In a previous video about creating custom functions in Sass, we created a function that converted pixel values to rem values by manipulating their units.

We can create our own functions for all sorts of applications in Sass but there are also a number of functions built into the Sass library which we can leverage to make our functions, mixins, loops and logic more robust and more comprehensive.

In this video we're going to look at a couple of unit functions from the Sass library and show how they can be used to enhance our typographic functions from the previous video.

Converting Units

In the Creating Custom Functions video we created a custom `px-to-rem` function that - perhaps unsurprisingly - converts a pixel value into a `rem` value.

We started by returning the result of dividing a pixel value by the base font size and then multiplying by 1rem.

```
$base-font-size: 16px;

@function px-to-rem( $value ) {
    @return ( $value / $base-font-size ) * 1rem;
}
```

We then made this function slightly more flexible by accommodating pixel values or plain numbers passes as arguments to the function by first running it through a function that strips the units.

```
@function strip-units( $length ) {
    @return $length / ( ( $length * 0 ) + 1 );
}
@function px-to-rem( $value ) {
    $value: strip-units( $value );
    $base-font-size: strip-units( $base-font-size );

    @return ( $value / $base-font-size ) * 1rem;
```

```
}
```

This is a fairly comprehensive function - and it works - but we can leverage a couple of additional Sass functions to make it even more robust and handle even more use cases.

The Unitless Function

Let's first take a look at the `unitless` function. This is a built-in Sass function that returns `true` or `false` depending on whether a value has units.

If we take a look at the
Sass Function Reference for unitless
we'll see a description of the function and a couple of examples of its use.

If we call `unitless` and pass in the value `100` the function returns `true` because `100` has no units - ie. it is unitless.

If we call `unitless` and pass in the value `100px` the function returns `false` because `100px` does have units - ie. it's not unitless.

The function will also handle errors and raise an argument error if we try to pass anything other than a number into the function.

So that's how the unitless function works but how would this be useful to us and our `px-to-rem` function?

Well, currently our function strips units from any value passed in and then converts the result into rems. But if we pass in a plain number value then we don't need to remove any units (because it doesn't have any).

```
@function px-to-rem( $value ) {
    @if not unitless( $value ) {
        $value: strip-units( $value );
    }
    $base-font-size: strip-units( $base-font-size );

    @return ( $value / $base-font-size ) * 1rem;
}
```

So we can modify our function to check if the value pass in is unitless or not. If it is, we can go ahead and convert it to rems. If not, we can first strip the units and then convert to rems.

The Unit Function

Another useful Sass function for working with units is the `unit()` function. This can help us when we need to perform different functionality depending on what type of unit a value has.

As we did last time, let's have a look at the Sass Function Reference for the unit function.

This function returns a string containing the unit associated with a number. If we call the `unit` function and pass in `100` the result is an empty string. If we call the function with `100px` the result is `"px"`. If we pass a mathematical expression to the unit function we get a complex unit.

We can use this `unit` function in a similar way to our `px-to-rem` function but with a slightly different use case.

Instead of converting numbers into `rem` values for use in font-sizes, paddings, margins etc. we can create a function that converts length values into `em` for use in media query breakpoints. This is often a desirable way to write your media queries so they apply as the size of the text changes as well as when other factors like browser width and height change.

Let's create a functions called `bp-to-em` which converts flexible values to `em` for use in breakpoints. This and the `px-to-rem` function are inspired by similar functionality baked into Zurb's Foundation framework.

```
@function bp-to-em( $value ) {
    @if unit( $value ) == 'px' or unitless( $value ) {
        $value: px-to-rem( $value );
    }
    @return strip-units( $value ) * 1em;
}

@media screen and ( max-width: bp-to-em( 300px ) ) {
    body { background:red }
}
```

Our `bp-to-em` function takes one argument: the value we want to convert.

The first thing we do is check what units this value has. If it has `px` units or has no units at all, we can modify our value and convert it to rems using our existing function.

Otherwise, if we pass in any other kind of value we'll strip the units and then multiply by `1em` to convert the number to `em`.

We can then use this function within our media queries and any unitless or pixel value will be converted to a relative `em` unit.

There are a number of functions like `unit` and `unitless` built into the Sass libarry and combining them together with custom functions and if/else logic like this can be a powerful way of creating your own library of helper functions for use across your projects.

If you're interested in a handful of other U-functions have a search through the Sass Function Reference for `unquote` and `unique_id` and see what else you can do!