Checkpoint 5

Reduce

You've seen several powerful array methods so far. In this checkpoint, you'll learn another: `reduce()`. As you'll see, `reduce()` is a tool that generalizes the accumulator pattern that you learned earlier in this program.

Outline

By the end of this checkpoint, you will be able to use `reduce()` to solve different problems.

The `reduce()` method 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 basics of `reduce()`

Earlier in this program, you learned about the *accumulator pattern*. In this pattern, you use a loop to build up a new value. Each step of the loop gets one item from the array and combines it with an accumulator, like a running total.

Take a look at an example with a `for` loop:

```
const areas = [768, 1004.2, 433.1];
let result = 0;
for (let i = 0; i < areas.length; i++) {
  result += areas[i];
}
console.log(result); //> 2205.3
```

The `reduce()` method is another way to express the accumulator pattern. This method turns an array of values into a single value. Like many of the other methods here, it will pass each value into a function, step by step. The `reduce()` method will also pass in the return value from the previous step.

For example, take a look at the following code. This code uses `reduce()` to achieve the same results as the code above.

```
const areas = [768, 1004.2, 433.1];
let accumulator = 0;
let result = areas.reduce((acc, area) => acc + area,
console.log(result); //> 2205.3
```

The `reduce()` method takes two arguments: a function and an optional starting value. The function passed into the `reduce()` function is similar to the other functions that you've seen, except that it includes an additional parameter: the accumulator. The accumulator parameter represents the following:

- On the first iteration, the accumulator value (`acc` in the above function) is set to the second parameter (`0` in the above function).
- On every subsequent iteration, the accumulator value is set to whatever was returned from the previous iteration.

So, the above code works as follows:

1. The `areas` and `accumulator` values are defined.

2. The `reduce()` method takes a function that adds the accumulator and the current element. The first iteration will add `0` and `768`.
3. The result of the first iteration will *become* the accumulator in the next iteration.
4. Once all iterations are finished, the value is stored in the `result` variable.
5. The `result` is logged out.

Do this

Add logging for understanding

The `reduce()` method can be difficult to understand. Try running the following code, and look at the logged statements.

Outline

```
const areas = [768, 1004.2, 433.1];
let accumulator = 0;
let result = areas.reduce((acc, area, index) => {
  console.log(`index: ${index}`, `acc: ${acc}`, `area
  return acc + area;
}, accumulator);
console.log(result); //> 2205.3
```

You should see something like this:

```
index: 0 acc: 0 area: 768
index: 1 acc: 768 area: 1004.2
index: 2 acc: 1772.2 area: 433.1
2205.3
```

As you can see above, the value of `area` is added to `acc` at each step.

Remove the initial value

The second argument in `reduce()` is optional. So, what happens if it is removed? Try running the following code to find out.

```
const areas = [768, 1004.2, 433.1];
let result = areas.reduce((acc, area, index) => {
  console.log(`index: ${index}`, `acc: ${acc}`, `area
  return acc + area;
});
console.log(result); //> 2205.3
```

You should see something like this:

```
index: 1 acc: 768 area: 1004.2
index: 2 acc: 1772.2 area: 433.1
2205.3
```

Notice that for the first iteration, `acc` is set to the first value of the `areas` array, and `area` is set to the second value.

Complex usage

The initial value that `reduce()` uses can be *any* data type, just like how the accumulator could be anything when you're using the accumulator pattern. This allows for some pretty powerful but complex usage of the `reduce()` method.

For example, take a look at the code below. This code accumulates array values into an object.

```
const parks = [
  { name: "Acadia", areaInSquareKm: 198.6 },
  { name: "Crater Lake", areaInSquareKm: 741.5 },
  { name: "Kenai Fjords", areaInSquareKm: 2710 },
  { name: "Zion", areaInSquareKm: 595.9 },
];

const result = parks.reduce((acc, park) => {
  acc[park.name] = park.areaInSquareKm;
  return acc;
}, {});
```

Running the code above produces the following value for the `result` variable:

Outline

```
{
  Acadia: 198.6,
  'Crater Lake': 741.5,
  'Kenai Fjords': 2710,
  Zion: 595.9
}
```

At each step in the function given to `reduce()`, a new key is created in the given object. The value assigned to that key is the `areaInSquareKm` value. Then, the overall object is returned so that it can serve as the accumulator (`acc`) in the next iteration.

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 **Approved**

Completed Next checkpoint

Outline

How would you rate this content?

Report a typo or other issue

Go to Overview