INTRO TO SASS

CLASS 2

DOWNLOAD CLASS 2 FILES:

cfarm.co/sass2

REVIEW: MIXINS

If you haven't already, create these mixins in _utilities:

- dropshadow/text-shadow style
- clearfix
- gradient

```scss
@mixin example($optionalArgument) {
  property: value;
  property: $optionalArgument;
}
p {
  @include example($optionalArgument);
}
```

MULTIPLE ARGUMENTS

Mixins can take multiple arguments, as we saw with the gradient mixin:

```
@mixin gradient($color1, $color2) {
  background-image: -webkit-linear-gradient($color1, $color2, $color1);
  background-image: -moz-linear-gradient($color1, $color2, $color1);
  background-image: linear-gradient($color1, $color2, $color1);
}
```

DEFAULT ARGUMENTS

You can also specify *default* values for arguments, for next-level laziness (a good thing):

```
@mixin gradient($color1: #fff, $color2: #666) {
  background-image: -webkit-linear-gradient($color1, $color2, $color1);
  background-image: -moz-linear-gradient($color1, $color2, $color1);
  background-image: linear-gradient($color1, $color2, $color1);
}
```

INTERPOLATION

With some styles, we may want to use variables right next to text, like with our image rotate style - try it in _utilities:

```
@mixin rotate($degree, $position) {
  -webkit-transform: rotate(-5deg);
  -moz-transform: rotate(-5deg);
  transform: rotate(-5deg);
  -webkit-transform-origin: $position;
  -moz-transform-origin: $position;
  transform-origin: $position;
}
```

INTERPOLATION

We do this by using interpolations, which is a special syntax for variables that appear inside regular CSS text:

```
@mixin rotate($degree, $position) {
  -webkit-transform: rotate(#{$degree}deg);
  -moz-transform: rotate(#{$degree}deg);
  transform: rotate(#{$degree}deg);
  -webkit-transform-origin: $position;
  -moz-transform-origin: $position;
  transform-origin: $position;
}
```

@EXTEND

Sometimes we have styles that duplicate another class, then add to it. We can use a mixin for these, OR we can use @extend:

```
.headline {
  font-size: 2em;
  font-weight: bold;
}
.lead-story-headline {
  @extend .headline;
  text-decoration: underline;
  text-transform: uppercase;
}
```

LET'S DEVELOP IT

- Create new mixins using the following:
  - Multiple arguments
  - Default arguments
  - Interpolation
  - @extend to include common styles
- Use these mix-ins in your styles.scss file
- Compile to CSS and refresh your index page to see your changes

CSS3: BROWSER PREFIXES

Lots of CSS3 features require browser prefixes or have complex syntaxes. These are perfectly suited to mixins, so we can include them in our styles without having to copy and paste from sites like CSS3 Please.

```scss
@mixin rotate($degree, $position) {
  -webkit-transform: rotate(#{$degree}deg);
  -moz-transform: rotate(#{$degree}deg);
  transform: rotate(#{$degree}deg);
  -webkit-transform-origin: $position;
  -moz-transform-origin: $position;
  transform-origin: $position;
}
```

CSS3: BROWSER PREFIXES

Gradients are CSS3 and require browser prefixes to work in Firefox, Chrome, Safari, & Internet Explorer.

```
@mixin gradient($color1: $bodyBackground, $color2: $accentBackground) {
  background-image: -webkit-linear-gradient($color1, $color2, $color1);
  background-image: -moz-linear-gradient($color1, $color2, $color1);
  background-image: linear-gradient($color1, $color2, $color1);
}
@include gradient(#fff, #000);
```

CSS3 MIXINS

Other CSS3 features that require browser prefixes to work in most popular browsers include:

- @font-face
- transitions
- box-shadow
- opacity

MIXINS FOR UI ELEMENTS

Mixins are very useful for common design patterns. A great example is the CSS sprite design pattern.

Here's YouTube's icon sprite:

Sprite image:

CSS for icon:

```
.icon {
display: block;
background: no-repeat url(images/youtube_sprite.png) -395
background-size: auto;
width: 18px;
height: 18px; }
```

SPRITE MIXINS

If you set up your Photoshop sprite to use grid lines, you can use Sass to easily position your sprite image without lots of trial and error.

```scss
@mixin sprite-position($x:0, $y:0) {
  $gridSize:  -32px;
  $offsetX:   $x * $gridSize;
  $offsetY:   $y * $gridSize;
  background-position: $offsetX $offsetY;
}
@mixin sprite-image($file, $x:0, $y:0) {
  background-image: url("../../images/icons/#{$file}");
  @include sprite-position($x, $y);
}
```

CSS3 MIXINS

Other common UI elements that Sass mixins make easy:

- buttons
- image replacement
- error messages
- CSS shapes (triangles, arrows)

LET'S DEVELOP IT

- Create new mixins for the following use cases, combining as many as possible:
  - Sprite images
  - Opacity for all browsers, including IE
  - CSS3 box-shadow
  - @font-face
  - transitions
  - CSS shapes
- Use these mix-ins in your styles.scss file
- Compile to CSS and refresh your index page to see your changes

BREAK TIME!

Stand up and stretch - we'll resume in 5 minutes

SPRITE MIXINS

Sass has logic statements that you can use to create conditionals. They are @if, @else if, and @else

```
@mixin opacity($value: 0.5) {
  @if $value == transparent {
    opacity: 0;
  } @else if $value == opaque {
    opacity: 1;
  } @else {
    opacity: $value;
  }
}
@include opacity(transparent);
```

IF/ELSE IF

Rewrite your existing mixins to use if/else if statements, so that they output different CSS depending on different arguments.

```
@mixin arrow($direction: right) {
  @if $direction == right {
    //right arrow styles
  }
  @else if $direction == left {
    // left arrow styles
  }
}
```

FOR LOOPS

With @for loops, you can make Sass write your classes and styles for you.

Sass code:

```
@for $i from 1 through 3 {
  .column-#{$i} { width: 2em * $i
}
```

CSS output:

```
.column-1 {
  width: 2em;
}
.column-2 {
  width: 4em;
}
.column-3 {
  width: 6em;
}
```

FOR LOOPS

- Use loops to write classes and styles for you! Use them to make columns that fit within in our layout, and add links to our footer nav in these columns.
- Experiment with math in your loop's styles - change the width, padding, even font-size

```
@for $i from 1 through 3 {
  .column-#{$i} { width: 2em * $i; }
}
```

@EACH

With @each, you can loop through a list of item and create styles for each item in the list.

Sass code:

```
@each $icon in youtube, twitter, facebook {
  .icon-$icon {
    background-image: url('#{$icon}.png');
  }
}
```

CSS output:

```
.icon-youtube {
  background: url('youtube.png');
}
.icon-twitter {
  background: url('twitter.png');
}
.icon-facebook {
  background: url('facebook.png');
}
```

@EACH

- Create background image styles for each Famous Woman article
- Try SubtlePatterns.com for images to use
- Write these background styles with @each and a list
- Bonus: style your social media icons using @each and @if logic

```
each $woman in ada, grace, frances, barbara, anita, maria {
 .#{$woman}-bg {
   background-image: url('images/#{$woman}.png');
 }
```

EXTEND SASS

The following tools can help you write Sass even faster:

- CodeKit: Compile Sass & more without command line
- Sass Sleuth: See Sass line numbers in your browser
- Live Reload: Compile Sass without refreshing your browser
- Compass

MORE TO EXPLORE

- Haml: like Sass for HTML
- Rails: Rails is a software development framework written in Ruby. Sass & Haml are used in Rails apps. Try the Rails Girls tutorials to build your own app!
- LESS: a CSS preprocessor like Sass with different syntax

QUESTIONS?