Intro to Vue 3

Lessons

1. Intro to Vue 3

2:22

2. Creating the Vue App

6:56

3. Attribute Binding

3:53

4. Conditional Rendering

5:11

5. List Rendering

3:31

6. Event Handling

4:31

7. Class & Style Binding

6:37

8. Computed Properties

6:23

9. Components & Props

6:38

10. Communicating Events

3:57

11. Forms & v-model

8:27

Class & Style Binding

In this lesson, we're going to look at the concept of class and style binding. If you're coding along with the repo, you can checkout the `L7-start` branch or the starting code on Codepen.

Our Goal

Bind classes and styles to our elements based on our app's data.

Style Binding

In the last lesson, we added the feature where if you hover over "green" or "blue", you update the image that is being displayed; the green or blue socks, respectively. But wouldn't the user experience be nicer if instead of hovering over the *word "*green" or "blue", we hovered over the actual *colors* green and blue?

Let's create green and blue circles that we can hover on. We can achieve this by using style binding.

First, to style our divs like circles, we'll we need to add a new class `.color-circle` to the variant div.

index.html

```
<div
  v-for="variant in variants"
  :key="variant.id"
  @mouseover="updateImage(variant.image)"
  class="color-circle"
</div>
```

This class already lives in our css file. As you can see, it simply transform our divs into a circle with a 50px diameter:

📄styles.css

```
.color-circle {
  width: 50px;
  height: 50px;
  margin-top: 8px;
  border: 2px solid #d8d8d8;
  border-radius: 50%;
}
```

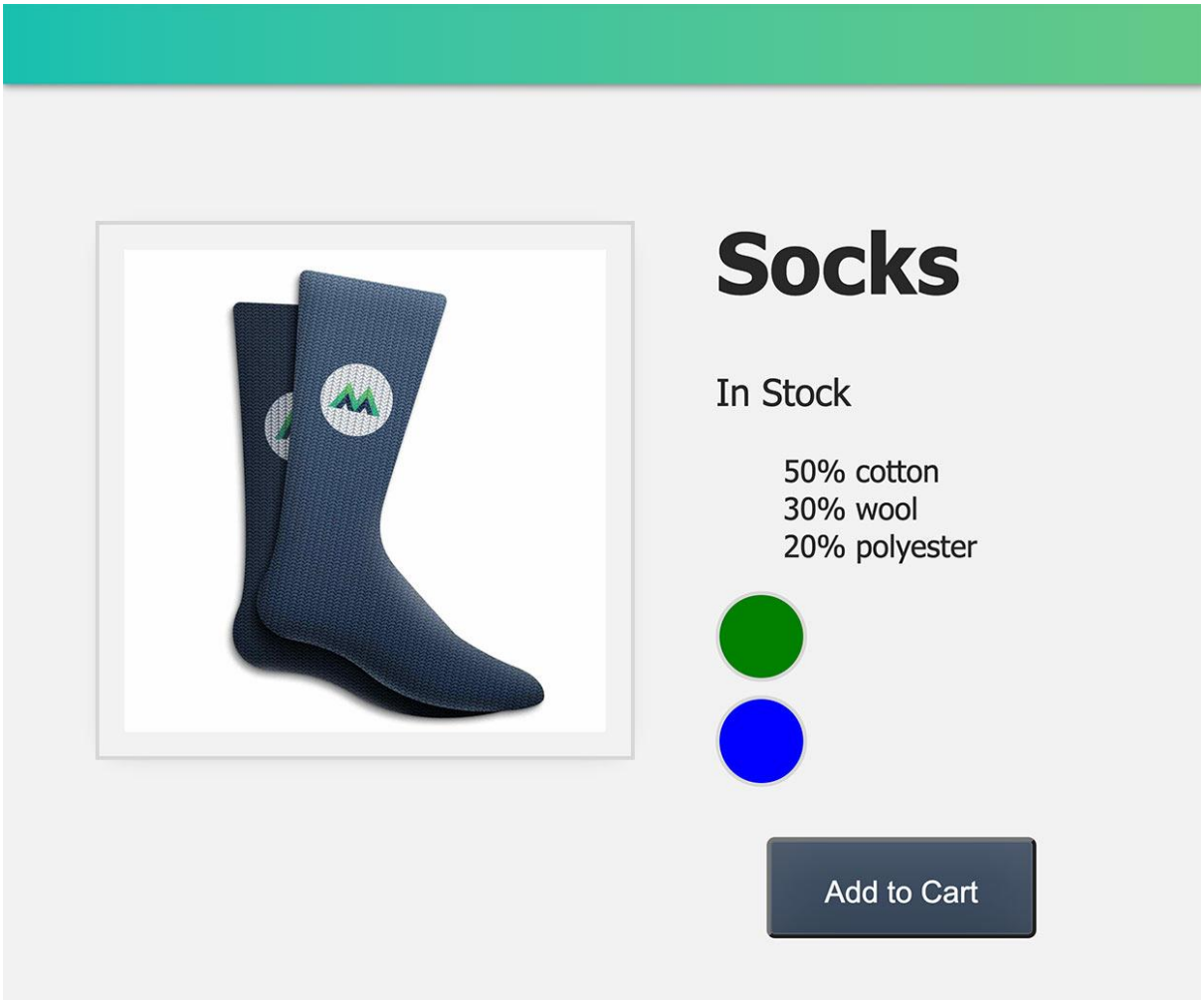
Now that we've got that out of the way, we can move on to the actual style binding. Just like it sounds, we want to bind styles to the variant divs. We do so by using `v-bind` (or its shorthand: `:`) on the `style` attribute, and binding a style object to it.

📄index.html

```
<div
  v-for="variant in variants"
  :key="variant.id"
  @mouseover="updateImage(variant.image)"
  class="color-circle"
  :style="{ backgroundColor: variant.color }">
</div>
```

Here, we're setting the divs' `backgroundColor` equal to the `variant.color`. So instead of printing out those strings, "green" and "blue", we're using them to set the background color of our circles.

Checking this out in the browser, we should now see two color circles filled in with a green and blue background.



Cool! Now let's understand, on a deeper level, how this is all working.

Understanding Style Binding

On our variant `div`, we added the `style` attribute and bound a style object to it.

index.html

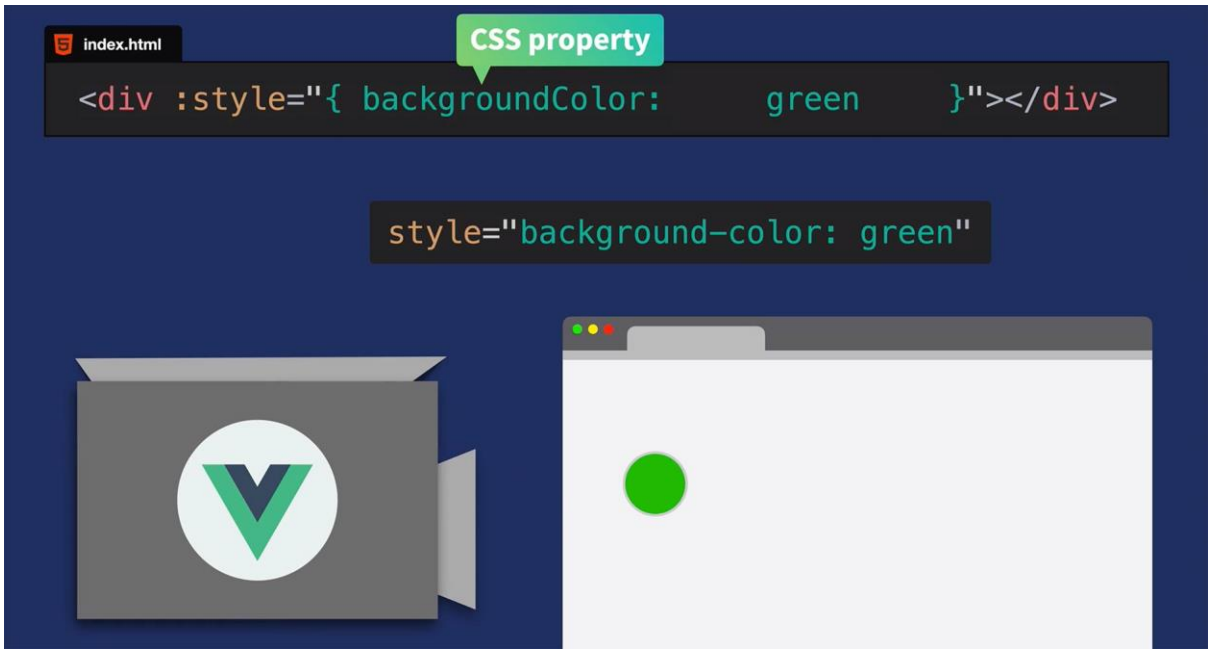
```
<div
  ...
  :style="{ backgroundColor: variant.color }">
</div>
```

That style object has the CSS property of `backgroundColor`, and we're setting that equal to whatever the variant color is at the time of that `v-for` iteration.

In the first iteration, `variant.color` is `"green"`

Vue takes that information and converts it into the code:`style="{ backgroundColor: green }"`

Then prints out a green background circle.



It repeats this process for the second variant color to create the blue circle.

Camel vs Kebab

There are some important things to consider when using style binding like this.

```
<div :style="{ backgroundColor: variant.color }></div>
```

Inside of this expression, remember that this style object is all JavaScript. That's why I used camelCase in the property name. If I had said `background-color`, that `-` would've been interpreted as a minus sign. But we're not doing any math here. We're setting a CSS property name.

So since we're in this JavaScript object, we have to use camelCased unless we want to use 'kebab-cased' in quotes to avoid the mathematical misinterpretation, like so:

```
<div :style="{ 'background-color': variant.color
}></div>
```

Both options will work, as long as you remember your quotation marks.

Style Binding: Objects

Sometimes you might want to add a bunch of styles to an element, but adding them all in-line could get messy. In these situations, we can bind to an entire style object that lives within our data.

Now that we've taken a look into the topic of style binding, let's look at a similar topic: class binding.

Class Binding

Back in our app, you'll notice that when our `inStock` data value is false, we can still click the Add to Cart button and increment the value of the cart. But if the product is out of stock, maybe we don't want the user to be able to add the product to the cart. So let's change up this behavior, disabling the button whenever `inStock` is `false` AND making the button *appear* disabled, using class binding.

To get this started, we'll use the shorthand for `v-bind` on the `disabled` attribute to add that attribute whenever our product is not in stock.

index.html

```
<button
  class="button"
  :disabled="!inStock"
  @click="addToCart">
  Add to Cart
</button>
```

Now, whenever `inStock` is `false` and we click the Add to Cart button, nothing will happen since it's disabled. But the button still *appears* active, which is misleading to our users. So let's use class binding to add a `disabledButton` class as well, whenever `inStock` is `false`.

You'll see in our CSS file that we already have this `disabledButton` class, which sets the `background-color` to gray and makes the `cursor` not-allowed.

styles.css

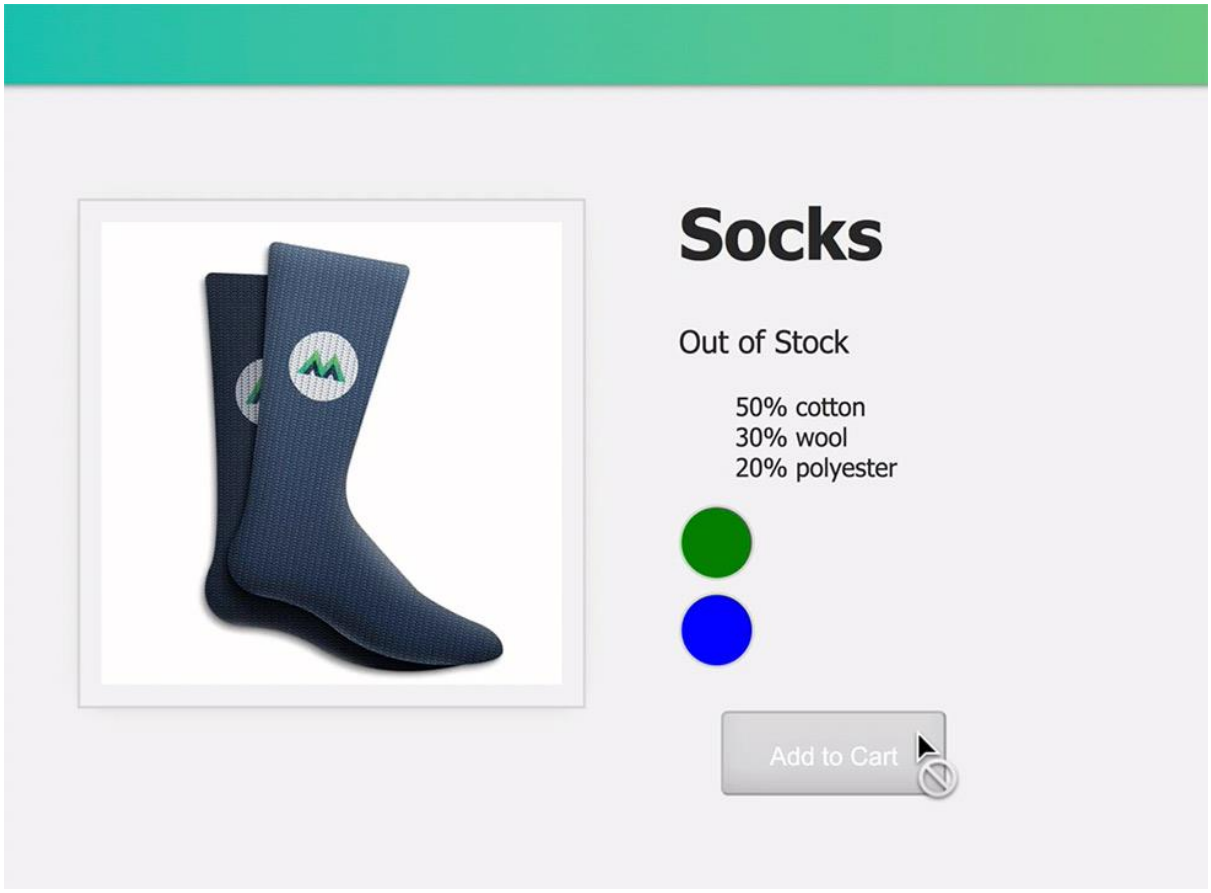
```
.disabledButton {
  background-color: #d8d8d8;
  cursor: not-allowed;
}
```

To apply this class conditionally, based on the value of `inStock`, we'll use the shorthand for `v-bind` on the `class` attribute, and use an expression that adds the `disabledButton` class (or not) whenever `!inStock`.

index.html

```
<button
  class="button"
  :class="{ disabledButton: !inStock }"
  :disabled="!inStock"
  @click="addToCart">
  Add to Cart
</button>
```

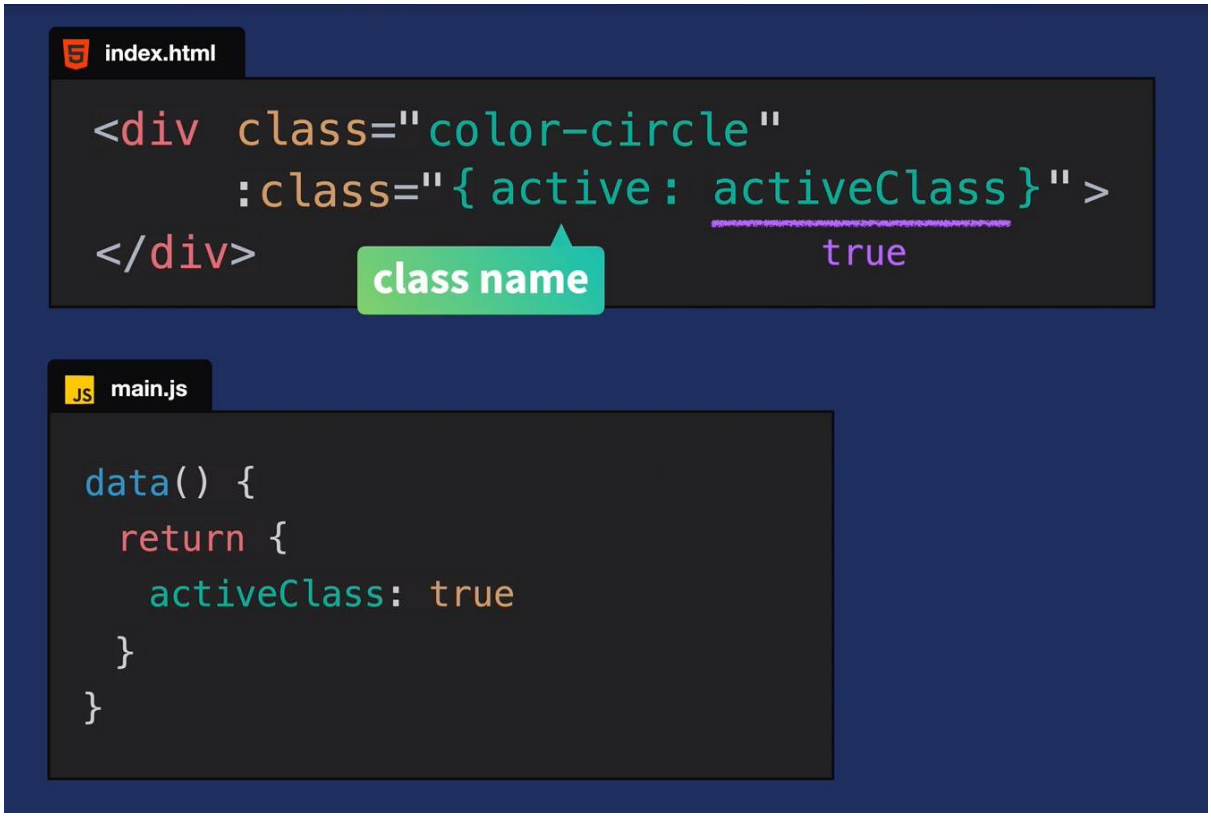
Now whenever `inStock` is `false`, not only will the button be disabled, it will also appear disabled.



Multiple Class Names

When getting started with class binding, there are some things to note. For example, what happens when we already have an existing class and we want to conditionally add another class based on a data value?

For example, if we already have the `color-circle` class on this `div`, and we conditionally add the `active` class, how will this look?



Those classes are going to be combined like so:

```
<div class="color circle active"></div>
```

Ternary Operators

A helpful tool that class binding gives us is the ability to use in-line ternary operators to add different classes based upon a condition.

In this case, because `isActive` is `true`, we are indeed adding the `activeClass`. If it were `false`, we'd add no class (`''`); alternatively, we could have added an entirely different class.

The variations in syntax and use cases that I just showed you with class and style binding is only the start. So I recommend checking out the Vue docs for more use cases and examples.

Coding Challenge

We've reached the end of the lesson and we're onto our challenge:

Bind the `out-of-stock-img` class to the image whenever `inStock` is `false`.

As a reminder, if you're coding along with our repo, you can check out `L7-end` branch, and you can view the solution code on Codepen.

Download Video

Share Lesson

Lesson Resources

- Starting Code
- Ending Code

Discuss in our Facebook Group Send us Feedback

Previous LessonNext Lesson

Vue mastery

As the ultimate resource for Vue.js developers, Vue Mastery produces weekly lessons so you can learn what you need to succeed as a Vue.js Developer.

VUE MASTERY

-
-
-
-
-
-
-
-

ABOUT US

-
-
-
-