Outline

Checkpoint 6

Sort

Sorting is often a starting step when you're solving other problems. But sorting can be a deep and difficult topic on its own. Until you're ready to dive into building your own sort function, you can use JavaScript's built-in `sort()` method.

By the end of this checkpoint, you will be able to use `sort()` to sort arrays in various ways.

Basics of `sort()`

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 `sort()` method takes a function and uses that function to sort an array based on the given criteria. For example, take a look at the function below. This function sorts the parks based on the rating of each park.

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

parks.sort((parkA, parkB) => (parkA.rating > parkB.ra
console.log(parks);
```

In the above code, you can see that the function passed in to `sort()` has two parameters. Each parameter, on the first iteration, represents the first and second element of the array. The ratings are then compared for each park.

If the number returned is negative, the first item (`parkA`) will be moved before the second item (`parkB`). The opposite is true if the number is positive. This iteration then continues, but with the second and third items.

If `0` is returned, the items won't change places.

You may also see code that follows the format shown below, particularly when you're sorting based on a number. The effect of this code is similar to that of the previous code sample.

Outline

```
parks.sort((parkA, parkB) => parkA.rating - parkB.rai
console.log(parks);
```

Sorting strings

You can compare strings with the greater-than `>` and less-than `<` symbols in JavaScript.

```
"Biscayne" < "Grand Canyon"; //> true
```

The above code returns `true` because the letter *B* comes before the letter *G*. However, it isn't quite as simple as it seems. For examp ook at this code:

```
"biscayne" < "Grand Canyon"; //> false
```

Strings do not get compared alphabetically; instead, they're compared based on their character value. This means that sorting strings can be a bit more unreliable than you might expect. Still, with some ingenuity, it is possible to do a *passable* job of sorting strings. For example, you could modify your function as follows:

```
parks.sort((parkA, parkB) =>
  parkA.name.toLowerCase() > parkB.name.toLowerCase(
);
console.log(parks);
```

Outline

Caveats for using `sort()`

Be careful with `sort()`. It has several peculiarities that can lead to bugs. Here are a few to watch out for:

- It expects you to return a negative number for items that should be earlier in the list, and a positive number for items that should be later in the list.
- It changes the array in place. In other words, it doesn't return a new array—it mutates the existing array. The `parks` dataset changed its order in the above example.
- It has a default behavior if you don't pass in a function, but this default behavior might not be what you want.

For more insight into the default behavior, take a look at the following code.

```
["Biscayne", "grand canyon", "Gateway arch"].sort();
//> [ 'Biscayne', 'Gateway arch', 'grand canyon' ]
```

This *could* be what you want, but it is almost always better to provide a function so that you can determine the effect of `sort()`.

Checkpoint

Outline

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

04.05.21 **Approved**

Completed **Next checkpoint**

How would you rate this content?

Report a typo or other issue

Go to Overview

Outline