strategy | consulting | digital | technology | operations

SASS

WEEK 2 DAY 1

accenture

AGENDA

- Introduction
- Nesting
- Variables
- Partials
- Imports
- Mixins
- Operators
- Inheritance
- Functions

Common CSS Problems

Duplication

Colors

Cascading

Calculations

Import

Duplication Problem

- Any styling required for a group of elements needs to be defined separately.
- This can lead to a large amount of duplication
- A small change in the way the elements are referenced results in a large refactoring.
  - E.g. adding a class to the 'article' element
- Many CSS3 declarations require vendor prefixes.
- Applying these vendor prefixes to the properties makes the code clumsy and difficult to read

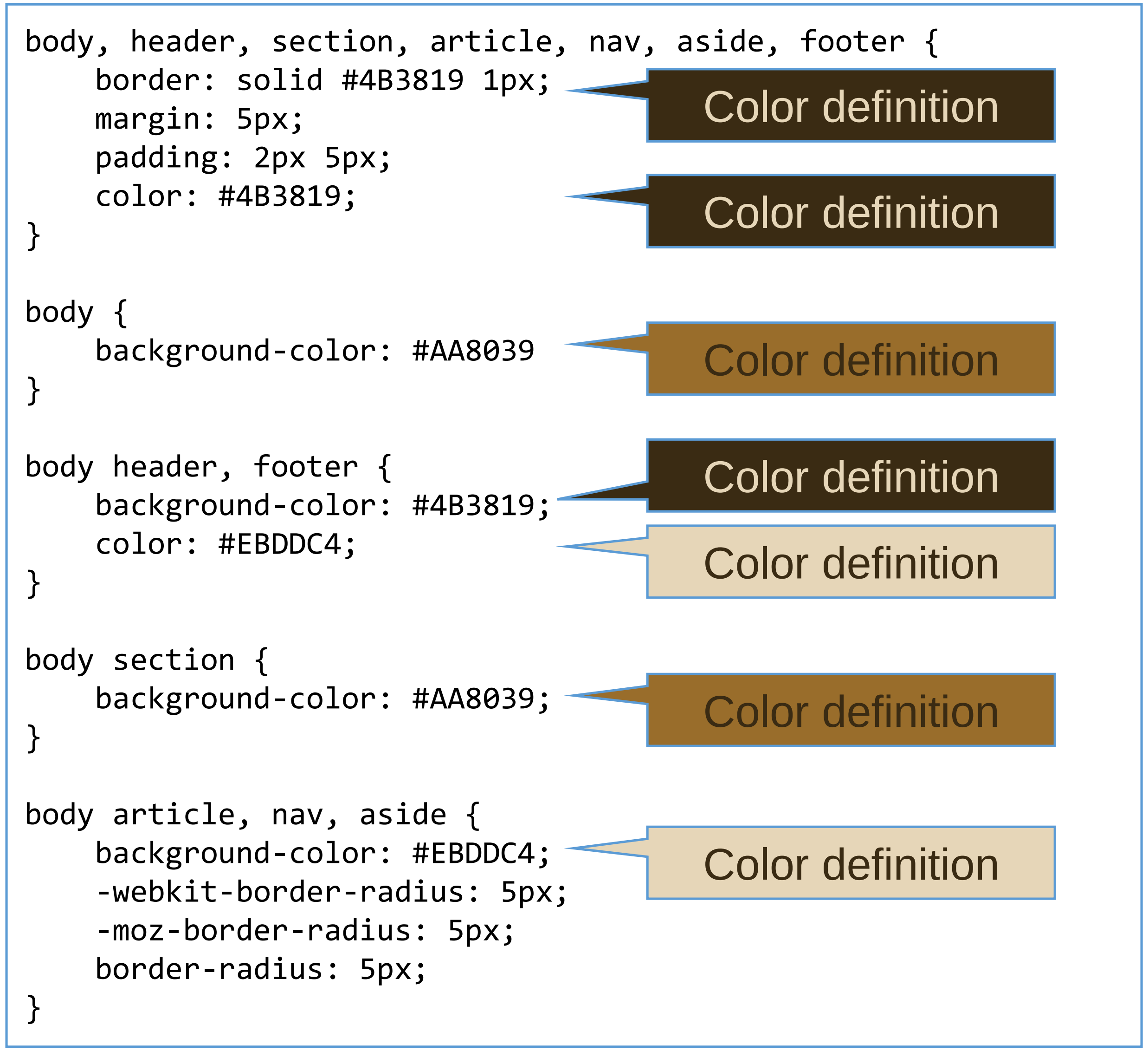
```css
body, header, section, article, nav, aside,
footer {
    border: solid #4B3819 1px;
    margin: 5px;
    padding: 2px 5px;
    color: #4B3819;
}

body header, footer {
    background-color: #4B3819;
    color: #EBDDC4;
}

body section {
    background-color: #AA8039;
    -webkit-border-radius: 5px;
    -moz-border-radius: 5px;
    border-radius: 5px;
}

body article, nav, aside {
    background-color: #EBDDC4;
    -webkit-border-radius: 5px;
    -moz-border-radius: 5px;
    border-radius: 5px;
}
```

Duplicate property declarations

accenture

Color Problem

- Web pages present their content according to a pre-defined theme and typically a theme consists of less than 10 different colors and shades.
- Colors must be defined individually for each element group
- This leads to a large amount of duplication
- Changing the entire color theme requires significant refactoring
- Using search and replace to change colors can lead to errors


```
body, header, section, article, nav, aside, footer {
    border: solid #4B3819 1px;
    margin: 5px;
    padding: 2px 5px;
    color: #4B3819;
}

body {
    background-color: #AA8039
}

body header, footer {
    background-color: #4B3819;
    color: #EBDDC4;
}

body section {
    background-color: #AA8039;
}

body article, nav, aside {
    background-color: #EBDDC4;
    -webkit-border-radius: 5px;
    -moz-border-radius: 5px;
    border-radius: 5px;
}
```

Cascading Problem

- Structured HTML leads to structured CSS
- Specificity rules tend to encourage the creep of unique identifiers
- CSS can get very complex very quickly
- Manual management of specificity on medium or large projects becomes difficult
- Changes in HTML structure can have significant refactoring implications

```
body, header, section, article, nav, aside, footer {
    border: solid #4B3819 1px;
    margin: 5px;
    padding: 2px 5px;
    color: #4B3819;
}

body {
    background-color: #AA8039
}

body > header {
    background-color: #AA8039;
}

body header, footer {
    background-color: #4B3819;
    color: #EBDDC4;
}

body section {
    background-color: #AA8039;
}
...

}
```

Specificity: 0-0-2

Specificity: 0-0-2

This takes precedence because of its position

accenture

Calculation Problem

- Calculations are often useful to define sizes relative to other elements such as window size or resolution
- The current candidate proposal for CSS is for a calc() function to be used in place of component values
- It is an experimental method that's not yet widely implemented
- Only supports a limited set of operators; +, -, *, /

```
body, header, section, article, nav, aside, footer {
    border: solid #4B3819 1px;
    margin: 5px;
    padding: 2px calc(2% + 5px);
    color: #4B3819;
}

body {
    background-color: #AA8039
}

body > header {
    background-color: #AA8039;
}

body header, footer {
    background-color: #4B3819;
    color: #EBDDC4;
}

body section {
    background-color: #AA8039;
}
...

}
```

Simple operator but with a mix of units

Importing Problem

- CSS already allows you to include a stylesheet from within another stylesheet using the @import rule
- The imports must be at the very top of the document
- Stylesheets referenced in this way will be loaded synchronously which may slow the initial loading time
- Where <link> and @import are used together
- There may be support issues with older browsers

index.html

```
<head>
  <link rel='stylesheet' type='text/css' href='one.css'>
  <link rel='stylesheet' type='text/css' href='two.css'>
</head>

...
```

Both stylesheets would load concurrently

index.html

```
<head>
  <link rel='stylesheet' type='text/css' href='one.css'>
</head>

...
```

The stylesheets would load sequentially

one.css

```
@import url('two.css');

selector { property: value; }

...
```

CSS Preprocessor

- A preprocessor is a program that takes one type of data and converts it to another type of data.
- CSS pre-processor is a language on top of CSS introducing powerful features like variables, mixins, nesting, etc,.
- Few popular CSS processor include SASS, LESS, Stylus, etc,.

SASS Introduction

- Sass or ‘Syntactically Awesome StyleSheet’ is a CSS pre-processor.
- Stylesheets written using Sass syntax are processed into CSS before use.
- Sass is a superset of CSS
  – This means CSS code is valid Sass code
- There are two versions of Sass syntax
  – SCSS: “Sassy CSS”
  – SASS: indented syntax (older)
- Only the compass *flavour* of Sass requires Ruby to compile.
- There are a number of ways to invoke Sass…

Invoking SASS

On the Client

- Include a script reference to Sass.js in the HTML document

On the command line

- Sass can be installed via the Node Package Manager and invoked via the CLI

Programmatically

- A JavaScript API can be called from a script to compile Sass into CSS

Automated compilation

- Using a tools likes Grunt ,Gulp or Webpack

Online compilers

Nesting Selectors

- Nesting is the process of placing selectors inside the scope of another selector:
- In Sass, it's helpful to think of the scope of a selector as any of the code between its opening { and closing } curly brackets.
- Selectors that are nested inside the scope of another selector are referred to as children. The former selector is referred to as the parent. This is just like the relationship observed in HTML elements.

```
.parent {
            color: blue;
 }
.parent .child {
             font-size: 12px;
}
```

```
.parent {
            color: blue;
            .child {
                        font-size: 12px;
                    }
 }
```

accenture

Nested Rules

Cascade Problem SOLVED

```
$base-color: #AA8039;

body {
  color: darken($base-color, 25%);
  background-color: $base-color;

  header, footer {
    background-color: darken($base-color, 25%);
    color: lighten($base-color, 40%);
  }

  section {
    background-color: lighten($base-color, 20%);

    article, nav, aside {
      background-color: lighten($base-color, 40%);
    }
  }
}
```

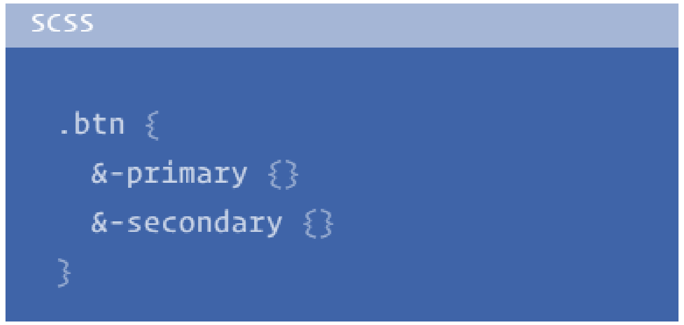
```
body {
  color: #4b3819;
  background-color: #aa8039;
}
body header,
body footer {
  background-color: #4b3819;
  color: #ebddc4;
}
body section {
  background-color: #d2b077;
}
body section article,
body section nav,
body section aside {
  background-color: #ebddc4;
}
```

- Sass allows selections to be nested
- It follows the HTML nesting more closely

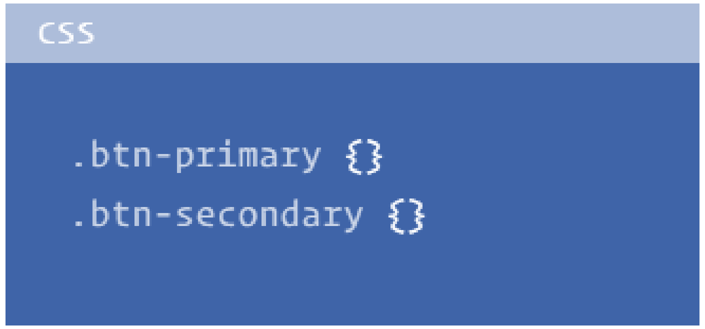
- Specificity issues are more easily resolved
- '&' can be used to select the parent

Ampersand (&) Selector

- Ampersand is used to reference the parent selector.

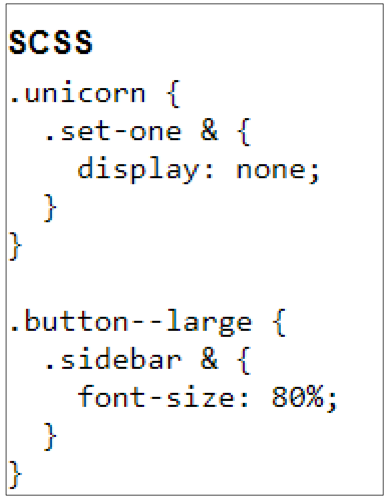


```
SCSS

.btn {
  &-primary {}
  &-secondary {}
}
```



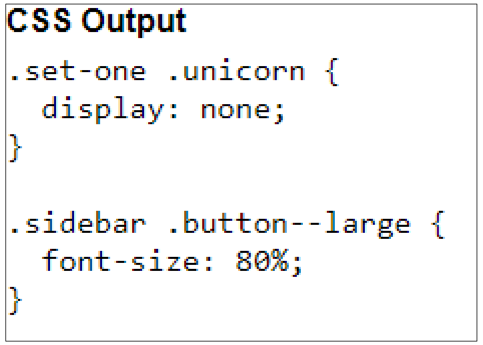
```
CSS

.btn-primary {}
.btn-secondary {}
```

- The trailing ampersand does the same thing as the prepended one – it switches out with the parent element when outputted.



SCSS

```
.unicorn {
  .set-one & {
    display: none;
  }
}

.button--large {
  .sidebar & {
    font-size: 80%;
  }
}
```



CSS Output

```
.set-one .unicorn {
  display: none;
}

.sidebar .button--large {
  font-size: 80%;
}
```

Variables

- Variable types are flexible
  - Numbers with or without units
  - Strings
  - Complex types
- Variables can be used in other places
- Variables are actually treated as constants in Sass
  - The colon : is used as a single use assignment
  - There is no equals = assignment

```
$growth-factor: 2;
$big-quote: 3em;

$wall-color: PaleGreen;
$main-fonts: Arial, Helvetica, sans-serif;

$bright-box: border-image: url(../res/button.gif) 19 fill stretch;
```

```
$resources: "../res";

$bright-box: border-image: url(${resources}/button.gif) 19 fill stretch;
```

```
// This will work
$left-edge: 20px;
div { margin-left: $left-edge + 1; }

// This won't work
$left-edge: $left-edge + 1;
div { margin-left: $left-edge; }
```

Variables

```
$color-base: #AA8039;
$color-dark: #4B3819;
$color-light: #EBDDC4;
$padding-std: 2px;
$margin-std: $padding-std + 3;

body, header, section, article, nav, aside, footer {
    border: solid $color-dark 1px;
    margin: $margin-std;
    padding: $padding-std $padding-std + 3px;
    color: $color-dark;
}

body {
    background-color: $color-base;
}

body header, footer {
    background-color: $color-dark;
    color: $color-light;
}

body section {
    background-color: $color-base;
}
```

Units are implied

```
body, header, section, article, nav, aside, footer {
    border: solid #4b3819 1px;
    margin: 5px;
    padding: 2px 5px;
    color: #4b3819;
}

body {
    background-color: #aa8039;
}

body header, footer {
    background-color: #4b3819;
    color: #ebddc4;
}

body section {
    background-color: #aa8039;
}
```

Operators

Assignment Operator

- Sass uses the colon (:) operator to define a variable.
- Ex : $main-color: lightgray;

Arithmetic Operators

- You can use the basic + – * and / symbols for addition, subtraction, multiplication and division in SASS.
- '+' can be used for addition and string concatenation.
- The operators work for only number with compatible types.
- Ex :

```
h2 {
        font-size: 5px + 2em; // error incompatible units
        font-size: 5px + 2; // 7px
}
```

accenture

Operators

Equality Operator

- == And !=
- Ex:

```
@if(type-of($font) == string) {
   content: 'My font is: #{$font}.';}
 @else {
            content: 'Sorry, the argument #{$font} is a #{type-of($font)}.';
   }
```

Comparison Operator

- > , >=, <, <=
- Ex :

```
$padding: 50px;
h2{
@if($padding <= 20px) {
          padding: $padding;}
 @else {
          padding: $padding / 2;}}
}
```

Operators

Logical Operators

- and,or , not
- Ex :

```
@if (length($list-map) > 2 or length($list-map) < 5) {
   background-color: map-get($list-map, $btn-state);
 }
```

Interpolation

- #{ }

```
h2 {
        font-size: 16px / 24px // Outputs as CSS
        font-size: (16px / 24px) // Uses parentheses, does division
        font-size: #{$base-size} / #{$line-height}; // Uses interpolation, outputs as CSS
        font-size: $base-size / $line-height // Uses variables, does division
        opacity: random(4) / 5; // Uses a function, does division
        padding-right: 2px / 4px + 3px // Uses an arithmetic expression, does division
}
```

accenture

Operators

String

- Sass uses the addition operator () to concatenate strings.
  - If we add a quoted string (that's said before the + operator) to an unquoted string, the result is a quoted string.
  - If we add an unquoted string (that's said before the + operator) to a quoted string, the result is an unquoted string
- Ex :

```
        content: "My favorite language is "+ Sass;
        font: Arial + " sans-serif";
Results in,
        content: "My favorite language is Sass";
        Font: Arial sans-serif;
```

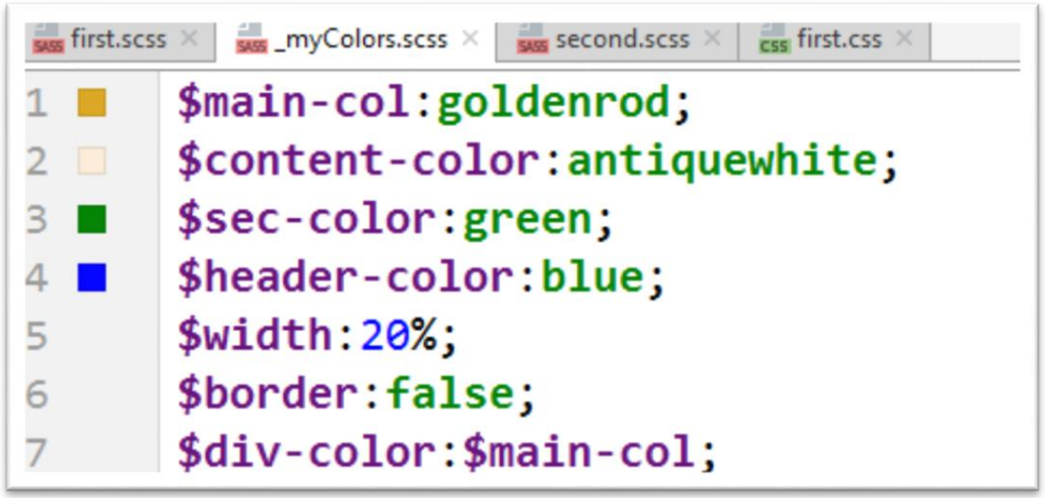
Color

- Arithmetic operators can be used on color values
- Ex :

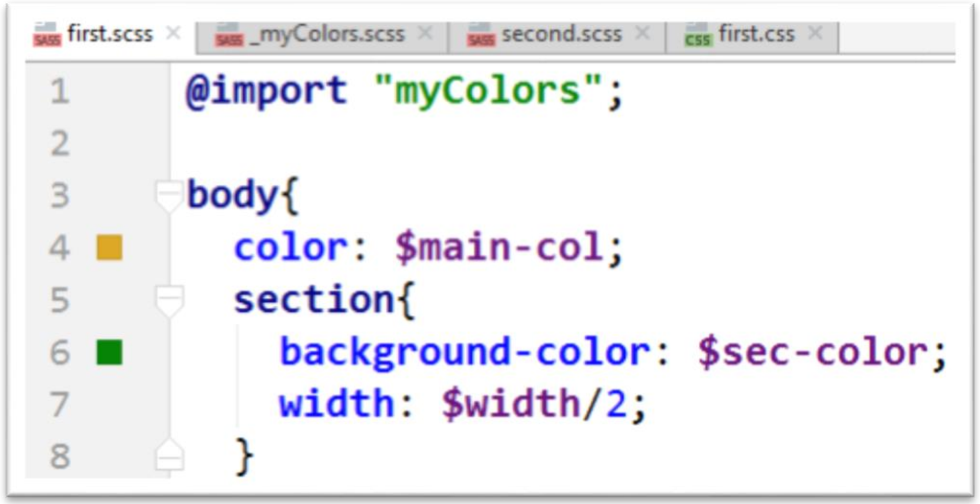
```
        color: #468499 + #204479;
        color: rgba(70, 132, 153, 1) + rgba(32, 68, 121, 1);
        color: rgba(70, 132, 153, .9) + rgba(32, 68, 121, .7); // error alpha channels must be equal
```

Partials

- Helps in creating modular and maintainable code.
- The partial files start with an _. The underscore symbol is an indicator to the compiler the file is not required to be compiled.



```scss
$main-col:goldenrod;
$content-color:antiquewhite;
$sec-color:green;
$header-color:blue;
$width:20%;
$border:false;
$div-color:$main-col;
```



```scss
@import "myColors";

body{
  color: $main-col;
  section{
    background-color: $sec-color;
    width: $width/2;
  }
```

Functions Galore

- In addition to the standard operands, Sass offers many other functions
- Math
  - Maximum, minimum, rounding up, rounding down, powers, roots, trigonometry functions, etc.
- String
  - URL encoding, regular expression pattern matching and replacement
- List
  - Find the length of a list and return values at given positions
- Type
  - Evaluate variables to test for specific types

- There's a whole subset of functions associated with color
- Colors can be defined as
  - RGB, RGBA, HSL, HSLA, HSV, HSVA
- Color channel function allow the extraction of information from a defined color
  - hue, lightness, alpha, red component, etc.
- Some operations are specifically concerned with color
  - saturate/unsaturate, adjust_color, mix, opacify, transparentize, etc.

Calculation Problem SOLVED

Sass Colors

```
$color-base: #AA8039;
$color-dark: darken($color-base, 25%);
$color-light: lighten($color-base, 40%);

body, header, section, article, nav, aside, footer {
    border: solid $color-dark 1px;
    margin: 5px;
    padding: 2px 5px;
    color: $color-dark;
}

body {
    background-color: $color-base;
}

body header, footer {
    background-color: $color-dark;
    color: $color-light;
}

body section {
    background-color: $color-base;
}
```

- Magic numbers in the code can be largely eliminated
- Theme color can be set in one place only
- Sass treats colors as HSL but they can be defined as using other models
- Standard math operands can also be used

Color Problem SOLVED

Mixins

```
@mixin rounded {
    -webkit-border-radius: 5px;
    -moz-border-radius: 5px;
    border-radius: 5px;
}

body, header, section, article, nav, aside, footer {
    border: solid #4B3819 1px;
    margin: 5px;
    padding: 2px 5px;
    color: #4B3819;
}

body {
    background-color: #AA8039
}

body header, footer {
    @include rounded
    background-color: #4B3819;
}

body section {
    @include rounded
    background-color: #AA8039;
}
```

- Sass allows us to "mix in" properties from one rule-set into another rule-set
- Allows to create reusable pieces of code
- The dot prefix in the name identifies it as a mixin
- It can be used with vendor prefixes to
  - Make the code cleaner
  - Help with future maintenance

Duplication Problem SOLVED

Parametric Mixins

```
@mixin rounded($radius: 5px) {
    -webkit-border-radius: $radius;
    -moz-border-radius: $radius;
    border-radius: $radius;
}

body, header, section, article, nav, aside, footer {
    border: solid #4B3819 1px;
    margin: 5px;
    padding: 2px 5px;
    color: #4B3819;
}

body {
    background-color: #AA8039
}

body header, footer {
    @include rounded
    background-color: #4B3819;
}

body section {
    @include rounded(3px)
    background-color: #AA8039;
}
```

- Mixins are able to take arguments
- We can, optionally, set a default value
- Mixins can also take multiple arguments that should be separated by semi-colons
  - Arguments can be arranged as comma-separated multi-parameter groups

```
.emboss($light; $dark) {
  border-top: solid $light 1px; border-left: solid $light 1px;
  border-bottom: solid $dark 1px; border-right: solid $dark 1px;
}
```

- Mixins can also behave more like functions and set a "return" value

```
 @mixin average($x, $y) { $average: (($x + $y) / 2); }

div {
  @include average(16px, 50px); // "call" the mixin
  padding: $average;    // use its "return" value
}
```

Importing with Sass

Import Problem SOLVED

- The @import directive is more flexible in Sass
  - It can placed anywhere it's required in the stylesheet
  - It can be used to import Sass or CSS if the document has a ".css" file extension
  - You can create partial Sass files that contain snippets that you can include in other Sass files.
  - A partial must be named with a leading underscore (e.g. _partial.scss).

Using namespaces and scoping

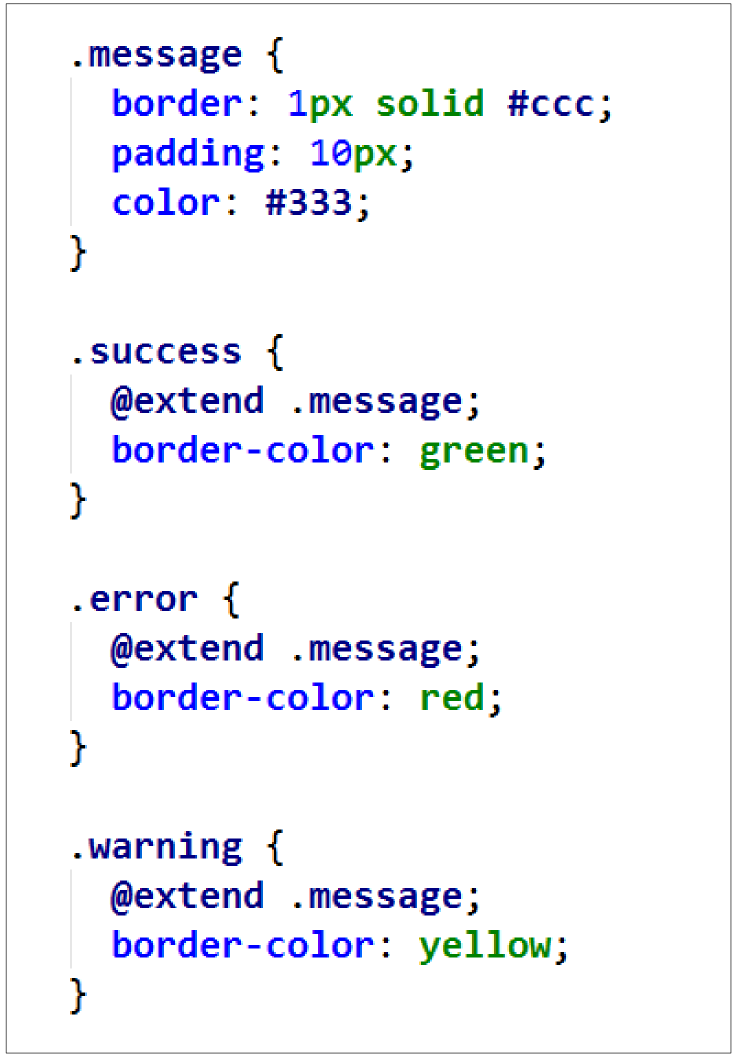
```
#core {
  @mixin button($param; $color) {
    margin: 5px;
    width: $param;
    height: $param / 2;
    background-color: $color;
  }

  #box2d {
    @mixin edge($color) {
      border: solid darken($color, 20%) 2px;
    }
  }

  #box3d {
    @mixin edge($color) {
      border-right: solid darken($color, 20%) 2px;
      border-bottom: solid darken($color, 20%) 2px;
      border-left: solid lighten($color, 20%) 2px;
      border-top: solid lighten($color, 20%) 2px;
    }
  }
}
```

- Mixins can be grouped
  - For greater organizational clarity
  - To introduce encapsulation
  - Ensuring name conflicts are avoided
- A mixin declared inside the namespace can use mixins and variables declared in its declaration and caller scope.
  - The declaration scope takes precedence over the caller scope.

```
.box1 {
  #core > @include button(100px, lightgreen);
  #core > #box2d > @include edge(lightgreen);
}

.box2 {
  #core > @include button(100px, lightblue);
  #core > #box3d > @include edge(lightblue);
}
```

Inheritance - @extend

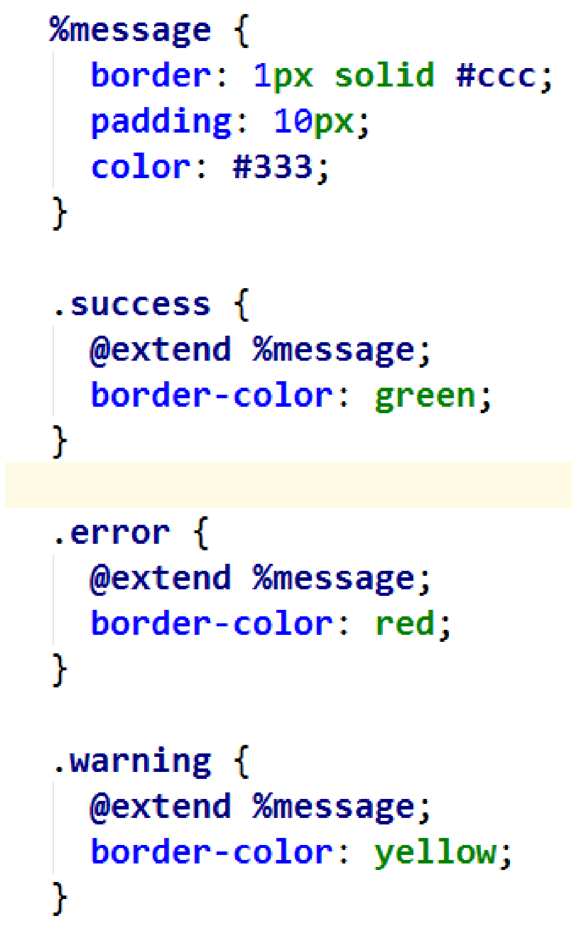
- **@extend** is a feature of Sass that allows classes to share a set of properties with one another.
- Though a similar result can be achieved using mixins. Mixins can do things that extend cannot though, namely take parameters and process/use them in the output.



```
.message {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

.success {
  @extend .message;
  border-color: green;
}

.error {
  @extend .message;
  border-color: red;
}

.warning {
  @extend .message;
  border-color: yellow;
}
```

Inheritance : Placeholders

- CSS code is not generated for the placeholders.



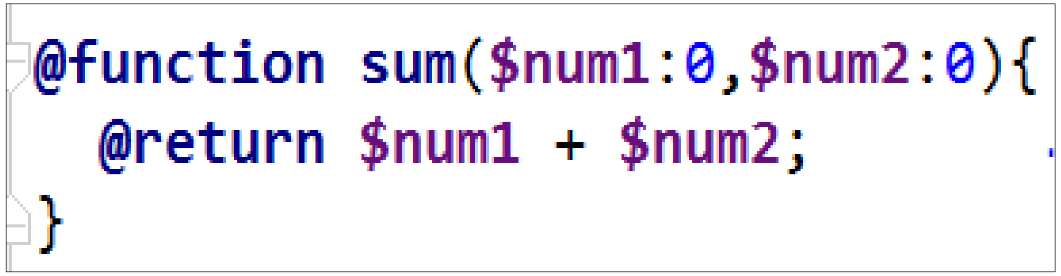
```
%message {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

.success {
  @extend %message;
  border-color: green;
}

.error {
  @extend %message;
  border-color: red;
}

.warning {
  @extend %message;
  border-color: yellow;
}
```

Custom Functions

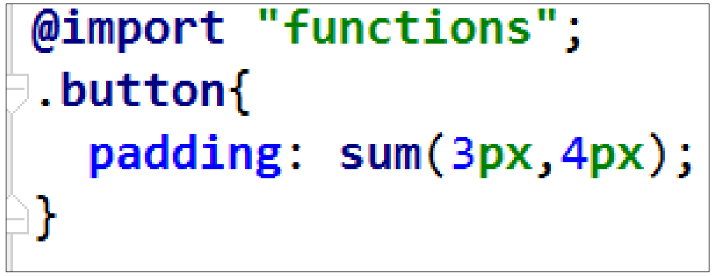
- Functions are blocks of code that return a single value of any Sass data type.
- Functions and mixins are very similar in nature.
- Functions return value instead of outputting lines of Sass the like mixins.
- To create a custom function you need two Sass directives, @function and @return.

```
@function function-name($args) {
		@return value-to-be-returned;
}
```

Custom Functions



```
@function sum($num1:0,$num2:0){
  @return $num1 + $num2;
}
```



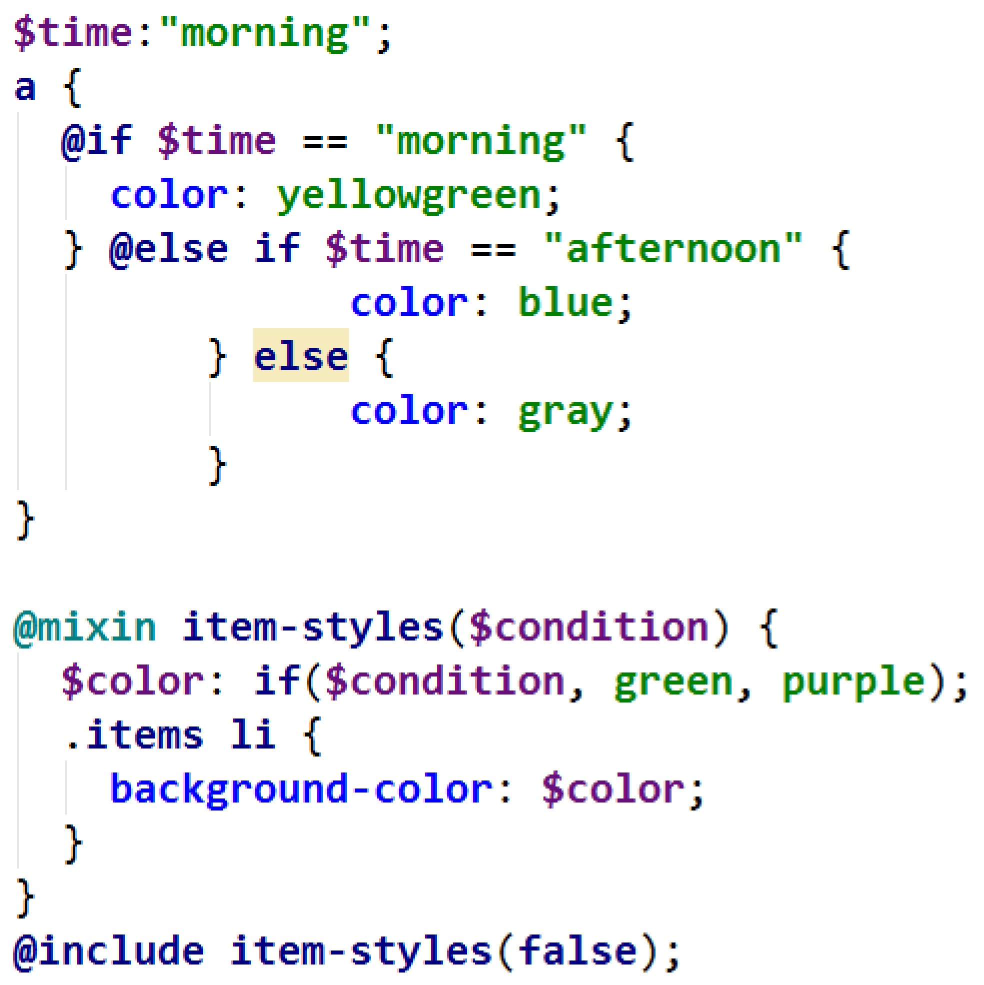
```
@import "functions";
.button{
  padding: sum(3px,4px);
}
```

Conditionals - @if and @else directives

- @if
  - The @if control directive takes a SassScript expression and processes its block of styles if the expression returns anything other than false.
  - Syntax :

```
@if expression {
  // CSS codes
} @else if condition {
  // CSS codes
} @else {
  // CSS codes
}
```

Conditionals - @if and @else directives



```
$time:"morning";
a {
  @if $time == "morning" {
    color: yellowgreen;
  } @else if $time == "afternoon" {
              color: blue;
        } else {
              color: gray;
        }
}

@mixin item-styles($condition) {
  $color: if($condition, green, purple);
  .items li {
    background-color: $color;
  }
}
@include item-styles(false);
```

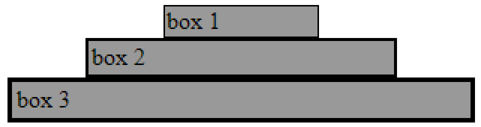
accenture

Loops - @for

- The Sass @for directive facilitates you to generate a style in a loop.
- It is used when you require repeatedly set of styles.
- The @for directive comes in two forms.
  - **@for $var from <start> through <end>**
    - which starts at <start> and loops "through" each iteration and ends at <end>.
  - **@for $var from <start> to <end>**
    - which starts at <start> and loops through each iteration "to" <end> and stops. Once the directive hits the <end> it stops the looping process and does not evaluate the loop that one last time..

```
div{
  background-color: #999;
}
@for $i from 1 through 3 {
  .item_#{$i} {
    width: $i * 100px;
    border:#{$i}px solid black;
    padding: #{$i}px;
    margin: auto;
  }
}
```

```
<div class="item_1 ">box 1</div>
<div class="item_2 ">box 2</div>
<div class="item_3 ">box 3</div>
```

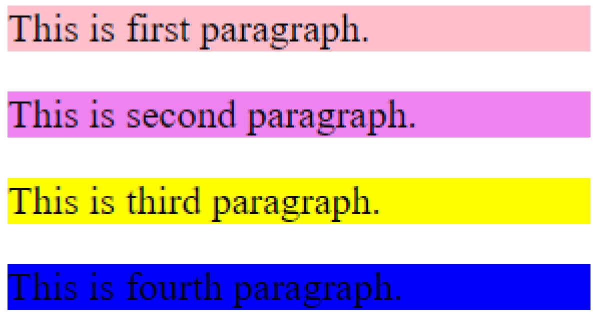


Loops - @for

@each

- The @each directive takes the form @each $var in <list>

```
@each $color in pink, violet, yellow, blue {
  .p_#{$color} {
    background-color: #{$color};
  }
}
```


This is first paragraph.

This is second paragraph.

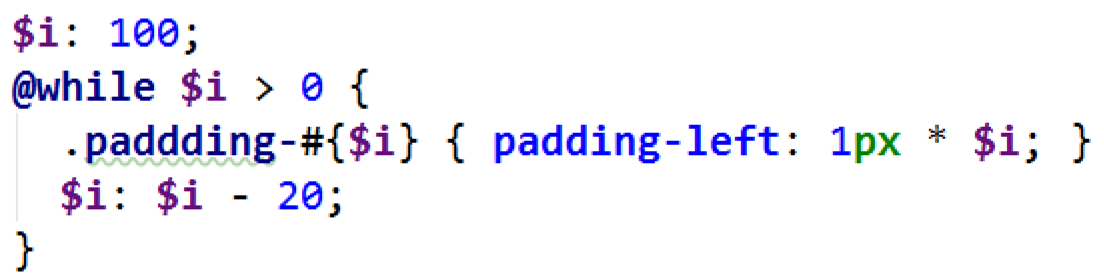
This is third paragraph.

This is fourth paragraph.

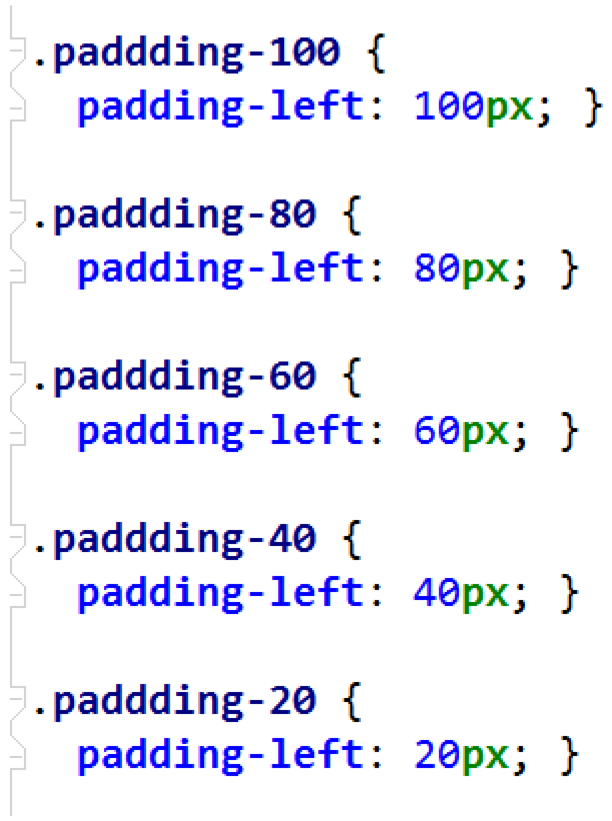
```
<p class="p_pink">This is first paragraph.</p>
<p class="p_violet">This is second paragraph.</p>
<p class="p_yellow">This is third paragraph.</p>
<p class="p_blue">This is fourth paragraph.</p>
```

@while

- The @each directive takes the form @each $var in <list>



```scss
$i: 100;
@while $i > 0 {
  .paddding-#{$i} { padding-left: 1px * $i; }
  $i: $i - 20;
}
```



```css
.paddding-100 {
  padding-left: 100px; }

.paddding-80 {
  padding-left: 80px; }

.paddding-60 {
  padding-left: 60px; }

.paddding-40 {
  padding-left: 40px; }

.paddding-20 {
  padding-left: 20px; }
```

BREAK : 15 MINUTES

accenture

LUNCH BREAK
45 MINUTES

accenture

Lesson Summary

Here are some key points to take away from this lesson:

- SASS as CSS Preprocessor
- Nesting
- Variables and Data Types
- Partials and imports
- Operators
- Mixins
- Inheritance
- List , Map, at-root
- Functions
- Conditional loopss

Questions?
Shift
PgDn

THANK YOU

accenture