Sass brings a lot more power to CSS and starts to make it feel like a real programming language rather than just a series of property and value declarations. One key feature that enables this programmatic field is **Sass variables** which can hold a number of different types of data.

In this episode, you'll learn the seven core data types in Sass, and how we can manipulate and operate on these various values. Sass variables can store a number of different data types, including numbers, strings, colors, booleans, nulls, lists and maps. These different data types allows to create variables for all sorts of different CSS values.

A variable is just a handy label for some arbitrary value, and they can help make our code easier to read and quicker to update. Let's look at each of these different data types in turn to clarify what they are, how to use them and the kinds of values that we can put inside each of them.

Sass variables can store a number of different data types including:

- numbers
- strings
- colors
- booleans
- nulls
- lists
- and maps

These different data types allow us to create variables for all sorts of different CSS values. A variable is just a handy label for an arbitrary value and they can help make our code easier to read and quick to update.

Let's look at each of the different data types in turn to clarify what they are and how to use them.

Numbers

Numbers are used a lot in CSS from setting margins and font-sizes to border-radius and line height. Numbers come in a few different flavours including integers (whole numbers), floating points (decimals) and length values like `10px`, `20em` or `50%`.

{% highlight scss %}
1, 2, 3, 0.1, 5.3, 10px, 20em, 50%
{% endhighlight %}

Strings

Strings of text are found in a number of CSS values including font families, colours and URLs. String in CSS are found both with and without surrounding quotation marks and are said to be "quoted" or unquoted strings.

```
{% highlight scss %}
"Times New Roman", '../images/logo.png', red, sans-serif
{% endhighlight %}
```

Sass (just like CSS) doesn't differentiate between strings with quotes and strings without quotes. The only exception is when dealing with some specific CSS values like `sans-serif` or `inherit` which should not be quoted. Colour names also have special meaning in Sass and should be unquoted.

In all other cases, strings should be quoted and I favour single quotes because they're easier to type and I personally find them neater to look at.

Colors

Colors come in many forms such as named colours, hex, `rgb` and `hsl`.

Colours have their own data type because they can be manipulated by color functions and be added and subtracted
from each other to produce new colours.

```
{% highlight scss %}
$named-color: black
$hex-color: #000
$rgba-color: rgba( 0, 0, 0, 1 )
$hsla-color: hsl( 0, 0%, 0%, 1 )
{% endhighlight %}
```

Boolean

Booleans are either `true` or `false` and aren't particularly useful for setting values for CSS properties but are very useful when combined with logical flow control whilst building complex components, frameworks or library code. There's a whole video coming up on loops and flow control - keep your eye out for that to see a good use case for boolean variables

Null

`null` is a special data type which represents and empty value. If a variable contains a null value, applying this to a CSS property will cause that line of code to be removed at compile time:

```
{% highlight scss %}
$font-size: null;
$line-height: 1.2;

.title {
font-size: $font-size;
line-height: $line-height;
}

/* compiles to
.title {
line-height: 1.2;
}
*/
{% endhighlight %}
```

Lists

Lists contain multiple values separated by spaces or commas. They are used to represent shorthand declarations like `margin` and `padding` or for font stacks. You can have a list that contains other variables or even a list of lists.

```
{% highlight scss %}
$space-separated-list: 0 0 20px 0;
$comma-separated-list: Helvetica, Arial, sans-serif;
$variable-list: $space-separated-list, $comma-separated-list;
{% endhighlight %}
```

Maps

For more complex lists of variables, one final data type is “map” variables. These contain a store of key and value pairs like a JavaScript object or Ruby hash and are a very powerful new feature of Sass 3.3.

```
{% highlight scss %}
$font-sizes: (
h1: 50px,
h2: 36px,
h3: 24px,
h4: 20px,
p: 18px
);
{% endhighlight %}
```

Because there’s a lot discuss regarding maps, we’ll cover them in detail in their own video in Episode 13.

So, with all these different types of variable values, what can we do with them?

Sass Maths

The different types of Sass variables can be used as they are, or we can apply a number of different mathematical operations to them to make their use more flexible.

With whole numbers or decimals, we can add, subtract, multiply, divide and modulo (which returns the remainder after a division).

When using mathematical operations with length values in Sass (values like 10px or 2em), we do need to be mindful of the length units as many units are incompatible.

I’ve set up a local project with my Sass watching for changes on the command line. I’ve got Sublime Text open with the `style.scss` file on the left and the compiled `style.css` file on the right.

We’ll use this setup to demonstrate some Sass maths and show the compiled CSS each time a change is made.

Lengths with matching units can be added and subtracted from each other. However, lengths with matching units can’t be multiplied together.

```
{% highlight scss %}
$a: 20px;
$b: 10px;

div { width: $a + $b }
div { width: $a - $b }
{% endhighlight %}
```

Dividing lengths with the same units results in a number with no units. This can be a handy trick for stripping the units from any length value so you can perform further Sass operations on the plain number value.

```
{% highlight scss %}
$a: 100px;
$b: 20px;

$number: $a / $b; // 5
{% endhighlight %}
```

When dealing with mixed units such as `px` and `em`, Sass will throw an error when trying to perform operations on them.

```
{% highlight scss %}
$a: 20em;
$b: 10px;

div {
padding: $a + $b; // error
}
{% endhighlight %}
```

If you do need to perform calculations with mixed units, this can actually be done in normal CSS with the `calc()` function which is supported in IE9+.

When working with mathematical operations in Sass, it’s always a good practice to leave whitespace around the operator. This makes the code more readable and reduces the risk of confusing mathematical operators for other normal CSS characters like hyphens or forward slashes.

Sass variables and maths are a great combination and can really help to add meaning and clarity to your code. And instead of the result of a calculation being declared as a value, you can see the maths write there in the code. I’m not a huge fan of maths in general but Sass certainly sweetens the deal.