MODULE 01 - 009: SCSS Variables (ii)

In this guide, we'll build on what we've learned about SCSS variables and explore how to set **default variable values** in SCSS.

Why Use Default Variables?

The `!default` flag in SCSS makes your styles **dynamic**, **scalable**, and **maintainable**.
Whether you're building multi-client solutions or simply aiming for flexibility, this tool ensures your variables work smarter, not harder.

Setting default variable values may not seem crucial at first, but it becomes highly valuable in scenarios like **white-labeling styles** for applications.

For example: - Platforms like **Wix** or **Squarespace** need this functionality to manage styles dynamically across their entire website maps.

Variable Scope

SCSS, like other programming languages, uses **variable scope** to control where variables can be accessed within a program. Using var. scope in a proper way helps you manage your variables effectively.

Benefits:

- **Effortless Updates**: Change one value, and it propagates across the application.
- **Dynamic Styles**: Easily customize designs for different clients or use cases.
- **Cleaner Code** : Avoid inline styles or redundant declarations.

Use Cases

Case 1: @mixin

What are `@mixins`?
A reusable block of code that make styles modular and flexible.

Imagine you're creating styles for a system that will be used across multiple websites.

For this example:
- Our **default color scheme** will use **dark red** for headings and section backgrounds.

Then:
> * We need to use **SCSS mixins** and the **`!default` flag** to set these values.

Case 2: Making Variables Dynamic with the `!default` Flag

- **Client 1** loves the default red color. No changes needed!
- **Client 2** wants a **dark cyan** color for their application.

Without the `!default` flag, this would require manually updating the entire application.
Worse, if the app is dynamic, you'd have to use inline styles or other hacks, which: - **Introduce bugs**
- **Violate best practices**

How the `!default` Flag Solves This ?:

1. **Set default values** using the `!default` flag.
2. **Override** the values by adding a single line at the top of the SCSS file.

3. SCSS will:
    - Ignore the default declarations.
    - Use the new value wherever the variable is called.
      ***

Code example

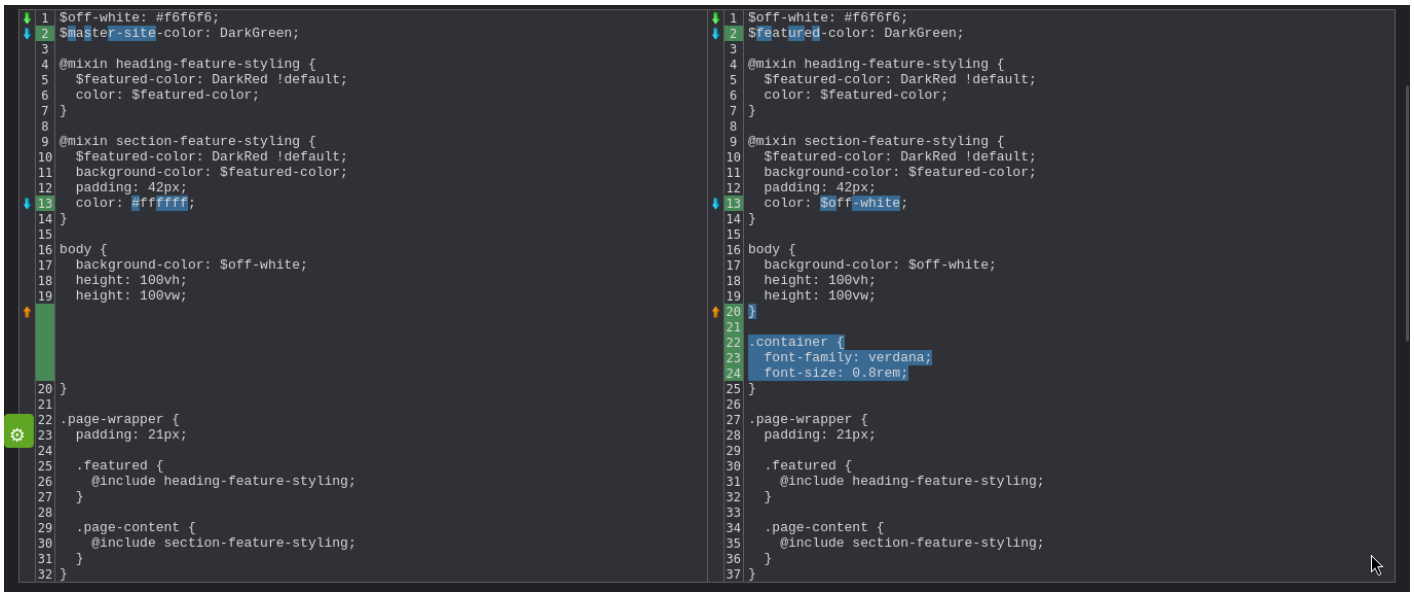

Figure 1: IMG_SCSS_snippets

Video Lesson Speech

[ENG]
Using Default Variables in SCSS We've walked through how to work with variables in SCSS to make it possible to make a single change that populates a number of style definitions. To take this knowledge a step further, in this guide we'll examine how to set default variable values in SCSS.
***

We've walked through how to work with variables in SCSS to make it possible to make a single change that populates a number of style definitions. To take this knowledge a step further, in this guide we'll examine how to set default variable values in SCSS.

This may not seem like a big topic to learn because it's mainly important in situations where you are white labeling styles in an application.
However, I think it can be a helpful skill to understand and it also is a good introduction to learning about how SCSS manages variable scope.

Variable scope is the process that programming languages use to control where variables can be accessed throughout a program.

Example of Using Variables in SCSS

For this example, let's imagine a scenario where you are building a set of styles that are going to be utilized on multiple websites. Services such as Wix and Squarespace have to implement this type of behavior throughout their entire systems. For our example, we want our color scheme to have a dark red set of colors for heading text and section backgrounds.

We'll implement SCSS `@mixin`s and add variables with the `!default` flag on them.
If you're not familiar with mixins, we'll dive into them later on, for right now just think of them like

reusable code snippets that can be called.
You can see the results below:

```
$off-white: #f6f6f6;
$master-site-color: DarkGreen;

@mixin heading-feature-styling {
  $featured-color: DarkRed !default;
  color: $featured-color;
}

@mixin section-feature-styling {
  $featured-color: DarkRed !default;
  background-color: $featured-color;
  padding: 42px;
  color: #ffffff;
}

body {
  background-color: $off-white;
  height: 100vh;
  height: 100vw;
}

.page-wrapper {
  padding: 21px;

  .featured {
    @include heading-feature-styling;
  }

  .page-content {
    @include section-feature-styling;
  }
}
```

Don't worry about understanding all of the code if it looks different than you're used to.
Simply focus on seeing how we're setting the default colors in each of the mixins, which results in the dark red color that we're looking for.

Making the Variables Dynamic with the `!default` Flag

Now, let's imagine that we have two clients come along. The first client likes the red color, so we don't have to make any changes.
However, the second client wants to have a dark cyan color for their application.

If we didn't use the !default flag, we'd have to go throughout the entire application and change all of the values.
Additionally, if the application is dynamic we'd need to figure out a way to perform an override on the value, such as using inline styles, which can be buggy and breaks a number of best practices.

However, because we used the `!default` flag in our mixins, we can simply add a single line at the top of the file that will be viewed as the new 'master' color.
Because the other color variables have the `!default` flag, SCSS will ignore those variable declarations and will only use the new one that's provided.

```
$off-white: #f6f6f6;
$featured-color: DarkGreen;

@mixin heading-feature-styling {
  $featured-color: DarkRed !default;
  color: $featured-color;
```

```
}

@mixin section-feature-styling {
  $featured-color: DarkRed !default;
  background-color: $featured-color;
  padding: 42px;
  color: $off-white;
}

body {
  background-color: $off-white;
  height: 100vh;
  height: 100vw;
}

.container {
  font-family: verdana;
  font-size: 0.8rem;
}

.page-wrapper {
  padding: 21px;

  .featured {
    @include heading-feature-styling;
  }

  .page-content {
    @include section-feature-styling;
  }
}
```

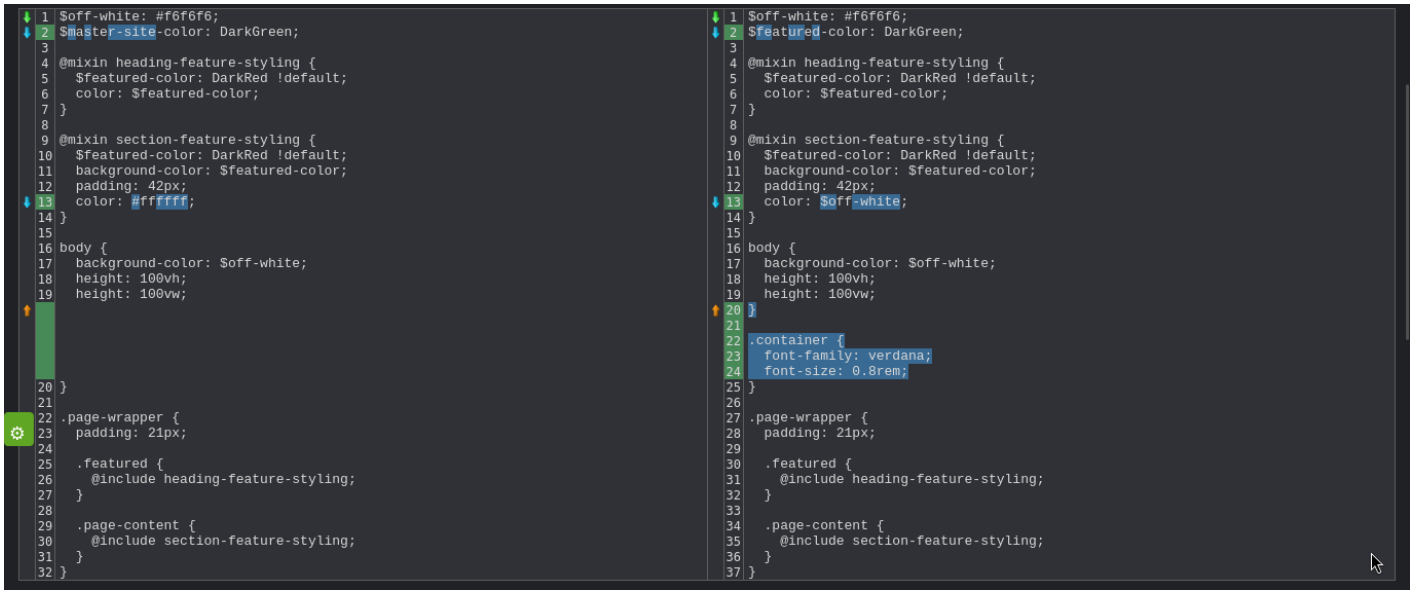

Figure 2: IMG_SCSS_snippets

[SPA]

Usando Variables Predeterminadas en SCSS.

Hemos visto cómo trabajar con variables en SCSS permite realizar un solo cambio que se refleja en numerosas d

Puede parecer un tema menor, ya que su principal importancia radica en situaciones donde se personalizan esti

El ámbito de una variable es el proceso que utilizan los lenguajes de programación para controlar dónde se pu

Ejemplo de Uso de Variables en SCSS

Para este ejemplo, imaginemos un escenario en el que estamos creando un conjunto de estilos que se utilizarán

Implementaremos mixins de SCSS y agregaremos variables con la bandera !default. Si no estás familiarizado con

Ahora puedes visualizar el primer bloque de código SCSS de esta lección.

No te preocupes por entender todo el código si parece diferente a lo que estás acostumbrado. Simplemente conc

Haciendo las Variables Dinámicas con la Bandera !default

Ahora, imaginemos que tenemos dos clientes. Al primer cliente le gusta el color rojo, por lo que no tenemos q

Sin embargo, debido a que usamos la bandera !default en nuestros mixins, simplemente podemos agregar una sola

Ahora, por favor, lee el segundo fragmento de código SCSS.