条件分岐や繰り返し処理

Sassにもプログラミングと同様に条件分岐や繰り返し処理が行えます。

現場で必ず使うとは限りませんが覚えておきましょう。

@ifを使って条件分岐を作る

やっていることはほとんどJavaScriptと変わらないので簡単に説明します。

`@if 条件`でif文を作ることができます。

trueとfalseの処理を作る場合は、`@else`を使用、3つ以上の条件分岐を作る場合は`@else if 条件`を書くことで条件を作ることができます。

```
$floatValue: true;

@if $floatValue {
  .is-float {
    float: left;
  }
}
@else {
  .is-float {
    float: right;
  }
}
```

上記は一番オーソドックスなパターンです。

次は比較条件を追加した例です。

```
$bgColor: 1;

@if $bgColor == 0 {
  .bg-color {
    background-color: red;
  }
}
@else if $bgColor == 1 {
  .bg-color {
    background-color: blue;
  }
}
@else {
  .bg-color {
    background-color: green;
  }
}
```

コンパイルしたCSSを確認すると2つ目の条件が追加されていると思います。

このように条件を決めて適用させるプロパティーを変更できます。

比較演算子はJavaScriptと同様のものが使えます。（`===`のような厳密な比較はできませんので気をつけてください。）

JavaScriptにもあった論理演算子ですが、Sassの場合は記号ではなく文字となります。(`and`、`or`、`not`)

注意点

if文が使えるのは一見便利に見えますが、変数などを使用して比較条件を作る場合、その変数は動的に値を変える必要があります。

理由は今回のような場合は値が上書きされることはないので誰かが手を加ない限りは通る条件が変わりません。

その都度人の手によって値が書き換えるのはあまり意味がないので動的に変える必要があります。

ロジックを組めばそれも可能かも知れませんがコーディングをやっている方が全員プログラミングに精通しているとは限らないので属人化するものを作るくらいならメンテナンス性に長けた書き方を取るのも1つの方法です。

@forを使って繰り返し処理をする

`@for 繰り返し条件`で繰り返し処理が行えます。

繰り返し条件には2つあるのでそれを紹介します。

`@for $変数名 from 開始の数値 through 終了の数値 {}`

`@for $変数名 from 開始の数値 to 終了の数値 {}`

この2つの違いは`through`と`to`です。この`through`と`to`では終了の数りが変わります。

`through`は指定した数値を含んで終了しますが、`to`は指定した数値を含まず終了します。

```
@for $value from 1 through 3 {
  .through-sample-#{$value * 10} {
    margin: 10px * $value;
  }
}

@for $value from 1 to 3 {
  .to-sample-#{$value * 10} {
    margin: 10px * $value;
  }
}
```

上記のスタイルをコンパイルしたCSSを見てもらうと違いがわかるかと思います。

@whileを使って繰り返し処理をする

`@while 繰り返しを継続する条件`で`@for`よりも複雑な処理ができます。

```
$value: 20;

@while $value > 0 {
  .while-sample-#{$value} {
    margin: $value + px;
  }
  $value: $value - 10;
}
```

`@for`では変数の値が1つずつしか増やせないのに対して、`@while`は条件となっている値を繰り返しの中で変更できます。

@eachを使ってリスト（配列）の要素に対して繰り返し処理をする

Sassでも配列を作ることができます。

```
// 配列
$bgColor:#acc1ed,#dc7668,#71d0f2;

// オブジェクト
$map: (
  "header": #acc1ed,
  "main": #dc7668,
  "footer": #71d0f2,
);
```

上記の`@each`で配列を用いて処理できます。

`@each $変数名 in リスト {}`

```
$bgColor:#acc1ed,#dc7668,#71d0f2;

@each $value in $bgColor {
  .bg-color {
    background: $value;
  }
}
```

コンパイルしたCSSを確認してください。

`$bgColor`へ書いた数だけスタイルが作成され、カラー番号が反映されています。

ただこれではすべて同じクラス名なので意味がないです。

クラス名を変える場合はオブジェクトを使用します。

```
$map: (
  "header": #acc1ed,
  "main": #dc7668,
  "footer": #71d0f2,
);

@each $key, $value in $map {
  .#{$key}-bg-color {
    background: $value;
  }
}
```

オブジェクトのkeyとvalueを第1引数、第2引数で渡し、それぞれ展開しています。

それによりクラス名も重複することのないクラス名が作れます。

繰り返し処理を実装するコツ

今回で繰り返し処理を覚えましたね！

ただ今回は単純な実装でしたが複雑な処理を行うこともあります。

エラーが起きてつまずくこともあると思います。

ただ、そう言った時に引数の中に実際何が入っているのかを確認しないままエラー解決するのはかなり苦労します。

そう言った時は`@debug`を使用してください。

`@debug 変数名;`としてコンパイルを実行するとコンソールに対象の中身を表示してくれます。

これで変数に期待した値が入っているか確認できますので実装の時は活用して見てください。

課題

1. ifの処理を`@mixin`の中で行い、対象のクラスの中で展開し、ifの条件で通ったCSSのプロパティーを反映させてください

```
.クラス名 {
  // この中に展開されプロパティーが反映される
}
```

2. `$snsClass`内にある配列の数だけクラスを作り、`$map`のkeyに対応したvalueの値をbackground-colorに反映させてください。

`_common.scss`ファイルを作成し以下のSassをコピペしてください。

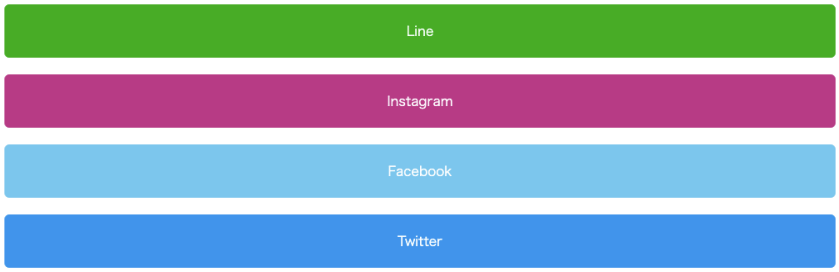
```
.inner {
  width: 100%;
  max-width: 1000px;
  margin: 0 auto;
  padding: 0 10px;
}

@mixin btn($color) {
  display: block;
  padding: 20px 30px;
  color: #fff;
  border-radius: 6px;
  cursor: pointer;
  background-color: $color;
  text-decoration: none;
  text-align: center;
}

$map: (
  "line": #00B900,
  "instagram": #CF2E92,
  "facebook": #71d0f2,
  "twitter": #1DA1F2,
  );
$snsClass:"line","instagram","facebook","twitter";
```

作業ファイルへインポートして使用してください。

作業して実際に表示されるか確認してください。



ボタンどうし余白があるのでそれはご自身でつけてください。

ボタンのタグは好きなものを選んでもらっても良いですがなぜそれを選んだのか理由も書いて提出してください。

```
/* 完成イメージ */
.content__sns__btn.is-line {
  display: block;
  padding: 20px 30px;
  color: #fff;
  border-radius: 6px;
  cursor: pointer;
  background-color: #00B900;
  text-decoration: none;
  text-align: center;
}
```

課題提出方法

1.と2.のファイルは別々にしてください。

2.の課題は_common.scss、作業ファイル、HTMLファイルを提出してください。