

Filter truthy values



filter truthy values:
`const arr = [0, 1, "", "hello", false, true, null];`

Here is the **simplest, most important interview-style answer** for **filtering truthy values** from an array:

✓ Filter Truthy Values in JavaScript

Javascript truthy → any value that is **not**:

`0, "", false, null, undefined, NaN`

Your array:

```
js
const arr = [0, 1, "", "hello", false, true, null];
```

✓ Solution

Method 1: Using Boolean (most common)

```
js
const truthyValues = arr.filter(Boolean);
console.log(truthyValues);
```

Output

```
js
[1, "hello", true]
```

- ✓ `Boolean` converts every element → `true / false`
- ✓ `filter` keeps only the elements that convert to `true`.

🧠 Beginner Explanation (Very Simple)

Value	Truthy?	Kept?
0	✗ false	Remove
1	✓ true	Keep
""	✗ false	Remove

Value	Truthy?	Kept?
"hello"	✓ true	Keep
false	✗ false	Remove
true	✓ true	Keep
null	✗ false	Remove

★ Method 2: Using custom check

```
js

const truthyValues = arr.filter(value => Boolean(value));
```

If you want, I can also give you:

- ✓ opposite: **filter falsy values**
- ✓ interview questions related to truthy/falsy
- ✓ more practice problems



```
const arr = [0, 1, "", "hello", false, true, null];

const result = arr?.filter((item) => Boolean)
console.log("result:", result)
```

why it is wrong

Your code is wrong because you are passing **the Boolean function itself**, not **calling it with item**.

✗ Your Code

```
js

const result = arr?.filter((item) => Boolean)
```

Here:

- **Boolean** is a function **reference**
- You are **not passing the item to Boolean**
- **filter** expects a function that returns **true/false for each element**
- But your callback returns **the Boolean function itself** (which is always truthy)

So this callback:

```
js
(item) => Boolean
```

ALWAYS returns the Boolean function, not Boolean(item).

Therefore filter keeps **all elements**, because a function is truthy.

✓ Correct Code

```
js
const result = arr.filter((item) => Boolean(item));
```

or the shorter & most common one:

```
js
const result = arr.filter(Boolean);
```

🧠 Why it works?

✓ `Boolean(item)` returns

- `true` for truthy values
- `false` for falsy values

✓ `filter` keeps only items where callback returns true.

★ Summary

Code	Correct?	Why
<code>(item) => Boolean</code>	✗ Wrong	Returning the function itself
<code>(item) => Boolean(item)</code>	✓ Correct	Calling Boolean with item
<code>filter(Boolean)</code>	✓ Correct	Short-hand syntax

If you want, I can also explain:

What does it mean that Boolean is being passed as a callback directly?



```
const result = arr.filter((item) => Boolean(item));
const result = arr.filter(Boolean);
```

why in one we have to pass value and not in other

Great question, and this is asked a LOT in interviews.

Let's explain **clearly and in beginner-friendly terms**.

✓ Short Answer

Both work the same because:

✓ In this one:

```
js

arr.filter((item) => Boolean(item))
```

You **manually call Boolean(item)**.

✓ In this one:

```
js

arr.filter(Boolean)
```

You **pass the Boolean function itself**, and **filter()** automatically passes each item into it.

So — you *don't* pass the value yourself, **filter()** does it behind the scenes.

🧠 Detailed Explanation

1 When you write:

```
js

arr.filter((item) => Boolean(item))
```

- You create your own callback function
- **filter** calls that function repeatedly
- Each time, it passes the item, and you call **Boolean(item)**

So you're doing:

- filter → passes value to your function
- your function → passes value to Boolean

2 When you write:

```
js
arr.filter(Boolean)
```

Here you give the **Boolean function directly**.

Filter internally does this:

```
js
Boolean(item)
```

Because `.filter()` automatically calls the callback like:

```
js
callback(element, index, array)
```

So if you pass **Boolean**, it becomes:

```
js
Boolean(element)
```

- ✓ You didn't call Boolean
- ✓ Filter called it for you
- ✓ That's why you don't pass the value
- ✓ Filter handles it

3 Why does this not work?

```
js
(item) => Boolean      // ✗ wrong
```

Because you are returning the **Boolean function itself**, not calling it.

Filter only checks if returned value is true/false →
A function is always truthy → so everything passes.

4 Perfect Summary (Interview-quality)

Syntax	Meaning	Who passes the item?
<code>filter((item) => Boolean(item))</code>	You create a callback and call Boolean manually	You
<code>filter(Boolean)</code>	You give Boolean function as callback	<code>filter()</code>

Both end up doing the same thing:

```
scss
```

```
Boolean(item)
```

If you want, I can also explain:

- ✓ **how filter passes arguments internally**
- ✓ **how passing a function reference works**
- ✓ **truthy/falsy tricky interview questions**



explain this line:

```
callback(element, index, array)
```

Sure Vipul, let me explain this line **very clearly and simply**, because this is one of the MOST important fundamentals of `map`, `filter`, `reduce`, etc.

✓ What does this mean?

```
js
```

```
callback(element, index, array)
```

This describes **how JavaScript internally calls your function** when you use `.filter()`, `.map()`, `.find()`, etc.

🧠 Simple Explanation

When you write:

```
js
```

```
arr.filter(callbackFunction)
```

JavaScript internally loops through the array and calls your callback like:

```
scss
```

```
callbackFunction(currentElement, currentIndex, originalArray)
```

This is the exact shape of the function call.



Meaning of each parameter

Parameter	Meaning	Example
element	Current item in the array	1, "hello", etc.
index	Position of the element	0, 1, 2...
array	The full original array	[0,1,"",...]

🎯 Visual Example

Array:

```
js
const arr = [10, 20, 30];
```

Your filter:

```
js
arr.filter((item, index, fullArray) => {
  console.log(item, index, fullArray);
});
```

Internally JavaScript does:

Step 1:

```
scss
callback(10, 0, [10,20,30])
```

Step 2:

```
scss
callback(20, 1, [10,20,30])
```

Step 3:

```
scss
callback(30, 2, [10,20,30])
```

So `.filter()` calls your function **one time per element**.

✨ Why this helps you understand:

Now the earlier line makes sense:

```
js
arr.filter(Boolean)
```

Internally JavaScript replaces it like:

```
pgsql
Boolean(element, index, array)
```

But the Boolean function **only uses the first argument**, so index and array are ignored.

🎉 Final Summary

`callback(element, index, array)` means:

- 👉 JS automatically passes
 - ✓ current element
 - ✓ its index
 - ✓ original array
- to your callback function when running filter or map.

You don't need to pass these manually.

If you want, I can also explain:

- ✓ the exact internal working of filter (step-by-step)
- ✓ difference between function reference vs function call
- ✓ how callback works in JS (very important topic for frontend interviews)



`undefined` is falsy or tuthy

`undefined` is **falsy** in JavaScript.

✓ Easy explanation

Falsy values are values that become **false** when converted to Boolean.

This is falsy:

```
js
Boolean(undefined) // false
```

📌 List of all falsy values in JS

These **7** values are falsy:

1. **false**
2. **0**
3. **-0**
4. **""** (empty string)
5. **null**
6. **undefined**
7. **NaN**

Everything else is **truthy**.

⭐ Quick check

```
js

if (undefined) {
  console.log("truthy");
} else {
  console.log("falsy");
}
```

Output:

```
nginx

falsy
```

If you want, I can give you a **full list of truthy values** as well.