MODULE 01 - 018 SCSS Lists, @each Directive, String interpolation

`@each` Directives

What is the `@each` Directive?

- `@each` directive: Allows iterating through items in a list, executing code for each item.

Syntax

```scss
@each $variable in $list {
  // Code block to execute
}
```

- `$variable`: Temporary variable assigned the value of the current list item.
- `$list`: The list being iterated over.

The given example, step-by-step

Lists - @each

```scss
$cars: 'maserati', 'tesla', 'porsche';

@each $car-name in $cars {
  .car-#{$car-name} {
    background-color: lightgrey;
  }
}
```

- On the first loop, `$car-name = 'maserati'`, generating .car-maserati.
- On subsequent loops, `$car-name = 'tesla'` and `'porsche'`.
  *** ## String Interpolation

String interpolation in SCSS dynamically incorporates variable values into strings.
Specially useful for generating CSS .class names, property values, URLs, and so on.

Syntax

```scss
#{$variable}
```

```scss
$cars: 'maserati', 'tesla', 'porsche';

@each $car-name in $cars {
  .car-#{$car-name} {
    background-image: url('./#{$car-name}.jpg');
  }
}
```

Now, Combining `@each` and String Interpolation

These two features work together to dynamically generate styles in SCSS, as shown below.
The given SCSS ...

```scss
$cars: 'maserati', 'tesla', 'porsche';

@each $car-name in $cars {
  .car-#{$car-name} {
    background-image: url('./#{$car-name}.jpg');
    height: 300px;
    width: 500px;
    float: left;
  }
}
```

… will output this SCSS:

```
.car-maserati {
  background-image: url('./maserati.jpg');
  height: 300px;
  width: 500px;
  float: left;
}
.car-tesla {
  background-image: url('./tesla.jpg');
  height: 300px;
  width: 500px;
  float: left;
}
.car-porsche {
  background-image: url('./porsche.jpg');
  height: 300px;
  width: 500px;
  float: left;
}
```

So, I think, this is a GREAT way to improve development time by prevenintg to repeat each task one by one, don't you?

Video speech lesson

[ENG]
# SCSS Lists, @each Directive, and String Interpolation This guide will help you demonstrate how to utilize lists, the SCSS @each directive, and string interpolation in order to dynamically generate classes.

I think we're going to have a lot of fun with this guide. We're going to learn many different new things. And part of the reason for it is because these would each be topics that really work best together. I have seen them where they're presented one at a time. And one thing that I came across with the feeling of was that it was very hard to understand why they were important in unless they were actually all grouped together so you could see how they interact with each other. The things that were going to cover in this guide are going to be lists and then we're also going to talk about each directive inside of SCSS and then we're going to finish off with string interpolation. If some of those things sound very fuzzy and you've never heard of them that is perfectly fine we're going to go through them. And I think we are going to have a really fun example.

So at the very top here I have a few images so I have a Tesla image a Maserati image and a Porsche image.
Now I am going to grab these URLs and I'll just paste each one of them in.
So we have a Porsche we have a Tesla and then we have a Maserati.
So we have each one of these. What I want is to be able to have HTML divs here and I want to dynamically generate the class names inside of our SCSS. That's something that is possible from within SCSS and it is a great way of performing tasks such as I imagine that you had a list of icons or you had something in your assets directory and you wanted to iterate through and you didn't want to manually just type the class name then pull in the image URL and do that for every single one. If you only have two or three like this example it may not seem like a big deal but imagine that you have an application with hundreds of icons or images and literally all you're trying to do is to change a certain number of values such as setting the background image or having a size or setting it to no-repeat or anything like that. You don't want to perform that task over and over again that's a bad programming practice.
Then also any time you add other icons you don't want to have to go and repeat all of it again.

So let's see how we can leverage each one of those things we're going to once again this guide is going to be all about lists. It's going to be about each directive and it's going to be about string interpellation.

So I'm going to bring this down here and in fact I'm just going to comment all of these out just so they don't get in the way of the rest of our code and I'm going to create a few more divs here so I'm going to create a div of class car dash Maserati and then I'm just going to repeat this a few times for each of the other cars so we have Mazarati, we have tesla, and then we have a Porsche so that is all we need on the HTML side which is going to be pretty cool.
This means that if you have an application that you have this type of behavior built in to with what we're going to build in SCSS then you could bring in entire code components just with these little div calls.

```html
<div class="car-maserati"></div>
<div class="car-tesla"></div>
<div class="car-porsche"></div>
```

So now with SCSS Let's talk about what we need to do.
The first thing we're going to implement is a list. So here I'm going to add comments.
You know if you follow many of my blogs and other courses that I'm not a big fan of comments but I will do implement comments here just so you can see them.
It shows what each one of these elements is for using it for your own reference.

So here we're going to create a list called cars.
I'm just going to pick out the names that we want. So are you going to say Mazarati then Tesla. And lastly, we're going to go with Porsche.
And so if you have this they are building a car Web site.
Then you could have each one of your images in the asset directory just like this and give it whatever name is going to be called and then it can iterate through and it can build all of these for you. So now I'm going to come down and we're going to use our each directive now or each directive is going to look a lot like the if conditional or the mix in.
So we're going to say each directive and hear what that means is it's going to start with an `@` symbol and then we're going to say each and the very first thing that we pass to each is going to be what's called an iterator variable. So here I'm going to say car name.

Now if you notice there's nothing called a car name that we've declared yet.
What each does is you the very first argument you pass it is going to be whatever the car is every time that it goes in its loop.
So with each, the first time that iterates over cars car name the first time is going to be Mazarati the second time it's going to be Tesla. And the third time it's going to be Porsche. So here we have each car name in and then I can just pass in our list so I can say each car name in cars and then from there I can now pass in everything that we want to happen.

So what I want to happen is I want to dynamically generate all of these class names. So if you were to build this by hand you do something like this. You could say car-tesla, and then you would build all of your classes just like this. However, because of the each directive because of string interpellation and lists we don't have to do that.
We can't actually just make one class and it will dynamically generate all of this for you if you come from any language programming language like Ruby and you've ever worked with metaprogramming. This is very similar to metaprogramming. And if you're about to start learning more about Ruby and metaprogramming this is a great introduction because essentially what we're doing here is something that many people would say is incredibly advanced development because we're dynamically creating code on the fly.
We're essentially writing code that is going in turn and it's writing code. So the next thing we're going to implement so so far we have lists. Now we have our each directive.

```scss
// List
$cars: 'maserati', 'tesla', 'porsche';

// Each directive
@each $car-name in $cars {
  // String interpolation
  .car-#{$car-name} {
    background-image: url('https://s3.amazonaws.com/bottega-devcamp/scss/cars/#{$car-name}.jpg');
    background-repeat: no-repeat;
    height: 300px;
    width: 500px;
    object-fit: fill;
    float: left;
  }
}
```

The next thing we're going to put in place is string interpellation. So here you use a string interpellation and let me just fix the indentation. There you go. So the way that you use string interrelation inside of SCSS is by using the same syntax as if you're coming from the Ruby programming language.
So what you do is you start with a hash then use the curly brackets and then you put anything inside of it that you want. In this case, we're going to use our iterator variable. So this car name it's going to be Mazarati the first time which means it's going to say car-Mazarati which you notice is right here.
So because we have access to a Mazarati that means that we get to place it right in here and this is going to generate this class for us whenever the SCSS compiles. So this is the first part. This is going to generate our three class names. Right now they're empty though so let's go and let's build it in with some more unstring interpellation.
So I'm going to say. Background image and pass in a URL and the URL we're going to pass in is going to be this one except we're going to change the name. So instead of it being Porche Tesla or Mazarati, we're not going to have to hardcode it. We're going to pass it in using string interpolation. So here again make sure you finish off with a semicolon. So here what we're going to do is take this car name and I'm going to take the entire thing here and we'll say instead of and you make sure you keep the jpeg on there.
But instead of Porsche, it's going to say car name and then dot Jpeg so the first time it comes around. It'll be Mazarati. So we'll get this image then it's going to hit Tesla. So it's going to grab this image. Then Porsche and it's going to grab this image. So that is the first part.

```scss
// List
$cars: 'maserati', 'tesla', 'porsche';

// Each directive
@each $car-name in $cars {
  // String interpolation
  .car-#{$car-name} {
    background-image: url('https://s3.amazonaws.com/bottega-devcamp/scss/cars/#{$car-name}.jpg');
    background-repeat: no-repeat;
    height: 300px;
    width: 500px;
    object-fit: fill;
    float: left;
  }
}
```

Now let's add a few more styles. I'm going to say background. Repeat set the sequel to no-repeat then height. Set to three hundred pixels height. And as you can see right there we're already pulling in our images and also all of these image URLs you can use your own or if you want to follow along exactly.
Then I am making them available in the show notes, then I'll say this is five hundred pixels and next one object-fit will be fill and then let's say we're going to float this to the left so that is all side by side and look at that. We have everything that we need right here. So what we've done here is pretty neat.
Instead of simply having to copy and paste this code like we would have technically if you didn't use each in these lists then you'd have to have one of these classes that's completely identical except for having a slightly different URL. Now, this is something you could also use a mixin for but I think this is in this case using the each directive is probably the most intuitive.

Now if you don't believe me about the classes being dynamically generated Let's take a look. If I come and click inspect right here this brings everything up and look at that. We have a class called car dash Mazarati. Even though we don't actually have anywhere in our SCSS file where we've created that class we've all we've done it all dynamically we've done it all when the preprocessor hit and started running.
So if we look at the other one here we have car-Tesla. And for the last one, we have car-Porsche.

So this is something that's pretty cool this is another way. Remember that one of the top goals that SCSS looks to accomplish is to help you as a developer be as efficient as possible with your code to remove duplication and to be able to organize your code in a way that it's intuitive and scalable. And that's what these three components do. ## HTML Code

```html
<div class="car-maserati"></div>
<div class="car-tesla"></div>
<div class="car-porsche"></div>
```

SCSS Code

```scss
// List
$cars: 'maserati', 'tesla', 'porsche';

// Each directive
@each $car-name in $cars {
  // String interpolation
  .car-#{$car-name} {
    background-image: url('https://s3.amazonaws.com/bottega-devcamp/scss/cars/#{$car-name}.jpg');
    background-repeat: no-repeat;
    height: 300px;
    width: 500px;
    object-fit: fill;
    float: left;
  }
}
```

Image URLs used

```
https://s3.amazonaws.com/bottega-devcamp/scss/cars/tesla.jpg
https://s3.amazonaws.com/bottega-devcamp/scss/cars/maserati.jpg
https://s3.amazonaws.com/bottega-devcamp/scss/cars/porsche.jpg
```

[SPA]
Al igual que en otros lenguajes, como bash, ruby, javascript, podemos programar listas, índices e iteradores, para automatizar tareas redundantes (como crear multitud de classes con idénticos atributos). La creación de classes, atributos, generación de URL's, la mejora en el mantenimiento y escalabilidad del proyecto y la concisión del código son las mayores ventajas.

Para ello usamos listas, como variables, que incluyen una serie de datos.
Después, hacemos la programación, muy similar a un bucle FOR - IN típico, pero con la directiva @each.

La interpolación de Strings, como contenido variable, con su propia sintaxis, nos hace el resto del trabajo.
“‘