Checkpoint 3

Variables

In your earlier work with JavaScript, you practiced the basics of declaring variables with the `let` keyword. But that was just the beginning! In this checkpoint, you'll learn about two other methods of declaring variables, and you'll explore when you should use each approach.

Outline

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

- Recognize the differences between `let`, `const`, and `var`, and use each method appropriately

The `let` keyword

As you've learned, the `let` keyword is used to declare variables in JavaScript. Here's some refresher code for your review:

```
let price = 9.99;
```

You can also reassign variables that have already been created using `let`. Here's an example:

```
let price = 9.99;
price = 8.5;
```

However, if you declare a variable with `let`, you *cannot* use `let` to later define that same variable. See the error below:

```
let price = 9.99;
let price = 8.5; //> Uncaught SyntaxError: Identifie
```

This error is pretty useful. It will stop you from accidentally recreating the same variable later on in your program when you don't intend to do so.

Outline

But here's something that might surprise you: `let` is actually a fairly new feature of JavaScript. Now, you'll take a step back to learn about two other variable-declaring features: `var` and `const`.

The `var` keyword

In the past, variables were declared with the keyword `var`. Here it is in action:

```
var productName = "Jogger Sweatpants";
```

Just like with `let`, variables created with the `var` keyword ca e reassigned. However, `var` does not have the same restrictions as `let` when it comes to declaring variables with the same name. See the example below:

```
var productName = "Jogger Sweatpants";
productName = "Men's Jogger Sweatpants";
var productName = "Woman's Jogger Sweatpants";
```

Because `var` is an older method of declaring variables, you will often see `var` referenced in older code. However, you *should not* use `var` in your code unless you have a very specific reason to do so. This checkpoint aims only to briefly introduce you to the `var` concept and how it works. Later in this program, you will learn more about when, where, and why you might use `var` in your work.

The `const` keyword

At first blush, the `const` keyword may seem very similar to `let`. Using `const`, you can declare variables much like you would with `let`. And just like `let`, you also can't use it to create another variable of the same name.

```
const size = "M";
const size = "L"; //> Uncaught SyntaxError: Identifie
```

However, with `const`, you also can't reassign the value. See below:

```
const size = "M";
size = "S"; //> Uncaught TypeError: Assignment      co
```

Ultimately, there is no performance benefit to using one method over another— `const`, `let`, and `var` will each allow you to declare a variable, which is what you want to do. But there is a workflow benefit. The intended purpose of both `const` and `let` is to help *developers make fewer mistakes*.

By allowing developers to create variables that cannot be declared again, `const` and `let` act a bit like a safety net. As you develop larger applications, you will find it challenging to create concise, descriptive, and unique variables names. Often, developers will attempt to use a variable name in their code, forgetting that they've already used that variable name higher up. That's where `const` and `let` can help. They prevent developers from making unintended mistakes when declaring their variables.

Outline

Warning: Using `const` with arrays and objects

But `const` also has some other traits that are worth discussing. And when it comes to arrays and objects, using `const` can be a bit tricky.

Take a look at the following code. It works as you may expect:

```
const product = { priceInCents: 2100, name: "Yellow E

product = { priceInCents: 2100, name: "Red Beanie", s
//> Uncaught TypeError: Assignment to constant varial
```

In the above case, you are not able to reassign the value of `const`, as intended. The following code, on the other hand, *does* run.

```
const product = { priceInCents: 2100, name: "Yellow E
product.name = "Red Beanie";
product.size = "L";
```

Although you can't reassign the variable completely with `const`, you *can* change the values inside of the object. The same goes for arrays. Take a look:

```
const sizes = [8, 10, 12, 14];
sizes[4] = 16;
```

The reasons behind this behavior will be covered in a later checkpoint. For now, you're just getting familiar with the way `const` works.

Outline

What to use?

At this point, you might be asking this important question: *Which method of declaring variables should I use?* Here's the process that you should follow when determining whether to use `let`, `const`, or `var`:

1. Use `const` primarily; this will be your go-to. Moving forward, you'll want to declare most of your variables using `const`.
2. Use `let` if you need to reassign a value. This is a common requirement during `for` loops and sometimes with `if` statements.
3. Don't use `var` unless necessary, like when working in a c base that uses it. However, that would likely only happen in the distant

future. As mentioned above, there are other reasons to use, and not to use, `var`, but those will be covered in a different checkpoint. For now, just avoid it.

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

Outline

03.22.21 **Approved**

Completed Next checkpoint

How would you rate this content?

Report a typo or other issue

Go to Overview

Outline