



# Fundamentals



Aug 21-23, 2017

**Mike North**

© Mike.Works, Inc. 2017. All rights reserved

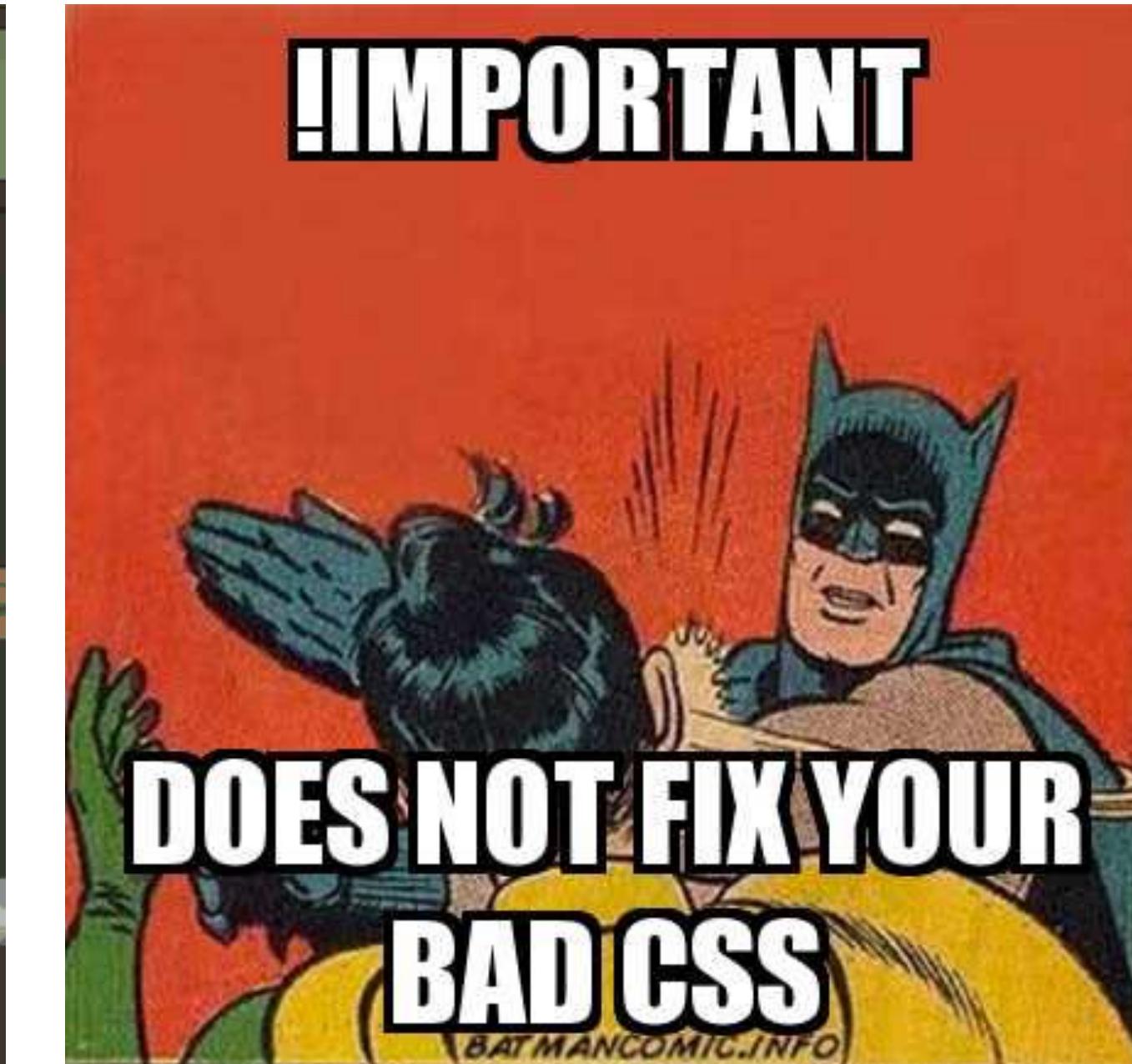


### CSS Pitfalls

- ▶ No encapsulation
- ▶ No variables
- ▶ Not composable
- ▶ Bad modularity primitives
- ▶ Globals
- ▶ Beating-into-submission-driven-development



CSS  
IS  
AWESOME



 24.9k





I AM NOT A

Trying to learn CSS  [\(i.imgur.com\)](https://i.imgur.com)  
submitted 2 years ago by [mcc5159](#)   
[902 comments](#) [share](#)

CSS  
IS  
AWESOME



A screenshot of a Reddit post. The thumbnail image shows Tom Hanks as Forrest Gump with the text "I AM NOT A". To the left of the thumbnail is a diamond-shaped karma counter showing "24.9k". To the right of the thumbnail is the post title "Trying to learn CSS" in large bold text, followed by "(i.imgur.com)" and a gold star icon. Below the title is the submission information "submitted 2 years ago by mcc5159" and "902 comments share".

### Preprocessors: Why use one?

- ▶ Adds stuff that should have been there
- ▶ Style is faster to build & easier to maintain
- ▶ D.R.Y.
- ▶ It's easier to keep your styles organized
- ▶ Easy to set up
- ▶ Toolkits on top of preprocessors





# Fundamentals

# *Sass* Fundamentals

- \* **Sass Origins & Basics**
- \* **Variables & Mixins**
- \* **Control flow**
- \* **CSS Architecture**
- \* **Defining your own Sass functions**

# HAMPTON CATLIN



Invented it

HAMPTON  
CATLIN



Invented it

CHRIS  
EPPSTEIN



NATALIE  
WEIZENBAUM



# Syntax

styles.scss

```
.foo {  
  color: green;  
  font-size: 1.3rem;  
}
```

styles.sass

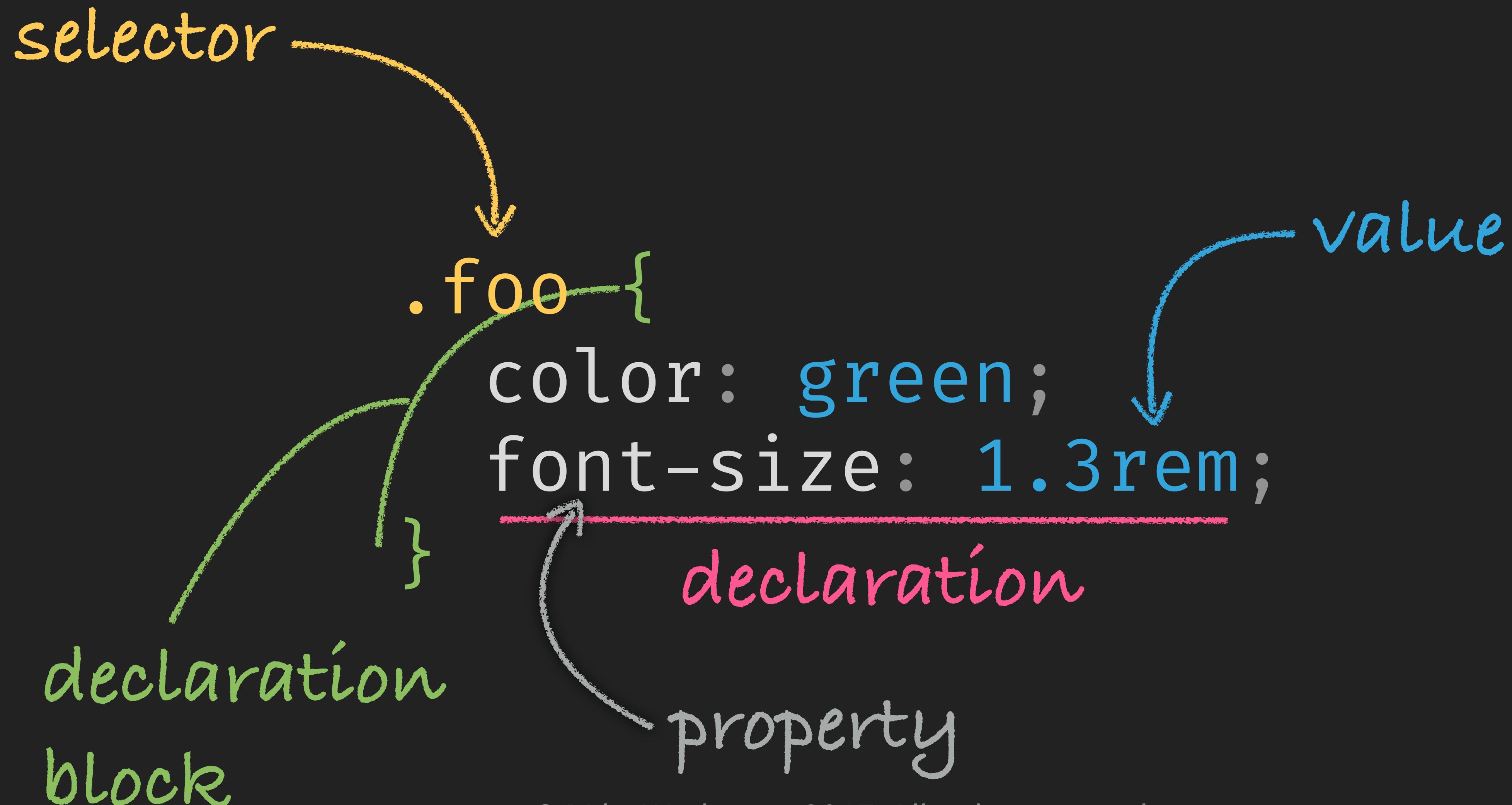
```
.foo  
  color: green  
  font-size: 1.3rem
```

# Syntax

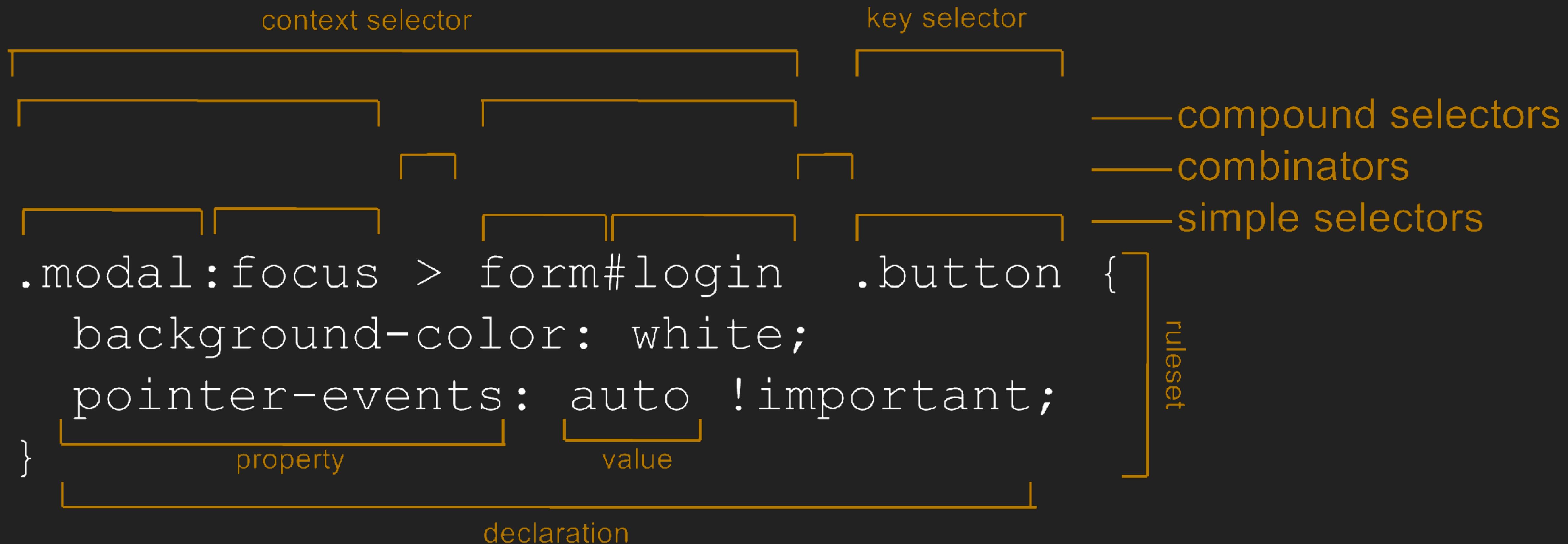
styles.scss

```
.foo {  
  color: green;  
  font-size: 1.3rem;  
}
```

# Terminology: A CSS "Rule"

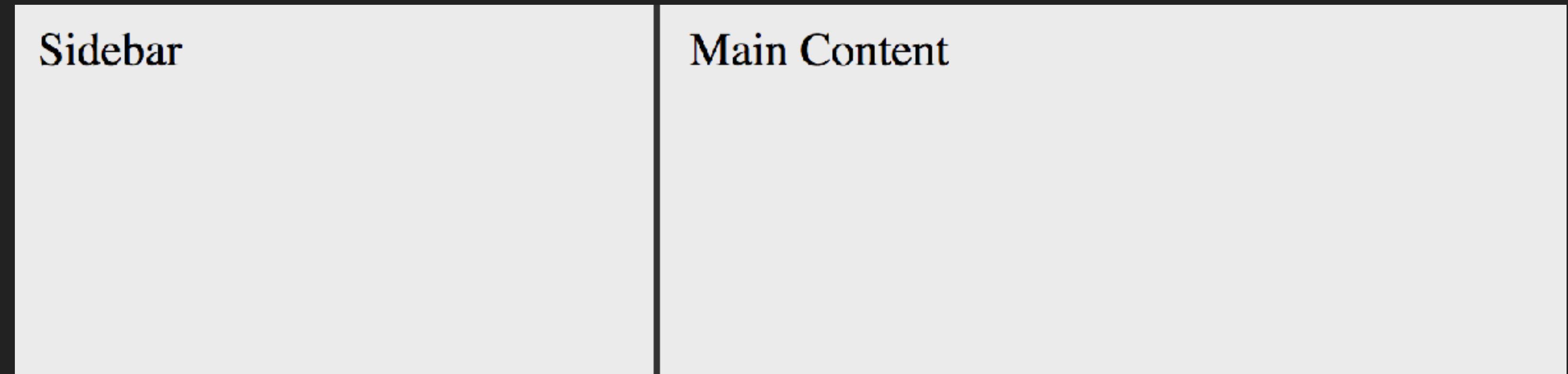


# Terminology: A CSS “Rule”



# Nesting & Scoping

```
<div class="container">  
  <div class="sidebar">  
    Sidebar  
  </div>  
  <div class="main">  
    Main Content  
  </div>  
</div>
```



# Nesting & Scoping

```
.container {  
    max-width: 600px;  
    width: 100%;  
    margin: auto;  
    background: #eee;  
}  
  
.container .sidebar,  
.cortainer .main {  
    padding: 10px;  
}  
  
.container .main {  
    margin-left: 220px;  
    min-height: 100vh;  
    border-left: 2px solid #333;  
}  
  
.container .sidebar {  
    width: 200px;  
    float: left;  
}
```

```
.container {  
  max-width: 600px;  
  width: 100%;  
  margin: auto;  
  background: #eee;  
  
.sidebar,  
.main { padding: 10px; }  
  
.main {  
  margin-left: 220px;  
  min-height: 100vh;  
  border-left: 2px solid #333;  
}  
  
.sidebar {  
  width: 200px;  
  float: left;  
}  
}
```

# Nesting & Scoping

- ▶ Styles can be placed in the declaration block of a parent element
- ▶ Everything is expanded as you would expect during the Sass compilation process
- ▶ This is already D.R.Y.'ed up quite a bit!

# Nesting & Scoping

```
.container {  
    max-width: 600px;  
    width: 100%;  
    margin: auto;  
    background: #eee;  
}  
  
.container .sidebar,  
.cortainer .main {  
    padding: 10px;  
}  
  
.container .main {  
    margin-left: 220px;  
    min-height: 100vh;  
    border-left: 2px solid #333;  
}  
  
.container .sidebar {  
    width: 200px;  
    float: left;  
}
```

# Descendant

```
<div class="container">
  <div class="left-area">...</div>
  <div class="other-thing">
    <div class="left-area"></div>
  </div>
</div>
```

SASS

```
.container {
  .left-area {
    ...
  }
}
```

CSS

```
.container .left-area {
  ...
}
```

# Direct Descendant (>)

```
<div class="container">
  <div class="left-area">...</div>
  <div class="other-thing">
    <div class="left-area"></div>
  </div>
</div>
```

SASS

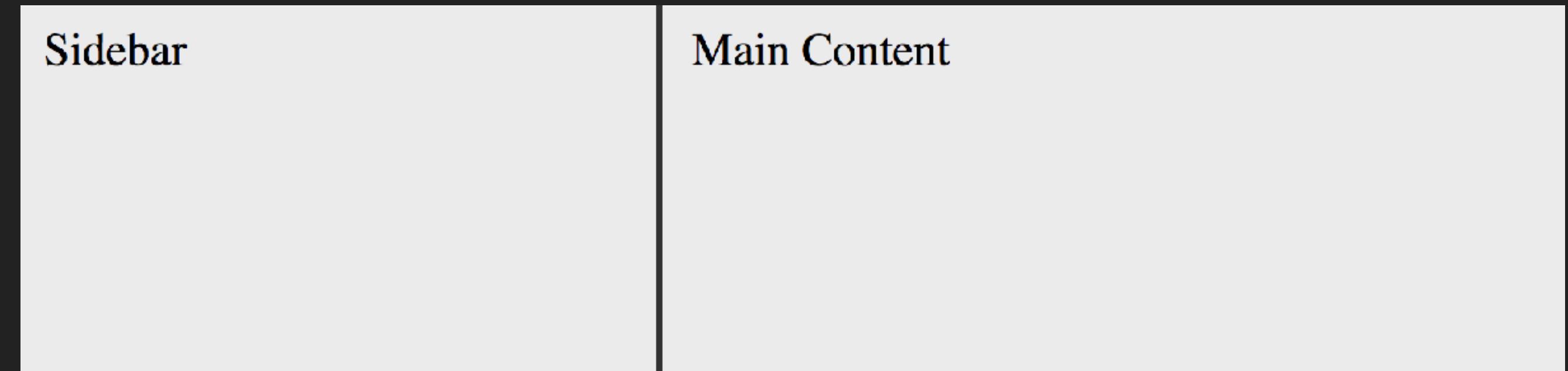
```
.container > .left-area {
  ...
}
```

CSS

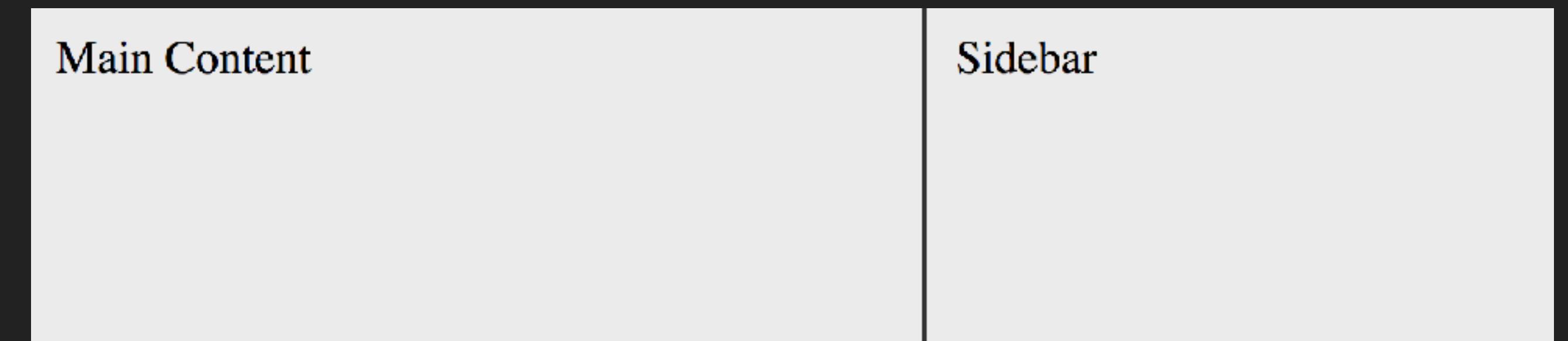
```
.container > .left-area {
  ...
}
```

# Parent Selector (&)

```
<div class="container">  
  ...  
</div>
```



```
<div class="container right-nav">  
  ...  
</div>
```



# Parent Selector (&)

SASS

```
.container {  
  &.right-nav {  
    color: #333;  
  }  
}
```

CSS

```
.container.right-nav {  
  color: '#333';  
}
```

- ▶ The parent selector is often useful in situations where adding a secondary class changes styles

# Parent Selector (&)

SASS

```
.button {  
  color: #333;  
  .theme-dark & {  
    color: #fff;  
  }  
}
```

CSS

```
.button {  
  color: #333;  
}  
.theme-dark .button {  
  color: #fff;  
}
```

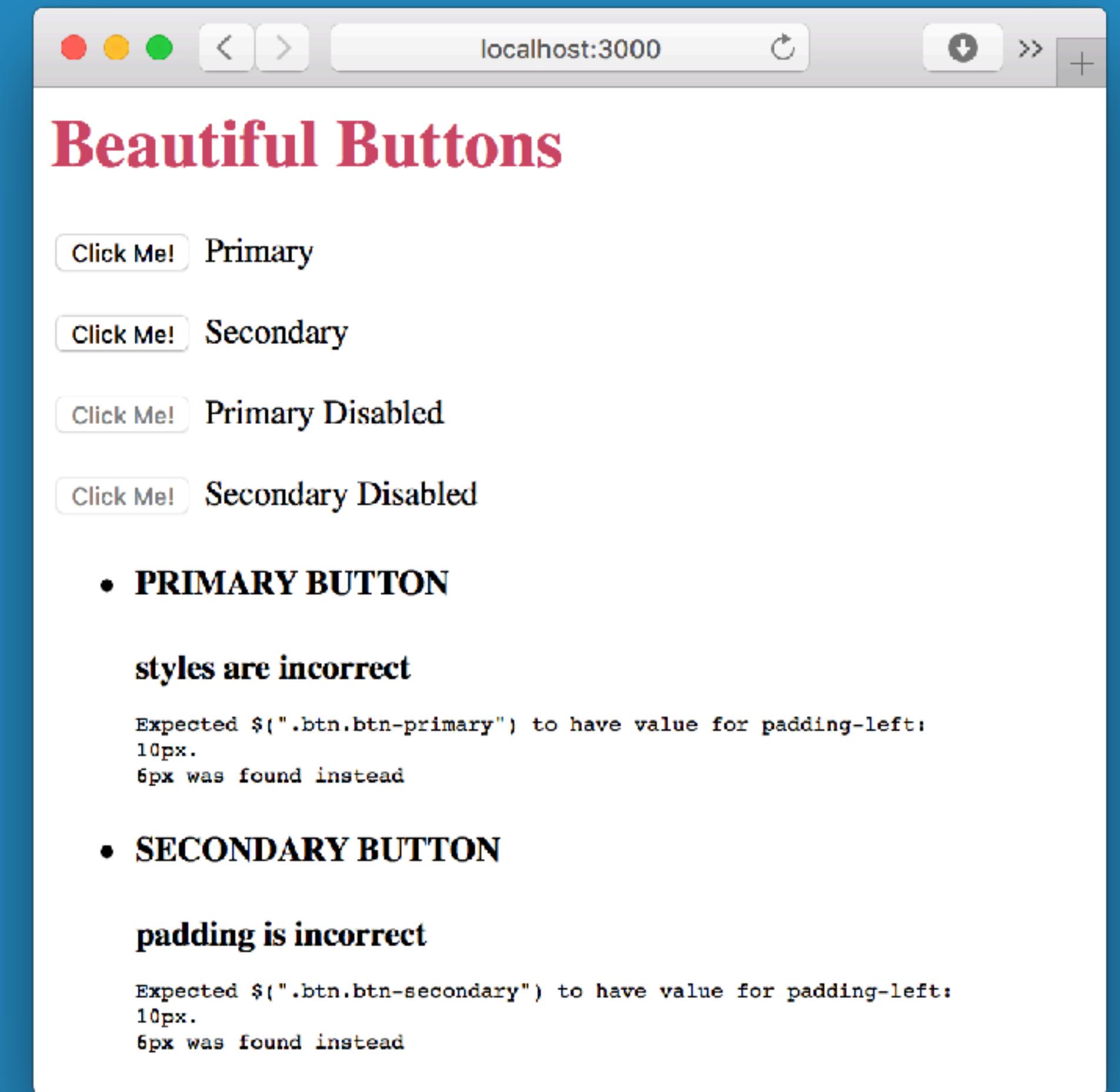
# Exercise: Parent Selectors I

- ▶ All buttons must have 10px padding on left and right, 2px on top and bottom
- ▶ All buttons should have a 1px #c46 solid border, with a 2px radius
- ▶ Adding the .btn-primary class to a button should turn its text white and background #c46
- ▶ Adding the .btn-secondary to a button should turn its background #edbcc8 and text black
- ▶ Disabling a button should cause its opacity to be set to 0.5

Time: 10 min

Command to run: `./run -e nesting`

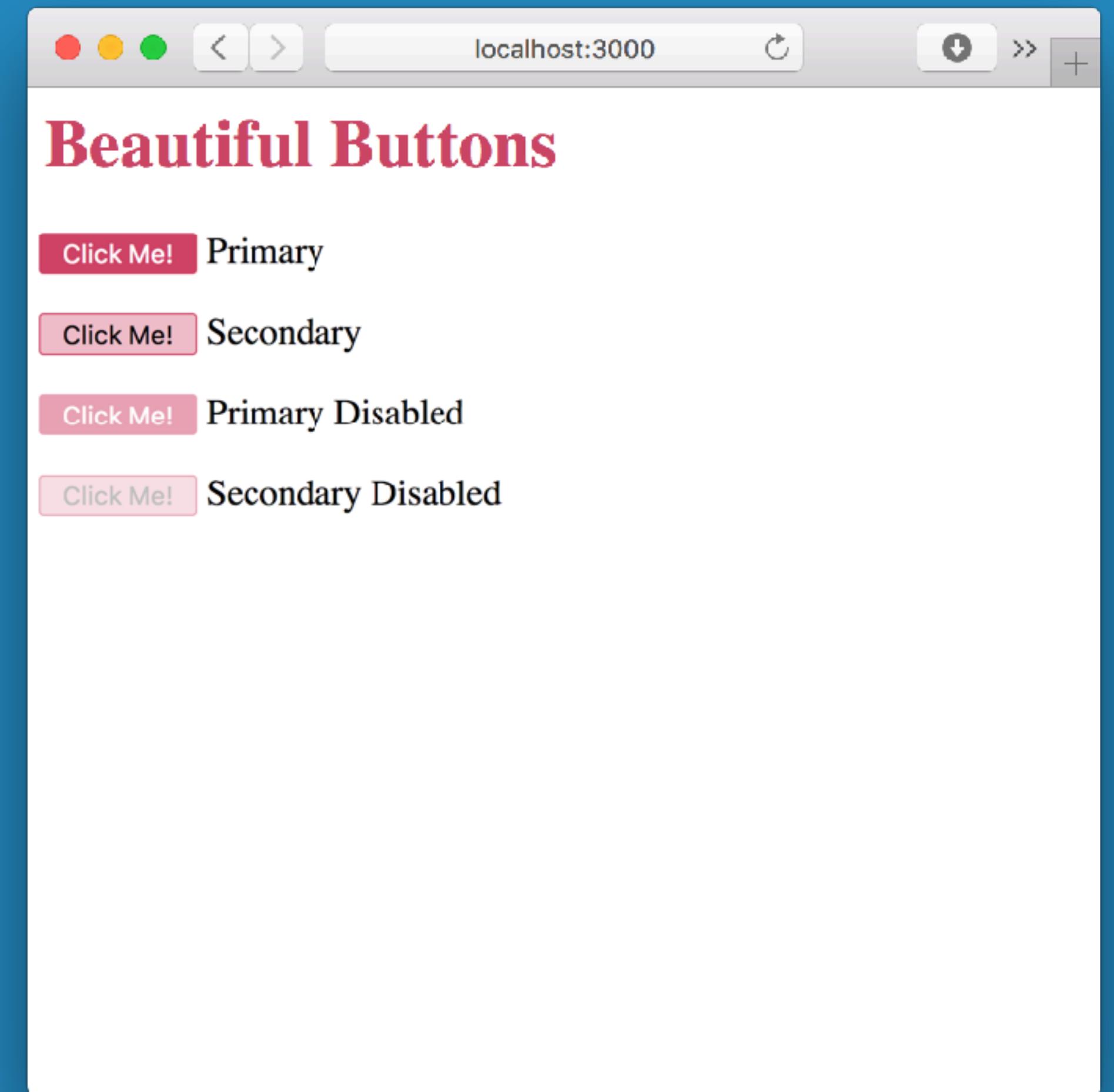
<http://localhost:3000/>



# Exercise: Parent Selectors I

- ▶ All buttons must have 10px padding on left and right, 2px on top and bottom
- ▶ All buttons should have a 1px #c46 solid border, with a 2px radius
- ▶ Adding the .btn-primary class to a button should turn its text white and background #c46
- ▶ Adding the .btn-secondary to a button should turn its background #edbcc8 and text black
- ▶ Disabling a button should cause its opacity to be set to 0.5

<http://localhost:3000/>



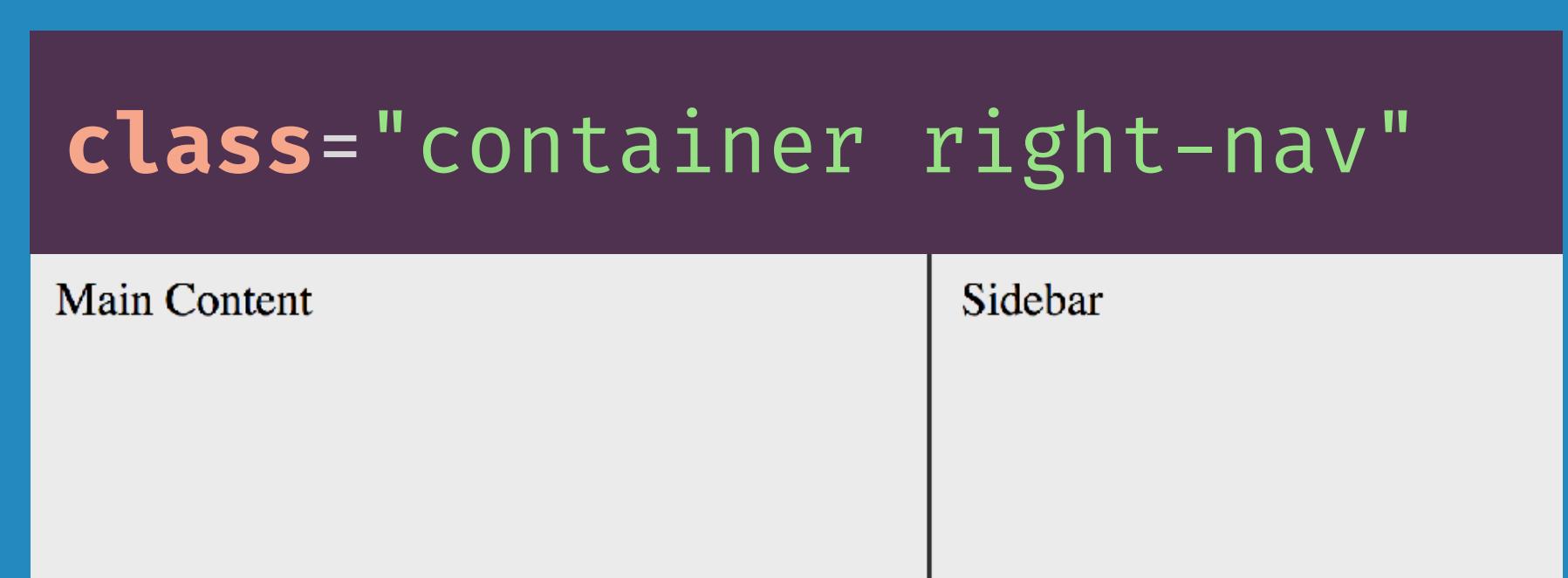
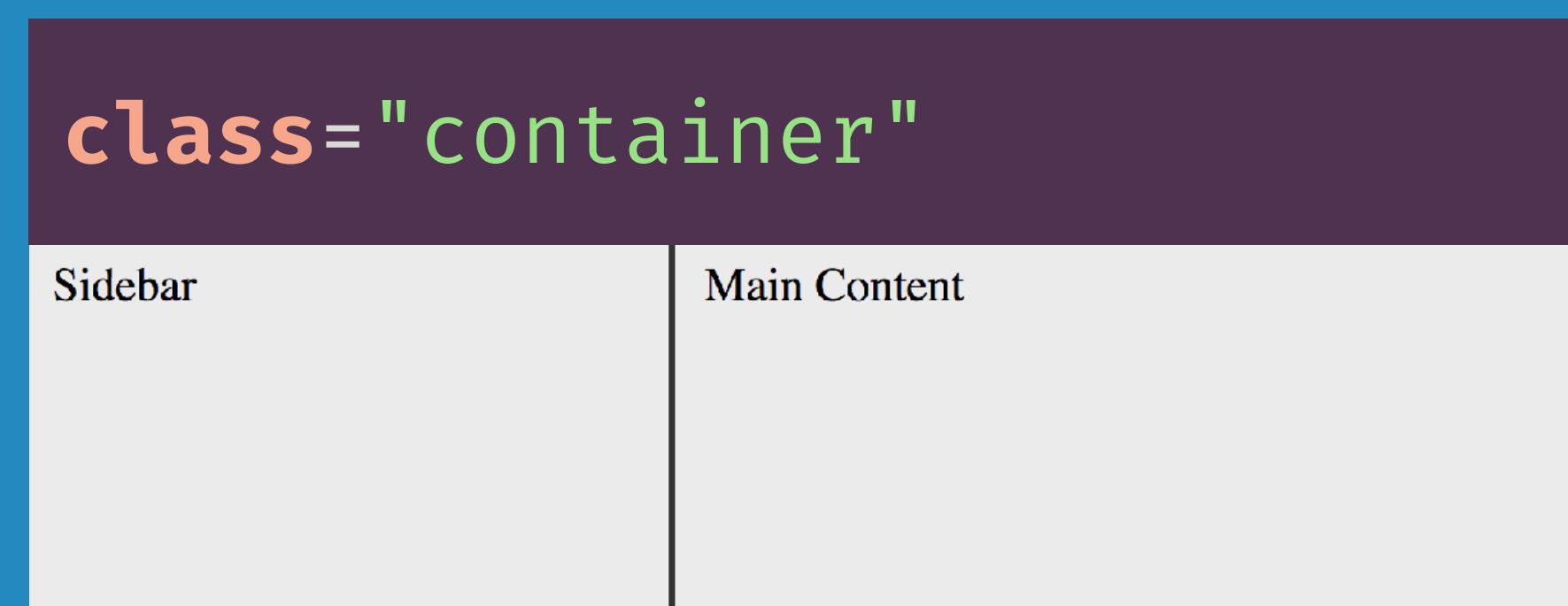
Time: 10 min

Command to run: `./run -e nesting`

© Mike.Works, Inc. 2017.  
All rights reserved

# Exercise: Parent Selectors II

- ▶ Expand upon our sidebar example, so that adding the `.right-nav` class to the `.container` div can be used to toggle between left and right sidenav alignment
- ▶ Ensure that your code passes the tests that are in place



Time: 10 min

Command to run: `./run -e parent`

# Importing

```
@import 'variables';
```

- ▶ Using an **@import** in CSS results in a new round-trip HTTP request, this can be a performance (load time) concern
- ▶ Modularity is a wonderful tool, but in CSS comes at a price
- ▶ Files our users download  $\neq$  files we use to manage source code
- ▶ **@import** in Sass does something else (it's more useful)

# Importing



src/sass

\_layout.scss

\_variables.scss

app.scss

other.scss

"partials"

will turn into CSS files

SASS

```
/* From a library */  
@import 'bootstrap';  
/* A "partial" in my project */  
@import 'layout';
```

# Variables

global  
variable declaration

```
$error_color: #f00 !default;
```

value

“unless set  
elsewhere”

```
.alert-error {  
    $text_color: #ddd;  
    background-color: $error-color;  
    color: $text-color;  
    text-shadow: 0 0 2px darken($text-color, 40%);  
}
```

local  
variable  
declaration }

# Variables

- ▶ Simple arithmetic for values is ok
- ▶ Sass can convert units, as long as dimension is the same
- ▶ Any CSS value should be ok

## Types

NUMBERS	COLORS	STRINGS	BOOLEANS	LISTS
<ul style="list-style-type: none"><li>▶ 10</li><li>▶ 200px</li><li>▶ 50%</li><li>▶ 14pt</li></ul>	<ul style="list-style-type: none"><li>▶ #FFF</li><li>▶ rgb(255, 0, 0)</li><li>▶ red</li></ul>	<ul style="list-style-type: none"><li>▶ “a.png”</li><li>▶ url(“a.png”)</li></ul>	<ul style="list-style-type: none"><li>▶ true</li><li>▶ false</li></ul>	<b>MAPS</b>

# Comments

SASS

```
/*!  
 * Will remain  
 */  
.foo {  
  color: red; // Will be removed  
}
```

CSS

```
/**  
 * Will remain  
 */  
.foo {  
  color: red;  
}
```

# String Interpolation

SASS

```
/**  
 * Hue is #{hue(green)}  
 */
```

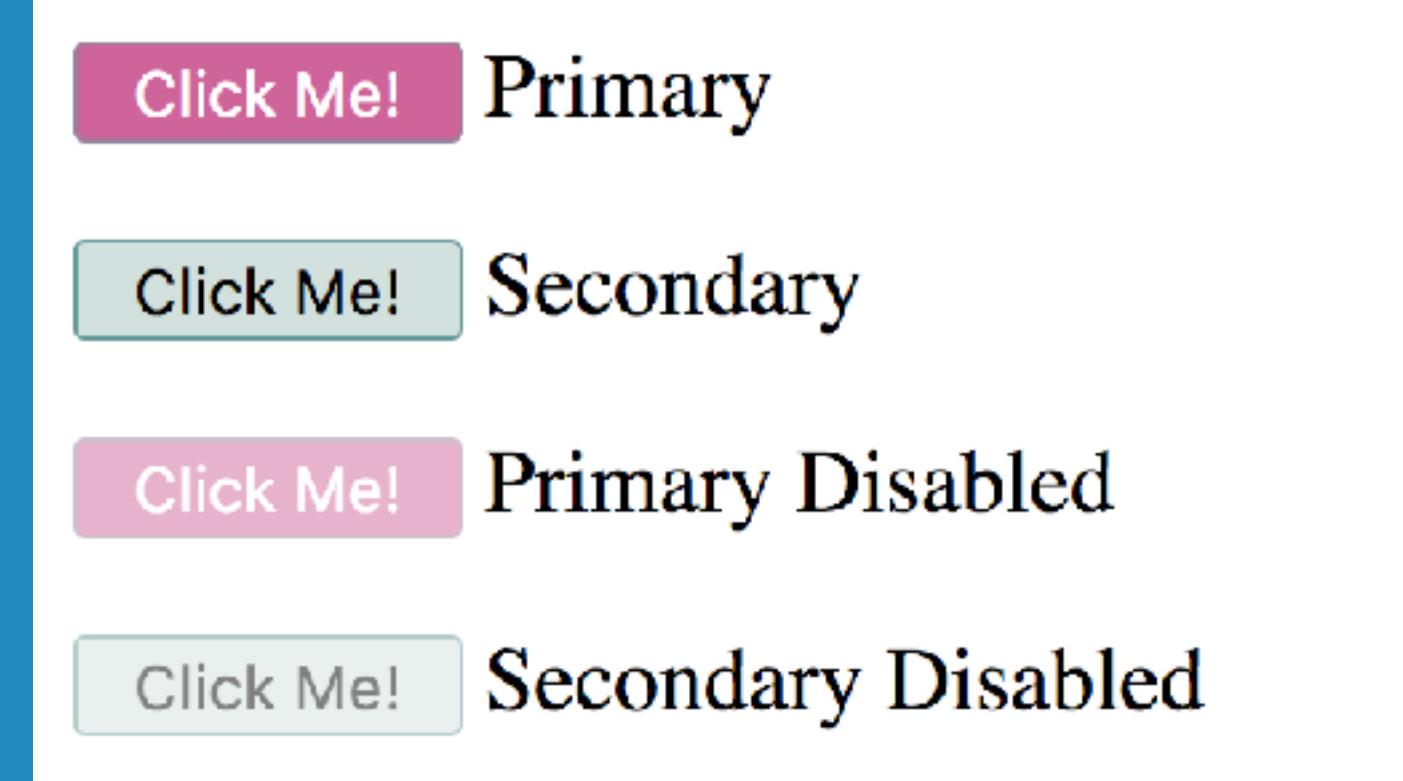
CSS

```
/**  
 * Hue is 120deg  
 */
```

**Only works with valid CSS values**

# Exercise: Imports and Variables

- ▶ This looks like our first exercise
- ▶ Except, there's a `_variables.scss`
- ▶ Primary button should have a `$hopbush` background and `$venus` border
- ▶ Secondary button should have a `$nebula` background and `$patina` border



```
$hopbush: ■#c69 !default;  
$patina: ■#699 !default;  
$venus: ■#998099 !default;  
$nebula: ■#d2e1dd !default;  
  
$black: □#000 !default;  
$white: ■#fff !default;  
  
$btn-disabled-opacity: 0.5 !default;
```

Time: 10 min

Command to run: `./run -e variables`

© Mike.Works, Inc. 2017.  
All rights reserved

# Mixins

- ▶ Allow for re-use of style
- ▶ **Mental model:** `@mixin` declarations are combined w/ declarations where you `@include` it
- ▶ **Best practice:** separate from styles, like variables

# Mixins

```
@mixin alert-text {  
  background-color: #f00;  
  color: white;  
  font-variant: small-caps;  
}
```

```
.error-text {  
  @include alert-text;  
}
```

```
.has-error:after {  
  @include alert-text;  
  display: inline-block;  
  content: attr(data-error);  
}
```

- ▶ Allow for re-use of style
- ▶ **Mental model:** `@mixin` declarations are combined w/ declarations where you `@include` it
- ▶ **Best practice:** separate from styles, like variables

```
@mixin alert-text($color) {  
  background-color: $color;  
  color: white;  
  font-variant: small-caps;  
}  
  
.error-text {  
  @include alert-text(blue);  
}  
  
.has-error:after {  
  @include alert-text(red);  
  display: inline-block;  
  content: attr(data-error);  
}
```

# Mixins: Arguments

► Variables (arguments) allow for parameterization!

```
@mixin shadow( ... $args ) {  
  box-shadow: $args;  
}
```

```
.card {  
  @include shadow(0 0 5px #000);  
}
```

# Mixins: Arguments

- ▶ Variables (arguments) allow for parameterization!
- ▶ Spread operator

# Exercise: Mixins I

- ▶ In your `app.scss` file, we're making use of a new (currently empty) mixin
- ▶ The goal is to define a means of defining the style of a button, given a color
- ▶ Implement the mixin, such that all failing tests pass

Time: 10 min

Command to run: `./run -e mixins`



# Mixins - Default Argument Values

```
@mixin alert-text($color: #f33) {  
background-color: $color;  
color: white;  
font-variant: small-caps;  
}  
  
h1 {  
@include alert-text(blue);  
}  
  
h2 {  
@include alert-text($color: green)  
}
```

- ▶ We can define default values for arguments
- ▶ Arguments can be provided in order, or by name
- ▶ When “keyword args” used, order is ignored

# Mixins - Null Values

SASS

```
@mixin foo($opacity: null) {  
  color: #333;  
  opacity: $opacity;  
}  
.btn {  
  @include foo();  
}
```

CSS

```
.btn {  
  color: #333;  
}
```

```
.other-btn {  
  @include foo(0.5);  
}
```

```
.other-btn {  
  color: #333;  
  opacity: 0.5;  
}
```

# Mixins - Passing a Declaration Block

SASS

```
@mixin foo($color) {  
  color: $color;  
  .inner {  
    @content  
  }  
}  
  
.btn {  
  @include foo(#c69) {  
    color: red;  
  }  
}
```

pass in a block

use the @content  
directive to place the  
block passed in

CSS

```
.btn {  
  color: #c69;  
}  
.btn .inner {  
  color: red;  
}
```

# Exercise: Mixins II

- ▶ The range input can be styled, but it involves lots of ugly vendor-prefixing
- ▶ This excellent tutorial from CSS tricks describes what needs to be done in CSS. Do it the Sass way!

<https://css-tricks.com/styling-cross-browser-compatible-range-inputs-css/>



**Time:** 20min

**Command to run:** `./run -e range`

# Functions

<http://sass-lang.com/documentation/Sass/Script/Functions.html>

**rgb(\$red, \$green, \$blue)**

Creates a [color](#) from red, green, and blue values.

**rgba(\$red, \$green, \$blue, \$alpha)**

Creates a [color](#) from red, green, blue, and alpha values.

**red(\$color)**

Gets the red component of a color.

**green(\$color)**

Gets the green component of a color.

**blue(\$color)**

Gets the blue component of a color.

**mix(\$color1, \$color2, [\$weight])**

Mixes two colors together.

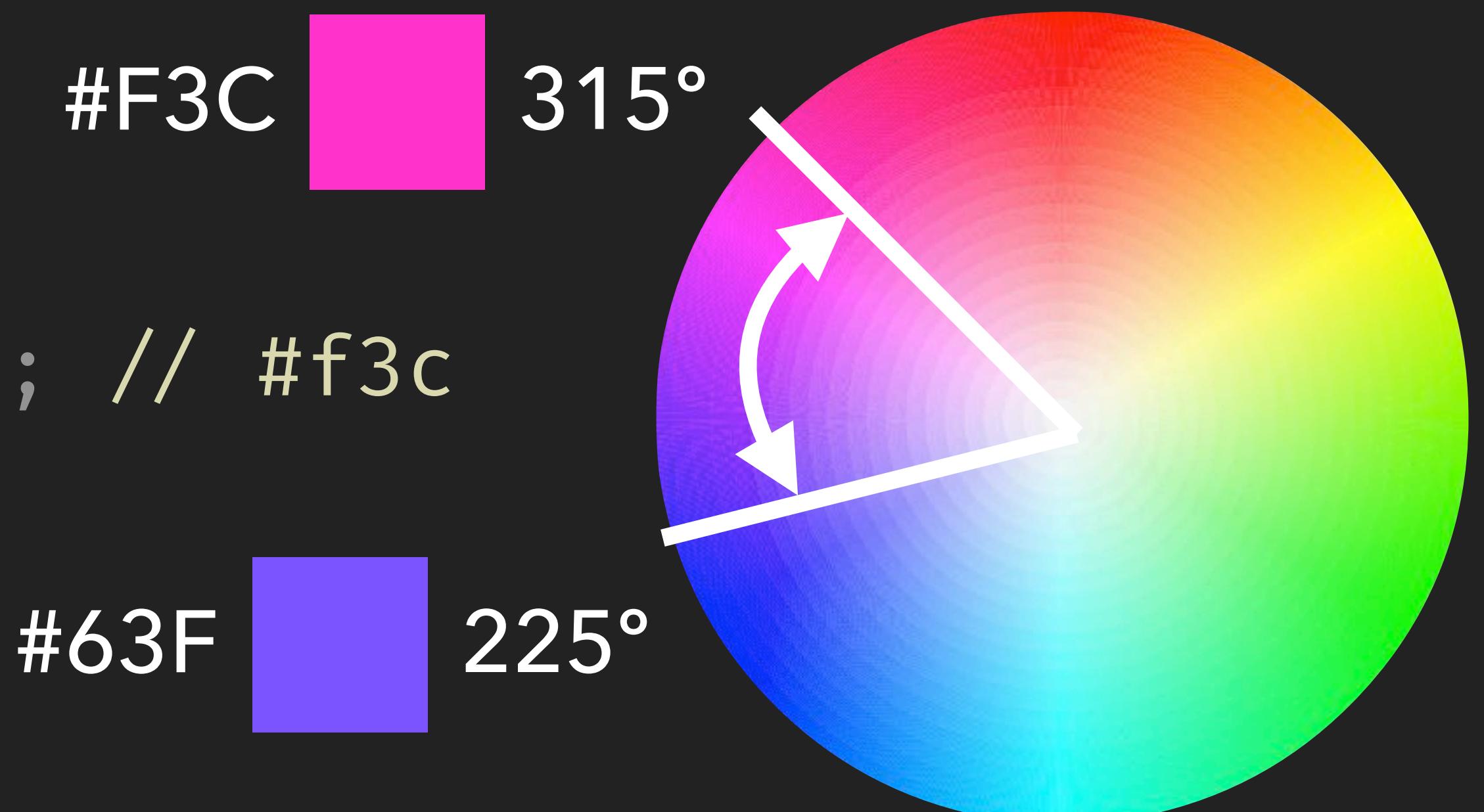
# Color Functions

<http://sass-lang.com/documentation/Sass/Script/Functions.html>

`adjust-hue( $color, $degrees );`

Rotates the hue of a color by a certain number of degrees

```
.foo {  
  color: adjust-hue(#63f, 60deg); // #f3c  
}
```



# Color Functions

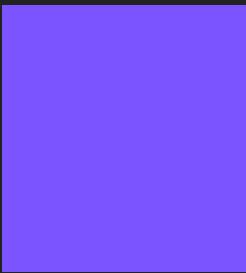
<http://sass-lang.com/documentation/Sass/Script/Functions.html>

```
darken($color, $percent);  
lighten($color, $percent);
```

Change the brightness of a color by a certain amount

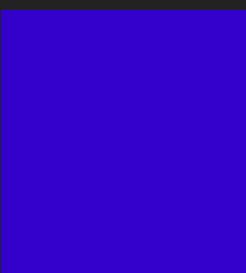
```
.foo {  
  color: darken(#63f, 20%); // #30C  
}
```

#63F



100%

#30C



80%

# Color Functions

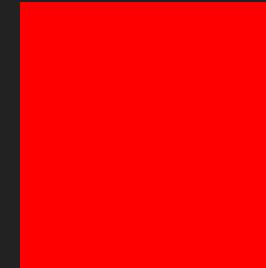
<http://sass-lang.com/documentation/Sass/Script/Functions.html>

```
desaturate($color, $percent);  
saturate($color, $percent);
```

Change the saturation of a color by a specified amount

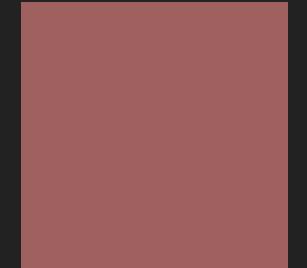
```
.foo {  
  color: desaturate(#f00, 75%); // #9F6060  
}
```

#F00



100%

#9F6060



25%

# Exercise: Color Functions

- ▶ We wish to apply some basic “theming” to a set of buttons

```
<div class="theme-1"><!-- Will be themed --></div>
```

- ▶ Theme mixin should take a **\$color** as first argument, **\$huerot** as second, and a **\$darkenpct** as third

## all buttons

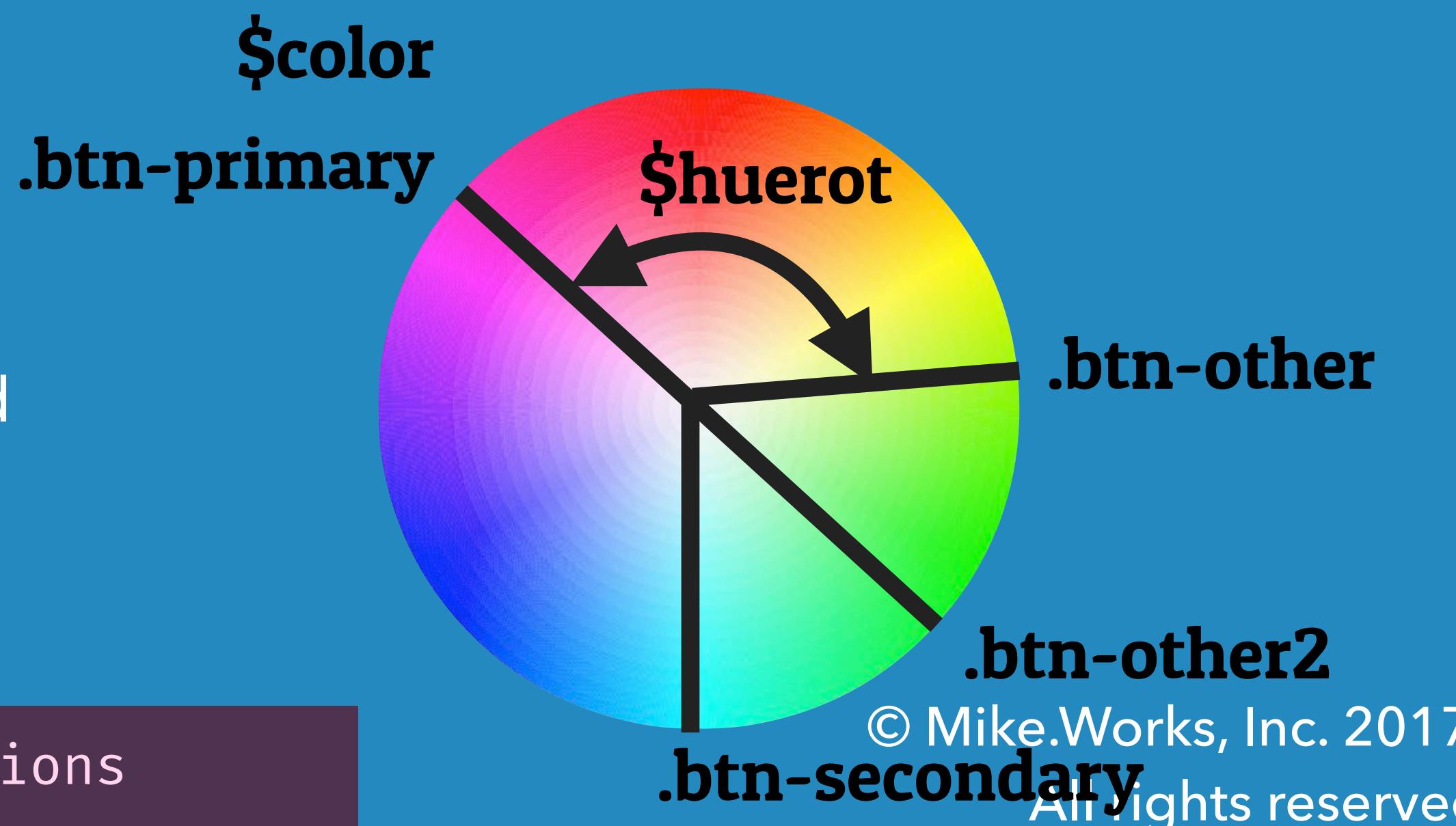
- ▶ border 20% darker color than background
- ▶ hover background: 20% more saturated then 10% lightened

## non-primary buttons

- ▶ darkened by **\$darkenpct**

**Time:** 30 min

**Command to run:** `./run -e functions`



# Control Flow - @if

SASS

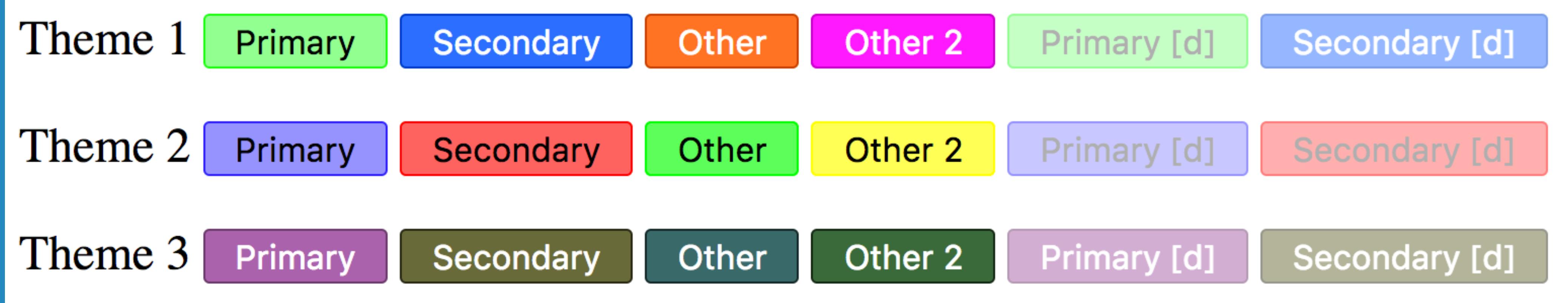
```
@mixin foo($size) {  
  font-size: $size;  
  @if $size > 20 {  
    line-height: $size;  
  }  
}  
  
.small {  
  @include foo(14px);  
}  
.large {  
  @include foo(24px);  
}
```

CSS

```
.small {  
  font-size: 14px;  
}  
.large {  
  font-size: 24px;  
  line-height: 24px;  
}
```

# Exercise: Control Flow w/ @if

- ▶ Only use white button text if the brightness value of a button background color is less than 70% (~~0.7~~)



Time: 10 min

Command to run: `./run -e if`

© Mike.Works, Inc. 2017.  
All rights reserved

# Control Flow - @for

SASS

```
@for $i from 1 through 5 {  
  h#{$i} {  
    font-size: 5rem - $i*0.75rem;  
  }  
}
```

CSS

```
h1 {font-size: 4.25rem;}  
h2 {font-size: 3.5rem;}  
h3 {font-size: 2.75rem;}  
h4 {font-size: 2rem;}  
h5 {font-size: 1.25rem;}
```

# Data Structures - Lists

SASS

```
$mylist: 0 0 2px #000;
```

```
.foo {  
  /**  
   * Shadow blur radius:  
   * #{nth($mylist, 3)}  
   */  
  box-shadow: $mylist;  
}
```

CSS

```
.foo {  
  /**  
   * Shadow blur radius:  
   * 2px  
   */  
  box-shadow: 0 0 2px #000;  
}
```

# Data Structures - Lists and @each

SASS

```
$mylist: 0 0 2px #000;
```

```
.foo {  
  @each $i in $mylist {  
    /* #{$i} */  
  }  
}
```

CSS

```
.foo {  
  /* 0 */  
  /* 0 */  
  /* 2px */  
  /* #000 */  
}
```

# Data Structures - nth

SASS

```
$gradients:  
  (to left top, blue, red),  
  (to left top, blue, yellow);  
  
.foo {  
  background: linear-gradient(nth($gradients, 2));  
}
```

SASS

```
$mymap: (  
    dark: #009,  
    light: #66f  
) ;
```

```
@mixin theme-button($t) {  
    /* Theme: #{$t} */  
    color: map-get($mymap, $t);  
}
```

```
.btn-dark {  
    @include theme-button('dark')  
}  
  
.btn-light {  
    @include theme-button('light')  
}
```

CSS

```
.btn-dark {  
    /* Theme: dark */  
    color: #009;  
}
```

```
.btn-light {  
    /* Theme: light */  
    color: #66f;  
}
```

# Data Structures - Maps and @each

SASS

```
$theme-colors: (  
    primary: #f00,  
    secondary: #0f0,  
    disabled: #999  
);  
  
@each $type, $color in $theme-colors {  
    .color-#$type {  
        color: $color  
    }  
}
```

CSS

```
.color-primary {  
    color: #f00;  
}  
  
.color-secondary {  
    color: #0f0;  
}  
  
.color-disabled {  
    color: #999;  
}
```

# Exercise: Nudging Classes

- ▶ We're going to build a bunch of tiny css rules, each with exactly one style declaration
- ▶ using @for, @each and Sass data structures, build a set of styles **in 5px increments**, for both **margin and padding**, in **each of the four directions** (top, bottom, left right).
- ▶ BONUS: Try to use 15 lines of Sass or less!

```
.m-t-5 {  
  margin-top: 5;  
}
```

Time: 25min

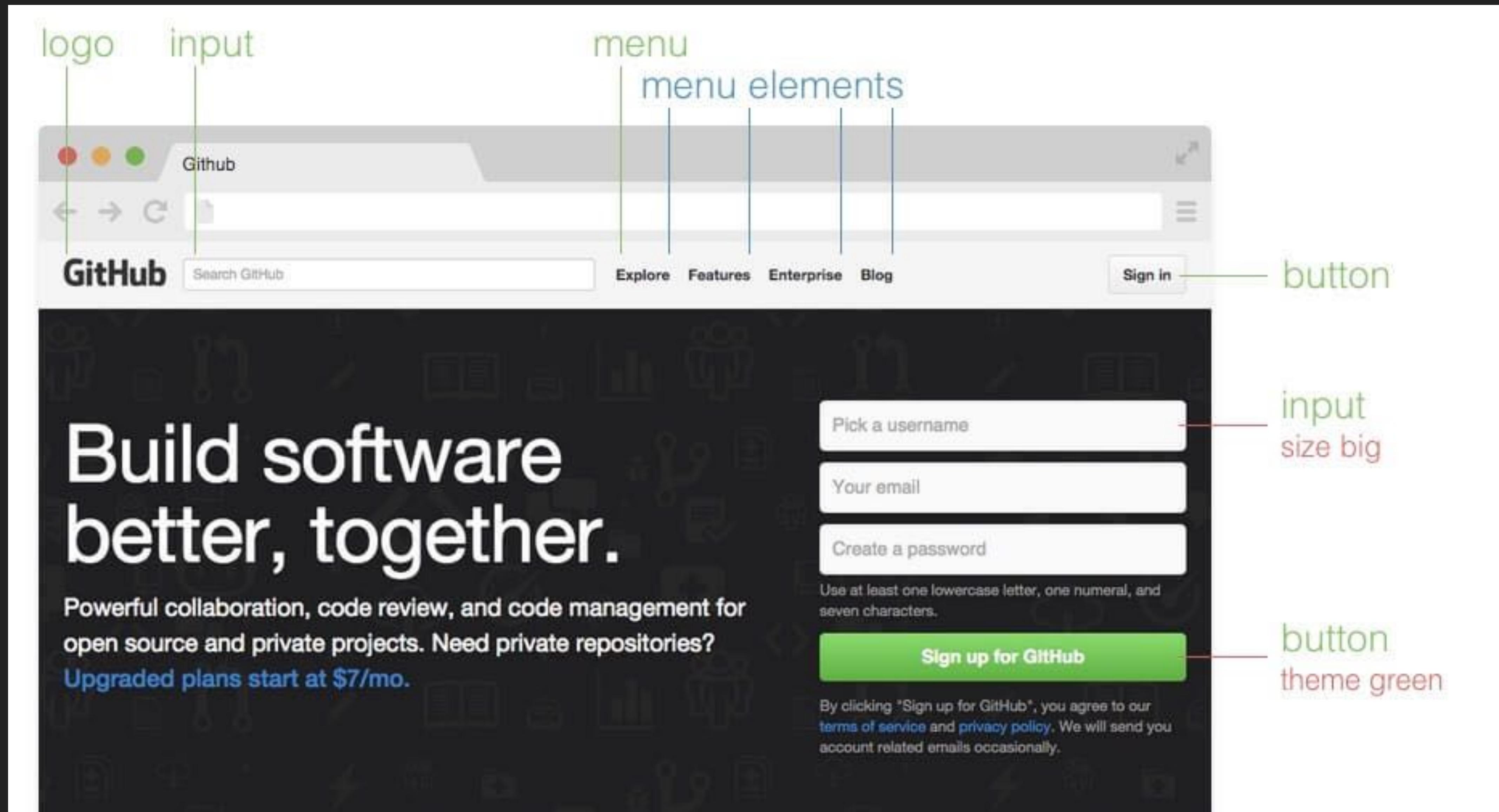
Command to run: ./run -e tiny

© Mike.Works, Inc. 2017.  
All rights reserved

# CSS Architecture - BEM

- ▶ **Block** - standalone entity, meaningful on its own
  - header, container, menu, checkbox, input, button
- ▶ **Element** - A part of a block that has no standalone meaning, and is semantically tied to its block
  - menu-item, list-item, checkbox-caption, header-title
- ▶ **Modifier** - A flag on a block or element, used to change appearance and/or behavior
  - disabled, highlighted, checked, size-big, color-yellow

# CSS Architecture - BEM



# CSS Architecture - BEM

First Name **Mike**  
Must be two characters or longer

```
<div class="textfield">
  <label for="first-name" class="textfield_label">
    First Name
  </label>
  <input name="first-name" type="email" class="textfield_input" />
  <span class="textfield_validation-error">
    Must be two characters or longer
  </span>
</div>
```

# CSS Architecture - BEM Elements

```
<div class="textfield">
  <label for="first-name" class="textfield_label">
    First Name
  </label>
  <input name="first-name" type="email" class="textfield_input" />
  <span class="textfield_validation-error">
    Must be two characters or longer
  </span>
</div>
```

```
.textfield {
  &_input { }
  &_label { }
  &_validation-error { }
}
```

# CSS Architecture - BEM Modifiers

```
<div class="textfield textfield--state-validated">
  <label for="first-name" class="textfield_label">
    First Name
  </label>
  <input name="first-name" type="email" class="textfield_input" />
  <span class="textfield_validation-error">
    Must be two characters or longer
  </span>
</div>
```

```
.textfield {
  &--state-error {}
  &--state-validated {}
}
```

# Exercise: BEM Buttons

- ▶ Build a “one-click checkout” button using BEM
- ▶ Basic color requirements for buttons, same as exercise 1
- ▶ Price should have bg of #080, but #aaa when button is disabled



Time: 30 min

Command to run: `./run -e bem`

© Mike.Works, Inc. 2017.  
All rights reserved

SASS

```
@mixin danger {  
  background: red;  
  color: white;  
}  
  
.btn-danger {  
  @include danger();  
  padding: 2px;  
}  
  
.alert-danger {  
  @include danger();  
  width: 100%;  
}
```

CSS

```
.btn-danger {  
  background: red;  
  color: white;  
  padding: 2px;  
}  
  
.alert-danger {  
  background: red;  
  color: white;  
  width: 100%;  
}
```

# @extend

SASS

```
.danger {  
  background: red;  
  color: white;  
}  
  
.btn-danger {  
  @extend .danger;  
  padding: 2px;  
}  
  
.alert-danger {  
  @extend .danger;  
  width: 100%;  
}
```

css

```
.danger,  
.btn-danger,  
.alert-danger {  
  background: red;  
  color: white;  
}  
  
.btn-danger {  
  padding: 2px;  
}  
  
.alert-danger {  
  width: 100%;  
}
```

# @extend - Placeholders

SASS

```
%danger {  
  background: red;  
  color: white;  
}  
  
.btn-danger {  
  @extend %danger;  
  padding: 2px;  
}  
  
.alert-danger {  
  @extend %danger;  
  width: 100%;  
}
```

CSS

```
.btn-danger,  
.alert-danger {  
  background: red;  
  color: white;  
}  
  
.btn-danger {  
  padding: 2px;  
}  
  
.alert-danger {  
  width: 100%;  
}
```

# ⚠️ @extend ⚠️

SASS

```
%danger {  
  background: red;  
  color: white;  
}  
  
.top, .middle, .footer {  
  .login-panel, .registration-panel {  
    .success-button, .cancel-button {  
      @extend %danger;  
    }  
  }  
}  
}
```

# ⚠️ @extend ⚠️

SASS

```
.danger {  
  background: red;  
  color: white;  
}  
  
.top, .middle, .footer {  
  .login-panel, .registration-panel {  
    .success-button, .cancel-button {  
      @extend .danger;  
    }  
  }  
}  
}
```

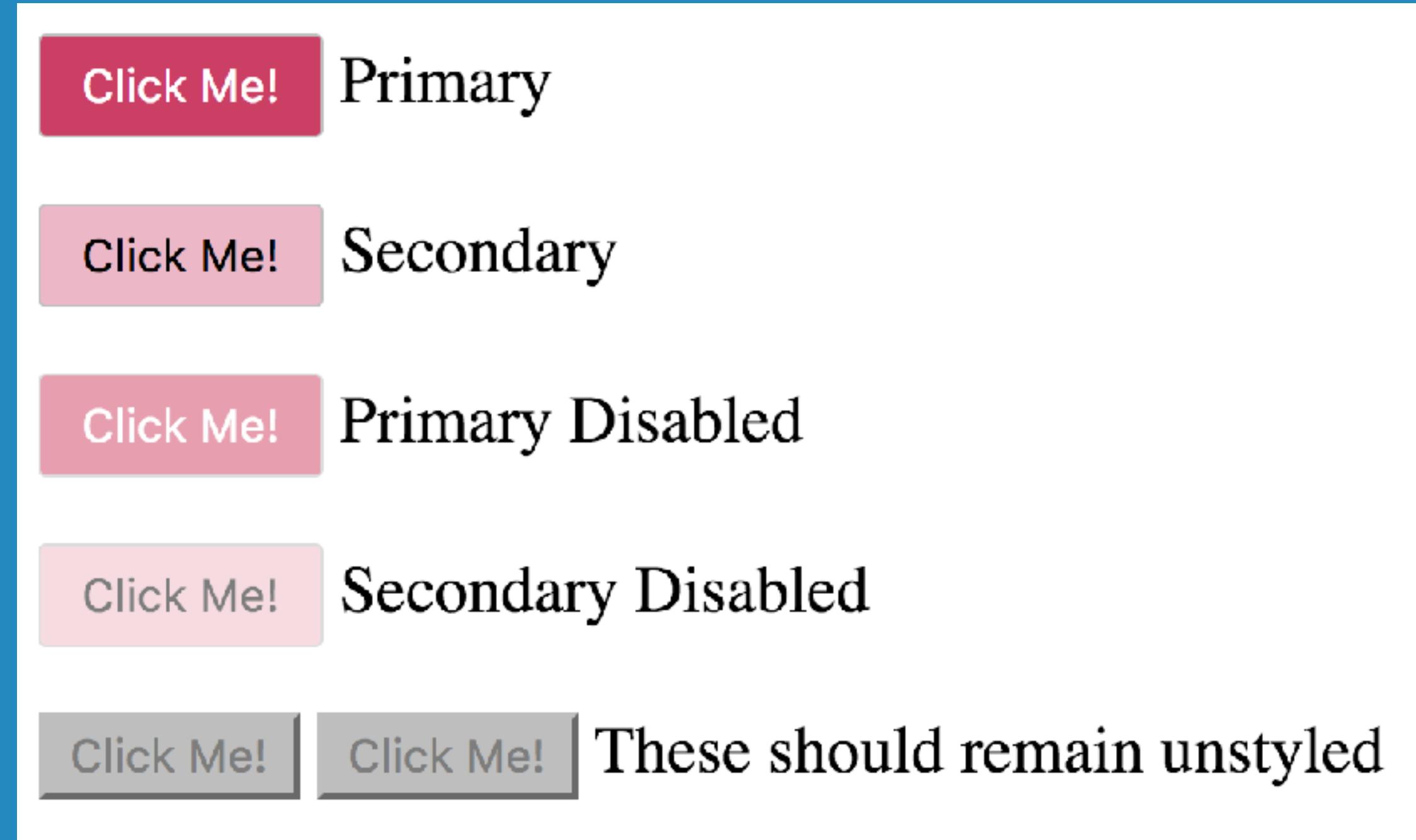
# ⚠️ @extend ⚠️

CSS

```
.top .login-panel .success-button,  
.top .login-panel .cancel-button,  
.top .registration-panel .success-button,  
.top .registration-panel .cancel-button,  
.middle .login-panel .success-button,  
.middle .login-panel .cancel-button,  
.middle .registration-panel .success-button,  
.middle .registration-panel .cancel-button,  
.footer .login-panel .success-button,  
.footer .login-panel .cancel-button,  
.footer .registration-panel .success-button,  
.footer .registration-panel .cancel-button {  
    background: red;  
    color: white;
```

# Exercise: @extend

- ▶ Achieve the same outcome as exercise 1, except...
- ▶ Do not change the appearance of the buttons at the bottom
- ▶ Use @extend and a %placeholder



Time: 30 min

Command to run: `./run -e extend`

# Debugging and Logging

SASS

```
@mixin btn-with-border($col) {  
  @if (lightness($col) < 50%) {  
    @debug "Base color #{$col} is too dark";  
  }  
  background-color: $col;  
  border: 2px solid darken($col, 50%);  
}  
.btn {  
  @include btn-with-border(#a00);  
}
```

CSS

```
.btn {  
  background-color: #900;  
  border: 2px solid black;  
}
```

... /src/sass/app.scss:3 DEBUG: Base color #a00 is too dark

# Debugging and Logging

SASS

```
@mixin btn-with-border($col) {  
  @if (lightness($col) < 50%) {  
    @warn "Base color #{$col} is too dark";  
  }  
  background-color: $col;  
  border: 2px solid darken($col, 50%);  
}  
.btn {  
  @include btn-with-border(#a00);  
}
```

CSS

```
.btn {  
  background-color: #900;  
  border: 2px solid black;  
}
```

WARNING: Base color #a00 is too dark

Backtrace:

```
... /sass/app.scss:3, in mixin `btn-with-border`  
... /sass/app.scss:10
```

# Debugging and Logging

SASS

```
@mixin btn-with-border($col) {  
  @if (lightness($col) < 50%) {  
    @error "Base color #{$col} is too dark";  
  }  
  background-color: $col;  
  border: 2px solid darken($col, 50%);  
}  
.btn {  
  @include btn-with-border(#a00);  
}
```

CSS



```
[sass]  error: Base color #a00 is too dark
```

```
in ... /src/sass/app.scss:3:5
```

```
Error: Base color #a00 is too dark
```

```
at options.error (.../node_modules/node-sass/lib/index.js:291:26)
```

# Defining your own functions

SASS

```
$w: 2px;  
@function double($x) {  
  @return 2 * $x;  
}  
.thin-border {  
  border-width: $w;  
}  
.thick-border {  
  border-width: double($w);  
}
```

CSS

```
.thin-border {  
  border-width: 2px;  
}  
.thick-border {  
  border-width: 4px;  
}
```

# Defining your own functions

SASS

```
@function biggest_channel_only($color) {  
    $r: red($color);  
    $g: green($color);  
    $b: blue($color);  
    @if $r > $g and $r > $b {  
        @return rgb($r, 0, 0);  
    }  
    @else if $g > $b and $g > $r {  
        @return rgb(0, $g, 0);  
    }  
    @else {  
        @return rgb(0, 0, $b);  
    }  
}
```

# Defining your own functions

SASS

```
.cadetblue { color: biggest_channel_only(cadetblue); }
.peru       { color: biggest_channel_only(peru); }
.lawngreen { color: biggest_channel_only(lawngreen); }
```

CSS

```
.cadetblue { color: #0000a0; }
.peru       { color: #cd0000; }
.lawngreen { color: #00fc00; }
```

# Exercise: Relative Luminance

- ▶ Exercise 6 involved using @if to ensure that the text/background color combination on a button was readable
- ▶ This quite the right concept. What we really want is to base the label text color on a difference in “relative luminance” between the label and background colors
- ▶ First, we’ll need to convert our RGB values from their current form (a gamma-compressed color space) to a linear color space.

@if

- ▶ A pow(\$base, \$exp) function has been provided

$$C_{\text{linear}} = \begin{cases} \frac{C_{\text{srgb}}}{12.92}, & \text{pow } C_{\text{srgb}} \leq 0.04045 \\ \left(\frac{C_{\text{srgb}} + a}{1+a}\right)^{2.4}, & C_{\text{srgb}} > 0.04045 \end{cases}$$

- where  $a = 0.055$  and where  $C$  is  $R$ ,  $G$ , or  $B$ .

Time: 35min

Command to run: `./run -e luminance`

# Exercise: Relative Luminance

- ▶ Next, we'll want to weigh these linear RGB values as follows

$$Y = 0.2126R + 0.7152G + 0.0722B.$$

- ▶ Finally, we'll want to change the @if involved with determining button text color, so that instead of the existing comparison, **it sets a white text color when the background color and the color white differ in luminance by more than 0.7**