Checkpoint 7

Destructuring and rest

In this program and throughout your career, you'll learn about important principles and best practices to guide your development work. But here's an important one to keep in mind: aim to make your code as clear and concise as possible. And in this checkpoint, you'll learn about some strategies to do just that. Specifically, you will explore two tools, destructuring and the rest operator, that will help you write modern JavaScript that is clearer and more concise. These tools will be valuable additions to your toolkit.

Outline

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

- Use the destructuring assignment syntax to write more concise code
- Use the rest operator to assign multiple values to a new array

Destructuring objects

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

Now, take a moment to examine the object in the code sample below.

```
const product = {
  title: "The Golden Compass",
  priceInCents: 799,
  author: {
    firstName: "Philip",
    surname: "Pullman",
  },
};
```

When working with an object, you'll often need to use some, but not all, of the available keys. Consider this example:

```
function printAuthorAndTitle(product) {
  return `${product.title} by ${product.author.firstN
}
```

The above function works just fine. However, the return statement ends up being a bit long because you have to repeat the `product` variable multiple times. In this case, you might update the above function to the one below, hoping to make it easier to read:

```
function printAuthorAndTitle(product) {
  const author = product.author;
  const title = product.title;

  return `${title} by ${author.firstName} ${author.su
}
```

Outline

The second function is more legible than the first. But it still looks clunky; after all, the `product` variable continues to be repeated multiple times. Fortunately, there's another option: *destructuring*. When you destructure an object or array, you're unpacking the properties or values and assigning them into distinct variables.

Take a look at the following version of the function above, which makes use of destructuring. What do you not


```
function printAuthorAndTitle(product) {
  const { author, title } = product;
```

```
  return `${title} by ${author.firstName} ${author.su
}
```

In this example, two new `const` variables are being created: `author` and `title`. It's important to notice how these new variables are wrapped in curly brackets `{}` *before* the `=` sign. This is the destructuring syntax. Those variables are set to whatever keys can be found inside of the `product` object. If a key is not found, it will be set to `undefined`. Take a look:

```
const { yearPublished } = product;
console.log(yearPublished); //> undefined
```

And what's more, you can destructure multiple levels into an object. Check it out.

```
function printAuthorAndTitle(product) {
  const {
    author: { firstName, surname },
    title,
  } = product;
  return `${title} by ${firstName} ${surname}`;
}
```

In the above function, the `author` key, which also points to an object, is further destructured to access the `firstName` and `surname` keys. Note that there is *no* `author` *variabl* variables that are created are `firstName`, `surname`, and `title`.

Do this

Destructure an object

Take a look at the following object. Practice destructuring all of the keys from this object.

```
const author = {
  name: {
    firstName: "Philip",
    surname: "Pullman",
  },
  birthday: "1946-10-19",
};
```

Missing keys

Review the following code. What do you expect will happen? Make a prediction, and then run the code for yourself.

Outline

```
const author = {
  name: {
    firstName: "Philip",
    surname: "Pullman",
  },
  birthday: "1946-10-19",
};

const { firstName } = author;
console.log(firstName);
```

As you can see, `firstName` ends up

there is no key.

Deep destructuring

Examine the code sample below. What do you expect will happen? Make a prediction, and then run the code for yourself.

```
const author = {
  name: {
    firstName: "Philip",
    surname: "Pullman",
  },
  birthday: "1946-10-19",
};

const {
  name: { firstName },
} = author;
console.log(firstName);
console.log(name);
```

In this case, `firstName` will be printed out as `"Philip"`. But trying to print `name` will cause a `ReferenceError`.

Outline

When you use the syntax above to do *deep destructuring*, you're able to dive into and unpack multiple levels of an object. However, you will *not* have access to every value along the way. Furthermore, deep destructuring can cause some issues if the key is missing.

For example, take a look at the code below.

```
const author = {
  birthday: "1946-10-19",
};

const {
  name: { firstName },
} = author;
```

If you run this code, you will receive the following error:

```
Uncaught TypeError: Cannot read property 'firstName'
```

If you use deep destructuring, you'll need to make sure that the inputted object or array is of the right shape. This will help prevent errors like this one.

Destructuring arrays

As you know, objects are accessed by their keys. Arrays, on the other hand, are accessed by their index. In the above example, you used object keys to destructure objects and create variables. The process for arrays is similar, except that you'll use those specific positions in the array to create the variables. To begin, check out the array below:

Outline

```
const genres = [
  "Fantasy",
  "Fiction",
  "Nonfiction",
  "Science Fiction",
  "Young Adult",
];
```

You can destructure this array by doing the following:

```
const [first, second] = gen
console.log(first); //> 'Fa
console.log(second); //> 'Fiction'
```

In the above example, two new variables are created: `first` and `second`. Those variable names are set to the elements at index `0` and index `1`. The rest of the elements are ignored.

Do this

Destructure an array

Take a look at the following array. Practice destructuring the first, second, and third values from this array.

```
const authors = [
  "Ursula K. Le Guin",
  "Brandon Sanderson",
  "Terry Pratchett",
  "Neil Gaiman",
  "J. R. R. Tolkien",
];
```

Outline

The rest operator

As you've seen, destructuring is an extremely valuable tool. But that being said, destructuring an array on its own is only so useful. It's much more useful when it pairs up with another tool: the *rest* operator.

Take a look at the code sample below. What do you notice?

```
const [first, second, ...otherGenres] = genres;
console.log(first); //> 'Fa
console.log(second); //> 'F
console.log(otherGenres); //> [ 'Nonfiction', 'Scienc
```

In this example, there are two variables, `first` and `second`. Those are followed by the syntax that makes up the rest operator: the three periods `...` and a variable name, which in this case is `otherGenres`. The variable that follows `...` will contain all of the remaining array elements that were not destructured. This can be very useful for splitting apart an array.

Destructuring parameters

You can also use destructuring in functions in order to destructure the parameters. Take a look at this example:

```
function printAuthorAndTitle({ author, title }) {
  return `${title} by ${author.firstName} ${author.su
}

printAuthorAndTitle(product); //> 'The Golden Compass
```


The above syntax, once understood, is useful for at least two reasons:

1. You know that the expected input into the function is an object.
2. The function is concise and easy to read.

However, there is a downside to this approach: if you need to access the entire inputted object, you have no way to do so.

The assignment below includes a quiz to test your knowledge. Here is the answer key for you to verify.

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

03.25.21 **Approved**

Completed Next checkpoint

Outline

How would you rate this content?

Report a typo or other issue

Go to Overview

Feeling stuck?
Chat live with an expert now. Beta