KAPLAN TEST PREP

# BEM, Sass, Mixins

## Front End Workshop

KAPLAN TEST PREP

# Sass 101

What is Sass?
How can we use this to write more efficient CSS?

# Mixins

What are mixins?
How can we use mixins to keep our code clean and DRY?

# BEM (Block Element Modifier)

Why use BEM architecture?
How we use this to our advantage?

KAPLAN TEST PREP

# Sass 101

# Sass 101
## What is Sass?

**Sass** lets you use feature that don't exist in regular CSS such as:

- Nesting
- Variables
- Mixins
- More (inheritance, built in functions, etc)

Helps to keep stylesheets organized and allows you to develop faster

To install: https://sass-lang.com/install

# Sass 101
What is Sass?

## Nesting

We can utilize the ampersand (**&**) in Sass to nest children into its parent selectors.

Very useful for pseudo classes!

styles.scss

```
.submit {
    color: green;

    &:hover {
        color: red;
    }

    &:after {
        overflow: hidden;
    }
}
```

styles.css

```
.submit {
    color: green;
}

.submit:hover {
    color: red;
}

.submit:after {
    overflow: hidden
}
```

# Sass 101
What is Sass?

## Nesting

styles.scss

```
.car {
    padding: 10px;
    border: 1px solid black;

    &__wheel {
        color: gray;
        margin: 0;
    }

    &__door {
        color: red;
        text-align: center;
    }
}
```

styles.css

```
.car {
    padding: 10px;
    border: 1px solid black;
}

.car__wheel {
    color: gray;
    margin: 0;
}

.car__door {
    color: red;
    text-align: center;
}
```

## Variables

Traditionally, we would create css classes like this

styles.css

```
header {
    background-color: #42327F;
}

footer {
    background-color: #005DE8;
}

h1, h2 {
    color: #FFFFFF;
}
```

# Variables

What if we had multiple stylesheets?

- buttons.css
- dividers.css
- iconography.css
- notifications.css
- slider.css
- tabs.css

If branding or designs change, you would have to find and replace each and every property!

# Sass 101
What is Sass?

## Variables

styles.scss

```
$primary-color: #42327F;
$secondary-color: #005DE8;
$text-color: #FFFFFF;

header {
    background-color: $primary-color;
}

footer {
    background-color: $secondary-color;
}

h1, h2 {
    color: $text-color;
}
```

# Sass 101
What is Sass?

## Variables

styles.scss

```
$primary-color: #005DE8;
$secondary-color: #434343;
$text-color: #FFFFFF;

header {
    background-color: $primary-color;
}

footer {
    background-color: $secondary-color;
}

h1, h2 {
    color: $text-color;
}
```


If designers asked you to change colors or padding, you can simply modify the variable

## Importing Variables

Sass lets us import other Sass files (scss) into our style sheets using

```
@import
```

## Built-in Functions

$base-color: #AD141E;

- lighten($base-color, 10%);
- darken($base-color, 10%);

KAPLAN TEST PREP

# Mixins

## Mixins

Think of them like CSS functions that you can include wherever you want.

Mixins can take in arguments like Javascript functions

Help you reuse code throughout your app

# Mixins
What are mixins?

## Reusing code

Let's say you have the same overlay style in your modal-overlay, image-overlay, and menu-overlay. One way to reuse this code is creating a **mixin.**

mixins.scss

```
@mixin overlay() {
    bottom: 0;
    left: 0;
    right: 0;
    top: 0;
    position: absolute;
    background: black;
    opacity: 0.9;
}
```

# Mixins
What are mixins?

## Reusing code

styles.scss

```
.modal__background {
    @include overlay();
}

.header__image--dimmed {
    @include overlay();
}

.menu__overlay {
    @include overlay();
}
```

# Mixins
What are mixins?

## Reusing code

styles.css

```
.modal__background {
    bottom: 0;
    left: 0;
    right: 0;
    top: 0;
    position: absolute;
    background: black;
    opacity: 0.9;
}

.header__image--dimmed {
    bottom: 0;
    left: 0;
    right: 0;
    top: 0;
    position: absolute;
    background: black;
    opacity: 0.9;
}
```

```
.menu__overlay {
    bottom: 0;
    left: 0;
    right: 0;
    top: 0;
    position: absolute;
    background: black;
    opacity: 0.9;
}
```

# Mixins
What are mixins?

## Passing Arguments

mixins.scss

```
@mixin roundButton($radius, $padding){
    padding: $padding;
    border-radius: $radius;
}
```

styles.scss

```
.primary-button {
    background-color: $primary-color;
    @include roundButton(4px, 10px);
}
```

KAPLAN TEST PREP

# BEM

## Block Element Modifier

The Block, Element, and Modifier (BEM) methodology is a common naming convention for CSS classes developed by Yandex.

The core idea behind BEM is to provide a strict way to arrange your CSS classes into modules.

https://medium.com/@adamaso/friends-with-bem-efits-3237d12b8142

## Block

**Block**: Represents a module or context where the element finds itself.

(“***What*** *is it?*”)

- navigation
- footer
- item-review

## Element

Describes a component within the block that performs a particular function.

(*"What does this **do/show**?"*)

- item
- link
- button

## Modifier

Defines the appearance, state, or behavior of a block or element.

(*"What **state** is it in?"*)

- active
- red
- large

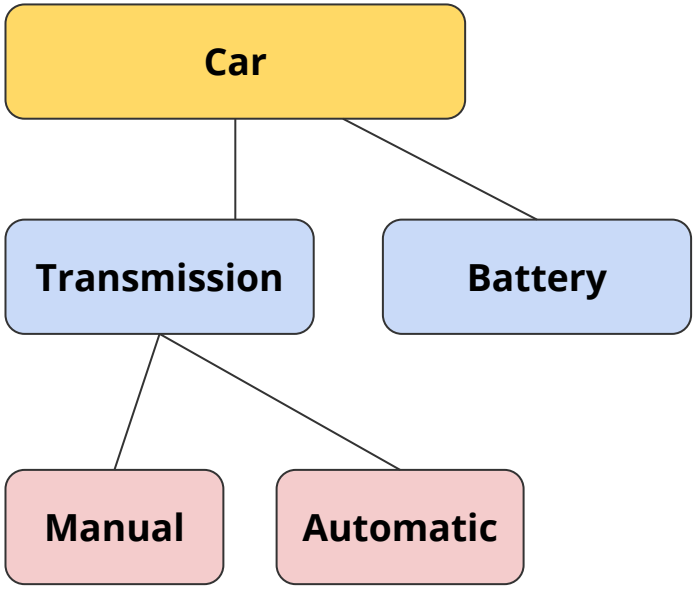
## Syntax

We represent this syntax with two underscores for elements and two dashes for modifiers:

```
.BLOCK__ELEMENT--MODIFIER
```

# Further Understanding

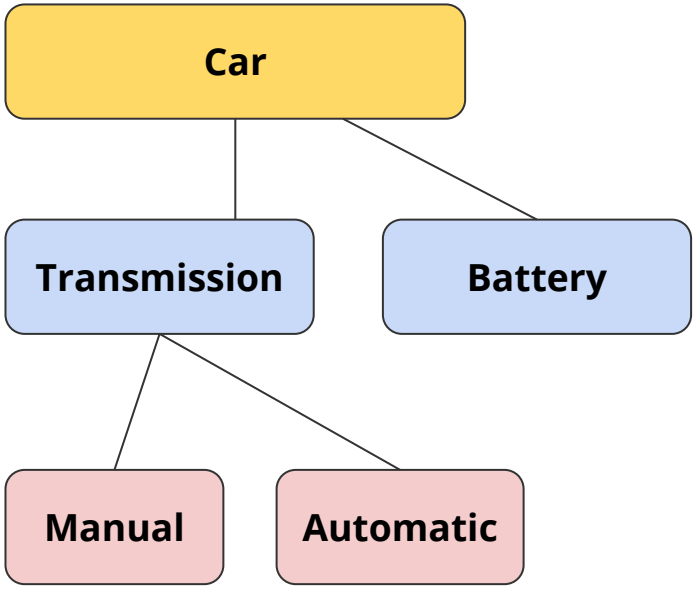
To grasp a further understanding of how these are all related, think of it like an analogy. Let's say we want to display different classes to describe a car's transmission.




The **block** or module would be the Car. The **element** we are showing is a Transmission, and is **modified** or described by Automatic and Manual.

# Further Understanding

To grasp a further understanding of how these are all related, think of it like an analogy. Let's say we want to display different classes to describe a car's transmission.




```
.car {}
.car__transmission {}
.car__transmission--automatic {}
.car__transmission--manual {}
```

# Why BEM?

- Self-documenting CSS! It displays the hierarchy between different nodes' relationships in the DOM.
- Strict class naming promotes encapsulation and reduce naming collisions or using classes outside a given context.
- Tells other developers more about what a piece of markup is doing from its name alone.
- Removes the need for descendant selectors.
- Component styles are decoupled and highly portable from project to project. In practice, this means an improvement in code quality and development speed!

# Practice

Let's take a deeper dive and see how this can be used practically:

- Use BEM, nesting, and variables to style the primary and secondary buttons
- Style the primary and secondary buttons so they have the same colors as table rows

Starter app:
https://stackblitz.com/edit/sass-variable-demo

Finished app:
https://stackblitz.com/edit/sass-variable-demo-finished