# Tip 5: Create Flexible Collections with Arrays

In this tip, you'll learn how arrays maximize flexibility and give you a foundation for understanding all other collections.

**We'll cover the following**



In JavaScript, there used to be only two structures for collections of data: **arrays** and **objects**. That list is growing. Now there are *maps, sets, weakmaps, weaksets, objects, and arrays*.

## Working with arrays #

When choosing a collection, you have to ask yourself what you need to do with the information. If you need to manipulate it in any way (*add, remove, sort, filter, alter all members*), then arrays are often the best collection. And even when you don't use an array, you'll almost certainly use ideas that you'd associate with arrays.

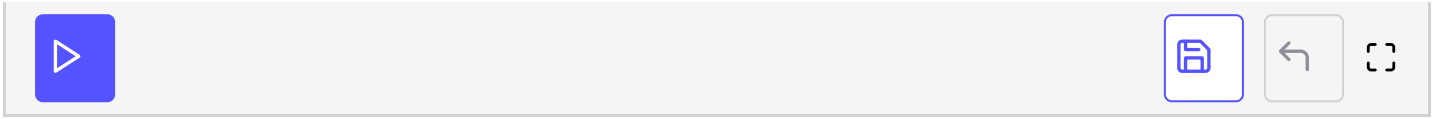
Arrays have a remarkable amount of *flexibility*. Because arrays preserve order, you can add and remove items according to their position or determine if they have a position at all. You can sort to give the array a new order as you'll see in Tip 9, Avoid Sort Confusion with the Spread Operator.

## Example: Using `sort` #

```
const team = [
    'Joe',
    'Dyan',
    'Bea',
    'Theo',
];
```

```
function alphabetizeTeam(team) {
    return [...team].sort();
}

console.log(alphabetizeTeam(team));
```



Interestingly, order is not technically guaranteed, but it's safe to assume that it will work in nearly all circumstances.

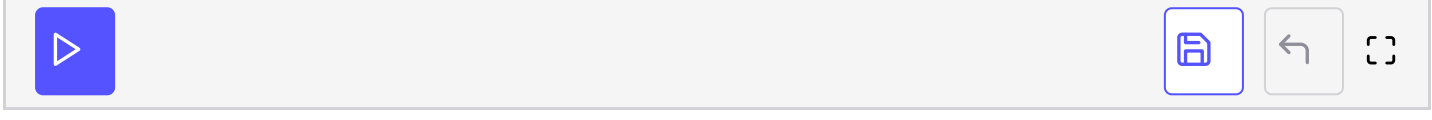
With array methods such as `map()`, `filter()`, and `reduce()`, you can alter or update the information easily with single lines, as you'll see starting with Tip 22, Create Arrays of a Similar Size with `map()`.

## Example: Using `filter` #

```
const staff = [
    {
        name: 'Wesley',
        position: 'musician',
    },
    {
        name: 'Davis',
        position: 'engineer',
    },
];

function getMusicians(staff) {
    return staff.filter(member => member.position === 'musician');
}

console.log(getMusicians(staff));
```



You may notice some strange looking syntax. Don't worry—you'll get to it soon. A lot of the new syntax in ES5 and ES6 is related to arrays. That should be a clue that they're valued highly in the JavaScript community.

Still, you'll need to use other collections. Yet, a solid understanding of arrays will greatly improve your code because arrays are at the heart of many popular data manipulations. For example, if you need to iterate over an object, the first thing you'd do is get the keys into an array with `Object.keys()` and then iterate over those. You're using an array as a bridge between the object and a loop.

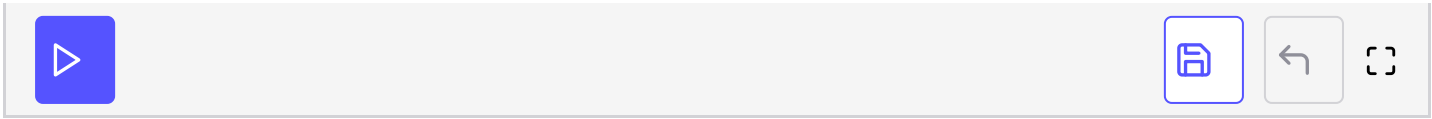
## Example: Iterating objects & using arrays #

```
const game1 = {
    player: 'Jim Jonas',
    hits: 2,
    runs: 1,
    errors: 0,
};

const game2 = {
    player: 'Jim Jonas',
    hits: 3,
    runs: 0,
    errors: 1,
};

const total = {};
const stats = Object.keys(game1);

for (let i = 0; i < stats.length; i++) {
    const stat = stats[i];
    if (stat !== 'player') {
        total[stat] = game1[stat] + game2[stat];
    }
}
console.log(total);
```



Arrays seem to pop up everywhere because they have a built-in iterable. An **iterable** is merely a way for the code to go through a collection one item at a time while knowing its current position. Any action you can perform on an array you can also perform on any data type that has an iterable (such as strings) or one that you can quickly transform into an iterable (as with `Object.keys()`).

If you know that you can create a new array with the spread operator, as you'll see in Tip 7, Mold Arrays with the Spread Operator, then you know that you can create a new `Map` with the spread operator because it also has a built-in iterable, as you'll see in Tip 14, Iterate Over Key-Value Data with Map and the Spread Operator.

## Converting to & from arrays #

Finally, you can express nearly every collection concept in the form of an array, which means you can easily convert from an array to a specialized collection and back again. Think about an object as a *key-value* store.

```
const dog = {
    name: 'Don',
```

```
    color: 'black',
};
console.log(dog.name);
```

You can describe that same concept, a key-value store, as an *array of arrays*. The internal arrays contain only two items.

- The first item is a *key*.
- The second item is the *value*.

This particular structure, an array consisting of two items, is also called a **pair**. Finding the value for a specific key is merely a matter of finding the pair with the correct key name and then returning the second item.

```
const dogPair = [
    ['name', 'Don'],
    ['color', 'black'],
];

function getName(dog) {
    return dog.find(attribute => {
        return attribute[0] === 'name';
    })[1];
}

console.log(getName(dogPair));
```

Admittedly, that's a lot of extra code for something so simple. You certainly wouldn't put this in a code base, but it's good to know that an object could be an array of pairs.

In fact, you'll use pairs to convert data between the `Map` object and an array. And now that the TC39 committee has finalized the spec to convert an object to an array of pairs using Object.entries(), you'll be able to use any array technique on objects with a quick conversion.


Having a deep understanding of arrays, and most iterables by proxy, will let you

grasp not only many of the new ES6 features that we're about to explore, but also many new features that are coming soon.

Study the code below. What does the following program do?

```
const people = [ { name: "John", age: 16 },
                 { name: "Thomas", age: 19 },
                 { name: "John", age: 20 },
                 { name: "Jess", age: 18 },];

const func = (collection) => {return collection.filter((person) =
> person.name == 'John');}
```

Retake Quiz

In the next tip, you'll begin to work with arrays by learning how testing existence in arrays has become even easier with includes().