MODULE 1 - 014: @mixin with (ARGUMENTS)

This guide focuses on integrating **($arg:)** into SCSS mixins, allowing for dynamic and reusable styling.

Arguments in mixins enable you to: - Pass specific values to customize styles. - Define **default values** for common use cases. - Avoid repetitive code by reusing the same mixin dynamically.

---

1 Defining a Mixin with Arguments

Use parentheses to specify arguments for a mixin. You can also assign default values.

```scss
@mixin mixin-name($arg: default-value) {
  // Use $arg within the mixin
}
```

Example

Defining a @mixin to style links based on '$link-color'

```scss
@mixin featured($link-color: Black) {
  color: Tomato;

  .subheading a {
    color: $link-color;
    text-decoration: none;

    &:hover {
      color: $link-color;
      text-decoration: underline;
    }
  }
}
```

---

2 Using this @mixin with (Arguments)

When calling the mixin, pass a value for the argument:

```scss
@include featured(Red);
```

Otherwise, **if no values were given, the default vaule would be used**.

Entire SCSS Code Example for Mixin Arguments with Default Values

```scss
$off-white: #f6f6f6;
$featured-color: DarkRed;

@mixin featured($link-color: Black) {
  color: Tomato;
  .subheading a {
    color: $link-color;
    text-decoration: none;
    &:hover {
      color: $link-color;
      text-decoration: underline;
    }
  }
}
```

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

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

.page-wrapper {
  padding: 21px;
  $featured-color: RoyalBlue;

  .featured {
    @include featured;
  }

  .page-content {
    background-color: $featured-color;
    padding: 42px;
    color: $off-white;

    .container {
      height: 60px !important;
      font-family: courier;

      .description {
        float: left;
        width: 75%;
      }

      .sidebar {
        font-family: Verdana;
        text-align: right;
        float: right;
        width: 25%;
        @include featured(MintCream);
      }
    }
  }
}
```

EXTRA

These references will be furtherly explained, but I think it's necessary to expand the SCSS scope by adding these noted:

Multiple Arguments // Logical Syntax

`@mixin`'s ($Arg:)'s may include multiple arguments, following these rules:

- **POSITIONAL ARGUMENTS in order**
- **NAMED ARGUMENTS not in order, but always the same**

```scss
@mixin iAmABox($width:  100px, $height:  200px, $color: Lime, $border: 2px solid black) {
  width: $width;
```

```
  height: $height;
  background-color: $color;
  border: $border;
}

.iAmAOrderedBox {
  // Here, Positional Args. IN ORDER included.
  @include iAmABox(100px, 200px, Lime, 2px solid black);
}

.iAmANamedBox{
  // Here we are NAMED args.
  @include iAmABox($width:  100px, $height:  200px, $color: Lime, $border: 2px solid black);
}
```

Both aren't oppossite, but complementary !!!

* Positional Args. are passed in the same order as declared as params. into the mixin, being **fast** while coding. It lacks on **scalability**, for example. * Named args., are a more verbosing mode, improving the **readibility**. These aslo complies with the usual *strong typed language* others than CSS/SCSS, being **more concise** and, the best part, it's not needed to remember the exact order of the arguments when replying, including, and so on.

`@mixin`'s (SCSS in fact) allows using logical operators as:

- **Conditionals**: `@if`, `@else` for logics within the mixin.
- **Loops**: `@for`, `@each`, or `@while` for reusing styles.

```
// Example of logical operators on SCSS
@mixin iAmUsedToShowConditional($iAmACondition: true) {
@if $iAmACondition {
  color: Lime;
} @else {
  color: Red;
}
}

.title {
@include iAmUsedToShowConditional(false);
}
```

```
// This code snippet doesn't belong to the actual Scss code we're using as example

@mixin box($width, $height, $color: LightGray) {
  width: $width;
  height: $height;
  background-color: $color;
}

.container {
  @include box(100px, 200px, LightBlue);
}

.alt-container {
  @include box($height: 300px, $width: 150px);
}
```

An expanded explanation will be provided on the following guides.

Video lesson Speech

Introduction to Mixin Arguments in SCSS

In this guide you'll learn how to pass standard and default arguments to SCSS Mixins to generate dynamic behavior.
***

After I finished the last section I realize that I may have jumped the gun a little bit in what we were planning to do next. Some of the topics that I was about to introduce for our more complicated feature with mixins might be a little bit challenging to understand if you've never seen them before. So we are going to break it down and go with a few more straightforward topics that lead up to that and we'll get to that in a couple of sections.

There are times where you may simply want to have access to a set of styles and to be able to call this mix and from anywhere and that's perfectly fine. However, there are many times where you want to change that behavior just a little bit. You may want to have some type of dynamic behavior and that's what we can leverage arguments that we can pass right into our mixins. I'm going to start off with a basic example and this is going to lead directly into how we can implement conditionals. We are going to declare a variable and with this variable, I'm just going to call this link color.

The goal of this is is that it should be able to replace the color on both of the links and it should be dynamic. So if we want one color for one link but a completely different color for the other then we can still call featured and have that different behavior. So the way that we're going to do it is we can copy link color and instead of having the hardcoded light blue steel I'm going to paste it in here.

So now that we have this link color you may notice if you're following along using code pen that we now have an error. And if we click on this error, what it's going to show us that the mixin featured is missing the argument link color. That's a nice helpful way of seeing what's going wrong. So what this means is that when we pass in or when we call featured right here we need to pass in that argument. So here I'm going to pass in. Let's just say black.

And now what is going to happen is after I fix the other one it's going to change it for this featured. I'm going to copy this come down here and the same issue. Now they're both black. Now if I want to change this one so save for this one I want to change this to something like Mint cream.

Now you can see that this heading link is still black but this one now has been changed to mint cream. And so even though we're using the exact same mix and we can now make this completely dynamic.

Now the other thing and we're going to get into is we also **have the ability to have default arguments** and so say that we for 99 percent of the time we want to have black for this featured mixin. But there are a few times like when we have a dark background where we may want to throw in a different color. So the way that you can do this is we can pull out the color black out of featured first.
It's going to throw an error right now because we haven't implemented this fix yet.

Up in our mixin, we can add out default argument like this:

So even though we're not passing in an argument with featured it's still working. If you come down here mint cream is still overwriting this link color.

Now, this starts to get **a little bit more complicated when you go into having multiple arguments**.
In our more advanced section when we talk about how we're going to implement flex-box here then I'm going to show you how you can actually use named arguments to make it very clear what values you're setting.

SCSS Code for Mixin Arguments with Default Values

```scss
$off-white: #f6f6f6;
$featured-color: DarkRed;

@mixin featured($link-color: Black) {
  color: Tomato;
  .subheading a {
    color: $link-color;
    text-decoration: none;
    &:hover {
      color: $link-color;
      text-decoration: underline;
```

```
    }
  }
}

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

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

.page-wrapper {
  padding: 21px;
  $featured-color: RoyalBlue;

  .featured {
    @include featured;
  }

  .page-content {
    background-color: $featured-color;
    padding: 42px;
    color: $off-white;

    .container {
      height: 60px !important;
      font-family: courier;

      .description {
        float: left;
        width: 75%;
      }

      .sidebar {
        font-family: Verdana;
        text-align: right;
        float: right;
        width: 25%;
        @include featured(MintCream);
      }
    }
  }
}
```

[SPA]

Nuestros @mixin's pueden pasar atributos como argumentos, de una manera similar a cómo lo hacemos en otros lenguajes.

Pasar argumentos facilita aún más evitar redundancia en nuestro código y en nuestro tiempo a la hora de codificarlo.

Cuando invocamos un mixin (@include), aún podemos usar un valor personalizado como argumento, deshabilitando lo que dicho mixin hace.

Al pasar argumentos, lo habitual es que éstos sean múltiples. Más adelante se irán explicando cómo hacer, pero para iniciar, los argumentos pueden pasarse de dos formas distintas, con sus propias normas:
1. Argumentos posicionales. Que siempre siempre llevan el mismo orden, cuya casi única ventaja radica en que no deben

ser nombrados explícitamente, facilitando un código algo más breve y conciso. 2. Argumentos con su etiqueta (named). Pueden incluirse en distinto orden al especificado, pero siempre deben ser precedidos por sus respectivas etiquetas de atributo.

Visualiza el código de ejemplo para comprender mejor este concepto.

Por último, otra ventaja que se irá desgranando respecto a SCSS es que, como lenguaje preprocesado, permite utilizar operadores lógicos típicos (como Condicionales o bucles). Esto mejorará aún más los tiempos de desarrollo.