In this series we've looked at all the fundamental, powerful features of Sass from variables and nesting to functions and flow control.

Further to these features, Sass has a whole host of useful functions and utilities which turns CSS from a purely declarative language into something more like a scripting language.

In this short episode we're going to look at one such feature and some practical applications: the Sass `random()` function.

How Random

Many programming languages have the ability to generate random numbers. These are useful for building games, generating dummy data or creating algorithms.

Sass is a slightly special case because even though it can generate a random number at compile time, this number will remain constant until the styles are recompiled again.

To illustrate this, let's take a look at using the `random` function in a simple example.

I've got a box here with a minimum width and height so we can see something on the rendered page to the left.

Using `random`, let's generate some random dimensions for the box.

```
.box {
    min-width:50px;
    min-height:50px;

    width: random(500) * 1px;
    height: random(500) * 1px;
}
```

The random function takes a single argument for the upper limit of the random number to be generated. When using this function, the lower limit is always 1.

If we save this code and compile, we'll see the randomly generated dimensions for the box. If I refresh the page... the dimensions remain

exactly the same.

This is because the random number is only generated at compile time.

If I force the code to recompile by saving... two new numbers are generated and the box dimensions change once more.

Practical Use Cases

This functino is simple to use but how likely is it that you'll ever need to generate a random number in CSS?

In the past I've used this for generating random colours for a grid of boxes when prototyping an image grid layout. Instead of searching for or creating a whole series of different images, I used Sass to generate a whole host of different coloured boxes.

Combining a `@for` loop with `rgb` backgrounds, each with a random number from 1 to 255, provides some interesting effects.

```
@for $i from 1 through 100 {
    .box:nth-child( #{ $i } ) {
        background-color: rgb( random( 255 ), random( 255 ), random( 255 ) );
    }
}
```

Another potential usecase that I discovered recently is creating particle effects.

https://css-tricks.com/animate-different-end-states-using-one-set-css-keyframes/

This uses the `random` function to randomise the starting position of a series of particles and then animate them with random animation duration and animation delay to create a very impressive effect.