MODULE 01 - 019: SCSS @content directive FOR FLEXIBLE `@mixin`'s

The `@content` directive allows mixins to be more flexible by yielding additional styles without the need to specify all values in the mixin itself. It lets you pass custom content into the mixin.

Use @content when you need more flexibility in your mixins, especially if you want to pass in dynamic or custom styles without dealing with too many parameters. It's a great tool for creating adaptable and reusable mixins.

- **More flexible** mixins.
- **No need for static arguments**
- Easily override default styles.
- Custom styles directly within the mixin without pre-defined parameters.

Example

1. Standard Mixin (Before `@content`)

```
@mixin notification($background-color, $color, $border) {
  width: 99%;
  height: 35px;
  text-align: center;
  padding-top: 10px;
  font-size: 1.2em;
  font-family: Verdana;
  border-radius: 3px;
  margin: 10px;
  background-color: $background-color;
  color: $color;
  border: $border;
}

.error {
  @include notification(DarkRed, white, 1px solid LightSlateGray);
}

.success {
  @include notification(MediumSeaGreen, MintCream, 1px solid LightSalmon);
}
```

Observations

- `notification` is called with specific values.
- Argumensts are passed `($background-color, $color, $border)` directly when including the mixin.

It's ok, but **not flexible when you want to override styles**.

---

2. Using `@content` for Flexibility

```
@mixin notification {
  width: 99%;
  height: 35px;
  text-align: center;
  padding-top: 10px;
  font-size: 1.2em;
  font-family: Verdana;
  border-radius: 3px;
  margin: 10px;
  @content;
}
```

```scss
.error {
  @include notification {
    background-color: DarkRed;
    color: white;
    border: 1px solid LightSlateGray;
  }
}

.success {
  @include notification {
    background-color: MediumSeaGreen;
    color: MintCream;
    border: 1px solid LightSalmon;
  }
}
```

Observations

- No arguments are passed to the mixin.
- `@content` lets define **custom styles inside the mixin call**.

So, overriding styles becomes easier and more flexible.

Video lesson Speech

[ENG]
# How to Use the @content Directive in Scss to Allow for Mixin Flexibility This lesson examines the `@content` directive in Scss and specifically walks through how to refactor a mixin by yielding to a set of custom styles.

In this guide, we're going to talk about the content directive now the content directive is a very helpful little tool that we can use with mixins that allow us to **make a little bit more flexible kinds of interfaces for those mixins**.
So right here I already have a standard mix in with three arguments right here and this is for a set of an alert device.

So we have HTML with these divs right here and then we have a mixin called notification that has a number of styles.
Everything from a width and height to a border-radius and the three that are dynamic the three values that are dynamic are the background color the text font color and then the border.
We're bringing those in as arguments.

Now, this is fine this works perfectly fine our end result is going to result in the exact same CSS code once it gets compiled.
However, there is another way of doing this:
In certain circumstances using the content directive can be a little bit more intuitive than what we're doing right here.
Here we are manually setting all these values and we're setting all these arguments and then we're calling them.
In this case, I would take a different approach.
Instead of placing these content items inside of the mixin and I would actually use the content directive.

We can get rid of all of the code right here. So now it looks like this.

What I'm going to do is pass in `@content` and this is going to do is it's going to allow us to essentially place any other styles that we want and this content directive is then going to yield to whatever we pass in.

In regards to this first error notification, the way that **you can call this content directive or how you can pass data to** it is **by opening up curly brackets and setting our values to what we want like below.**

So now all of this is working perfectly.
The important thing to know is **we're not limited by these items** I know we started with these but if I wanted to do anything else then I could place all of that right here in this content.

So if I wanted to, I could actually override the width here and I could say 15 pixels or something like that it would shrink. Obviously that would be a little bit weird because you wouldn't be able to see it but the whole point is to show that **it is now completely dynamic so we can override any of those values that we want.**

The whole point of using the content directive is not to replace the arguments for mixins because there are plenty of times where you're going to want to do that.

I personally love using arguments in mixins. **It makes it very easy, it's very much like writing Ruby code or javascript code. However, there are times where I want a little bit more flexibility and that's what this content directive allows** me to have.
Here I don't have to worry about passing in all kinds of crazy variables and arguments and worrying about default types and those kinds of things.
I can simply **call the content directive right here and pass in any other types of styles overrides additional items right in there and it'll all work very nicely.**

So that is how you can use the content directive in SCSS.

HTML CODE

```html
<div class="error">
  There was an error processing your request.
</div>

<div class="success">
  Your request was processed successfully.
</div>
```

STARTER SCSS CODE

```scss
@mixin notification($background-color, $color, $border) {
  width: 99%;
  height: 35px;
  text-align: center;
  padding-top: 10px;
  font-size: 1.2em;
  font-family: Verdana;
  border-radius: 3px;
  margin: 10px;
  background-color: $background-color;
  color: $color;
  border: $border;
}

.error {
  @include notification($background-color: DarkRed, $color: white, $border: 1px solid LightSlateGray);
}

.success {
  @include notification($background-color: MediumSeaGreen, $color: MintCream, $border: 1px solid LightSalmon)
}
```

Refactored Scss Code with @content Directive

```scss
@mixin notification {
  width: 99%;
  height: 35px;
  text-align: center;
  padding-top: 10px;
  font-size: 1.2em;
  font-family: Verdana;
  border-radius: 3px;
  margin: 10px;
  @content;
}

.error {
  @include notification {
    background-color: DarkRed;
    color: white;
    border: 1px solid LightSlateGray;
  }
}

.success {
  @include notification {
    background-color: MediumSeaGreen;
    color: MintCream;
    border: 1px solid LightSalmon;
  }
}
```

[SPA] La directiva @content es una herramienta poderosa en SCSS para hacer los mixins mucho más flexibles. En lugar de pasar todos los valores como parámetros, puedes dejar que el mixin se complete con los estilos que tú des.

Usamos contents cuando necesitamos más control sobre lo que pasa dentro del mixin y cuando quieras pasar estilos dinámicos sin complicarte con demasiados parámetros.

Los usamos por:
1.flexibilidad. Puedes pasar cualquier estilo dentro del mixin, sin tener que predefinir todo con variables.
2. Si necesitas cambiar algún estilo por defecto, puedes sobreescribirlo de forma rápida y fácil.

Por último, el código realizado es más limpio y conciso, eficiente en tiempo de desarrollo.
No necesitas tantos parámetros ni repeticiones, lo que lo hace más simple y manejable.