Check if an array contains any element of another array in JavaScript

Asked 6 years, 5 months ago Active 2 months ago Viewed 301k times

300

74

I have a target array `["apple","banana","orange"]` , and I want to check if other arrays contain any one of the target array elements.

For example:

```
["apple","grape"] //returns true;

["apple","banana","pineapple"] //returns true;

["grape", "pineapple"] //returns false;
```

How can I do it in JavaScript?

javascript arrays

edited Jul 21 '16 at 6:29
shA.t
13.6k 4 41 79

asked May 1 '13 at 3:58
Alex
2,377 10 27 35

2 Use a `for` loop and iterate over the target array. If every element is contained within the current array (use `current.indexOf(elem) !== -1)` , then they're all in there. – Blender May 1 '13 at 3:59

11 @Alex pick an answer dude, it is rude to leave an answer unchecked, especially with this many good answers. I'd pick the underscore / lodash one if I were you. – Leon Gaban Aug 15 '16 at 20:33

24 Answers

Vanilla JS

```
const found = arr1.some(r=> arr2.includes(r))
```

ES6:

```
const found = arr1.some(r=> arr2.indexOf(r) >= 0)
```

How it works

`some(..)` checks each element of the array against a test function and returns true if any element of the array passes the test function, otherwise, it returns false. `indexOf(..) >= 0` and `includes(..)` both return true if the given argument is present in the array.

edited Feb 11 at 9:53

answered Oct 6 '16 at 10:30
Paul Grimshaw
8,659 5 28 49

30 This code needs an explanation - this might work, but there's no value because noone is learning anything – TolMera Jul 19 '17 at 21:50

6 Array.prototype.some() checks each element of the array against a test function and returns `true` if any element of the array passes the test function. Otherwise, it returns `false` . – Hendeca Aug 15 '17 at 19:25

2 this should be the correct answer! Great one using ES6 – Nacho Aug 23 '17 at 15:00

1 Is it expected that my result is `[false, false, false]` instead of an empty array `[]` ? – Batman Sep 24 '18 at 8:00

@Batman: The result is true/false, but you can adapt the solution from Mr. skyisred – 0zkr PM Jun 19 at 22:58

219

vanilla js

```
/**
 * @description determine if an array contains one or more items from another array.
 * @param {array} haystack the array to search.
 * @param {array} arr the array providing items to check for in the haystack.
 * @return {boolean} true|false if haystack contains at least one item from arr.
 */
var findOne = function (haystack, arr) {
    });
};
```

edited May 25 '16 at 10:42
Community ♦
1 1

answered Sep 19 '14 at 4:52
skyisred
3,724 5 25 47

9 Nice solution! `some()` is rad. Quits as soon as something matches. – averydev Mar 2 '16 at 22:04

6 event tidier like this: `arr.some(v=> haystack.indexOf(v) >= 0)` – Paul Grimshaw Oct 6 '16 at 10:27

79 Also available in ES2016 `arr.some(v => haystack.includes(v))` – loganfsmyth Oct 30 '16 at 20:30

5 in one line `arr1.some(v => arr2.indexOf(v) >= 0)`. – webjay Jan 12 '17 at 18:14

1 For now, it might be best to avoid using `includes`, as apparently it is not supported in IE: stackoverflow.com/questions/36574351/… – Shafique Jamal Mar 3 '18 at 2:04

70

If you're not opposed to using a libray, http://underscorejs.org/ has an intersection method, which can simplify this:

```
var _ = require('underscore');

var target = [ 'apple', 'orange', 'banana'];
var fruit2 = [ 'apple', 'orange', 'mango'];
var fruit3 = [ 'mango', 'lemon', 'pineapple'];
var fruit4 = [ 'orange', 'lemon', 'grapes'];

console.log(_.intersection(target, fruit2)); //returns [apple, orange]
console.log(_.intersection(target, fruit3)); //returns []
console.log(_.intersection(target, fruit4)); //returns [orange]
```

The intersection function will return a new array with the items that it matched and if not matches it returns empty array.

answered May 1 '13 at 4:16
willz
1,455 2 17 22

JHH Jan 22 '16 at 8:35

1 lodash is much more readable then vanilla Javascript, libs like it at Ramda should always be used instead of vanilla imho. Better for all devs... – Leon Gaban Aug 15 '16 at 20:32

41

If you don't need type coercion (because of the use of `indexOf`), you could try something like the following:

```
var arr = [1, 2, 3];
var check = [3, 4];

var found = false;
for (var i = 0; i < check.length; i++) {
    if (arr.indexOf(check[i]) > -1) {
        found = true;
        break;
    }
}
console.log(found);
```

Where `arr` contains the target items. At the end, `found` will show if the second array had **at least one** match against the target.

Of course, you can swap out numbers for anything you want to use - strings are fine, like your example.

And in my specific example, the result should be `true` because the second array's `3` exists in the target.

UPDATE:

Here's how I'd organize it into a function (with some minor changes from before):

```
var anyMatchInArray = (function () {
    "use strict";

    var targetArray, func;

    targetArray = ["apple", "banana", "orange"];
    func = function (checkerArray) {
        var found = false;
        for (var i = 0, j = checkerArray.length; !found && i < j; i++) {
        }
        return found;
    };

    return func;
}());
```

DEMO: http://jsfiddle.net/u8Bzt/

In this case, the function could be modified to have `targetArray` be passed in as an argument instead of hardcoded in the closure.

UPDATE2:

While my solution above may work and be (hopefully more) readable, I believe the "better" way to handle the concept I described is to do something a little differently. The "problem" with the above solution is that the `indexOf` inside the loop causes the target array to be looped over completely for every item in the other array. This can easily be "fixed" by using a "lookup" (a map...a JavaScript object literal). This allows two simple loops, over each array. Here's an example:

```
var anyMatchInArray = function (target, toMatch) {
    "use strict";

    var found, targetMap, i, j, cur;

    found = false;
    targetMap = {};

    // Put all values in the `target` array into a map, where
    //  the keys are the values from the array
    for (i = 0, j = target.length; i < j; i++) {
        cur = target[i];
        targetMap[cur] = true;
    }

    // Loop over all items in the `toMatch` array and see if any of
    //  their values are in the map from before
    for (i = 0, j = toMatch.length; !found && (i < j); i++) {
        cur = toMatch[i];
        found = !!targetMap[cur];
        // If found, `targetMap[cur]` will return true, otherwise it
        //  will return `undefined`...that's what the `!!` is for
    }
```

DEMO: http://jsfiddle.net/5Lv9v/

The downside to this solution is that only numbers and strings (and booleans) can be used (correctly), because the values are (implicitly) converted to strings and set as the keys to the lookup map. This isn't exactly good/possible/easily done for non-literal values.

edited Mar 13 '14 at 20:19

answered May 1 '13 at 4:04

Ian
41.5k 11 87 102

2 Excelent example, the second example is simply brilliant. – Sorin Haidau Apr 28 '14 at 9:36

Why are you using for loops while you could use some or findIndex? – It's me ... Alex Oct 24 '14 at 11:59

1 "some" simplifies the code greatly. Also, anyMatchInArray([1,2,3,"cats","4"], ["1",4]) would be true. Lastly, this may be more performant IF you had a large number of lookups and cached the targetMap. Even so, there could probably be performance increases. For example, I would guess that "found = toMatch[i] !== undefined" would be more performant, and in some cases better (so that you don't evaluate "" or 0 to false) – csga5000 Jan 12 '16 at 20:17

"otherwise it will return `undefined` ...that's what the `!!` is for" - that's wrong. It will return the boolean opposition of `!` . – AlienWebguy Sep 30 '17 at 14:42

41

ES6 (fastest)

```
const a = ['a', 'b', 'c'];
const b = ['c', 'a', 'd'];
a.some(v=> b.indexOf(v) !== -1)
```

ES2016

```
const a = ['a', 'b', 'c'];
const b = ['c', 'a', 'd'];
a.some(v => b.includes(v));
```

Underscore

DEMO: https://jsfiddle.net/r257wuv5/

jsPerf: https://jsperf.com/array-contains-any-element-of-another-array

edited Sep 12 '18 at 7:25

answered Jul 25 '17 at 13:02

lusk
411 4 3

1 this is the most simple and declarative solution – bitstrider Oct 28 '17 at 16:31

I know I'm really late for this, but to check the console if JSFiddle add JQuery Edge and turn on Firebug Lite – Rojo Jun 20 at 15:29

28

Using filter/indexOf:

```
function containsAny(source,target)
{
    var result = source.filter(function(item){ return target.indexOf(item) > -1});
    return (result.length > 0);
}


//results

var fruits = ["apple","banana","orange"];


console.log(containsAny(fruits,["apple","grape"]));

console.log(containsAny(fruits,["apple","banana","pineapple"]));

console.log(containsAny(fruits,["grape", "pineapple"]));
```

Run code snippet Expand snippet

This suffers from the same problem as library functions such as _.intersection in that it will continue looking for matches even after finding one. For small arrays obviously it doesn't matter though. – JHH Jan 22 '16 at 8:37

27

You could use lodash and do:

```
_.intersection(originalTarget, arrayToCheck).length > 0
```

Set intersection is done on both collections producing an array of identical elements.

answered Jan 6 '16 at 19:31

Justin Cuaresma
401 5 7

9

I found this short and sweet syntax to match all or some elements between two arrays. For example

// OR operation. find if any of array2 elements exists in array1. This will return as soon as there is a first match as some method breaks when function returns TRUE

```
let array1 = ['a', 'b', 'c', 'd', 'e'], array2 = ['a', 'b'];

console.log(array2.some(ele => array1.includes(ele)));
```

// prints TRUE

// AND operation. find if all of array2 elements exists in array1. This will return as soon as there is a no first match as some method breaks when function returns TRUE

```
let array1 = ['a', 'b', 'c', 'd', 'e'], array2 = ['a', 'x'];

console.log(!array2.some(ele => !array1.includes(ele)));
```

answered May 22 '18 at 20:13
kashpatel
442 4 12

I really liked the second part of the question, to make it work for ES5, did it like: !array2.some(function(ele) {return array1.indexOf(ele) === -1 }); – Friesgaard Jun 19 '18 at 18:09

6

You can use a nested Array.prototype.some call. This has the benefit that it will bail at the first match instead of other solutions that will run through the full nested loop.

eg.

```
var arr = [1, 2, 3];
var match = [2, 4];

var hasMatch = arr.some(a => match.some(m => a === m));
```

answered Jan 19 '18 at 13:50
bingles
5,953 4 54 61

6

ES6 solution:

```
let arr1 = [1, 2, 3];
let arr2 = [2, 3];

let isFounded = arr1.some( ai => arr2.includes(ai) );
```

Unlike of it: Must contains all values.

```
let allFounded = arr2.every( ai => arr1.includes(ai) );
```

answered Jul 14 at 9:55

Tanvir Hasan
143 1 9

Is there any way to get index of *arra2* values from *array1*?? – Anzil khaN Jul 31 at 9:17

1 In that case, we can use "filter" instead of "some". Then it'll return an array instead of boolean and you can easily access value from there. – Tanvir Hasan Aug 7 at 6:50

4

Here is an interesting case I thought I should share.

Let's say that you have an array of objects and an array of selected filters.

```
let arr = [
  { id: 'x', tags: ['foo'] },
  { id: 'y', tags: ['foo', 'bar'] },
  { id: 'z', tags: ['baz'] }
];

const filters = ['foo'];
```

To apply the selected filters to this structure we can

```
if (filters.length > 0)
  arr = arr.filter(obj =>
    obj.tags.some(tag => filters.includes(tag))
  );

// [
//   { id: 'x', tags: ['foo'] },
//   { id: 'y', tags: ['foo', 'bar'] }
// ]
```

answered Mar 4 '18 at 14:17

user5470921
384 2 13

3

What about using a combination of some/findIndex and indexOf?

So something like this:

```
var array1 = ["apple","banana","orange"];
var array2 = ["grape", "pineapple"];

var found = array1.some(function(v) { return array2.indexOf(v) != -1; });
```

To make it more readable you could add this functionality to the Array object itself.

```
Array.prototype.indexOfAny = function (array) {
    return this.findIndex(function(v) { return array.indexOf(v) != -1; });
}

Array.prototype.containsAny = function (array) {
    return this.indexOfAny(array) != -1;
}
```

Note: If you'd want to do something with a predicate you could replace the inner indexOf with another findIndex and a predicate

edited Oct 24 '14 at 12:03

answered Oct 24 '14 at 11:55
It's me ... Alex
171 2 11

3

My solution applies Array.prototype.some() and Array.prototype.includes() array helpers which do their job pretty efficient as well

ES6

```
const originalFruits = ["apple","banana","orange"];

const fruits1 = ["apple","banana","pineapple"];

const fruits2 = ["grape", "pineapple"];

const commonFruits = (myFruitsArr, otherFruitsArr) => {
```

```
console.log(commonFruits(originalFruits, fruits1)) //returns true;
console.log(commonFruits(originalFruits, fruits2)) //returns false;
```

Run code snippet   Expand snippet

edited Jan 16 at 12:12
Emobe
371 4 21

answered Dec 23 '18 at 15:31
Svyatoslav Gerasimov
56 7

Is there any way to get index of includes items from originalFruits ?? – Anzil khaN Jul 31 at 9:26

2

It can be done by simply iterating across the main array and check whether other array contains any of the target element or not.

Try this:

```
function Check(A) {
    var myarr = ["apple", "banana", "orange"];
    var i, j;
    var totalmatches = 0;
    for (i = 0; i < myarr.length; i++) {
        for (j = 0; j < A.length; ++j) {
            if (myarr[i] == A[j]) {

                totalmatches++;

            }

        }
    }
    if (totalmatches > 0) {
        return true;
    } else {
        return false;
    }
}
var fruits1 = new Array("apple", "grape");
alert(Check(fruits1));
```

```
var fruits3 = new Array("grape", "pineapple");
alert(Check(fruits3));
```

DEMO at JSFIDDLE

answered May 1 '13 at 4:17
ritesh_NITW
3,129 3 31 62

2

With underscorejs

```
var a1 = [1,2,3];
var a2 = [1,2];

_.every(a1, function(e){ return _.include(a2, e); } ); //=> false
_.every(a2, function(e){ return _.include(a1, e); } ); //=> true
```

edited Sep 29 '14 at 12:57

answered Apr 3 '14 at 14:17
fguillen
23.1k 13 93 151

2 Personally, although I like underscorejs, this is a classic example of how convoluted code can look. Not only is it hard to comprehend as underscorejs code, but from a general coding point of view, the same is also true (e.g. the word "every" doesn't spring to mind when I want to find index of something in an array but "indexOf" does). We should avoid the needless use of third party tools when for a few characters extra, a pure JavaScript solution could be provided. Using underscorejs for the sake of it means your solution becomes tightly coupled with third party code. – csharpforevermore May 28 '15 at 15:17

@csharpforevermore I suppose this is a matter of taste, you say this solution is more *convoluted* than all of the others using `indexOf` I think the opposite :). In the other hand I agree in trying to not add external libraries if they are not really needed, but I'm not really obsessive with that, third-part libraries not only offer useful functionalities but also *solid* functionalities. For example: have you tested all the edge-cases and mayor-browsers with your solution?.. (by the way, `every` is not trying to find an index in a list but evaluating something in *every* element in the list) – fguillen May 29 '15 at 8:12

Adding to Array Prototype

Disclaimer: Many would strongly advise against this. The only time it'd really be a problem was if a library added a prototype function with the same name (that behaved differently) or something like that.

Code:

```
Array.prototype.containsAny = function(arr) {
    return this.some(
        (v) => (arr.indexOf(v) >= 0)
    )
}
```

Without using big arrow functions:

```
Array.prototype.containsAny = function(arr) {
    return this.some(function (v) {
        return arr.indexOf(v) >= 0
    })
}
```

Usage

```
var a = ["a","b"]

console.log(a.containsAny(["b","z"]))    // Outputs true

console.log(a.containsAny(["z"]))    // Outputs false
```

edited Apr 8 '16 at 2:34
Antonio Brandao
744 9 16

answered Jan 4 '16 at 8:31
csga5000
2,479 4 29 45

2

Vanilla JS with partial matching & case insensitive

The problem with some previous approaches is that they require an *exact match of every word*. But, **What if you want to provide results for partial matches?**

```
function search(arrayToSearch, wordsToSearch) {
    arrayToSearch.filter(v =>
        wordsToSearch.every(w =>
            v.toLowerCase().split(" ").
                reduce((isIn, h) => isIn || String(h).indexOf(w) >= 0, false)
            )
        )
}
//Usage
var myArray = ["Attach tag", "Attaching tags", "Blah blah blah"];
var searchText = "Tag attach";
var searchArr = searchText.toLowerCase().split(" "); //["tag", "attach"]

var matches = search(myArray, searchArr);
//Will return
//["Attach tag", "Attaching tags"]
```

This is useful when you want to provide a search box where users type words and the results can have those words in any order, position and case.

answered Aug 9 '17 at 19:46
SntsDev
71 2

Update @Paul Grimshaw answer, use `includes` insteed of `indexOf` for more readable

2

> let found = arr1.some(r=> arr2.indexOf(r) >= 0)
> let found = arr1.some(r=> arr2.includes(r))

answered Jan 17 '18 at 8:20
Đinh Anh Huy
129 9

I came up with a solution in node using underscore js like this:

```
        next();
    }
```

answered Apr 13 '16 at 13:44
charles
29 1

1

Just one more solution

```
var a1 = [1, 2, 3, 4, 5]
var a2 = [2, 4]
```

Check if a1 contain all element of a2

```
var result = a1.filter(e => a2.indexOf(e) !== -1).length === a2.length
console.log(result)
```

answered Jul 23 at 1:36
Cong Nguyen
881 1 9 10

0

Personally, I would use the following function:

```
var arrayContains = function(array, toMatch) {
    var arrayAsString = array.toString();
    return (arrayAsString.indexOf(','+toMatch+',') >-1);
}
```

The "toString()" method will always use commas to separate the values. Will only really work with primitive types.

answered May 11 '15 at 17:24
csharpforevermore

2 This won't work when the elements are at the beginning or at the end of the array, or in a different order. – DanielM Aug 10 '15 at 8:06

1 -1 because as DanielM said this is broken. You *could* prepend and append a comma to arrayAsString as a workaround, but honestly it just seems like an overly complicated solution to use strings. – JHH Jan 22 '16 at 8:38

0

Array .filter() with a nested call to .find() will return all elements in the first array that are members of the second array. Check the length of the returned array to determine if any of the second array were in the first array.

```
getCommonItems(firstArray, secondArray) {
  return firstArray.filter((firstArrayItem) => {
    return secondArray.find((secondArrayItem) => {
      return firstArrayItem === secondArrayItem;
    });
  });
}
```

answered Jan 31 '17 at 14:21
Neoheurist
1,094 3 19 37

Is there a way to "clean" the array? Like deleting the values in the second array if they exist in the first? – sandrooco Mar 10 '17 at 15:18

0

```
console.log("searching Array: "+finding_array);
console.log("searching in:"+reference_array);
var check_match_counter = 0;
for (var j = finding_array.length - 1; j >= 0; j--)
{
    if(reference_array.indexOf(finding_array[j]) > 0)
    {
        check_match_counter = check_match_counter + 1;
    }
}
 var match = (check_match_counter > 0) ? true : false;
console.log("Final result:"+match);
```

61 6

0

```
var target = ["apple","banana","orange"];
var checkArray = ["apple","banana","pineapple"];

var containsOneCommonItem = target.some(x => checkArray.some(y => y === x));`

["apple","grape"] //returns true;

["apple","banana","pineapple"] //returns true;

["grape", "pineapple"] //returns false;
```

answered Sep 24 '18 at 21:03

Vasudev
6 2