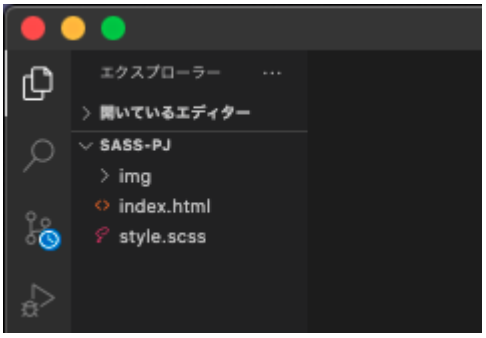
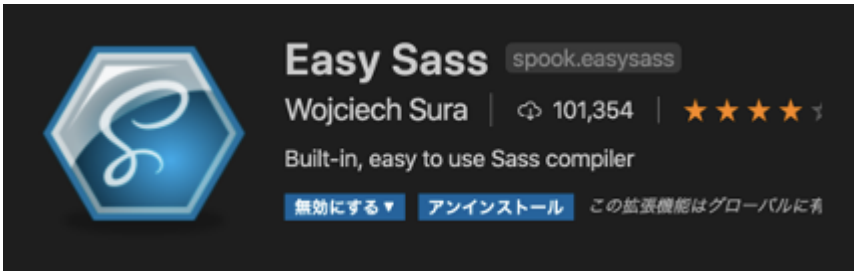
注意点
「教材全ての内容を期間内に学習するのが難しい」というご意見を多数いただいたため、この章は学習しない運びとなりました。
そのためこちらの章は学習せず次のカリキュラムに進んでください。

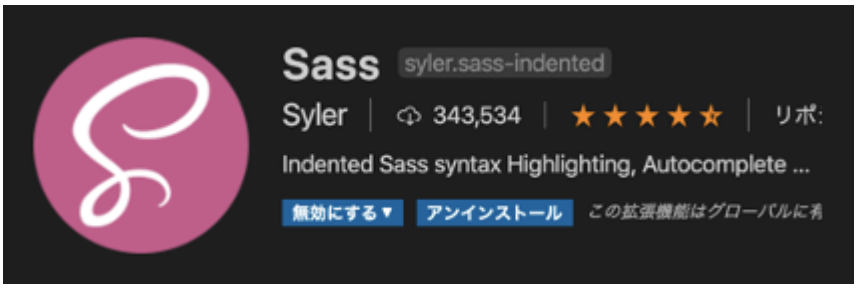
Sassとは
SassはCSS拡張メタ言語と呼ばれるCSSを便利に記述するための言語のひとつです。
SassにはSASSとSCSSという二つの記法があり、SCSSの方が広く普及しています。
Sassは関数や定数を利用することで再利用性や保守性を高めることや、ネストにすることで作業効率を上げることができる言語なのでしっかりと理解しましょう。

セットアップ
まずは「sass-pj」というプロジェクトを作成し、その直下に「index.html」と「style.scss」というファイルを作成します。



Sassで書いたコードのままではスタイルを適用することができません。
SassはCSSに変換する必要があります。
Sassで書いたコードをCSSに変換するために、Visual Studio Codeの「Easy Sass」と「Sass」というプラグインをダウンロードしましょう。





「sass-pj」のindex.htmlとstyle.scssに下記コードを記述して保存（Command+S）すると「style.css」と「style.min.css」というファイルが自動で作成されます。

「style.min.css」は「style.css」のコメントや不要なスペースを取り除くことで、ファイルサイズが圧縮されたテキストファイルです。
そのため「index.html」でCSSを読み込む時は、読み込みの高速化を図るためにリンク先のURLには「style.min.css」を設定しましょう。

index.html

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Document</title>
  <link rel="stylesheet" href="style.min.css"/>
</head>
<body></body>
</html>
```

style.scss

```
.test{
  color: #000;
  a{
    background: #ffffff;
  }
}
```

これをstyle.cssで見てみると、入れ子構造のSCSSのコードが下のようにCSSに変換されていることがわかります。

```
.test {
  color: #000;
}
.test a {
  background: #ffffff;
}
```

ネスト

ネストとは、あるものの中にそれと同じ形や種類の（一回り小さい）ものが入っている状態や構造のことです。
例えばヘッダーをCSSで装飾したいとき、CSSには以下のように記述します。

```
header{
  height: 80px;
}
header ul{
  display: flex;
}
header ul a{
  color: #000;
}
header ul a::before{
  content: "・";
}
```

しかしSassでは以下のようにネストさせて書くことができます。
また&を使用することで今までのネストしてきた全てのセレクタを継承することができます。

```
header {
  height: 80px;
  ul {
    display: flex;
    a {
      color: #000;
      &::before {
        content: "・";
      }
    }
  }
}
```

変数

「$変数名:値」で変数を設定することで、同じカラーコードの複数回のコピー&ペーストや、全体に適用される大幅な修正があった際に全て手作業で直す手間を省くことが可能です。
testというclass、ヘッダー、そしてメインを装飾したい時にCSSでは以下のように記述します。

```
.test {
  color: #1234ab;
```

```
}
.test li {
  border: 1px solid #1234ab;
}
.test li p {
  background-color: #1234ab;
}
.test a {
  color: #1234ab;
}
header {
  height: 60px;
}
main {
  margin-top: 80px;
}
```

Sassでは以下のように変数を定義することで、再利用したい時に呼び出すことができます。
また、変数で定義した値に対しては四則演算を行うことができます。

```
$darkBlue: #1234ab;
$headerHeight: 60px;

.test {
  color: $darkBlue;
  li {
    border: 1px solid $darkBlue;
    p {
      background-color: $darkBlue;
    }
  }
  a {
    color: $darkBlue;
  }
}
header {
  height: $headerHeight;
}
main {
  margin-top: $headerHeight + 20px;
}
```

mixin

基本

mixinとは、よく使うスタイル等をあらかじめ変数として定義しておき、利用する際に呼び出す機能です。
「@mixin 変数名」で定義し、「@include 変数名」で呼び出すことで何度も同じ記述をする必要がなくなります。
mixinはセレクタもプロパティも定義できるので、定義するときにプロパティも組み込みましょう。

```
@mixin buttonStyle {
  a {
    background-color: blue;
    color: #fff;
    display: flex;
    align-items: center;
    justify-content: center;
    border-radius: 5px;
    height: 50px;
    width: 200px;
  }
}
.buttons {
  .modalopen {
    @include buttonStyle;
  }
}
.test {
  @include buttonStyle;
}
```

引数の使い方

「@mixin 変数名($別の変数名)」で定義し、「@include 変数名(値)」で呼び出すことで、定義したもののの一部を簡単に変更することが可能です。

```
@mixin buttonStyle($color) {
  a {
    background-color: $color;
    color: #fff;
    display: flex;
    align-items: center;
    justify-content: center;
    border-radius: 5px;
    height: 50px;
```

```
    width: 200px;
  }
}
.buttons {
  .modalopen {
    @include buttonStyle(red);
  }
}
.test {
  @include buttonStyle(purple);
}
```

初期値の設定
「@mixin 変数名($別の変数名: 初期値)」で定義することで初期値を設定することができます。
一番使用するものを初期値として設定して、イレギュラーなものを使用するときは引数をつけましょう。

```
@mixin buttonStyle($color: white, $height: 50px) {
  a {
    background-color: $color;
    color: #fff;
    display: flex;
    align-items: center;
    justify-content: center;
    border-radius: 5px;
    height: $height;
    width: 200px;
  }
}
.buttons {
  .modalopen {
    @include buttonStyle;
  }
}
.test {
  @include buttonStyle(purple, 80px);
}
```

任意の引数にのみ値を代入
「@include 変数名($代入したい箇所の変数名: 値)」で呼び出すことで任意の引数のみに指定することができます。

```scss
@mixin buttonStyle($color: white, $height: 50px, $width: 200px) {
  a {
    background-color: $color;
    color: #fff;
    display: flex;
    align-items: center;
    justify-content: center;
    border-radius: 5px;
    height: $height;
    width: $width;
  }
}
.buttons {
  .modalopen {
    @include buttonStyle($width: 180px);
  }
}
.test {
  @include buttonStyle($height: 80px);
}
```

実践

Sassを使用するにあたって必要なポイントを下記のコードを書きながら理解していきましょう。
「lighten/darken(色, ○%); 」
定義した色に対して、○%明るく/暗く変更します。

```scss
$red: #f00;

.button{
  width: 300px;
  height:50px;
  display: flex;
  align-items: center;
  justify-content: center;
  border-radius: 5px;
  font-size: 20px;
  color: black;
  background-color: lighten($red, 20%);/*redより20%明るい色に変更*/
  cursor: pointer;
}
```

「 transparentize(色, 0～1の値);」
定義した色×値の透明度に変更します。

```
$red: #f00;

.button {
  width: 300px;
  height: 50px;
  display: flex;
  align-items: center;
  justify-content: center;
  border-radius: 5px;
  font-size: 20px;
  color: black;
  background-color: transparentize($red, 0.5);/*redを50%透明に変更
*/
  cursor: pointer;
}
```

「adjust-hue(色, 角度); 」
定義した色から○度色相を回転させた色に変更します。

```
$red: #f00;

.button {
  width: 300px;
  height: 50px;
  display: flex;
  align-items: center;
  justify-content: center;
  border-radius: 5px;
  font-size: 20px;
  color: black;
  background-color: adjust-hue($red, 90);/*redから90度色相を回転させ
た色に変更*/
  cursor: pointer;
}
```

「 linear-gradient(色1, 色2); 」
色1から色2へとグラデーションをかけます。

```
$red: #f00;
```

```
.button {
  width: 300px;
  height: 50px;
  display: flex;
  align-items: center;
  justify-content: center;
  border-radius: 5px;
  font-size: 20px;
  color: black;
  background-image: linear-gradient($red, adjust-hue($red,
90));/*red→redから90度色相を回転させた色へとグラデーションをかける*/
  cursor: pointer;
}
```

「 box-shadow: αpx βpx γpx lighten/darken(色, ○%); 」
定義した色より○%明るく/暗くした影をboxから水平方向にαpx、垂直方向にβpx、ぼかし距離γpxで作成します。

```
$color: blue;

.button {
  width: 300px;
  height: 50px;
  display: flex;
  align-items: center;
  justify-content: center;
  border-radius: 5px;
  font-size: 20px;
  color: black;
  background-image: linear-gradient($color, adjust-hue($color,
90));
  box-shadow: 5px 5px 3px lighten($color, 10%); /*影をつける*/
  cursor: pointer;
}
```

条件分岐

以下のようなボタンを条件分岐を用いて作りましょう。

index.html

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Document</title>
  <link rel="stylesheet" href="style.min.css"/>
</head>
<body>
  <a href="" class="button">前へ</a>
  <a href="" class="button2">次へ</a>
</body>
</html>
```

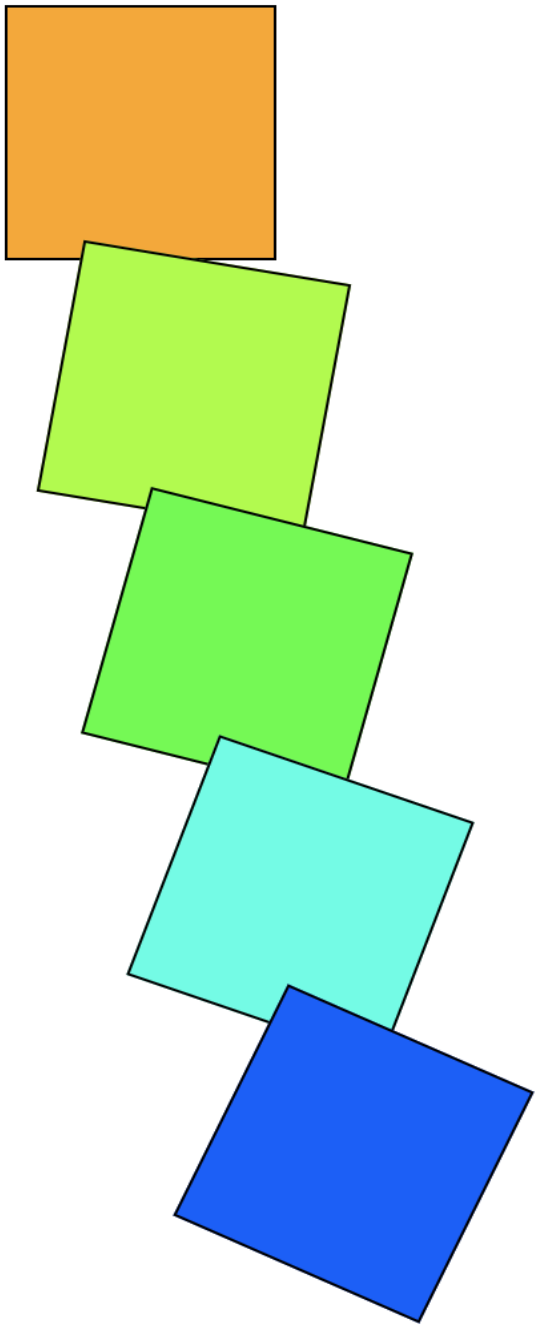
style.scss

```
a {
  text-decoration: none; /*下線を消す*/
}

@mixin arrowStyle{
  position: absolute;
  top: 50%;
  transform: translateY(-50%);
  font-weight: normal;
}

@mixin buttonStyle($direction: "left", $arrowMargin: 20px){
  width: 200px;
  height: 40px;
  display: flex;
  justify-content: center;
```

```
  align-items: center;
  background-color: blue;
  color: white;
  font-weight: bold;
  border-radius: 5px;
  transition: 0.3s;
  position: relative;
  margin-bottom: 20px;

  @if $direction == "left"{
    &::before {
      @include arrowStyle;
      content: "<";
      left: $arrowMargin;
    }
  } @else if $direction == "right"{
    &::after {
      @include arrowStyle;
      content: ">";
      right: $arrowMargin;
    }
  }

  &:hover {
    opacity: 0.7; /*要素の透明度の指定*/
  }
}

.button{
  @include buttonStyle;
}

.button2{
  @include buttonStyle("right");
}
```

ループ処理

以下のような要素をループ処理を用いて作成してみましょう。



index.html

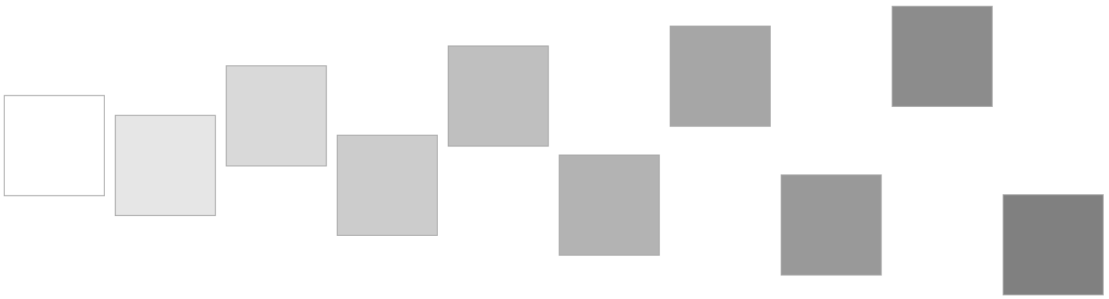
```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
```

```
    <title>Document</title>
    <link rel="stylesheet" href="style.min.css" />
  </head>
  <body>
    <ul class="boxes">
      <li></li>
      <li></li>
      <li></li>
      <li></li>
      <li></li>
    </ul>
  </body>
</html>
```

style.scss

```
ul {
  list-style: none;
}

.boxes {
  li {
    $startColor: orange;
    width: 100px;
    height: 100px;
    background-color: $startColor;
    border: 1px solid black;
    @for $i from 2 through 5 {
      /*2から5番目のボックスに以下を適用する*/
      &:nth-of-type(#{$i}) {
        transform: translateX(($i - 1) * 20px) rotate($i * 5deg);
        background-color: adjust-hue($startColor, ($i - 1) *
45deg);
      }
    }
  }
}
```

以下のような要素をループ処理を用いて作成してみましょう。



index.html

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="style.min.css" />
  </head>
  <body>
    <ul class="boxes">
      <li></li>
      <li></li>
      <li></li>
      <li></li>
      <li></li>
      <li></li>
      <li></li>
      <li></li>
      <li></li>
      <li></li>
    </ul>
  </body>
</html>
```

style.scss

```
@charset "UTF-8";

ul {
  list-style: none;
```

```
}

.boxes {
  display: flex;
  margin: 100px 0;
  li {
    $startColor: white;
    width: 100px;
    height: 100px;
    background-color: $startColor;
    border: 1px solid #aaa;
    &:not(:first-of-type) {
      margin-left: 10px;
    }
    @for $i from 2 through 10 {
      &:nth-of-type(#{$i}) {
        background-color: darken($startColor, 5% * $i);

        $translateYValue: 10px;
        @if $i % 2 == 0 {
          transform: translateY($i * $translateYValue);
        } @else {
          transform: translateY($i * $translateYValue * -1);
        }
      }
    }
  }
}
```

レスポンシブ対応

Sassを用いてレスポンシブ対応を行うことで、メディアクエリを複数書く必要がなく管理性が高いコードを実装することができます。今回は、Sassでメディアクエリを管理する方法を紹介します。

ブレイクポイントの変数を宣言

ブレイクポイントを変数にまとめておくことで、ブレイクポイントを変更する際にこちらの変数の値を変更するだけで全てに適用することができます。

```
$sp: 481px;
$tab: 769px;
```

メディアクエリの記述
@mixin sp,@mixin tabと記述しておくことで、@includeを使用してコードを呼び出すことができます。
@includeに記述したコードが@contentの部分に入ります。

```
@mixin sp {
  @media (min-width: ($sm)) {
    @content;
  }
}
@mixin tab {
  @media (min-width: ($tab)) {
    @content;
  }
}
```

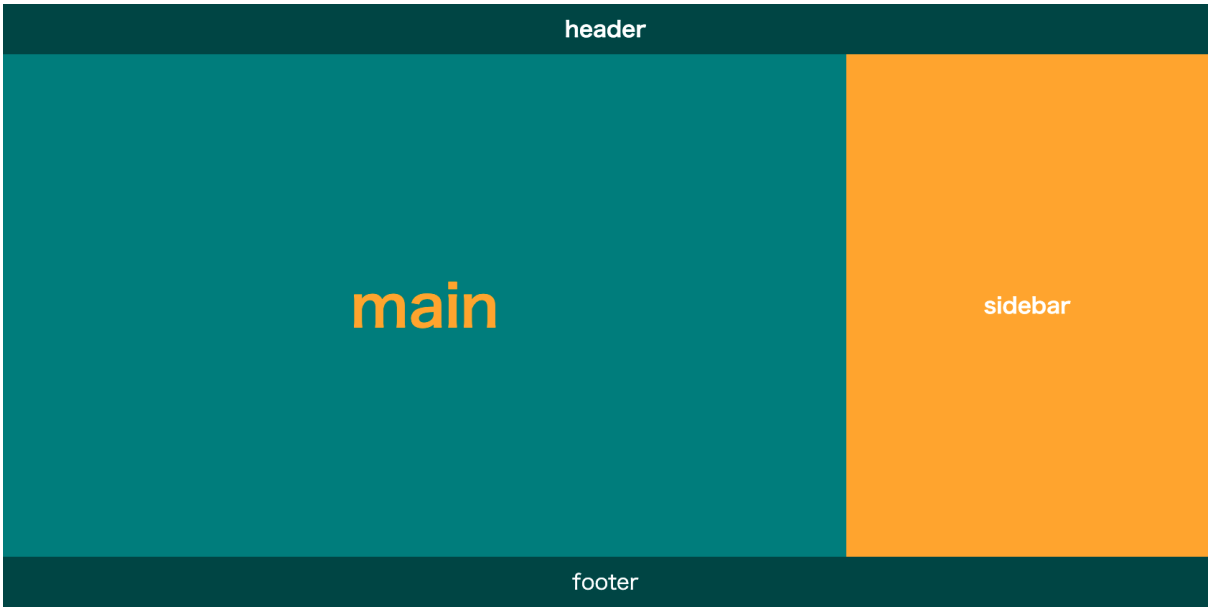
実際に使用
spというクラスは、画面幅がタブレット以上(769px)の場合はdisplay:none;を指定するというコード

```
.sp {
  display: block;
  @include tab {
    display: none;
  }
}
```

問題

以下の画面構造をSassを用いて作成してください。

- ベースカラー:#004e4e
- メインカラー:#008888
- アクセントカラー: #ffad12
- ヘッダー高さ:70px
- フッター高さ:70px
- メイン高さ:700px
- メイン幅:70%
- サイドバー高さ:700px
- サイドバー幅:30%



解答

index.html

```
<!DOCTYPE html>
<html lang="ja">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="css/reset.css" />
    <link rel="stylesheet" href="style.min.css" />
</head>

<body>
    <header class="header">
        <h1 class="header__ttl">header</h1>
    </header>
    <div class="wrap">
        <div class="main">
            <h2 class="main__ttl">main</h2>
        </div>
        <aside class="sidebar">
            <h2 class="sidebar__ttl">sidebar</h2>
        </aside>
    </div>
```

```
    <footer class="footer">
        <p class="footer__txt">footer</p>
    </footer>
</body>

</html>
```

style.scss

```
$base-clr: #004e4e;
$main-clr: #008888;
$accent-clr: #ffad12;

body {
  color: #fff;
  text-align: center;
}

.header {
  height: 70px;
  background-color: $base-clr;
  &__ttl {
    font-size: 30px;
    line-height: 70px;
  }
}

.wrap {
  display: flex;
  width: 100%;
  height: 700px;
}

.main {
  background-color: $main-clr;
  width: 70%;
  &__ttl {
    color: $accent-clr;
    line-height: 700px;
    font-size: 80px;
  }
}
```

```
.sidebar {
  background-color: #ffad12;
  width: 30%;
  &__ttl {
    line-height: 700px;
    font-size: 30px;
  }
}

.footer {
  background-color: $base-clr;
  height: 70px;
  &__txt {
    line-height: 70px;
    font-size: 30px;
  }
}
```