Why is null an object and what's the difference between null and undefined?

Ask Question

Why is `null` considered an `object` in JavaScript?

Is checking

```
if ( object == null )
      Do something
```

the same as

```
if ( !object )
      Do something
```

?

And also:

What is the difference between `null` and `undefined` ?

javascript object null undefined typeof

edited Jul 20 '17 at 10:10

Rahul

3,768 2 15 43

asked Apr 29 '09 at 5:47

rahul

144k 39 198 243

6 It would be great if the last question is added in the title. The answers here are awesome, but the title may not point that you will find this answer here – marcos.borunda Nov 27 '12 at 20:12

6 Null is not an object in JavaScript! `typeof null === 'object'` but that's a bug! Here's a link Watch all the videos

and enjoy :) – CoR Jun 30 '15 at 14:42

1 I like to use C/C++ as the axiomatic truth for what an undefined and NULL valued variable is, since it's very straightforward. Then see how those definitions compare to other languages' specifications. – sαmosΛris Feb 3 '16 at 19:24

20 Answers

```
(name is undefined)
```

You: What is `name` ? (*)
JavaScript: `name` ? What's a `name` ? I don't know what you're talking about. You haven't ever mentioned any `name` before. Are you seeing some other scripting language on the (client-)side?

```
name = null;
```

You: What is `name` ?
JavaScript: I don't know.

In short; `undefined` is where no notion of the thing exists; it has no type, and it's never been referenced before in that scope; `null` is where the thing is known to exist, but it's not known what the value is.

One thing to remember is that `null` is not, conceptually, the same as `false` or `""` or such, even if they equate after type casting, i.e.

```
name = false;
```

You: What is `name` ?
JavaScript: Boolean false.

```
name = '';
```

You: What is `name` ?
JavaScript: Empty string

*: `name` in this context is meant as a variable which has never been defined. It could be be any undefined variable. However, name is a property of just about any HTML form element. It goes way, way back and was instituted well before id. It is useful because ids

must be unique but names to not have to be.

edited Oct 3 '13 at 18:27

Ken Kin
2,189 3 28 66

answered Apr 29 '09 at 13:31

Rob
42.7k 2 63 86

13 But in JavaScript, the empty string '' is still a boolean false. – Andreas Grech Feb 21 '10 at 21:11

103 The empty string is not boolean false, but in the context of a conditional it is interpreted as a false(y) value (coercion). – Bryan Matthews Dec 21 '10 at 19:23

21 For the second case, where name = null, instead of 'I don't know', JavaScript could answer: The null object. Other than that I like the style of the answer. – Jean Vincent Jun 18 '11 at 10:05

13 Rather than “I don't know”, I'd say the answer for `null` is “nothing”. Null is precisely defined as no value. Void, nil, nada. Nothing. – mikl Aug 13 '11 at 20:09

44 Love that style of answering. In general, *"You haven't ever mentioned any* `name` *before"* is true. However, declaring a variable without assigning a value to it (`var somevar;`), will surprisingly enough still result in `undefined` . – MC Emperor Jan 10 '12 at 15:40

The difference can be summarized into this snippet:

```
alert(typeof(null));      // object
alert(typeof(undefined)); // undefined

alert(null !== undefined) //true
alert(null == undefined)  //true
```

Checking

`object == null` is different to check `if ( !object )` .

The latter is equal to `! Boolean(object)` , because the unary `!` operator automatically cast the right operand into a Boolean.

Since `Boolean(null)` equals false then `!false === true` .

So if your object is **not null**, *but* **false** or **0** or **""**, the check will pass because:

```
alert(Boolean(null)) //false
alert(Boolean(0))    //false
alert(Boolean(""))   //false
```

edited Nov 23 '14 at 13:59

Peter Mortensen

12.7k 19 82 110

answered Apr 29 '09 at 6:44

kentaromiura

5,371 2 16 15

8 +1, for the explanation of !Boolean(object) – c4il Sep 1 '10 at 17:14

@kentaromiura For us Javascript noobs...maybe only me...what is this Boolean(value) syntax? Casting to a Boolean? – xr280xr Jun 12 '12 at 20:37

@xr280xr Yes, it's casting. Try `String(null)` to see another example of casting. You can even do silly things like `Number(null + 2)` ... but you shouldn't :-). Excellent answer from kentaromiura. – squidbe Jan 4 '13 at 7:06

Remember `typeof` is an operator. You wouldn't wrap the operand in parenthesis for the same reason you wouldn't write `var sum = 1 +(1);` . – alex Jan 27 '16 at 10:51

`null` is not an object, it is a *primitive value*. For example, you cannot add properties to it. Sometimes people wrongly assume that it is an object, because `typeof null` returns `"object"` . But that is actually a bug (that might even be fixed in ECMAScript 6).

The difference between `null` and `undefined` is as follows:

- `undefined` : used by JavaScript and means “no value”. Uninitialized variables, missing parameters and unknown variables have that value.

```
> var noValueYet;
> console.log(noValueYet);
undefined

> function foo(x) { console.log(x) }
> foo()
undefined

> var obj = {};
```

```
> console.log(obj.unknownProperty)
undefined
```

Accessing unknown variables, however, produces an exception:

```
> unknownVariable
ReferenceError: unknownVariable is not defined
```

- `null` : used by programmers to indicate “no value”, e.g. as a parameter to a function.

Examining a variable:

```
console.log(typeof unknownVariable === "undefined"); // true

var foo;
console.log(typeof foo === "undefined"); // true
console.log(foo === undefined); // true

var bar = null;
console.log(bar === null); // true
```

As a general rule, you should always use === and never == in JavaScript (== performs all kinds of conversions that can produce unexpected results). The check `x == null` is an edge case, because it works for both `null` and `undefined` :

```
> null == null
true
> undefined == null
true
```

A common way of checking whether a variable has a value is to convert it to boolean and see whether it is `true` . That conversion is performed by the `if` statement and the boolean operator ! (“not”).

```
function foo(param) {
    if (param) {
        // ...
    }
}
function foo(param) {
    if (! param) param = "abc";
}
function foo(param) {
    // || returns first operand that can't be converted to false
    param = param || "abc";
}
```

Drawback of this approach: All of the following values evaluate to `false`, so you have to be careful (e.g., the above checks can't distinguish between `undefined` and `0`).

- `undefined`, `null`
- Booleans: `false`
- Numbers: `+0`, `-0`, `NaN`
- Strings: `""`

You can test the conversion to boolean by using `Boolean` as a function (normally it is a constructor, to be used with `new`):

```
> Boolean(null)
false
> Boolean("")
false
> Boolean(3-3)
false
> Boolean({})
true
> Boolean([])
true
```

edited Nov 7 '11 at 2:56

answered Nov 1 '11 at 15:06

Axel Rauschmayer
13.5k 3 16 13

1 Why reference `+0` and `-0` seperately if `+0 === -0` ? – Raynos Nov 7 '11 at 3:09

5 Probably because you can still distinguish `+0` and `-0`: `1/+0 !== 1/-0`. – neo Aug 1 '13 at 8:21

1 This answer links to a post that links back to this answer as source... probably meant to link to this instead 2ality.com/2013/10/typeof-null.html – Ricardo Tomasi Nov 27 '13 at 17:47

Oh right, like in C, where uninitialized variable declarations are considered undefined (old versions could actually contain a previous programs data). – sɑmosʌris Feb 3 '16 at 19:19

4 "But that is actually a bug (that might even be fixed in ECMAScript 6)" – source? – Gajus Jan 31 '17 at 11:47

> What is the difference between null and undefined??

A property when it has no definition, is undefined. null is an object. Its type is object. null is a special value meaning "no value. undefined is not an object, it's type is undefined.

You can declare a variable, set it to null, and the behavior is identical except that you'll see "null" printed out versus "undefined". You can even compare a variable that is undefined to null or vice versa, and the condition will be true:

```
undefined == null
null == undefined
```

Refer to JavaScript Difference between null and undefined for more detail.

and with your new **edit yes**

```
if (object == null)  does mean the same  if(!object)
```

when testing if object is false, they both only meet the condition when testing if **false**, but not when true

Check here: Javascript gotcha

edited Jun 23 '17 at 8:14

CodingBarfield

2,634 1 19 48

answered Apr 29 '09 at 5:52

TStamper

25.5k 9 55 70

3 You should use ===, then undefined !== null :D – olliej Apr 29 '09 at 6:42

1 pay attention to the last part, is incorrect see my answer ;) – kentaromiura Apr 29 '09 at 7:00

5 !object is not the same as "object == null" ... In fact, they're quite different. !object will return true if is object is 0, an empty string, Boolean false, undefined or null. – James Apr 29 '09 at 8:38

3 Is null really an object? The 1st link you have provided checks null by typeof, but typeof(null) evaluates to 'object' because of an language design error. – c4il Sep 1 '10 at 17:21

2 `null` is not an object. That `typeof null == 'object';` returns true is because of a bug that can't be fixed in JavaScript (presently, but may change in the future). – Mohamad Dec 30 '14 at 17:51

First part of the question:

> Why is null considered an object in JavaScript?

> It is a JavaScript design error they can't fix now. It should have been type null, not type object, or not have it at all. It necessitates an extra check (sometimes forgotten) when detecting real objects and is source of bugs.
>
> Second part of the question:
>
> > Is checking
> >
> > ```
> > if (object == null)
> > Do something
> > ```
> >
> > the same as
> >
> > ```
> > if (!object)
> > Do something
> > ```
>
> The two checks are always both false except for:
>
> - object is undefined or null: both true.
> - object is primitive, and 0, `""` , or false: first check false, second true.
>
> If the object is not a primitive, but a real Object, like `new Number(0)` , `new String("")` , or `new Boolean(false)` , then both checks are false.
>
> So if 'object' is interpreted to mean a real Object then both checks are always the same. If primitives are allowed then the checks are different for 0, `""` , and false.
>
> In cases like `object==null` , the unobvious results could be a source of bugs. Use of `==` is not recommended ever, use `===` instead.
>
> Third part of the question:
>
> > And also:
> >
> > What is the difference between null and undefined?
>
> In JavaScript, one difference is that null is of type object and undefined is of type undefined.

In JavaScript, `null==undefined` is true, and considered equal if type is ignored. Why they decided that, but 0, `""` and false aren't equal, I don't know. It seems to be an arbitrary opinion.

In JavaScript, `null===undefined` is not true since the type must be the same in `===` .

In reality, null and undefined are identical, since they both represent non-existence. So do 0, and `""` for that matter too, and maybe the empty containers `[]` and `{}` . So many types of the same nothing are a recipe for bugs. One type or none at all is better. I would try to use as few as possible.

'false', 'true', and '!' are another bag of worms that could be simplified, for example, `if(!x)` and `if(x)` alone are sufficient, you don't need true and false.

A declared `var x` is type undefined if no value is given, but it should be the same as if x was never declared at all. Another bug source is an empty nothing container. So it is best to declare and define it together, like `var x=1` .

People are going round and round in circles trying to figure out all these various types of nothing, but it's all just the same thing in complicated different clothes. The reality is

```
undefined===undeclared===null===0===""===[]==={}===nothing
```

And maybe all should throw exceptions.

edited Nov 23 '14 at 16:00

Peter Mortensen
12.7k 19 82 110

answered Jul 14 '11 at 22:15

Anon
197 2 2

3 +1 for that statement at the end. – dmackerman Sep 9 '11 at 20:44

+1 for answering ALL the questions – Pakman Mar 14 '12 at 16:33

6 Great answer, but it goes a little too far with []: "*In reality, null and undefined are identical... and maybe the empty containers [] and {}*." You could probably talk me into thinking {} should be some flavor of null, but my gut impression is that it shouldn't. But less controversially, **an empty array `[]` understandably has a `.push()` function**, so there's no good argument for [] being null. $0.02. – ruffin Oct 11 '12 at 20:56

```
var x = null;
```

x is defined as null

y is not defined; // because I did not define it

```
if (!x)
```

null is evaluated as false

answered Apr 29 '09 at 5:56

Chad Grant

31.4k 7 48 69

One way to make sense of null and undefined is to understand where each occurs.

Expect a null return value in the following situations:

- Methods that query the DOM

```
console.log(window.document.getElementById("nonExistentElement"));
//Prints: null
```

- JSON responses received from an Ajax request

```
{
  name: "Bob",
  address: null
}
```

- RegEx.exec.
- New functionality that is in a state of flux. The following returns null:

```
var proto = Object.getPrototypeOf(Object.getPrototypeOf({}));

// But this returns undefined:

Object.getOwnPropertyDescriptor({}, "a");
```

All other cases of non-existence are denoted by undefined (as noted by @Axel). Each of the following prints "undefined":

```
var uninitalised;
console.log(uninitalised);

var obj = {};
console.log(obj.nonExistent);

function missingParam(missing){
    console.log(missing);
}

missingParam();

var arr = [];
console.log(arr.pop());
```

Of course if you decide to write var unitialised = null; or return null from a method yourself then you have null occurring in other situations. But that should be pretty obvious.

A third case is when you want to access a variable but you don't even know if it has been declared. For that case use typeof to avoid a reference error:

```
if(typeof unknown !== "undefined"){
    //use unknown
}
```

In summary check for null when you are manipulating the DOM, dealing with Ajax, or using certain ECMAScript 5 features. For all other cases it is safe to check for undefined with strict equality:

```
if(value === undefined){
  // stuff
}
```

edited Jan 6 '14 at 20:27

answered May 7 '12 at 10:06

Noel Abrahams

5,930 2 25 30

Comparison of many different null checks in JavaScript:

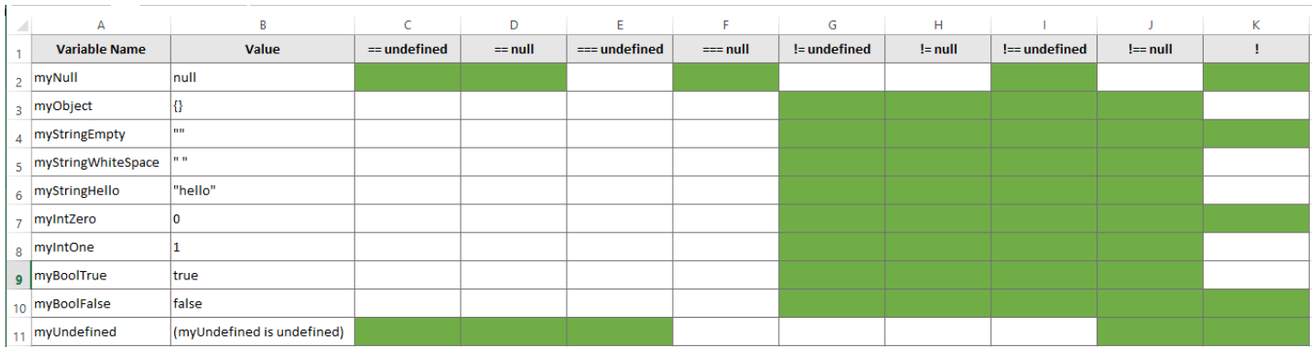
http://jsfiddle.net/aaronhoffman/DdRHB/5/

```
// Variables to test
var myNull = null;
```

```
var myObject = {};
var myStringEmpty = "";
var myStringWhiteSpace = " ";
var myStringHello = "hello";
var myIntZero = 0;
var myIntOne = 1;
var myBoolTrue = true;
var myBoolFalse = false;
var myUndefined;

...trim...
```

http://aaron-hoffman.blogspot.com/2013/04/javascript-null-checking-undefined-and.html



	A	B	C	D	E	F	G	H	I	J	K
1	Variable Name	Value	== undefined	== null	=== undefined	=== null	!= undefined	!= null	!== undefined	!== null	!
2	myNull	null									
3	myObject	{}									
4	myStringEmpty	""									
5	myStringWhiteSpace	" "									
6	myStringHello	"hello"									
7	myIntZero	0									
8	myIntOne	1									
9	myBoolTrue	true									
10	myBoolFalse	false									
11	myUndefined	(myUndefined is undefined)									

edited Nov 23 '14 at 16:03

Peter Mortensen
12.7k 19 82 110

answered Apr 29 '13 at 15:23

Aaron Hoffman
4,308 5 38 49

does "var myUndefined;" define the variable (as a known variable with unknown type)? – Qi Fan Apr 28 '15 at 8:22

@QiFan yes, that is correct. – Aaron Hoffman Apr 30 '15 at 2:42

null and undefined are both false for value equality (null==undefined): they both collapse to boolean false. They are not the same object (null!==undefined).

undefined is a property of the global object ("window" in browsers), but is a primitive type and not an object itself. It's the default value for uninitialized variables and functions ending without a return statement.

null is an instance of Object. null is used for DOM methods that return collection objects to indicate an empty result, which provides a false value without indicating an error.

answered Apr 29 '09 at 5:58

Anonymous

26.8k 1 19 19

Some precisions:

null and undefined *are* two different values. One is representing the absence of a value for a name and the other is representing the absence of a name.

What happens in an `if` goes as follows for `if( o )`:

The expression in the parentheses o is evaluated, and then the `if` kicks in type-coercing the value of the expression in the parentheses - in our case `o`.

Falsy (that will get coerced to false) values in JavaScript are: *'', null, undefined, 0, and false*.

edited Nov 23 '14 at 14:08

Peter Mortensen

12.7k 19 82 110

answered Apr 30 '09 at 7:55

Laurent Villeneuve

106 2

To add to the answer of *What is the difference between* `undefined` *and* `null`, from JavaScript Definitive Guide on this page:

> You might consider `undefined` to represent system-level, unexpected, or error-like absense of value and `null` to represent program-level, normal, or expected absence of value. If you need to assign one of these values to a variable or property or pass one of these values to a function, `null` is almost always the right choice.

edited Nov 23 '14 at 16:02

Peter Mortensen

12.7k 19 82 110

answered Oct 7 '12 at 14:47

太極者無極而生

66.3k 91 361 615

`null` is an object. Its type is null. `undefined` is not an object; its type is undefined.

edited Nov 23 '14 at 13:56

Peter Mortensen
12.7k 19 82 110

answered Apr 29 '09 at 5:51

Vikas
12.4k 31 100 158

11 wrong - both `null` and `undefined` are primitive values - `typeof null === 'object'` is a language bug, because `Object(null) !== null` – Christoph Apr 29 '09 at 18:16

No, is not. Object() cast work in that way, see ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf -- 15.2.2.1 new Object ([value]) ... 8. (The argument value was not supplied or its type was Null or Undefined.) Create a new native ECMAScript object. The [[Prototype]] property of the newly constructed object is set to the Object prototype object. The [[Class]] property of the newly constructed object is set to "Object". The newly constructed object has no [[Value]] property. Return the newly created native object. – kentaromiura Apr 30 '09 at 6:39

3 @Christoph: I mean, you're right. null *should* be a type, what I means is that you cannot check it in that way because: alert(new Object() !== new Object()); /* true, new istance is not the same istance */ *alert(Object(null).constructor == {}.constructor);* /* true as in spec */ *alert(Object(null).prototype == {}.prototype);* /* true as in spec */ *alert(null instanceof Object);* /* obviously false, null means not instantiated */ But basically is a spec bug XD see: uselesspickles.com/blog/2006/06/12/… and javascript.crockford.com/remedial.html – kentaromiura Apr 30 '09 at 7:10

Why don't they just make everything an object... – Rolf Apr 21 '11 at 15:24

The following function shows why and is capable for working out the difference:

```
function test() {
        var myObj = {};
        console.log(myObj.myProperty);
        myObj.myProperty = null;
        console.log(myObj.myProperty);
}
```

If you call

```
test();
```

You're getting

> undefined

> null

The first `console.log(...)` tries to get `myProperty` from `myObj` while it is not yet defined - so it gets back "undefined". After assigning null to it, the second `console.log(...)` returns obviously "null" because `myProperty` exists, but it has the value `null` assigned to it.

In order to be able to query this difference, JavaScript has `null` and `undefined` : While `null` is - just like in other languages an object, `undefined` cannot be an object because there is no instance (even not a `null` instance) available.

answered May 8 '14 at 13:00
user3615347

For example `window.someWeirdProperty` is undefined, so

`"window.someWeirdProperty === null"` evaluates to false while

`"window.someWeirdProperty === undefined"` evaluates to true.

Moreover checkif `if (!o)` is not the same as checking `if (o == null)` for `o` being `false` .

edited Jul 24 '15 at 10:14

Bruce
1,278 3 12 18

answered Apr 29 '09 at 5:55

Piotr Findeisen
3,885 15 29

Can you elaborate the difference between the two conditions – rahul Apr 29 '09 at 6:05

read my answer, he means that if o is equals to 0, false or "" the Boolean value is false: var undef, various = [0,"",null,false,undef]; for(var obj in various){ alert(!obj); //4 times false in IE,5 in FF ;) } – kentaromiura Apr 29 '09 at 9:55

The other fun thing about null, compared to undefined, is that it can be incremented.

```
x = undefined
x++
y = null
```

```
y++
console.log(x) // NaN
console.log(y) // 0
```

Run code snippet Expand snippet

This is useful for setting default numerical values for counters. How many times have you set a variable to -1 in its declaration?

answered Feb 10 '16 at 15:45

Tim Scollick
894 1 11 17

In **Javascript** `null` is not an `object` type it is a `primitave` type.

What is the difference? Undefined refers to a pointer that has not been set. **Null** refers to the null pointer for example something has manually set a variable to be of type `null`

answered Aug 28 '16 at 4:40

AnonDCX
816 1 5 19

From "The Principles of Object-Oriented Javascript" by Nicholas C. Zakas

> But why an object when the type is null? (In fact, this has been acknowledged as an error by TC39, the committee that designs and maintains JavaScript. You could reason that null is an empty object pointer, making "object" a logical return value, but that's still confusing.)

Zakas, Nicholas C. (2014-02-07). The Principles of Object-Oriented JavaScript (Kindle Locations 226-227). No Starch Press. Kindle Edition.

That said:

```
var game = null; //typeof(game) is "object"

game.score = 100;//null is not an object, what the heck!?
```

```
game instanceof Object; //false, so it's not an instance but it's type is object
//let's make this primitive variable an object;
game = {};
typeof(game);//it is an object
game instanceof Object; //true, yay!!!
game.score = 100;
```

Undefined case:

```
var score; //at this point 'score' is undefined
typeof(score); //'undefined'
var score.player = "felix"; //'undefined' is not an object
score instanceof Object; //false, oh I already knew that.
```

answered Aug 28 '14 at 19:49

Felix
1,752 6 30 51

The best way to think about 'null' is to recall how the similar concept is used in databases, where it indicates that a field contains "no value at all."

- Yes, the item's value is known; it *is* 'defined.' It *has* been initialized.
- The item's value is: *"there is no value."*

This is a very useful technique for writing programs that are more-easily debugged. An 'undefined' variable might be the result of a bug ... *(how would you know?)* ... but if the variable contains the value 'null,' you know that "someone, somewhere in this program, *set it* to 'null.'" Therefore, I suggest that, when you need to get rid of the value of a variable, don't "delete" ... set it to 'null.' The old value will be orphaned and soon will be garbage-collected; the new value is, "there is no value (now)." In both cases, the variable's state is certain: "it obviously, deliberately, got that way."

answered Jan 6 '16 at 19:47

Mike Robinson
3,623 1 9 21

---

1. Undefined means a variable has been declared but it has not been assigned any value while Null can be assigned to a variable representing "no value".(Null is an assignment operator)

2.Undefined is a type itself while Null is an object.

3.Javascript can itself initialize any unassigned variable to undefined but it can never set value of a variable to null. This has to be done programatically.

answered Mar 11 '16 at 12:27

the_unknown_spirit
920 10 32

Look at this:

```
   <script>
function f(a){
  alert(typeof(a));
  if (a==null) alert('null');
  a?alert(true):alert(false);
}
</script>
                                          //return:
<button onclick="f()">nothing</button>    //undefined    null    false
<button onclick="f(null)">null</button>   //object       null    false
<button onclick="f('')">empty</button>    //string               false
<button onclick="f(0)">zero</button>      //number               false
<button onclick="f(1)">int</button>       //number               true
<button onclick="f('x')">str</button>     //string               true
```

edited Jul 15 '17 at 11:57

answered Jul 15 '17 at 11:43

ZPKSoft
11 3

---

protected by xdazz Oct 7 '12 at 14:50

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?