Checkpoint 4

# Find, filter, map, some, and every

Now that you have some understanding of how to use `forEach()`, you are ready to learn other array methods that work in a similar fashion.

Outline

By the end of this checkpoint, you will be able to do the following:

- Use `find()`, `filter()`, `map()`, `some()`, and `every()` to solve various problems

## Overview

Start by watching the video below, which provides a brief introduction to this topic. Then, read through the rest of the checkpoint and complete the practice work required. This will give you a full understanding of these concepts.


Outline

# The `find()` method

In this checkpoint, you'll work with the `parks` data again.

```
const parks = [
  { name: "Biscayne", rating: 4.2 },
  { name: "Grand Canyon", rating: 5 },
  { name: "Gateway Arch", rating: 4.5 },
  { name: "Indiana Dunes", rating: 4.1 },
];
```

Sometimes, you'll want to write a loop to find an item in an array. Here's an example that uses a loop to find a park that has a specific na

```
let found = null;
for (let i = 0; i < parks.length; i++) {
  const park = parks[i];
  if (park.name === "Biscayne") found = park;
}
console.log(found); //> { name: "Biscayne", rating: 4
```

Once again, you can extract the logic from this loop into a function. Take a look at the function below. This function, `parkNameIsBiscayne()`, accepts a single `park` object and returns `true` if the name of the park is `"Biscayne"` and `false` otherwise. You can then call that function in the loop.

```
const parkNameIsBiscayne = (park) => park.name === "B

let found = null;
for (let i = 0; i < parks.length; i++) {
  const park = parks[i];
  if (parkNameIsBiscayne(park)) found = park;
}
console.log(found); //> { name: "Biscayne", rating: 4
```

Outline

Here, you are using the loop to visit each element in the array and perform a comparison. If the comparison is `true`, you select that item from the array. And if the comparison never returns `true`, you select nothing from the array.

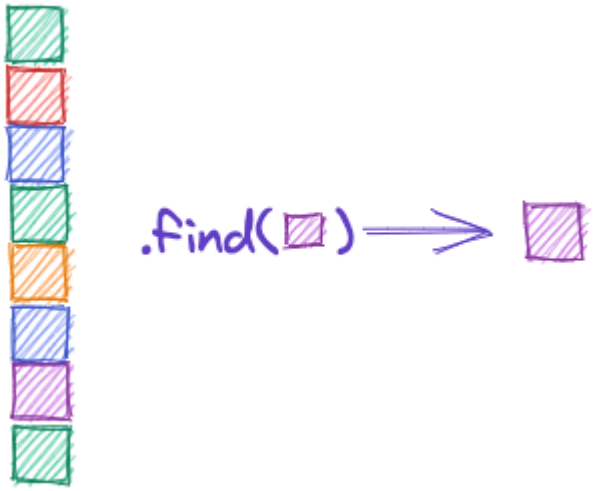
You could also use a built-in array method to do this; the `find()` method encapsulates this functionality. With `find()`, you can provide a callback function that implements the comparison that you wi[illegible] perform. Then, `find()` will apply this comparison to each element of the array.

Take a look at how you could perform the above task using the `find()` method:

```
let found = parks.find((park) => park.name === "Bisca
console.log(found); //> { name: "Biscayne", rating: 4
```

The `find()` method uses the callback function to decide whether each item matches, and it does the rest of the work from the loop. It returns the *first* item that matches the condition, even if more than one item matches. If there is no match, `find()` returns `undefined`.

Here's a quick summary of this method:




The `find()` method operates on an array of items and returns a single item.

## Do this

### Use `find()`

Using the above `parks` dataset, use the `find()` function to write code that finds the park with a rating of `4.1`.

Outline

# The `filter()` method

```
const parks = [
  { name: "Biscayne", rating: 4.2 },
  { name: "Grand Canyon", rating: 5 },
  { name: "Gateway Arch", rating: 4.5 },
  { name: "Indiana Dunes", rating: 4.1 },
];
```

Sometimes, you'll want to build up a new list of items that meet a particular condition. Take a look at some code that does that with a `for` loop:

```
let result = [];
for (let i = 0; i < parks.length; i++) {
  const park = parks[i];
  if (park.rating >= 4.5) result.push(park);
}
console.log(result); // [ { name: "Grand Canyon", rat
```

Outline

You can use the `filter()` method to achieve the same goal. This method builds a new array of only the items that match a condition. Take a look at the example below to see how this works. Once again, the comparison that was done in the body of the loop has been refactored into a callback function.

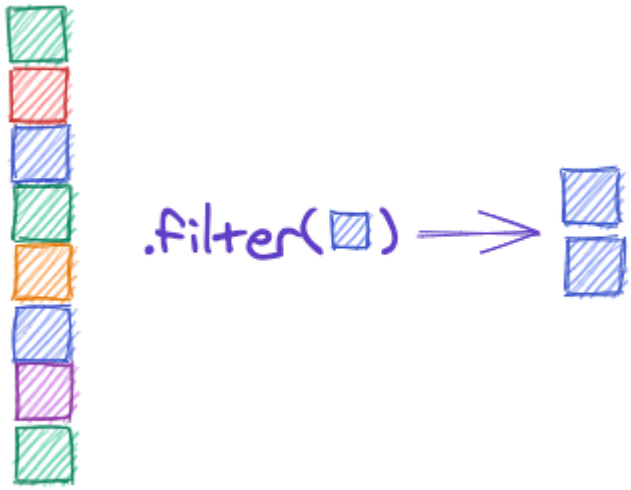
```
let result = parks.filter((park) => park.rating >= 4.
console.log(result); // [ { name: "Grand Canyon", rat
```

This method filters the array so that you have only some of the [illegible]s. It's pretty similar to `find()`, except that it returns an array of all the

matching items, instead of just the first match.

Keep in mind that `filter()` returns a new array. The old array will still have the same items. It is also possible for the returned array to be empty, if no element matches the criterion that's given in the function.

Here's a quick summary of this method:




Outline

The `filter()` method operates on an array of items and returns a new array of items that match the criterion.

## Do this

### Use `filter()`

Using the above `parks` dataset, write code using the `filter()` function to find the parks that begin with the letter `"G"`.

# The `map()` method

The loop below creates a new array that just contains the names of all of the parks.

```
const parks = [
  { name: "Biscayne", rating: 4.2 },
  { name: "Grand Canyon", rating: 5 },
  { name: "Gateway Arch", rating: 4.5 },
  { name: "Indiana Dunes", rating: 4.1 },
];

const result = [];
for (let i = 0; i < parks.length; i++) {
  result.push(parks[i].name);
}
console.log(result); // [ "Biscayne", "Grand Canyon",
```

Basically, this code processes each item in the array and creates a new value for each item in the original array. Each item in the original array *maps* to an item in the new array.

You can achieve this same goal with `map()`, as follows:

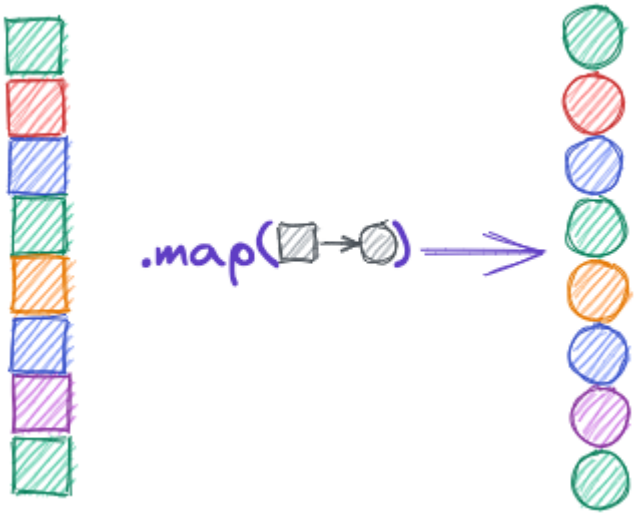


```
const result = parks.map((park) => park.name);
console.log(result); // [ "Biscayne", "Grand Canyon",
```

Just like the other array methods that you've been looking at, `map()` will run the callback function for each of the items in the array. The `map()` method uses the callback function to create the items for the new array. In this case, the code adds the `park.name` for each park to the new array.

The `map()` method works in lots of situations—not just ones where you want to pick one value from each of a list of objects. This method works in any situation where you want to transform each value in an array into another value.

Here's a quick summary of this method:




The `map()` method operates on an array of items and creates a new array of items the same size as the original array.

## Do this

Outline

### Use `map()`

Using the above `parks` dataset, use the `map()` method to write code that returns an array of strings, where each string is the name of the park next to its rating. The end result should look something like this:

```
["Biscayne: 4.2", "Grand Canyon: 5", "Gateway Arch: 4
```

# The `some()` method

Sometimes, you'll just want to check if some condition is met in your array. The following code checks whether or not any of the par ve a rating of greater than `4`.

```
const parks = [
  { name: "Biscayne", rating: 4.2 },
  { name: "Grand Canyon", rating: 5 },
  { name: "Gateway Arch", rating: 4.5 },
  { name: "Indiana Dunes", rating: 4.1 },
];

let result = false;
for (let i = 0; i < parks.length; i++) {
  if (parks[i].rating > 4) result = true;
}
console.log(result); // true
```

You can achieve this same goal with `some()`, like this:

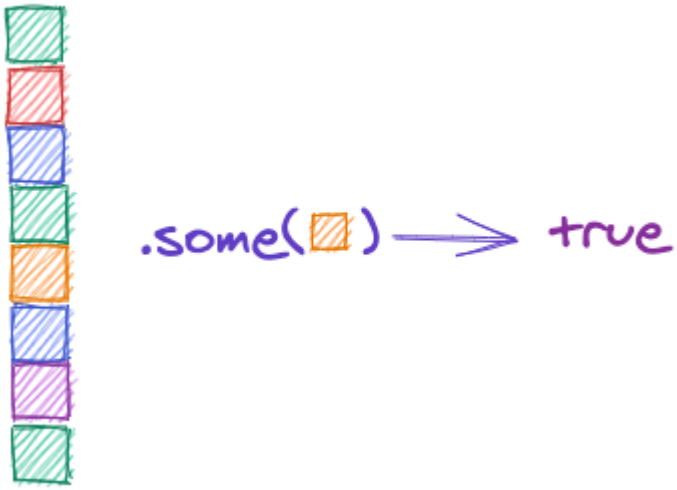
```
const result = parks.some((park) => park.rating > 4);
console.log(result); // true
```

Outline

The `some()` method accepts a callback function that implements a comparison that is executed for each item in the array, similar to the previous methods. If the callback function returns `true` for any item in the array, then the entire `some()` method returns `true`.

This method is more efficient than the `for` loop shown above, in that it returns immediately as soon as the condition is met. The `some()` method is useful for quick checks like this, and it's different from the other methods in this checkpoint in that it returns a boolean value instead of an array.

Here's a quick summary of this method:




The `some()` method operates on an array of items and returns a boolean value.

## Do this

### Use `some()`

Using the above `parks` dataset, use the `some()` function to write code that returns whether or not the `"Grand Arches"` park is included in the array.

# The `every()` method

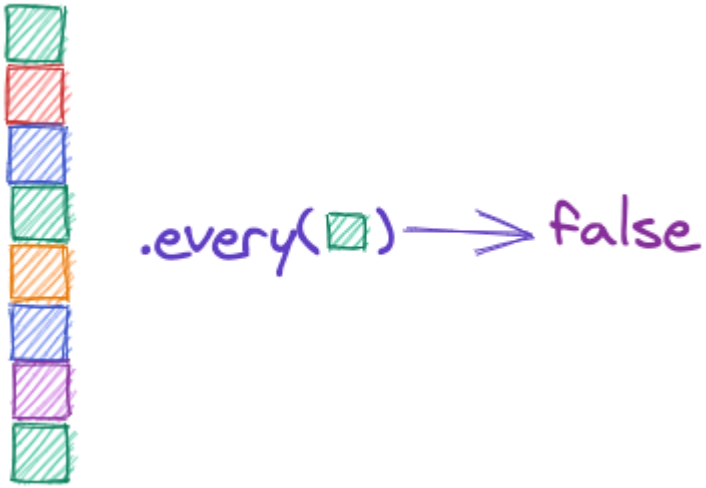
At times, you may want to check whether *every* item in an array matches some condition. The `every()` method has you covered. Take a look:

```
const parks = [
  { name: "Biscayne", rating: 4.2 },
  { name: "Grand Canyon", rating: 5 },
  { name: "Gateway Arch", rating: 4.5 },
  { name: "Indiana Dunes", rating: 4.1 },
];

const result = parks.every((park) => park.rating > 4)
console.log(result); // true
```

The `every()` method works by checking the condition given against every item in the array. If that condition ever fails, it will return `false`. Otherwise, it will return `true`. Once again, the comparator is implemented in the callback function passed to the method.

Here's a quick summary of this method:




Outline

The `every()` method operates on an array of items and returns a boolean value.

## Do this

### Use `every()`

Using the above `parks` dataset, use the `every()` function to write code that returns whether or not every park has a rating between `4.2` and `4.6`.

# Iteration over objects

Another way that you could store the `parks` data is with the following object:

```
let parks = {
  "Biscayne": 4.2,
  "Grand Canyon": 5,
  "Gateway Arch": 4.5,
  "Indiana Dunes": 4.1,
};
```

How might you use the methods that you've learned with an object? If you want to get all the keys as an array, you can use the built-in JavaScript method `Object.keys()`, like this:

```
Object.keys(parks); // => ["Biscayne", "Grand Canyon
```

With the keys as an array, you can use the array methods that you've learned, like this:

Outline

```
Object.keys(parks).filter((name) => {
  const rating = parks[name];
  return rating >= 4.5;
}); // => ["Grand Canyon", "Gateway Arch"]
```

**Note:** There's also a method to get the values from an object. To learn more, check out the MDN documentation on Object.values().

# Don't forget to return!

All of these methods require that you return some value inside the callback function. Not doing so could have unintended consequences, as demonstrated here:

```
const parks = [
  { name: "Biscayne", rating: 4.2 },
  { name: "Grand Canyon", rating: 5 },
  { name: "Gateway Arch", rating: 4.5 },
  { name: "Indiana Dunes", rating: 4.1 },
];

const result = parks.map((park) => {
  park.name;
});
console.log(result); // [ undefined, undefined, undef
```

In the above code, because `park.name` is not being returned, the inner function's return value is `undefined`. This means that `undefined` will take the place of each item.

So if you are seeing strange results, make sure that you are returning within the callback function.

Outline

---

# Checkpoint

This checkpoint will be autograded. Please click the link below to open your assignment in a new tab. Once you complete the assignment, you will see a button allowing you to submit your answers and move on to the next checkpoint.

## Your work

**04.01.21**

How would you rate this content?

Report a typo or other issue

Go to Overview