affect the validity of your code, you've got a typo: `getUndefiend` should be `getUndefined`.
– icktoofay May 14, 2013 at 3:02

9

There is a nice and elegant way to assign a defined property to a new variable if it is defined or assign a default value to it as a fallback if it's undefined.

```
var a = obj.prop || defaultValue;
```

It's suitable if you have a function, which receives an additional configuration property:

```
var yourFunction = function(config){

   this.config = config || {};
   this.yourConfigValue = config.yourConfigValue || 1;
   console.log(this.yourConfigValue);
}
```

Now executing

```
yourFunction({yourConfigValue:2});
//=> 2

yourFunction();
//=> 1

yourFunction({otherProperty:5});
//=> 1
```

Share Improve this answer Follow

edited Jul 25, 2020 at 0:23

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Mar 3, 2016 at 10:05

Marian Klühspies
17.5k ● 19 ● 108 ● 159

Using || (OR) will use type coertion for "undefined" and "null" values to false, and every other values to "true". ------- If property checked (in example "obj.prop" and "config") is undefined or null it will be assigned 'default value'. In any other the value will not be changed. -------- This soultion doesn't check if the object HAS property defined. -- To check if object has a property you can use: `let objAndPropCheck = (obj || { }).prop || 0;` – konieckropka Mar 4, 2022 at 0:44

8

Here is my situation:

I am using the result of a REST call. The result should be parsed from JSON to a JavaScript object.

There is one error I need to defend. If the arguments to the REST call were incorrect as far as the user specifying the arguments wrong, the REST call comes back basically empty.

While using this post to help me defend against this, I tried this:

```
if( typeof restResult.data[0] === "undefined" ) { thro
```

For my situation, if restResult.data[0] === "object", then I can safely start inspecting the rest of the members. If undefined then throw the error as above.

What I am saying is that for my situation, all the previous suggestions in this post did not work. I'm not saying I'm right and everyone is wrong. I am not a JavaScript master at all, but hopefully this will help someone.

Share Improve this answer

Follow

edited Jul 25, 2020 at 0:26

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Aug 15, 2013 at 13:56

wayneseymour

491 ● 5 ● 6

Your `typeof` guard doesn't actually guard against anything that a direct comparison couldn't handle. If `restResult` is undefined or undeclared, it'll still throw. – user8897421 Dec 18, 2017 at 13:50

In your case you could more simply check if the array is empty: `if(!restResult.data.length) { throw "Some error"; }` – Headbank Feb 28, 2019 at 15:36

7

All the answers are incomplete. This is the right way of knowing that there is a property 'defined as undefined':

```
var hasUndefinedProperty = function hasUndefinedProper
  return ((prop in obj) && (typeof obj[prop] == 'undef
};
```

Example:

```
var a = { b : 1, e : null };
a.c = a.d;

hasUndefinedProperty(a, 'b'); // false: b is defined a
hasUndefinedProperty(a, 'c'); // true: c is defined as
hasUndefinedProperty(a, 'd'); // false: d is undefined
hasUndefinedProperty(a, 'e'); // false: e is defined a

// And now...
delete a.c ;
hasUndefinedProperty(a, 'c'); // false: c is undefined
```

Too bad that this been the right answer and is buried in wrong answers >_<

So, for anyone who pass by, I will give you undefined's for free!!

```
var undefined ; undefined ; // undefined
({}).a ;                    // undefined
[].a ;                      // undefined
''.a ;                      // undefined
(function(){}()) ;          // undefined
void(0) ;                   // undefined
eval() ;                    // undefined
1..a ;                      // undefined
/a/.a ;                     // undefined
(true).a ;                  // undefined
```

Share Improve this answer edited Jul 25, 2020 at 0:20

Follow

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Jun 18, 2014 at 5:21

Juan Garcia
706 ● 7 ● 23

7

There is a very easy and simple way.

You can use **optional chaining**:

```
x = {prop:{name:"sajad"}}

console.log(x.prop?.name) // Output is: "sajad"
console.log(x.prop?.lastName) // Output is: undefined
```

or

```
if(x.prop?.lastName) // The result of this 'if' statem
throwing an error
```

You can use optional chaining even for functions or arrays.

As of mid-2020 this is not universally implemented. Check the documentation at https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Optional_chaining

Share Improve this answer

edited Aug 22, 2020 at 5:04

Follow

user1527225
1,089 ● 10 ● 8

answered May 5, 2020 at 6:41

S.Saderi
5,495 ● 4 ● 23 ● 24

6

Going through the comments, for those who want to check both is it undefined or its value is null:

```
//Just in JavaScript
var s; // Undefined
if (typeof s == "undefined" || s === null){
    alert('either it is undefined or value is null')
}
```

If you are using jQuery Library then `jQuery.isEmptyObject()` will suffice for both cases,

```
var s; // Undefined
jQuery.isEmptyObject(s); // Will return true;

s = null; // Defined as null
jQuery.isEmptyObject(s); // Will return true;

//Usage
if (jQuery.isEmptyObject(s)) {
    alert('Either variable:s is undefined or its value
} else {
     alert('variable:s has value ' + s);
}

s = 'something'; // Defined with some value
jQuery.isEmptyObject(s); // Will return false;
```

Share Improve this answer Follow

edited Nov 9, 2014 at 12:26
Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Jul 8, 2014 at 8:19
Angelin Nadar
9,300 ● 10 ● 45 ● 54

jQuery will also take care of any cross-browser compatibility issues with the different JavaScript APIs. – Henry Heleine Dec 9, 2014 at 22:12

6

If you are using Angular:

```
angular.isUndefined(obj)
angular.isUndefined(obj.prop)
```

Underscore.js:

```
_.isUndefined(obj)
_.isUndefined(obj.prop)
```

Share Improve this answer Follow

answered Dec 14, 2014 at 22:35
Vitalii Fedorenko
114k ● 29 ● 150 ● 111

2 How do I add `1` to variable `x`? Do I need Underscore or jQuery? (amazing that people will use libraries for even the most elementary operations such as a `typeof` check) – Stijn de Witt Nov 1, 2015 at 2:43

6

I provide three ways here for those who expect weird answers:

```
function isUndefined1(val) {
    try {
        val.a;
    } catch (e) {
        return /undefined/.test(e.message);
    }
    return false;
}

function isUndefined2(val) {
    return !val && val+'' === 'undefined';
}

function isUndefined3(val) {
    const defaultVal = {};
    return ((input = defaultVal) => input === defaul
}

function test(func){
    console.group(`test start :`+func.name);
    console.log(func(undefined));
    console.log(func(null));
    console.log(func(1));
    console.log(func("1"));
    console.log(func(0));
    console.log(func({}));
    console.log(func(function () { }));
    console.groupEnd();
}
test(isUndefined1);
test(isUndefined2);
test(isUndefined3);
```

Run code snippet   Expand snippet

isUndefined1:

Try to get a property of the input value, and check the error message if it exists. If the input value is undefined, the error message would be *Uncaught TypeError: Cannot read property 'b' of undefined*.

isUndefined2:

Convert the input value to a string to compare with `"undefined"` and ensure it's a negative value.

isUndefined3:

In JavaScript, an optional parameter works when the input value is exactly `undefined`.

Share Improve this answer Follow

edited Jul 25, 2020 at 0:11
Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Oct 28, 2017 at 9:24
blackmiaool
5,314 ● 2 ● 23 ● 39

5

I use `if (this.variable)` to test if it is defined. A simple `if (variable)`, recommended in a previous answer, fails for me.

It turns out that it works only when a variable is a field of some object, `obj.someField` to check if it is defined in the dictionary. But we can use `this` or `window` as the dictionary object since any variable is a field in the current window, as I understand it. Therefore here is a test:

```
if (this.abc)
    alert("defined");
else
    alert("undefined");

abc = "abc";
if (this.abc)
    alert("defined");
else
    alert("undefined");
```

Run code snippet   Expand snippet

It first detects that variable `abc` is undefined and it is defined after initialization.

Share Improve this answer

Follow

edited Jul 25, 2020 at 0:17

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Mar 6, 2015 at 15:58

Val

11.1k ● 9 ● 53 ● 65

4

```
function isUnset(inp) {
  return (typeof inp === 'undefined')
}
```

Returns false if variable is set, and true if is undefined.

Then use:

```
if (isUnset(var)) {
  // initialize variable here
}
```

Share Improve this answer Follow

edited Jun 15, 2011 at 1:43
Simon E.
58.4k ● 18 ● 144 ● 137

answered Jul 12, 2010 at 20:54
Rixius
2,293 ● 3 ● 24 ● 33

5 No. Don't do this. It only takes a very simple test to prove that you cannot meaningfully wrap a `typeof` test in a function. Amazing that 4 people upvoted this. -1. – Stijn de Witt Nov 1, 2015 at 2:39

4

I would like to show you something I'm using in order to protect the `undefined` variable:

```
Object.defineProperty(window, 'undefined', {});
```

This forbids anyone to change the `window.undefined` value therefore destroying the code based on that variable. If using `"use strict"`, anything trying to change its value will end in error, otherwise it would be silently ignored.

Share Improve this answer

Follow

edited Nov 9, 2014 at 12:27

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Oct 9, 2014 at 8:09

Seti
2,318 ● 17 ● 27

1 2 Next

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.