Checkpoint 12

Writing readable code

At this point, you've learned several important JavaScript concepts. You're familiar with some essential JavaScript syntax, statements, and code-writing tools. And now, it's time to take those skills and refine them.

Outline

As you know, developers should aim to write code that is clear, clean, and easy to read. And this checkpoint is all about helping you do that. Of course, there are different approaches and techniques, and the advice provided in this checkpoint reflects that subjectivity. None of the techniques included here are required. However, they will improve the readability of your code, and using them will show other developers (and potential employers) that you're thoughtful, careful, and professional.

For more advice on how to write code well, speak with experts in the field. Many developers have strong op

Feeling stuck?
Chat live with an expert now. Beta

is more legible and maintainable. Gathering information from different experts will help inform your preferences and practices.

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

- Refactor code to be more readable and more efficient

Don't repeat yourself

One important rule that you will often hear as a developer is not to repeat yourself. In fact, *don't repeat yourself* is often abbreviated to *DRY*, and the whole idea is often captured in the following advice: *write DRY code*. Generally, this idea is used specifically in the context of functions. Because functions allow you to wrap up repeated code within a function, they can be particularly helpful when it comes to writing DRY code.

However, this is only one way to think about the phrase "don't repeat yourself." Take a look at the following data and function. Spend a few moments reviewing it to make sure you understand what is happening. What do you notice? Then, try running the function yourself.

```
const authors = [
  {
    id: 1,
    name: {
      firstName: "Philip",
      surname: "Pullman",
    },
    series: ["His Dark Materials", "Sally Lockhart"],
  },
  {
    id: 2,
    name: {
      firstName: "Terry",
      lastName: "Pratchett",
    },
    series: ["Discworld", "Long Earth"],
  },
```

Feeling stuck?
Chat live with an expert now. Beta

```
];

function getAllSeries(authors) {
  const result = [];
  for (let i = 0; i < authors.length; i++) {
    for (let j = 0; j < authors[i].series.length; j++
      result.push(authors[i].series[j]);
    }
  }
  return result;
}

getAllSeries(authors);
//> [ 'His Dark Materials', 'Sally Lockhart', 'Discw
```

In the above function, there is some duplicate code. Can you see it? The code shows `authors[i]` multiple times.

But maybe that can be tightened up. Instead, you could assign this value to the variable. Check it out:

Outline

```
function getAllSeries(authors) {
  const result = [];
  for (let i = 0; i < authors.length; i++) {
    const author = authors[i];
    for (let j = 0; j < author.series.length; j++) {
      result.push(author.series[j]);
    }
  }
  return result;
}
```

Feeling stuck?
Chat live with an expert now. Beta

Although you've technically made the
now much clearer. It also avoids using `authors[i]` multiple ti .

Return early

Another important rule is to *return early*. At its most basic level, the return early mindset involves writing a function that terminates or throws an error as soon as something is wrong, with the goal of yielding the correct result—the one that you'd expect—at the end of the function.

To better understand this idea, take a look at the following function. This code sample expects inputted data that is similar to that `authors` array from above.

```
function getSeriesListById(authors, id) {
  let selected = null;
  for (let i = 0; i < authors.length; i++) {
    const author = authors[i];
    if (author.id === id) selected = author;
  }

  if (id) {
    if (selected) {
      const message = `Series list: ${selected.series
      return message;
    } else {
      return [];
    }
  } else {
    return "No ID provided.";
  }
}
```

In the above function, the `authors` array and an `id` are given to the function. If the `id` matches one of the `authors`, the code returns the series for that author. Otherwise, i[illegible] is inputted, it returns a string at the end saying so: `No ID provided`.

The above code isn't that complicated. But it can be made simp[illegible] by returning early. Check it out:

```
function getSeriesListById(authors, id) {
  if (!id) return "No ID provided.";

  let selected = null;
  for (let i = 0; i < authors.length; i++) {
    const author = authors[i];
    if (author.id === id) selected = author;
  }
  if (!selected) return [];

  const message = `Series list: ${selected.series.joi
  return message;
}
```

Outline

In this modified example, the function stops if there is no `id` inputted rather than running *despite* the lack of `id`. In fact, when the function stops, the same string from above, `No ID provided`, is returned. But you'll notice that this happens near the beginning of the function, rather than at the end. This means that the code below that point does not have to run, and it is letting you know as soon as possible that there's no `id` inputted.

This is sometimes referred to as a guard clause. A *guard clause* is a statement that evaluates to a boolean that determines whether or not a function should continue running. Implementing guard clauses in your code will make your code much more efficient and easier to read.

Avoid boolean returns

One final rule is to *avoid boolean retu*

Feeling stuck?
Chat live with an expert now. Beta

isn't *always* possible, you can often avoid explicitly returning `true` and `false` by returning the expression that is evaluating the state .

For example, take a look at the following function. What do you notice?

```
function moreThanThreeAuthors(authors) {
  if (authors.length > 3) {
    return true;
  } else {
    return false;
  }
}
```

The function above just checks if there are more than three authors in the given array. But you can actually write a stronger, shorter function with the following:

```
function moreThanThreeAuthors(authors) {
  return authors.length > 3;
}
```

Outline

The conditional statement will already be evaluated to a boolean, so you don't need to explicitly return `true` or `false`.

Checkpoint

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

Feeling stuck?
Chat live with an expert now. Beta

Your work

03.29.21 Approved

Completed

Next checkpoint

How would you rate this content?

Report a typo or other issue

Outline

Go to Overview

Feeling stuck?
Chat live with an expert now. Beta