Variables can be used to set values of CSS properties but if a project calls for using a variable in a selector or property name, we need **"interpolation"** to make everything work properly.

What Does Interpolation Mean?

Let's start off with a bit of a definition. Interpolation is a term often used in mathematics to refer to the practice of "constructing new data points within the range of an existing set of known data points".

That still sounds a bit complicated to me so how about this:

Interpolation inserts something into something else.

In the Sass world, this usually means inserting a Sass expression

- a variable, function call or mathematical calculation - into another string, a selector or property name.

Sass uses a special syntax for inserting values into other values that takes the form

{% highlight scss %}
"This string will have a #{ $variable } inserted into it."
{% endhighlight %}

This syntax is also found in the Ruby programming language where Sass has its roots.

Let's look at a series of examples to illustrate how and why to use interpolation in Sass.

Interpolation vs. Concattenation

Take the following example where I have a variable called `$image` that holds the value `logo.png`.

I've got a selector for the logo and I want to apply the image as a background image. If I were to write it out manually, the image URL would look as follows.

```
{% highlight css %}
.logo {
background: url( ‘images/logo.png’ );
}
{% endhighlight %}
```

But instead of writing it out manually, I’m going to use the variable.

If I add the variable name within the url string, Sass doesn’t throw an error but the image doesn’t show up.

```
{% highlight scss %}
$image: ‘logo.png’;

.logo {
background: url( ‘images/$image’ );
}
{% endhighlight %}
```

Adding `$image` within the quotes for the path creates the litteral string `images/$image` as a URL. If we want `$image` to be used as a Sass variable we have two options.

Firstly, we could use string concattenation. This is the process of joining two strings together using a + operator.

```
{% highlight scss %}
$logo: ‘logo.png’;

.logo {
background: url( ‘images/’ + $logo );
}
{% endhighlight %}
```

We add the string `”logo.png”` from the variable onto the end of the `”images/”` string in the URL.

Alternatively, we could use interpolation and insert the value of the variable within the url string. To do that, we can re-write the code as follows:

```
{% highlight scss %}
$logo: ‘logo.png’;

.logo {
background: url( ‘images/#{ $logo }’ );
}
{% endhighlight %}
```

Within the interpolation braces, the Sass expresion is interpreted first and then inserted within the string.

When using interpolation, any Sass expression such as function call or mathematical calculation can be performed and the result of the expression will be inserted in its place.

You may ask why there are two different approaches for doing almost exactly the same thing. Well, interpolation can be used for inserting values into more than just strings.

Interpolating Selectors & Properties

We've just seen two solutions for inserting a variable into a string but let's now take a look at working with selectors.

In the following example I have three variables, one called `$selector` and one called `$property` and one called `$value`. These variables could have been named anything, but I've named them according to where they'll be used for this example.

{% highlight scss %}
$selector: 'highlight';
$property: 'bottom';
$value: red;
{% endhighlight %}

We can use the `$value` variable in the typical way to set the `color` property. When Sass compiles, this variable name is substituted for its value "red".

If we wanted to use the `$selector` variable in the selector, we need to use interpolation as string concattenation with + doesn't work as + is actually used as an adjacent sibling selector.

{% highlight scss %}
// this won't work

p . + $selector {
color: red;
}
{% endhighlight %}

Instead, we use interpolation to create the selector `p .highlight` in the compiled CSS.

{% highlight scss %}
p .#{ $selector } {
color: red;
}
{% endhighlight %}

A similar thing can be done to use a variable within a property name.

{% highlight scss %}
p .#{ $selector } {

```
color: red;
border-#{ $property }: 2px solid;
}
{% endhighlight %}
```

Interpolation is a very useful feature - even if using variables in selectors or properties isn’t as common as using variables for values, it’s good to know that this tool is there when needed.

Interpolation in Media Queries

One final area where interpolation comes in very useful is when working with media queries. Variables in media queries is worth looking at because it can be a bit confusing to know when interpolation is or is not needed.

Take the following example of a `$breakpoint` variable set to a number of pixels.

This can be included in a media query as a value as part of a `min-width` or `max-width` media query:

```
{% highlight scss %}
$breakpoint: 500px;

@media screen and ( min-width: $breakpoint ) {
body {
font-size: 1.25rem;
}
}
{% endhighlight %}
```

Imagine we use these `min-width` media queries a lot; we might get tired of typing out `screen and ( min-width: $breakpoint )` all the time. Instead, we could create another variable for the whole query and even use interpolation to insert the value of our breakpoint:

```
{% highlight scss %}
$breakpoint: 500px;
$query: ‘screen and ( min-width: #{ breakpoint } )’;
{% endhighlight %}
```

If we were now to use this variable for the media query, Sass would throw an error.

```
{% highlight scss %}
$breakpoint: 500px;
$query: ‘screen and ( min-width: #{$breakpoint} )’;

@media $query {
body {
```

```
font-size: 1.25rem;
}
}
{% endhighlight %}
```

Instead, we need to use interpolation and everything works as planned.

```
{% highlight scss %}
@media #{$query} {
body {
font-size: 1.25rem;
}
}
{% endhighlight %}
```