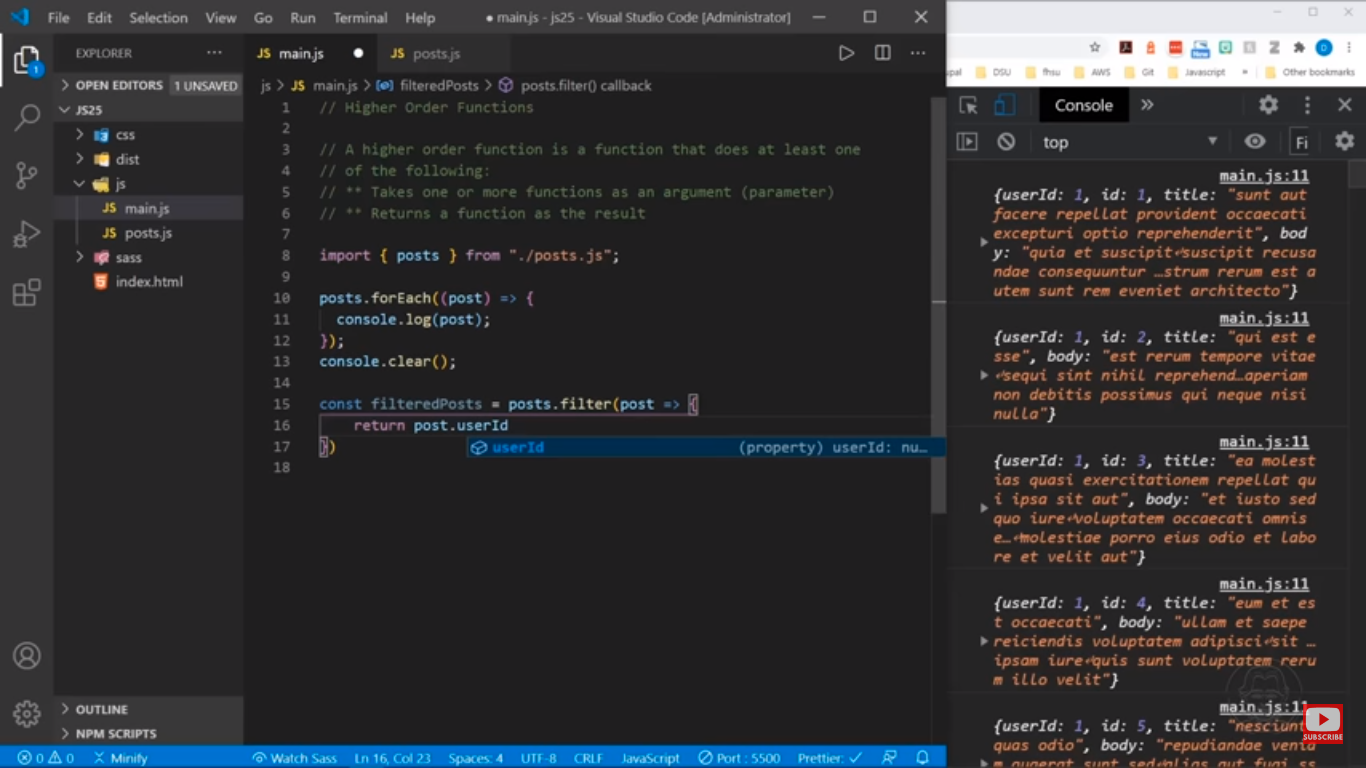
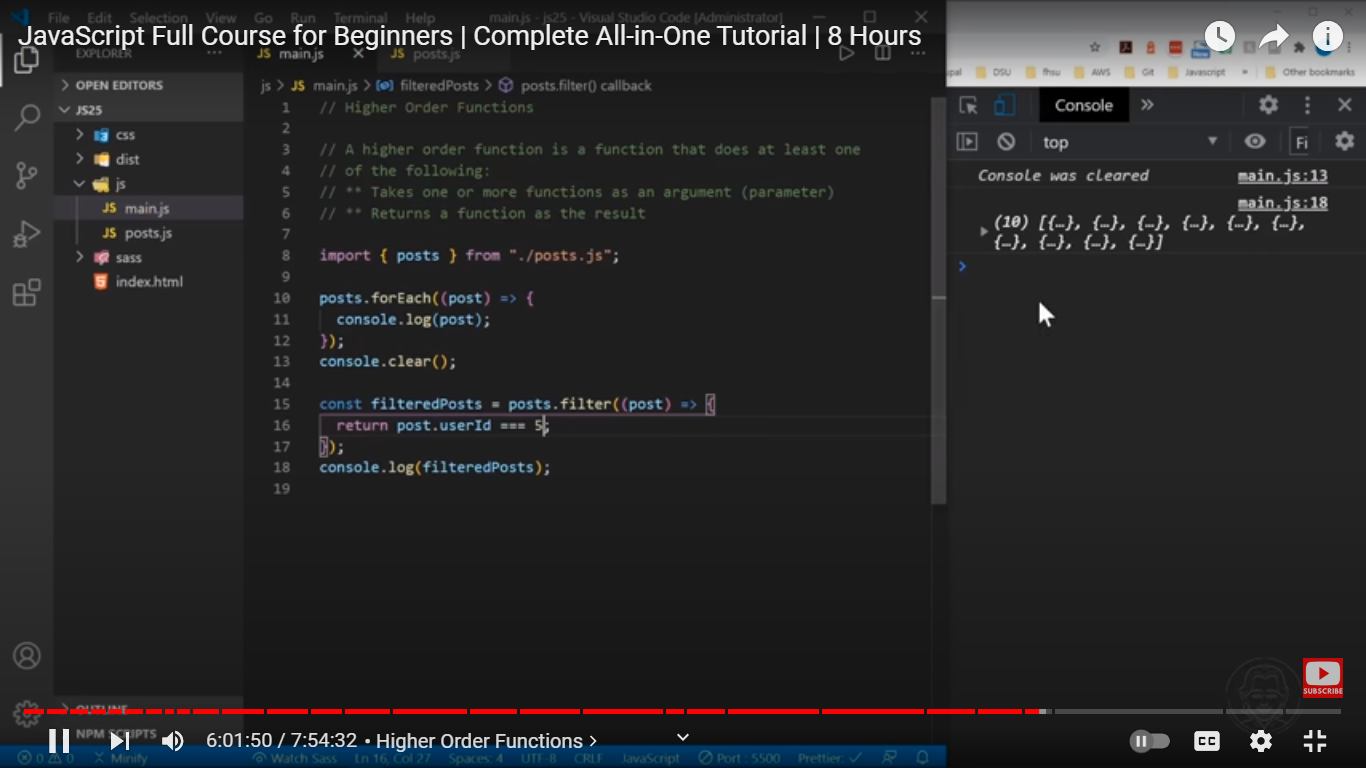
filter() function





The **filter()** method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances creates a [shallow copy](https://developer.mozilla.org/en-US/docs/Glossary/Shallow_copy) of a portion of a given array, filtered down to just the elements from the given array that pass the test implemented by the provided function.

const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];

const result = words.filter((word) => word.length > 6);

console.log(result);

// Expected output: Array ["exuberant", "destruction", "present"]

[**Parameters**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter#parameters)

callbackFn

A function to execute for each element in the array. It should return a [truthy](https://developer.mozilla.org/en-US/docs/Glossary/Truthy) value to keep the element in the resulting array, and a [falsy](https://developer.mozilla.org/en-US/docs/Glossary/Falsy) value otherwise. The function is called with the following arguments:

element

The current element being processed in the array.

index

The index of the current element being processed in the array.

array

The array filter() was called upon.

thisArg Optional

A value to use as this when executing callbackFn. See [iterative methods](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#iterative_methods).

[**Return value**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter#return_value)

A [shallow copy](https://developer.mozilla.org/en-US/docs/Glossary/Shallow_copy) of a portion of the given array, filtered down to just the elements from the given array that pass the test implemented by the provided function. If no elements pass the test, an empty array will be returned.

[**Description**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter#description)

The filter() method is an [iterative method](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#iterative_methods). It calls a provided callbackFn function once for each element in an array, and constructs a new array of all the values for which callbackFn returns a [truthy](https://developer.mozilla.org/en-US/docs/Glossary/Truthy) value. Array elements which do not pass the callbackFn test are not included in the new array.

callbackFn is invoked only for array indexes which have assigned values. It is not invoked for empty slots in [sparse arrays](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Indexed_collections#sparse_arrays).

The filter() method is a [copying method](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#copying_methods_and_mutating_methods). It does not alter this but instead returns a [shallow copy](https://developer.mozilla.org/en-US/docs/Glossary/Shallow_copy) that contains the same elements as the ones from the original array (with some filtered out). However, the function provided as callbackFn can mutate the array. Note, however, that the length of the array is saved *before* the first invocation of callbackFn. Therefore:

* callbackFn will not visit any elements added beyond the array's initial length when the call to filter() began.
* Changes to already-visited indexes do not cause callbackFn to be invoked on them again.
* If an existing, yet-unvisited element of the array is changed by callbackFn, its value passed to the callbackFn will be the value at the time that element gets visited. [Deleted](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete) elements are not visited.

**Examples**

Filtering out all small values

The following example uses filter() to create a filtered array that has all elements with values less than 10 removed.

JS

function isBigEnough(value) {

return value >= 10;

}

const filtered = [12, 5, 8, 130, 44].filter(isBigEnough);

// filtered is [12, 130, 44]

Find all prime numbers in an array

The following example returns all prime numbers in the array:

JS

Copy to Clipboard

const array = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13];

function isPrime(num) {

for (let i = 2; num > i; i++) {

if (num % i === 0) {

return false;

}

}

return num > 1;

}

console.log(array.filter(isPrime)); // [2, 3, 5, 7, 11, 13]

Filtering invalid entries from JSON

The following example uses filter() to create a filtered JSON of all elements with non-zero, numeric id.

JS

const arr = [

{ id: 15 },

{ id: -1 },

{ id: 0 },

{ id: 3 },

{ id: 12.2 },

{},

{ id: null },

{ id: NaN },

{ id: "undefined" },

];

let invalidEntries = 0;

function filterByID(item) {

if (Number.isFinite(item.id) && item.id !== 0) {

return true;

}

invalidEntries++;

return false;

}

const arrByID = arr.filter(filterByID);

console.log("Filtered Array\n", arrByID);

// Filtered Array

// [{ id: 15 }, { id: -1 }, { id: 3 }, { id: 12.2 }]

console.log("Number of Invalid Entries =", invalidEntries);

// Number of Invalid Entries = 5

Searching in array

Following example uses filter() to filter array content based on search criteria.

JS

const fruits = ["apple", "banana", "grapes", "mango", "orange"];

/\*\*

\* Filter array items based on search criteria (query)

\*/

function filterItems(arr, query) {

return arr.filter((el) => el.toLowerCase().includes(query.toLowerCase()));

}

console.log(filterItems(fruits, "ap")); // ['apple', 'grapes']

console.log(filterItems(fruits, "an")); // ['banana', 'mango', 'orange']

Using filter() on sparse arrays

filter() will skip empty slots.

JS

console.log([1, , undefined].filter((x) => x === undefined)); // [undefined]

console.log([1, , undefined].filter((x) => x !== 2)); // [1, undefined]

Calling filter() on non-array objects

The filter() method reads the length property of this and then accesses each property whose key is a nonnegative integer less than length.

JS

const arrayLike = {

length: 3,

0: "a",

1: "b",

2: "c",

3: "a", // ignored by filter() since length is 3

};

console.log(Array.prototype.filter.call(arrayLike, (x) => x <= "b"));

// [ 'a', 'b' ]

Affecting Initial Array (modifying, appending and deleting)

The following example tests the behavior of the filter method when the array is modified.

JS

// Modifying each word

let words = ["spray", "limit", "exuberant", "destruction", "elite", "present"];

const modifiedWords = words.filter((word, index, arr) => {

arr[index + 1] += " extra";

return word.length < 6;

});

console.log(modifiedWords);

// Notice there are three words below length 6, but since they've been modified one is returned

// ["spray"]

// Appending new words

words = ["spray", "limit", "exuberant", "destruction", "elite", "present"];

const appendedWords = words.filter((word, index, arr) => {

arr.push("new");

return word.length < 6;

});

console.log(appendedWords);

// Only three fits the condition even though the `words` itself now has a lot more words with character length less than 6

// ["spray" ,"limit" ,"elite"]

// Deleting words

words = ["spray", "limit", "exuberant", "destruction", "elite", "present"];

const deleteWords = words.filter((word, index, arr) => {

arr.pop();

return word.length < 6;

});

console.log(deleteWords);

// Notice 'elite' is not even obtained as it's been popped off 'words' before filter can even get there

// ["spray" ,"limit"]