Array

The **Array** object, as with arrays in other programming languages, enables storing a collection of multiple items under a single variable name, and has members for performing common array operations.

Description

In JavaScript, arrays aren't primitives but are instead Array objects with the following core characteristics:

- **JavaScript arrays are resizable** and **can contain a mix of different data types**. (When those characteristics are undesirable, use typed arrays instead.)
- **JavaScript arrays are not associative arrays** and so, array elements cannot be accessed using arbitrary strings as indexes, but must be accessed using nonnegative integers (or their respective string form) as indexes.
- **JavaScript arrays are zero-indexed**: the first element of an array is at index 0, the second is at index 1, and so on — and the last element is at the value of the array's length property minus 1.
- **JavaScript array-copy operations create shallow copies**. (All standard built-in copy operations with *any* JavaScript objects create shallow copies, rather than deep copies).

Array indices

Array objects cannot use arbitrary strings as element indexes (as in an associative array) but must use nonnegative integers (or their respective string form). Setting or accessing via non-integers will not set or retrieve an element from the array list itself, but will set or access a variable associated with that array's object property collection. The array's object properties and list of array elements are separate, and the array's traversal and mutation operations cannot be applied to these named properties.

Array elements are object properties in the same way that toString is a property (to be specific, however, toString() is a method). Nevertheless, trying to access an element of an array as follows throws a syntax error because the property name is not valid:

JSCopy to Clipboard

```
console.log(arr.0); // a syntax error
```

JavaScript syntax requires properties beginning with a digit to be accessed using bracket notation instead of dot notation. It's also possible to quote the array indices (e.g., years['2'] instead of years[2]), although usually not necessary.

The 2 in years[2] is coerced into a string by the JavaScript engine through an implicit toString conversion. As a result, '2' and '02' would refer to two different slots on the years object, and the following example could be true:

JSCopy to Clipboard

```
console.log(years["2"] !== years["02"]);
```

Only years['2'] is an actual array index. years['02'] is an arbitrary string property that will not be visited in array iteration.

Relationship between length and numerical properties

A JavaScript array's length property and numerical properties are connected.

Several of the built-in array methods (e.g., join(), slice(), indexOf(), etc.) take into account the value of an array's length property when they're called.

Other methods (e.g., push(), splice(), etc.) also result in updates to an array's length property.

JSCopy to Clipboard

```
const fruits = [];
fruits.push("banana", "apple", "peach");
console.log(fruits.length); // 3
```

When setting a property on a JavaScript array when the property is a valid array index and that index is outside the current bounds of the array, the engine will update the array's length property accordingly:

JSCopy to Clipboard

```
fruits[5] = "mango";
console.log(fruits[5]); // 'mango'
console.log(Object.keys(fruits)); // ['0', '1', '2', '5']
console.log(fruits.length); // 6
```

Increasing the length.

JSCopy to Clipboard

```
fruits.length = 10;
console.log(fruits); // ['banana', 'apple', 'peach', empty x 2, 'mango', empty x 4]
console.log(Object.keys(fruits)); // ['0', '1', '2', '5']
console.log(fruits.length); // 10
console.log(fruits[8]); // undefined
```

Decreasing the length property does, however, delete elements.

JSCopy to Clipboard

```
fruits.length = 2;
console.log(Object.keys(fruits)); // ['0', '1']
console.log(fruits.length); // 2
```

This is explained further on the length page.

Array methods and empty slots

Empty slots in sparse arrays behave inconsistently between array methods. Generally, the older methods will skip empty slots, while newer ones treat them as undefined.

Among methods that iterate through multiple elements, the following do an in check before accessing the index and do not conflate empty slots with undefined:

- concat()
- copyWithin()
- every()
- filter()
- flat()
- flatMap()
- forEach()
- indexOf()
- lastIndexOf()
- map()
- reduce()
- reduceRight()
- reverse()
- slice()
- some()
- sort()
- splice()

For exactly how they treat empty slots, see the page for each method.

These methods treat empty slots as if they are undefined:

- entries()
- fill()
- find()
- findIndex()
- findLast()
- findLastIndex()
- group() Experimental
- groupToMap() Experimental
- includes()
- join()
- keys()
- toLocaleString()
- values()

Copying methods and mutating methods

Some methods do not mutate the existing array that the method was called on, but instead return a new array. They do so by first constructing a new array and then populating it with elements. The copy always happens *shallowly* — the method never copies anything beyond the initially created array. Elements of the original array(s) are copied into the new array as follows:

- Objects: the object reference is copied into the new array. Both the original and new array refer to the same object. That is, if a referenced object is modified, the changes are visible to both the new and original arrays.
- Primitive types such as strings, numbers and booleans (not String, Number, and Boolean objects): their values are copied into the new array.

Other methods mutate the array that the method was called on, in which case their return value differs depending on the method: sometimes a reference to the same array, sometimes the length of the new array.

The following methods create new arrays by accessing this.constructor[Symbol.species] to determine the constructor to use:

- concat()
- filter()
- flat()
- flatMap()
- map()
- slice()
- splice() (to construct the array of removed elements that's returned)

The following methods always create new arrays with the Array base constructor:

- toReversed()
- toSorted()
- toSpliced()
- with()

group() and groupToMap() do not use @@species to create new arrays for each group entry, but always use the plain Array constructor. Conceptually, they are not copying methods either.

The following table lists the methods that mutate the original array, and the corresponding non-mutating alternative:

Mutating method	**Non-mutating alternative**
copyWithin()	No one-method alternative
fill()	No one-method alternative
pop()	slice(0, -1)
push(v1, v2)	concat([v1, v2])
reverse()	toReversed()
shift()	slice(1)
sort()	toSorted()
splice()	toSpliced()
unshift(v1, v2)	toSpliced(0, 0, v1, v2)

An easy way to change a mutating method into a non-mutating alternative is to use the spread syntax or slice() to create a copy first:

JSCopy to Clipboard

```
arr.copyWithin(0, 1, 2); // mutates arr
const arr2 = arr.slice().copyWithin(0, 1, 2); // does not mutate arr
const arr3 = [...arr].copyWithin(0, 1, 2); // does not mutate arr
```

Iterative methods

Many array methods take a callback function as an argument. The callback function is called sequentially and at most once for each element in the array, and the return value of the callback function is used to determine the return value of the method. They all share the same signature:

JSCopy to Clipboard

```
method(callbackFn, thisArg)
```

Where callbackFn takes three 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 that the method was called upon.

What callbackFn is expected to return depends on the array method that was called.

The thisArg argument (defaults to undefined) will be used as the this value when calling callbackFn. The this value ultimately observable by callbackFn is determined according to the usual rules: if callbackFn is non-strict, primitive this values are wrapped into objects, and undefined/null is substituted with globalThis. The thisArg argument is irrelevant for any callbackFn defined with an arrow function, as arrow functions don't have their own this binding.

All iterative methods are copying and generic, although they behave differently with empty slots.

The following methods are iterative:

- every()
- filter()
- find()
- findIndex()
- findLast()
- findLastIndex()
- flatMap()
- forEach()
- group()
- groupToMap()
- map()
- some()

In particular, every(), find(), findIndex(), findLast(), findLastIndex(), and some() do not always invoke callbackFn on every element — they stop iteration as soon as the return value is determined.

There are two other methods that take a callback function and run it at most once for each element in the array, but they have slightly different signatures from typical iterative methods (for example, they don't accept thisArg):

- reduce()
- reduceRight()

The sort() method also takes a callback function, but it is not an iterative method. It mutates the array in-place, doesn't accept thisArg, and may invoke the callback multiple times on an index.

Generic array methods

Array methods are always generic — they don't access any internal data of the array object. They only access the array elements through the length property and the indexed elements. This means that they can be called on array-like objects as well.

JSCopy to Clipboard

```
const arrayLike = {
  0: "a",
  1: "b",
  length: 2,
};
console.log(Array.prototype.join.call(arrayLike, "+")); // 'a+b'
```

Normalization of the length property

The length property is converted to an integer and then clamped to the range between 0 and 2^{53} - 1. NaN becomes 0, so even when length is not present or is undefined, it behaves as if it has value 0.

The language avoids setting length to an unsafe integer. All built-in methods will throw a TypeError if length will be set to a number greater than 2^{53} - 1. However, because the length property of arrays throws an error if it's set to greater than 2^{32} - 1, the safe integer threshold is usually not reached unless the method is called on a non-array object.

JSCopy to Clipboard

```
Array.prototype.flat.call({}); // []
```

Some array methods set the length property of the array object. They always set the value after normalization, so length always ends as an integer.

JSCopy to Clipboard

```
const a = { length: 0.7 };
Array.prototype.push.call(a);
console.log(a.length); // 0
```

Array-like objects

The term *array-like object* refers to any object that doesn't throw during the length conversion process described above. In practice, such object is expected to actually have a length property and to have indexed elements in the range 0 to length - 1. (If it doesn't have all indices, it will be functionally equivalent to a sparse array.) Any integer index less than zero or greater than length - 1 is ignored when an array method operates on an array-like object.

Many DOM objects are array-like — for example, NodeList and HTMLCollection. The arguments object is also array-like. You can call array methods on them even if they don't have these methods themselves.

JSCopy to Clipboard

```
function f() {
  console.log(Array.prototype.join.call(arguments, "+"));
}

f("a", "b"); // 'a+b'
```

Constructor

Array()

Creates a new Array object.

Static properties

Array[@@species]

Returns the Array constructor.

Static methods

Array.from()

Creates a new Array instance from an iterable or array-like object.

Array.fromAsync()

Creates a new Array instance from an async iterable, iterable, or array-like object.

Array.isArray()

Returns true if the argument is an array, or false otherwise.

Array.of()

Creates a new Array instance with a variable number of arguments, regardless of number or type of the arguments.

Instance properties

These properties are defined on Array.prototype and shared by all Array instances.

Array.prototype.constructor

The constructor function that created the instance object. For Array instances, the initial value is the Array constructor.

Array.prototype[@@unscopables]

Contains property names that were not included in the ECMAScript standard prior to the ES2015 version and that are ignored for with statement-binding purposes.

These properties are own properties of each Array instance.

length

Reflects the number of elements in an array.

Instance methods

Array.prototype.at()

Returns the array item at the given index. Accepts negative integers, which count back from the last item.

Array.prototype.concat()

Returns a new array that is the calling array joined with other array(s) and/or value(s).

Array.prototype.copyWithin()

Copies a sequence of array elements within an array.

Array.prototype.entries()

Returns a new *array iterator* object that contains the key/value pairs for each index in an array.

Array.prototype.every()

Returns true if every element in the calling array satisfies the testing function.

Array.prototype.fill()

Fills all the elements of an array from a start index to an end index with a static value.

Array.prototype.filter()

Returns a new array containing all elements of the calling array for which the provided filtering function returns true.

Array.prototype.find()

> Returns the value of the first element in the array that satisfies the provided testing function, or `undefined` if no appropriate element is found.

Array.prototype.findIndex()

> Returns the index of the first element in the array that satisfies the provided testing function, or `-1` if no appropriate element was found.

Array.prototype.findLast()

> Returns the value of the last element in the array that satisfies the provided testing function, or `undefined` if no appropriate element is found.

Array.prototype.findLastIndex()

> Returns the index of the last element in the array that satisfies the provided testing function, or `-1` if no appropriate element was found.

Array.prototype.flat()

> Returns a new array with all sub-array elements concatenated into it recursively up to the specified depth.

Array.prototype.flatMap()

> Returns a new array formed by applying a given callback function to each element of the calling array, and then flattening the result by one level.

Array.prototype.forEach()

> Calls a function for each element in the calling array.

Array.prototype.group() Experimental

> Groups the elements of an array into an object according to the strings returned by a test function.

Array.prototype.groupToMap() Experimental

> Groups the elements of an array into a Map according to values returned by a test function.

Array.prototype.includes()

> Determines whether the calling array contains a value, returning `true` or `false` as appropriate.

Array.prototype.indexOf()

Returns the first (least) index at which a given element can be found in the calling array.

Array.prototype.join()

Joins all elements of an array into a string.

Array.prototype.keys()

Returns a new *array iterator* that contains the keys for each index in the calling array.

Array.prototype.lastIndexOf()

Returns the last (greatest) index at which a given element can be found in the calling array, or -1 if none is found.

Array.prototype.map()

Returns a new array containing the results of invoking a function on every element in the calling array.

Array.prototype.pop()

Removes the last element from an array and returns that element.

Array.prototype.push()

Adds one or more elements to the end of an array, and returns the new length of the array.

Array.prototype.reduce()

Executes a user-supplied "reducer" callback function on each element of the array (from left to right), to reduce it to a single value.

Array.prototype.reduceRight()

Executes a user-supplied "reducer" callback function on each element of the array (from right to left), to reduce it to a single value.

Array.prototype.reverse()

Reverses the order of the elements of an array *in place*. (First becomes the last, last becomes first.)

Array.prototype.shift()

Removes the first element from an array and returns that element.

Array.prototype.slice()

Extracts a section of the calling array and returns a new array.

Array.prototype.some()

Returns `true` if at least one element in the calling array satisfies the provided testing function.

Array.prototype.sort()

Sorts the elements of an array in place and returns the array.

Array.prototype.splice()

Adds and/or removes elements from an array.

Array.prototype.toLocaleString()

Returns a localized string representing the calling array and its elements. Overrides the `Object.prototype.toLocaleString()` method.

Array.prototype.toReversed()

Returns a new array with the elements in reversed order, without modifying the original array.

Array.prototype.toSorted()

Returns a new array with the elements sorted in ascending order, without modifying the original array.

Array.prototype.toSpliced()

Returns a new array with some elements removed and/or replaced at a given index, without modifying the original array.

Array.prototype.toString()

Returns a string representing the calling array and its elements. Overrides the `Object.prototype.toString()` method.

Array.prototype.unshift()

Adds one or more elements to the front of an array, and returns the new `length` of the array.

Array.prototype.values()

Returns a new *array iterator* object that contains the values for each index in the array.

Array.prototype.with()

Returns a new array with the element at the given index replaced with the given value, without modifying the original array.

Array.prototype[@@iterator]()

An alias for the values() method by default.

Examples

This section provides some examples of common array operations in JavaScript.

Note: If you're not yet familiar with array basics, consider first reading JavaScript First Steps: Arrays, which explains what arrays are, and includes other examples of common array operations.

Create an array

This example shows three ways to create new array: first using array literal notation, then using the Array() constructor, and finally using String.prototype.split() to build the array from a string.

JSCopy to Clipboard

```
// 'fruits' array created using array literal notation.
const fruits = ["Apple", "Banana"];
console.log(fruits.length);
// 2

// 'fruits2' array created using the Array() constructor.
const fruits2 = new Array("Apple", "Banana");
console.log(fruits2.length);
// 2

// 'fruits3' array created using String.prototype.split().
const fruits3 = "Apple, Banana".split(", ");
console.log(fruits3.length);
// 2
```

Create a string from an array

This example uses the join() method to create a string from the fruits array.

JSCopy to Clipboard

```
const fruits = ["Apple", "Banana"];
const fruitsString = fruits.join(", ");
console.log(fruitsString);
// "Apple, Banana"
```

Access an array item by its index

This example shows how to access items in the fruits array by specifying the index number of their position in the array.

JSCopy to Clipboard

```
const fruits = ["Apple", "Banana"];

// The index of an array's first element is always 0.
fruits[0]; // Apple

// The index of an array's second element is always 1.
fruits[1]; // Banana

// The index of an array's last element is always one
// less than the length of the array.
fruits[fruits.length - 1]; // Banana

// Using an index number larger than the array's length
// returns 'undefined'.
fruits[99]; // undefined
```

Find the index of an item in an array

This example uses the indexOf() method to find the position (index) of the string "Banana" in the fruits array.

JSCopy to Clipboard

```
const fruits = ["Apple", "Banana"];
console.log(fruits.indexOf("Banana"));
// 1
```

Check if an array contains a certain item

This example shows two ways to check if the fruits array contains "Banana" and "Cherry": first with the includes() method, and then with the indexOf() method to test for an index value that's not -1.

JSCopy to Clipboard

```
const fruits = ["Apple", "Banana"];

fruits.includes("Banana"); // true
fruits.includes("Cherry"); // false

// If indexOf() doesn't return -1, the array contains the given item.
fruits.indexOf("Banana") !== -1; // true
fruits.indexOf("Cherry") !== -1; // false
```

Append an item to an array

This example uses the push() method to append a new string to the fruits array.

JSCopy to Clipboard

```
const fruits = ["Apple", "Banana"];
const newLength = fruits.push("Orange");
console.log(fruits);
// ["Apple", "Banana", "Orange"]
console.log(newLength);
// 3
```

Remove the last item from an array

This example uses the pop() method to remove the last item from the fruits array.

JSCopy to Clipboard

```
const fruits = ["Apple", "Banana", "Orange"];
const removedItem = fruits.pop();
console.log(fruits);
// ["Apple", "Banana"]
console.log(removedItem);
// Orange
```

Note: pop() can only be used to remove the last item from an array. To remove multiple items from the end of an array, see the next example.

Remove multiple items from the end of an array

This example uses the splice() method to remove the last 3 items from the fruits array.

JSCopy to Clipboard

```
const fruits = ["Apple", "Banana", "Strawberry", "Mango", "Cherry"];
const start = -3;
const removedItems = fruits.splice(start);
console.log(fruits);
// ["Apple", "Banana"]
console.log(removedItems);
// ["Strawberry", "Mango", "Cherry"]
```

Truncate an array down to just its first N items

This example uses the splice() method to truncate the fruits array down to just its first 2 items.

JSCopy to Clipboard

```
const fruits = ["Apple", "Banana", "Strawberry", "Mango", "Cherry"];
const start = 2;
const removedItems = fruits.splice(start);
console.log(fruits);
// ["Apple", "Banana"]
console.log(removedItems);
// ["Strawberry", "Mango", "Cherry"]
```

Remove the first item from an array

This example uses the shift() method to remove the first item from the fruits array.

JSCopy to Clipboard

```
const fruits = ["Apple", "Banana"];
const removedItem = fruits.shift();
console.log(fruits);
// ["Banana"]
console.log(removedItem);
// Apple
```

Note: shift() can only be used to remove the first item from an array. To remove multiple items from the beginning of an array, see the next example.

Remove multiple items from the beginning of an array

This example uses the splice() method to remove the first 3 items from the fruits array.

JSCopy to Clipboard

```
const fruits = ["Apple", "Strawberry", "Cherry", "Banana", "Mango"];
const start = 0;
const deleteCount = 3;
const removedItems = fruits.splice(start, deleteCount);
console.log(fruits);
// ["Banana", "Mango"]
console.log(removedItems);
// ["Apple", "Strawberry", "Cherry"]
```

Add a new first item to an array

This example uses the unshift() method to add, at index 0, a new item to the fruits array — making it the new first item in the array.

JSCopy to Clipboard

```
const fruits = ["Banana", "Mango"];
const newLength = fruits.unshift("Strawberry");
console.log(fruits);
// ["Strawberry", "Banana", "Mango"]
console.log(newLength);
// 3
```

Remove a single item by index

This example uses the splice() method to remove the string "Banana" from the fruits array — by specifying the index position of "Banana".

JSCopy to Clipboard

```
const fruits = ["Strawberry", "Banana", "Mango"];
const start = fruits.indexOf("Banana");
const deleteCount = 1;
const removedItems = fruits.splice(start, deleteCount);
console.log(fruits);
// ["Strawberry", "Mango"]
console.log(removedItems);
// ["Banana"]
```

Remove multiple items by index

This example uses the splice() method to remove the strings "Banana" and "Strawberry" from the fruits array — by specifying the index position of "Banana", along with a count of the number of total items to remove.

JSCopy to Clipboard

```
const fruits = ["Apple", "Banana", "Strawberry", "Mango"];
const start = 1;
const deleteCount = 2;
const removedItems = fruits.splice(start, deleteCount);
```

```
console.log(fruits);
// ["Apple", "Mango"]
console.log(removedItems);
// ["Banana", "Strawberry"]
```

Replace multiple items in an array

This example uses the splice() method to replace the last 2 items in the fruits array with new items.

JSCopy to Clipboard

```
const fruits = ["Apple", "Banana", "Strawberry"];
const start = -2;
const deleteCount = 2;
const removedItems = fruits.splice(start, deleteCount, "Mango", "Cherry");
console.log(fruits);
// ["Apple", "Mango", "Cherry"]
console.log(removedItems);
// ["Banana", "Strawberry"]
```

Iterate over an array

This example uses a for...of loop to iterate over the fruits array, logging each item to the console.

JSCopy to Clipboard

```
const fruits = ["Apple", "Mango", "Cherry"];
for (const fruit of fruits) {
  console.log(fruit);
}
// Apple
// Mango
// Cherry
```

But for...of is just one of many ways to iterate over any array; for more ways, see Loops and iteration, and see the documentation for the every(), filter(), flatMap(), map(), reduce(), and reduceRight() methods — and see the next example, which uses the forEach() method.

Call a function on each element in an array

This example uses the forEach() method to call a function on each element in the fruits array; the function causes each item to be logged to the console, along with the item's index number.

JSCopy to Clipboard

```
const fruits = ["Apple", "Mango", "Cherry"];
fruits.forEach((item, index, array) => {
  console.log(item, index);
});
// Apple 0
// Mango 1
// Cherry 2
```

Merge multiple arrays together

This example uses the concat() method to merge the fruits array with a moreFruits array, to produce a new combinedFruits array. Notice that fruits and moreFruits remain unchanged.

JSCopy to Clipboard

```
const fruits = ["Apple", "Banana", "Strawberry"];
const moreFruits = ["Mango", "Cherry"];
const combinedFruits = fruits.concat(moreFruits);
console.log(combinedFruits);
// ["Apple", "Banana", "Strawberry", "Mango", "Cherry"]

// The 'fruits' array remains unchanged.
console.log(fruits);
// ["Apple", "Banana", "Strawberry"]

// The 'moreFruits' array also remains unchanged.
console.log(moreFruits);
// ["Mango", "Cherry"]
```

Copy an array

This example shows three ways to create a new array from the existing fruits array: first by using spread syntax, then by using the from() method, and then by using the slice() method.

JSCopy to Clipboard

```
const fruits = ["Strawberry", "Mango"];

// Create a copy using spread syntax.
const fruitsCopy = [...fruits];
// ["Strawberry", "Mango"]

// Create a copy using the from() method.
const fruitsCopy2 = Array.from(fruits);
// ["Strawberry", "Mango"]

// Create a copy using the slice() method.
const fruitsCopy3 = fruits.slice();
// ["Strawberry", "Mango"]
```

All built-in array-copy operations (spread syntax, Array.from(), Array.prototype.slice(), and Array.prototype.concat()) create shallow copies. If you instead want a deep copy of an array, you can use JSON.stringify() to convert the array to a JSON string, and then JSON.parse() to convert the string back into a new array that's completely independent from the original array.

JSCopy to Clipboard

```
const fruitsDeepCopy = JSON.parse(JSON.stringify(fruits));
```

You can also create deep copies using the structuredClone() method, which has the advantage of allowing transferable objects in the source to be *transferred* to the new copy, rather than just cloned.

Finally, it's important to understand that assigning an existing array to a new variable doesn't create a copy of either the array or its elements. Instead the new variable is just a reference, or alias, to the original array; that is, the original array's name and the new variable name are just two names for the exact same object (and so will always evaluate as strictly equivalent). Therefore, if you make any changes at all either to the value of the original array or to the value of the new variable, the other will change, too:

JSCopy to Clipboard

```
const fruits = ["Strawberry", "Mango"];
const fruitsAlias = fruits;
// 'fruits' and 'fruitsAlias' are the same object, strictly equivalent.
fruits === fruitsAlias; // true
// Any changes to the 'fruits' array change 'fruitsAlias' too.
fruits.unshift("Apple", "Banana");
console.log(fruits);
// ['Apple', 'Banana', 'Strawberry', 'Mango']
console.log(fruitsAlias);
// ['Apple', 'Banana', 'Strawberry', 'Mango']
```

Grouping the elements of an array

The Array.prototype.group() methods can be used to group the elements of an array, using a test function that returns a string indicating the group of the current element.

Here we have a simple inventory array that contains "food" objects that have a name and a type.

JSCopy to Clipboard

```
const inventory = [
 { name: "asparagus", type: "vegetables" },
 { name: "bananas", type: "fruit" },
 { name: "goat", type: "meat" },
 { name: "cherries", type: "fruit" },
 { name: "fish", type: "meat" },
];
```

To use group(), you supply a callback function that is called with the current element, and optionally the current index and array, and returns a string indicating the group of the element.

The code below uses an arrow function to return the type of each array element (this uses object destructuring syntax for function arguments to unpack the type element from the passed object). The result is an object that has properties named after the unique strings returned by the callback. Each property is assigned an array containing the elements in the group.

JSCopy to Clipboard

```
const result = inventory.group(({ type }) => type);
console.log(result.vegetables);
// [{ name: "asparagus", type: "vegetables" }]
```

Note that the returned object references the *same* elements as the original array (not deep copies). Changing the internal structure of these elements will be reflected in both the original array and the returned object.

If you can't use a string as the key, for example, if the information to group is associated with an object that might change, then you can instead use Array.prototype.groupToMap(). This is very similar to group except that it groups the elements of the array into a Map that can use an arbitrary value (object or primitive) as a key.

Creating a two-dimensional array

The following creates a chessboard as a two-dimensional array of strings. The first move is made by copying the 'p' in board[6][4] to board[4][4]. The old position at [6][4] is made blank.

JSCopy to Clipboard

```
const board = [
  ["R", "N", "B", "Q", "K", "B", "N", "R"],
  ["P", "P", "P", "P", "P", "P", "P", "P"],
  [" ", " ", " ", " ", " ", " ", " ", " "],
  [" ", " ", " ", " ", " ", " ", " ", " "],
  [" ", " ", " ", " ", " ", " ", " ", " "],
  [" ", " ", " ", " ", " ", " ", " ", " "],
  ["p", "p", "p", "p", "p", "p", "p", "p"],
  ["r", "n", "b", "q", "k", "b", "n", "r"],
];

console.log(`${board.join("\n")}\n\n`);

// Move King's Pawn forward 2
board[4][4] = board[6][4];
board[6][4] = " ";
console.log(board.join("\n"));
```

Here is the output:

```
R,N,B,Q,K,B,N,R
P,P,P,P,P,P,P,P
 , , , , , , , 
 , , , , , , , 
 , , , , , , , 
 , , , , , , , 
p,p,p,p,p,p,p,p
r,n,b,q,k,b,n,r

R,N,B,Q,K,B,N,R
P,P,P,P,P,P,P,P
 , , , , , , , 
 , , , , , , , 
 , , , ,p, , , 
 , , , , , , , 
p,p,p,p, ,p,p,p
r,n,b,q,k,b,n,r
```

Using an array to tabulate a set of values

JSCopy to Clipboard

```
const values = [];
for (let x = 0; x < 10; x++) {
  values.push([2 ** x, 2 * x ** 2]);
}
console.table(values);
```

Results in

```
// The first column is the index
0  1    0
1  2    2
2  4    8
3  8    18
4  16   32
5  32   50
6  64   72
7  128  98
8  256  128
9  512  162
```

Creating an array using the result of a match

The result of a match between a RegExp and a string can create a JavaScript array that has properties and elements which provide information about the match. Such an array is returned by RegExp.prototype.exec() and String.prototype.match().

For example:

JSCopy to Clipboard

```
// Match one d followed by one or more b's followed by one d
// Remember matched b's and the following d
// Ignore case

const myRe = /d(b+)(d)/i;
const execResult = myRe.exec("cdbBdbsbz");

console.log(execResult.input); // 'cdbBdbsbz'
console.log(execResult.index); // 1
console.log(execResult); // [ "dbBd", "bB", "d" ]
```

For more information about the result of a match, see the RegExp.prototype.exec() and String.prototype.match() pages.

Specifications

Specification
ECMAScript Language Specification # sec-array-objects

Browser compatibility

Report problems with this compatibility data on GitHub

	desktop					mobile					
	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebV Andr
Array	1 Toggle history	12 Toggle history	1 Toggle history	4 Toggle history	1 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	37 Tog histo
@@iterator	38 Toggle history	12 Toggle history	36 more Toggle history	25 Toggle history	10 Toggle history	38 Toggle history	36 more Toggle history	25 Toggle history	10 Toggle history	3.0 Toggle history	38 Tog histo
@@species	51 Toggle history	79 Toggle history	48 Toggle history	38 Toggle history	10 Toggle history	51 Toggle history	48 Toggle history	41 Toggle history	10 Toggle history	5.0 Toggle history	51 Tog histo
@@unscopables	38 Toggle history	12 Toggle history	48 Toggle history	25 Toggle history	10 Toggle history	38 Toggle history	48 Toggle history	25 Toggle history	10 Toggle history	3.0 Toggle history	38 Tog histo
Array() constructor	1 Toggle history	12 Toggle history	1 Toggle history	4 Toggle history	1 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	37 Tog histo
at	92 Toggle history	92 Toggle history	90 Toggle history	78 Toggle history	15.4 Toggle history	92 Toggle history	90 Toggle history	65 Toggle history	15.4 Toggle history	16.0 Toggle history	92 Tog histo
concat	1 Toggle history	12 Toggle history	1 Toggle history	4 Toggle history	1 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	4.4 Tog histo
copyWithin	45 Toggle history	12 Toggle history	32 Toggle history	32 Toggle history	9 Toggle history	45 Toggle history	32 Toggle history	32 Toggle history	9 Toggle history	5.0 Toggle history	45 Tog histo
entries	38 Toggle history	12 Toggle history	28 Toggle history	25 Toggle history	8 Toggle history	38 Toggle history	28 Toggle history	25 Toggle history	8 Toggle history	3.0 Toggle history	38 Tog histo
every	1 Toggle history	12 Toggle history	1.5 Toggle history	9.5 Toggle history	3 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	37 Tog histo
fill	45 Toggle history	12 Toggle history	31 Toggle history	32 Toggle history	8 Toggle history	45 Toggle history	31 Toggle history	32 Toggle history	8 Toggle history	5.0 Toggle history	45 Tog histo
filter	1 Toggle history	12 Toggle history	1.5 Toggle history	9.5 Toggle history	3 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	37 Tog histo
find	45 Toggle history	12 Toggle history	25 Toggle history	32 Toggle history	8 Toggle history	45 Toggle history	4 Toggle history	32 Toggle history	8 Toggle history	5.0 Toggle history	45 Tog histo

	desktop					mobile					
	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebV Andr
[findIndex]()	45 Toggle history	12 Toggle history	25 Toggle history	32 Toggle history	8 Toggle history	45 Toggle history	4 Toggle history	32 Toggle history	8 Toggle history	5.0 Toggle history	45 Togg histo
[findLast]()	97 Toggle history	97 Toggle history	104 Toggle history	83 Toggle history	15.4 Toggle history	97 Toggle history	104 Toggle history	68 Toggle history	15.4 Toggle history	18.0 Toggle history	97 Tog histo
[findLastIndex]()	97 Toggle history	97 Toggle history	104 Toggle history	83 Toggle history	15.4 Toggle history	97 Toggle history	104 Toggle history	68 Toggle history	15.4 Toggle history	18.0 Toggle history	97 Tog histo
[flat]()	69 Toggle history	79 Toggle history	62 Toggle history	56 Toggle history	12 Toggle history	69 Toggle history	62 Toggle history	48 Toggle history	12 Toggle history	10.0 Toggle history	69 Tog histo
[flatMap]()	69 Toggle history	79 Toggle history	62 Toggle history	56 Toggle history	12 Toggle history	69 Toggle history	62 Toggle history	48 Toggle history	12 Toggle history	10.0 Toggle history	69 Tog histo
[forEach]()	1 Toggle history	12 Toggle history	1.5 Toggle history	9.5 Toggle history	3 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	37 Tog histo
[from]()	45 Toggle history	12 Toggle history	32 Toggle history	32 Toggle history	9 Toggle history	45 Toggle history	32 Toggle history	32 Toggle history	9 Toggle history	5.0 Toggle history	45 Tog histo
[fromAsync]()	No Toggle history	No Toggle history	115 Toggle history	No Toggle history	16.4 Toggle history	No Toggle history	115 Toggle history	No Toggle history	16.4 Toggle history	No Toggle history	No Tog histo
[group]() [Experimental]()	No Toggle history	No Toggle history	98 disabled Toggle history	No Toggle history	16.4 Toggle history	No Toggle history	No Toggle history	No Toggle history	16.4 Toggle history	No Toggle history	No Tog histo
[groupToMap]() [Experimental]()	No Toggle history	No Toggle history	98 disabled Toggle history	No Toggle history	16.4 Toggle history	No Toggle history	No Toggle history	No Toggle history	16.4 Toggle history	No Toggle history	No Tog histo
[includes]()	47 Toggle history	14 Toggle history	43 Toggle history	34 Toggle history	9 Toggle history	47 Toggle history	43 Toggle history	34 Toggle history	9 Toggle history	5.0 Toggle history	47 Tog histo
[indexOf]()	1 Toggle history	12 Toggle history	1.5 Toggle history	9.5 Toggle history	3 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	37 Tog histo
[isArray]()	4	12	4	10.5	5	18	4	14	5	1.0	4.4

	desktop					mobile					
	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebV Andr
	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Togg histo
[join](#)	1 Toggle history	12 Toggle history	1 Toggle history	4 Toggle history	1 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	4.4 Togg histo
[keys](#)	38 Toggle history	12 Toggle history	28 Toggle history	25 Toggle history	8 Toggle history	38 Toggle history	28 Toggle history	25 Toggle history	8 Toggle history	3.0 Toggle history	38 Togg histo
[lastIndexOf](#)	1 Toggle history	12 Toggle history	1.5 Toggle history	9.5 Toggle history	3 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	37 Togg histo
[length](#)	1 Toggle history	12 Toggle history	1 Toggle history	4 Toggle history	1 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	37 Togg histo
[map](#)	1 Toggle history	12 Toggle history	1.5 Toggle history	9.5 Toggle history	3 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	37 Togg histo
[of](#)	45 Toggle history	12 Toggle history	25 Toggle history	26 Toggle history	9 Toggle history	39 Toggle history	25 Toggle history	26 Toggle history	9 Toggle history	4.0 Toggle history	39 Togg histo
[pop](#)	1 Toggle history	12 Toggle history	1 Toggle history	4 Toggle history	1 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	4.4 Togg histo
[push](#)	1 Toggle history	12 Toggle history	1 Toggle history	4 Toggle history	1 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	4.4 Togg histo
[reduce](#)	3 Toggle history	12 Toggle history	3 Toggle history	10.5 Toggle history	4 Toggle history	18 Toggle history	4 Toggle history	14 Toggle history	3.2 Toggle history	1.0 Toggle history	37 Togg histo
[reduceRight](#)	3 Toggle history	12 Toggle history	3 Toggle history	10.5 Toggle history	4 Toggle history	18 Toggle history	4 Toggle history	14 Toggle history	3.2 Toggle history	1.0 Toggle history	37 Togg histo
[reverse](#)	1 Toggle history	12 Toggle history	1 Toggle history	4 Toggle history	1 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	4.4 Togg histo
[shift](#)	1 Toggle history	12 Toggle history	1 Toggle history	4 Toggle history	1 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	4.4 Togg histo
[slice](#)	1 Toggle history	12 Toggle history	1 Toggle history	4 Toggle history	1 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	4.4 Togg histo
[some](#)	1	12	1.5	9.5	3	18	4	10.1	1	1.0	37

	desktop					mobile					
	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebV Andr
	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Togg histo
[sort]()	1 Toggle history	12 Toggle history	1 Toggle history	4 Toggle history	1 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	4.4 Tog histo
Stable sorting	70 Toggle history	79 Toggle history	3 Toggle history	57 Toggle history	10.1 Toggle history	70 Toggle history	4 Toggle history	49 Toggle history	10.3 Toggle history	10.0 Toggle history	70 Tog histo
[splice]()	1 Toggle history	12 Toggle history	1 Toggle history	4 Toggle history	1 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	4.4 Tog histo
[toLocaleString]()	1 Toggle history	12 Toggle history	1 Toggle history	4 Toggle history	1 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	37 Tog histo
`locales` parameter	24 Toggle history	79 Toggle history	52 Toggle history	15 Toggle history	7 Toggle history	25 Toggle history	56 Toggle history	14 Toggle history	7 Toggle history	2.0 Toggle history	4.4 Tog histo
`options` parameter	24 Toggle history	79 Toggle history	52 Toggle history	15 Toggle history	7 Toggle history	25 Toggle history	56 Toggle history	14 Toggle history	7 Toggle history	2.0 Toggle history	4.4 Tog histo
[toReversed]()	110 Toggle history	110 Toggle history	115 Toggle history	96 Toggle history	16 Toggle history	110 Toggle history	115 Toggle history	74 Toggle history	16 Toggle history	21.0 Toggle history	11 Tog histo
[toSorted]()	110 Toggle history	110 Toggle history	115 Toggle history	96 Toggle history	16 Toggle history	110 Toggle history	115 Toggle history	74 Toggle history	16 Toggle history	21.0 Toggle history	11 Tog histo
[toSpliced]()	110 Toggle history	110 Toggle history	115 Toggle history	96 Toggle history	16 Toggle history	110 Toggle history	115 Toggle history	74 Toggle history	16 Toggle history	21.0 Toggle history	11 Tog histo
[toString]()	1 Toggle history	12 Toggle history	1 Toggle history	4 Toggle history	1 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	37 Tog histo
[unshift]()	1 Toggle history	12 Toggle history	1 Toggle history	4 Toggle history	1 Toggle history	18 Toggle history	4 Toggle history	10.1 Toggle history	1 Toggle history	1.0 Toggle history	4.4 Tog histo
[values]()	66 Toggle history	14 Toggle history	60 Toggle history	53 Toggle history	9 Toggle history	66 Toggle history	60 Toggle history	47 Toggle history	9 Toggle history	9.0 Toggle history	66 Tog histo
[with]()	110	110	115	96	16	110	115	74	16	21.0	11

desktop					mobile					
Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebV Andr
Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Toggle history	Togg histo