Table of Contents

Mixins and the & Selector

HTML Code

//Copy the code in your index.html

```
<link rel="stylesheet" type="text/css" href="main.css">

<div id="notecards">

  <div class="notecard">

    <div class="front">      <span class="word">Titanosaur</span>    </div>

    <div class="back">

      <div class="definition">

        122 ft, 70 tons, Herbivore, Patagonia about 100 to 95 million years ago

        <div class="photo"></div>

      </div>

    </div>

  </div>

</div>
```

SCSS Code

//Copy the code in your main.scss

```
@mixin stripes($direction, $width-percent, $stripe-color, $stripe-background: #FFF) {
  background: repeating-linear-gradient(
    $direction,
    $stripe-background,
    $stripe-background ($width-percent - 1),
    $stripe-color 1%,
    $stripe-background $width-percent  );
}
@mixin transform($transformation) {
  transform: $transformation;
  -webkit-transform: $transformation;
  -moz-transform: $transformation;
  -ms-transform: $transformation;
  -o-transform: $transformation;
}
@mixin photo-content {
  object-fit: cover;
}

//Add your own mixins here

.notecard {
  width: 300px;
  height: 180px;
```

```
border: 1px solid black;
display: inline-block;
margin: 20px;
font-family: Roboto, sans-serif;
box-shadow: 1px 1px 2px 2px rgba(0,0,0,.2);

.front, .back {
  width: 100%;
  height: 100%;
  position: absolute;
}
.front {
z-index: 3;
font-size: 20px;
  .word {
    display: block;
    text-align: center;
    position: relative;
    top: 40%;
  }
}
.back {
z-index: 1;
word-wrap: break-word;
line-height: 1.6;
  .definition {
```

```
    width: 100%;
    height: 100%;
    .photo {
      width: 60%;
      margin: 0px auto;
    }
  }
 }
}
```

The & Selector in Nesting

Recall that, in CSS, a pseudo-element is used to style parts of an element, for example:

- Styling the content `::before` or `::after` the content of an element.
- Using a pseudo class such as `:hover` to set the properties of an element when the user's mouse is touching the area of the element.

In Sass, the & character is used to specify exactly where a parent selector should be inserted. It also helps write psuedo classes in a much less repetitive way.

For example, the following Sass:

```
.notecard{
  &:hover{
      @include transform (rotatey(-180deg));
    }
  }
```

will compile to the following CSS:

```
.notecard:hover {
  transform: rotatey(-180deg);
}
```

1. In **main.scss**, inside of `.notecard`, nest and invoke the following & selector:

```
&:hover{
  @include transform (rotatey(-180deg));
}
```

Compile to see your changes in the browser and inspect them in the output of **main.css**. Hover over your card and watch the contents rotate! We will be styling it further in the next exercises.

What is a Mixin?

In addition to variables and nesting, Sass has multiple constructs that reduce repetition.

In Sass, a *mixin* lets you make groups of CSS declarations that you want to reuse throughout your site.

The notation for creating a mixin is as follows:

```
@mixin backface-visibility {
  backface-visibility: hidden;
  -webkit-backface-visibility: hidden;
  -moz-backface-visibility: hidden;
  -ms-backface-visibility: hidden;
  -o-backface-visibility: hidden;
}
```

Note: Mixin names and all other Sass identifiers use hyphens and underscores interchangeably. The following code:

```
.notecard {
.front, .back {
    width: 100%;
    height: 100%;
    position: absolute;

    @include backface-visibility;
  }
}
```

is equivalent to the following CSS:

```
.notecard .front, .notecard .back {
  width: 100%;
  height: 100%;
  position: absolute;

   backface-visibility: hidden;
  -webkit-backface-visibility: hidden;
  -moz-backface-visibility: hidden;
  -ms-backface-visibility: hidden;
  -o-backface-visibility: hidden;
}
```

1. At the top of **main.scss**, create the following mixin:

```
@mixin backface-visibility {
  backface-visibility: hidden;
  -webkit-backface-visibility: hidden;
  -moz-backface-visibility: hidden;
  -ms-backface-visibility: hidden;
  -o-backface-visibility: hidden;
}
```

Next, include the mixin inside the joint `.front, .back` selector:

```
.front, .back {
```

```
  width: 100%;
  height: 100%;
  position: absolute;
  @include backface-visibility;
}
```

Note: `backface-visibility` is a vendor specific property that is perfect for mimicking an index card with a front and a back.

https://developer.mozilla.org/en-US/docs/Web/CSS/backface-visibility

https://www.w3schools.com/CSSref/css3_pr_backface-visibility.asp

https://css-tricks.com/almanac/properties/b/backface-visibility/

Compile to see your changes in the browser and inspect them in **main.css**. Hover over you card and see the effects of a hidden backface in motion.

Mixins: Arguments

Mixins also have the ability to take in a value.

An *argument*, or parameter, is a value passed to the mixin that will be used inside the mixin, such as `$visibility` in this example:

```
@mixin backface-visibility($visibility) {
  backface-visibility: $visibility;
  -webkit-backface-visibility: $visibility;
  -moz-backface-visibility: $visibility;
  -ms-backface-visibility: $visibility;
  -o-backface-visibility: $visibility;
}
```

In fact, you should only ever use a mixin if it takes an argument. We will learn more about this in a later exercise.

The syntax to pass in a value is as follows:

```
@include backface-visibility(hidden);
```

In the code above, `hidden` is passed in to the `backface-visibility` mixin, where it will be assigned as the value of its argument, `$visibility`.

1. In **main.scss**, modify `backface-visibility` mixin to take in a parameter, like so:

```
@mixin backface-visibility($visibility) { //Add an argument
  backface-visibility: $visibility;
  -webkit-backface-visibility: $visibility;
  -moz-backface-visibility: $visibility;
  -ms-backface-visibility: $visibility;
  -o-backface-visibility: $visibility;
}
```

Pass in the value of `hidden` inside the `.front, .back` selector:

```
.front, .back {
    @include backface-visibility(hidden);
}
```

Compile to see your changes in the browser and inspect them in **main.css**.

Default Value Arguments

Mixin arguments can be assigned a *default value* in the mixin definition by using a special notation.

A default value is assigned to the argument if no value is passed in when the mixin is included. Defining a default value for each argument is optional.

The notation is as follows:

```
@mixin backface-visibility($visibility: hidden) {
   backface-visibility: $visibility;
  -webkit-backface-visibility: $visibility;
  -moz-backface-visibility: $visibility;
  -ms-backface-visibility: $visibility;
  -o-backface-visibility: $visibility;
}
```

In the example above, if no value is passed in when `backface-visibility` is included, `hidden` would be assigned to all properties.

1. In **main.scss** add a default value of `hidden` to the argument in `backface-visibility`:

```
@mixin backface-visibility($visibility: hidden) {
  // Backface properties
}
```

Inside of `.front, .back` remove the argument you previously passed to `backface-visibility`.

Compile to see your changes in the browser and inspect them in the output of **main.css**.

Mixin Facts

In general, here are 5 important facts about arguments and mixins:

1. Mixins can take multiple arguments.
2. Sass allows you to explicitly define each argument in your `@include` statement.
3. When values are explicitly specified you can send them out of order.
4. If a mixin definition has a combination of arguments with and without a default value, you should define the ones with no default value first.
5. Mixins can be nested.

Here are some concrete examples of the rules:

```
@mixin dashed-border($width, $color: #FFF) {
  border: {
     color: $color;
     width: $width;
     style: dashed;
  }
}

span { //only passes non-default argument
    @include dashed-border(3px);
}

p { //passes both arguments
    @include dashed-border(3px, green);
}

div { //passes out of order but explicitly defined
   @include dashed-border(color: purple, width: 5px);
}
```

In the example above, the color of the border of `span` elements would be white, the border of `paragraph` elements would be green, while the `div` elements would have a thicker purple border.

Instructions

1. Practice makes perfect! In **main.scss** inside of the `.back` selector, include the following:

```
@include transform(rotatey(-180deg));
```

Compile to see your changes in the browser and inspect them in the output of **main.css**.

2. In addition to flipping the back, we also want to make sure that the notecard preserves a 3D effect during all of its transformations. At the top of **main.scss**, add the following mixin:

```
@mixin transform-style($style){
   transform-style: $style;
  -moz-transform-style: $style;
  -o-transform-style: $style;
  -ms-transform-style: $style;
  -webkit-transform-style: $style;
}
```

Invoke the mixin inside of `.notecard`, add the following:

```
@include transform-style(preserve-3d);
```

Compile to see your changes in the browser and inspect them in the output of **main.css**.

3. Last but not least, add the following at the top of main.scss:

```
@mixin transition($time){
  transition: $time;
  -webkit-transition: $time;
  -moz-transition: $time;
  -o-transition: $time;
}
```

Add the following inside the `.notecard` selector:

```
@include transition(0.4s);
```

Compile to see your changes in the browser and inspect them in the output of **main.css**.

List Arguments

Sass allows you to pass in multiple arguments in a list or a map format.

For example, take the multiple properties needed to create the college-ruled stripes in the back of our notecard.

```
@mixin stripes($direction, $width-percent, $stripe-color, $stripe-background:
#FFF) {
  background: repeating-linear-gradient(
    $direction,
    $stripe-background,
    $stripe-background ($width-percent - 1),
    $stripe-color 1%,
    $stripe-background $width-percent
  );
}
```

In this scenario, it makes sense to create a map with these properties in case we ever want to update or reference them.

```
$college-ruled-style: (
    direction: to bottom,
    width-percent: 15%,
    stripe-color: blue,
    stripe-background: white
);
```

When we include our mixin, we can then pass in these arguments in a map with the following `...` notation:

```
.definition {
      width: 100%;
      height: 100%;
      @include stripes($college-ruled-style...);
 }
```

Note: Remember to always prioritize readability over writing less code. This approach is only useful if you find it adds clarity to your codebase.

1. At the top of **main.scss**, make a new list variable:

```
$stripe-properties: to bottom, 15%, blue, white;
```

Include the `stripes` mixin inside of the `.definition` selector:

```
.definition {
    @include stripes($stripe-properties...);
  }
```

Compile to see your changes in the browser and inspect them in the output of **main.css**.

String Interpolation

In Sass, **string interpolation** is the process of placing a variable string in the middle of two other strings.

In a mixin context, interpolation is handy when you want to make use of variables in selectors or file names. The notation is as follows:

```
@mixin photo-content($file) {
  content: url(#{$file}.jpg); //string interpolation
  object-fit: cover;
}

//....

.photo {
  @include photo-content('titanosaur');
  width: 60%;
  margin: 0px auto;
  }
```

String interpolation would enable the following CSS:

```
.photo {
  content: url(titanosaur.jpg);
  width: 60%;
  margin: 0px auto;
}
```

1. In **main.scss**, edit the `photo-content` mixin as follows:

```
@mixin photo-content($file) {
  content: url(#{$file}.jpg);
  object-fit: cover;
}
```

Include the mixin inside the `.photo` selector and pass in a file name of `'titanosaur'`:

```
.photo {
        @include photo-content('titanosaur');
        width: 60%;
        margin: 0px auto;
 }
```

Compile to see your changes in the browser and inspect them in the output of **main.css**.

The & Selector in Mixins

Now it's time to tie in how the & selector plays into mixins.

Sass allows & selector usage inside of mixins. The flow works much like you think it would:

1. The & selector gets assigned the value of the parent at the point where the mixin is included.
2. If there is no parent selector, then the value is null and Sass will throw an error.

```
@mixin text-hover($color){
  &:hover {
      color: $color;
  }
}
```

In the above, the value of the parent selector will be assigned based on the parent at the point where it is invoked.

```
  .word { //SCSS:
    display: block;
    text-align: center;
    position: relative;
    top: 40%;
    @include text-hover(red);
  }
```

The above will compile to the following CSS:

```
  .word{
    display: block;
    text-align: center;
    position: relative;
    top: 40%;
  }
  .word:hover{
    color: red;
  }
```

Notice that the mixin inherited the parent selector `.word` because that was the parent at the point where the mixin was included.

Instructions

1. In **main.scss**, create the new mixin:

```
@mixin hover-color($color) {
   &:hover{
       color: $color;
   }
}
```

Include it inside the `.word` selector:

```
@include hover-color(red);
```

Compile to see your changes in the browser and inspect them in the output of **main.css**. In particular, hover over the word "Titanosaur" in the browser and watch it turn red before it flips over.

Generalizations

This exercise has introduced you to some of the most powerful concepts Sass has to offer!

- *Mixins* are a powerful tool that allow you to keep your code DRY. Their ability to take in arguments, assign default values to those arguments, and accept said arguments in whatever format is most readable and convenient for you makes the mixin Sass's most popular directive.
- The & selector* is a Sass construct that allows for expressive flexibility by referencing the parent selector when working with CSS psuedo elements and classes.
- *String interpolation* is the glue that allows you to insert a string in the middle of another when it is in a variable format. Its applications vary, but the ability to interpolate is especially useful for passing in file names.

Let's keep up the awesome job in the next exercise, where we will learn about functions, arithmetic, and color operations in Sass!