CSS設計

2019/6/21 勉強会

CSSの設計とは

・CSSのメンテナンス性や作業効率を向上させるためのクラス名の付け方

・CSSのコードの管理方法の考え方

CSS設計すると嬉しいこと

- メンテナンス性が向上する
- 大規模なWebサイトでも整ったコードで作れる
- 複数人でのコーディングが行いやすくなる
- 命名ルールを作ることで名前付けで悩む時間の短縮になる

なぜCSS設計が必要なのか

- HTML 側の内容とセレクタがマッチすればスタイルが適用される単純な仕組み
- 簡単であるがゆえに、自由に書けてしまう
- その場限りではそれらしい見た目を強引にでも作れてしまう

↓

コードの破綻がしやすい

↙ ↘

①CSSフレームワークを使おう！

②CSS設計のアイデアを
元にガイドラインを作成して解決しよう！

CSSが破綻する原因

- セレクタに詳細度がある
- スタイルが子要素に継承される
- 同じセレクタを多重定義できる
- スタイルが複合的に適用される

セレクタに詳細度がある問題

HTML

```
<p id="foo" class="bar">
ここの文字は何色でしょう？
</p>
```

CSS

```
#foo{ color: red; }
.bar{ color: blue; }
```

ここの文字は何色でしょう？

- IdとclassがIdで記述したスタイルが優先される
- あとからclassでスタイルを定義しても上書きできない
- 細かな詳細度を理解して管理しないとCSS破綻の原因となる

スタイルが子要素に継承される問題

HTML

```
<section>
  <p>文字サイズはいくつかな？</p>
</section>
```

CSS

```
body{ font-size: 16px; }
section p{ font-size: 32px; }
```

文字サイズはいくつかな？

- 親要素（body）で16pxに指定しているため、子要素であるpもスタイルを受け継いで16pxになる
- 親要素で定義したスタイルが小要素全てに適用されてしまうので、子要素をあとから上書きする必要がある
- フォントサイズの指定方法（px, rem, em, %）を統一しておかないと、メンテナンスしにくくなる

同じセレクタを多重定義できる問題

HTML

```
<p class="text">何色かな？</p>
```

CSS

```
.text{ color: black; }
.text{ color: red; }
```

何色かな？

- 親要素（body）で16pxに指定しているため、子要素であるpもスタイルを受け継いで16pxになる
- 親要素で定義したスタイルが小要素全てに適用されてしまうので、子要素をあとから上書きする必要がある
- フォントサイズを相対的な指定方法で定義してしまうと、1箇所修正するごとに子要素も修正しなくてはいけなくなってしまいCSS破綻の原因になる

スタイルが複合的に適用される問題

HTML

```
<p class="cls1 cls2 cls3">スタイルが合成された文字</p>
```

CSS

```
.cls1{ font-size: 16px; }
.cls2{ color: red; }
.cls3{ font-weight: bold; }
```

スタイルが合成された文字

- CSSは複合的にスタイルを受け入れるので意図的でなくマルチクラスにしてしまうとメンテナンスが難しくなる
- クラスの影響範囲を把握していないとCSS破綻の原因になる

CSS破綻を解決する代表的な4つのCSS設計アイデア

1. OOCSS
2. SMACSS
3. BEM
4. APBCSS

CSS設計アイデアを取り入れているWebサイト

1. **BEM**

   - Dropbox https://www.dropbox.com/

   - 任天堂WiiU https://www.nintendo.co.jp/hardware/wiiu/index.html

2. **SMACSS**

   - （参考フレームワーク）Pure.css https://purecss.io/

3. **OOCSS**

   - Bootstrap https://getbootstrap.com/

4. **APBCSS**

   - AbemaTV https://abema.tv/

5. **OOCSS + BEM**

   - Airbnb https://ja.airbnb.com/

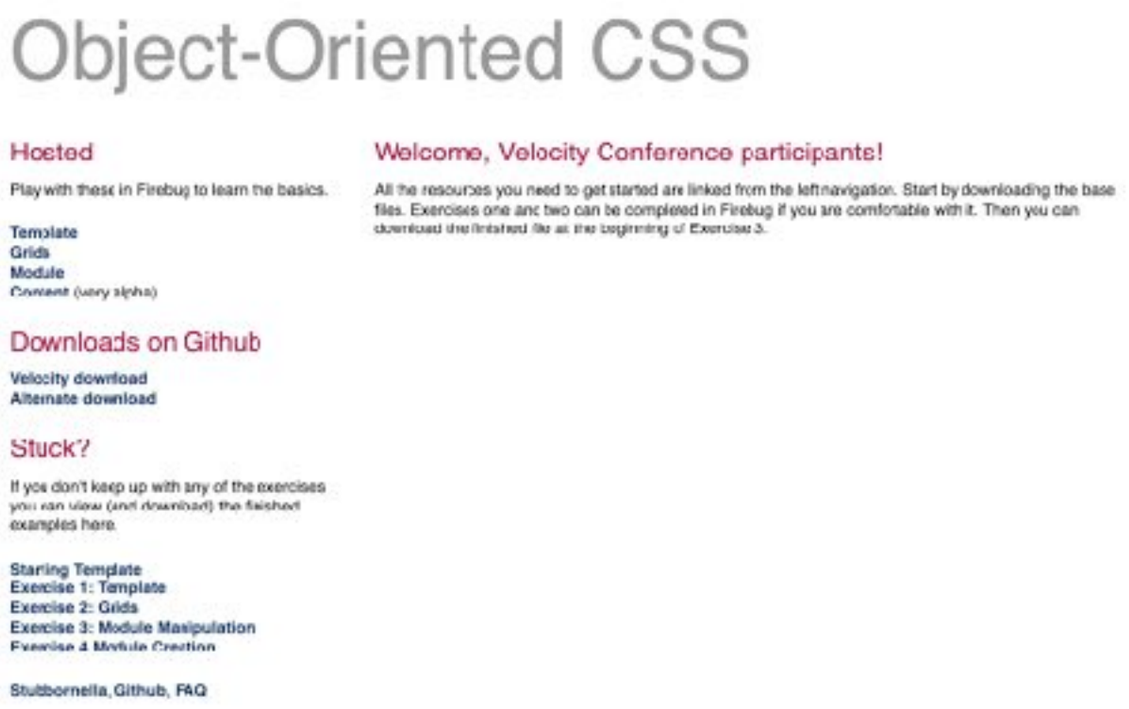
1. OOCSS

・Object-Oriented CSS

・オブジェクト指向の考え方を取り入れたCSS設計の方法

・オブジェクト指向プログラミングの概念を取り入れて、その実現に近づけたCSS設計法

・多くのCSS設計のアイデアの基礎



1. OOCSS

特徴

・オブジェクト指向
・複数のクラスを付与してスタイルを作る（マルチクラス）
・構造と見た目を分離する
・コンテナとコンテンツを分離する
・Bootstrapと同じ

メリット

・構造と見た目を分離するため、パーツの使い回しがしやすい
・コンテナとコンテンツを分離するので、作ったパーツを場所に依存せずどこでも使える

デメリット

・パーツがどこで使われているかわかりにくい
・コンテンツと分離するので微妙に違うパーツがあるときにパーツを増やしてしまうことになる

構造と見た目の分離

ダメな例

- 構造と見た目が一緒に記述されている

HTML

```html
<div>
  <a href="#" class="button">SUBMIT</a>
</div>
```

CSS

```css
.button {
  display: inline-block;
  padding: 0.5rem 1rem;
  text-decoration: none;
  background: #668ad8;
  color: #FFF;
  border-bottom: solid 4px #627295;
  border-radius: 3px;
  font-size: 1rem;
}
```

- 構造と見た目が一緒に記述されている
- 色違いの同じボタンを作る場合、新たなクラスでほぼ同じスタイルを定義しないといけないので無駄なコードが増えてしまう

Goodな例

- 構造と見た目が分離

HTML

```html
<div>
  <a href="#" class="button button-submit">SUBMIT</a>
</div>
<br>
<div>
  <a href="#" class="button button-cancel">CANCEL</a>
</div>
```

CSS

```css
.button {
  display: inline-block;
  padding: 0.5rem 1rem;
  text-decoration: none;
  border-bottom: solid 4px #627295;
  border-radius: 3px;
}
.button-submit {
  background: #668ad8;
  color: #FFF;
  font-size: 1rem;
}
.button-cancel {
  background: #ff6666;
  color: #FFF;
  font-size: 1rem;
}
```

SUBMIT

CANCEL

- 構造と見た目が分離できている
- ボタンの構造（padding,borderなど）は1つのクラスに定義し、2つの種類のボタンで使いまわしているため無駄なコードを生まない
- 見た目（color、font-sizeなど）は分けることによりメンテナンス性も向上

コンテナとコンテンツを分離

ダメな例

- コンテナとコンテンツが分離できていない

HTML

```html
<header class="header">
  <h1 class="logo">logo</h1>
</header>
```

CSS

```css
.header .logo {
  background-color: pink;
  color: white;
  padding: 1rem;
  width: 10rem;
  text-align: center;
}
```


- コンテナとコンテンツが分離できていない
- 場所に依存しているため、logoクラスを他の場所で使いまわしができない

Goodな例

- コンテナとコンテンツが分離

HTML

```html
<header class="header">
  <h1 class="logo">logo</h1>
</header>
<br>
<footer class="footer">
  <h1 class="logo logo-small">logo</h1>
</footer>
```

CSS

```css
.logo {
  background-color: pink;
  color: white;
  padding: 1rem;
  width: 10rem;
  text-align: center;
}

.logo-small {
  width: 5rem;
}
```


- コンテナとコンテンツが分離できている
- 場所に依存しないため、logoをフッターにも使い回すことができる。
- logoをヘッダーより小さくしたい場合は、マルチクラスで対応できる

2. SMACSS

- Scalable and Modular Architecture for CSS
- SMACSS は CSS を 5 種類のルールにカテゴライズして記述する

http://smacss.com/

2. SMACSS

特徴

- base/layout/module/state/themeの5つにCSSをカテゴライズ
- 命名ルールがある
- CSSをより体系立て、より構造化
- マルチクラス

メリット

- 共通で使えるものは共通化するので使い回ししやすい
- 繰り返し項目のパターン化によってコード量が少なくなる

デメリット

- 場所に依存する
- パーツが多すぎるサイトでは良いところを活かせられない
- 何がレイアウトで何がモジュールなのかわかりにくい

2. SMACSS

ルール

base	デフォルトのスタイル	body, html
layout	ページをエリアごとに分割するためのスタイルを定義します（位置の定義）	l-main,l-grid
module	再利用可能なパーツの指定	*-box,nav,logo
state	moduleやlayoutの状態を指定	is-active,is-hidden
theme	デザインテーマを指定（大体色を指定する）	theme-sea

上記の5つのフォルダを作成し、

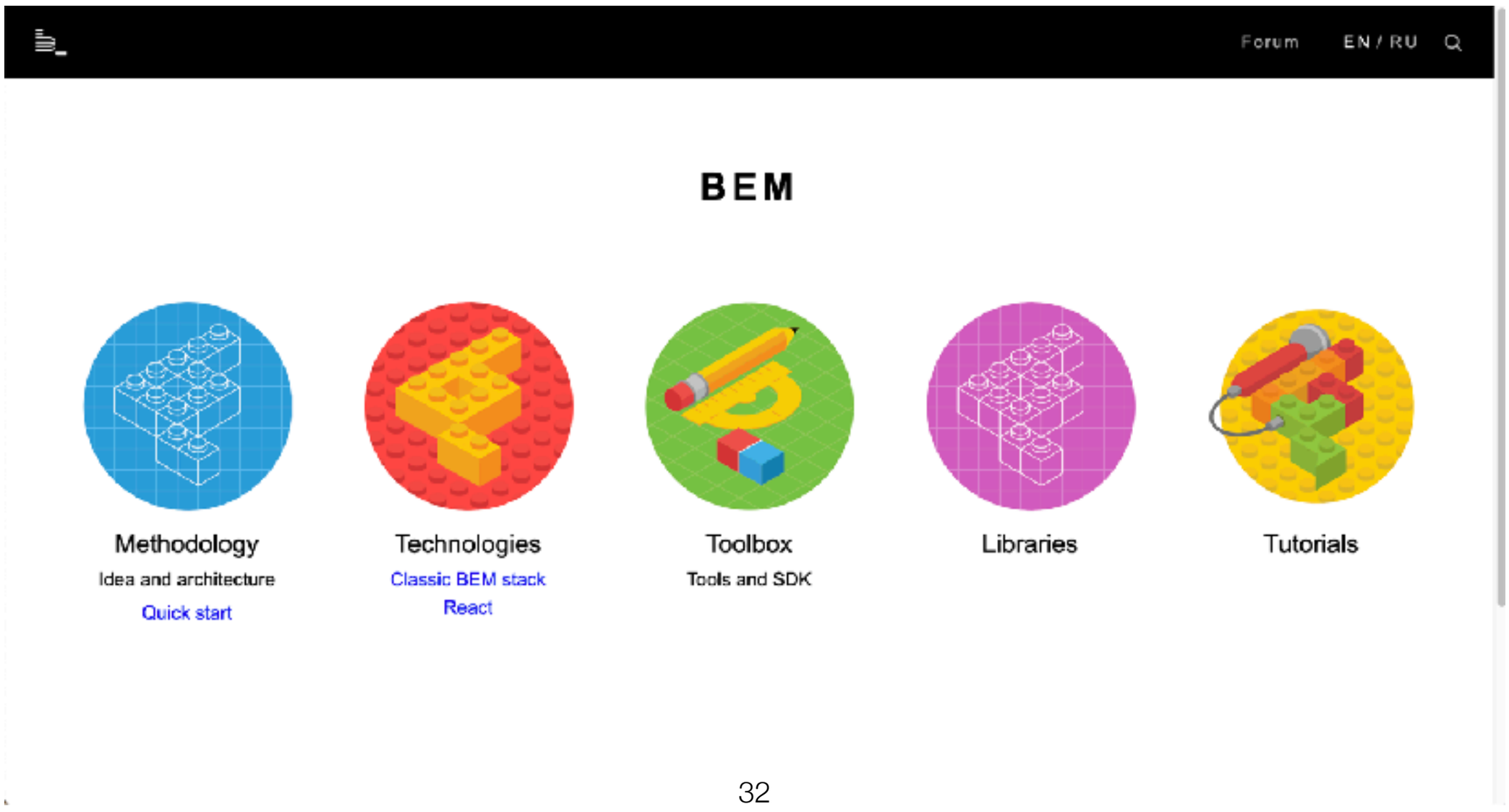
その中に必要なCSSファイルを保存していくような形を取ります

2. SMACSS

```
├── style.scss
│
├── base //ベースに関すること
│   ├── _base.scss
│   └── _mixins.scss
│
├── layout //レイアウトに関すること
│   ├── _top.scss
│   ├── _header.scss
│   └── _footer.scss
│
└── module //モジュールに関すること
    ├── _btn.scss
    ├── _logo.scss
    └── _nav.scsss
```

3. BEM

- ”Block”-"Element"-"Modifier"
- BEM = 命名規則



3. BEM

特徴

- **Block（塊）、Element (要素)、Modifier (修飾) の３つに分ける**
- **命名規則は.block__element--modifier**
- **付与されるクラスは一つ（シングルクラス）**

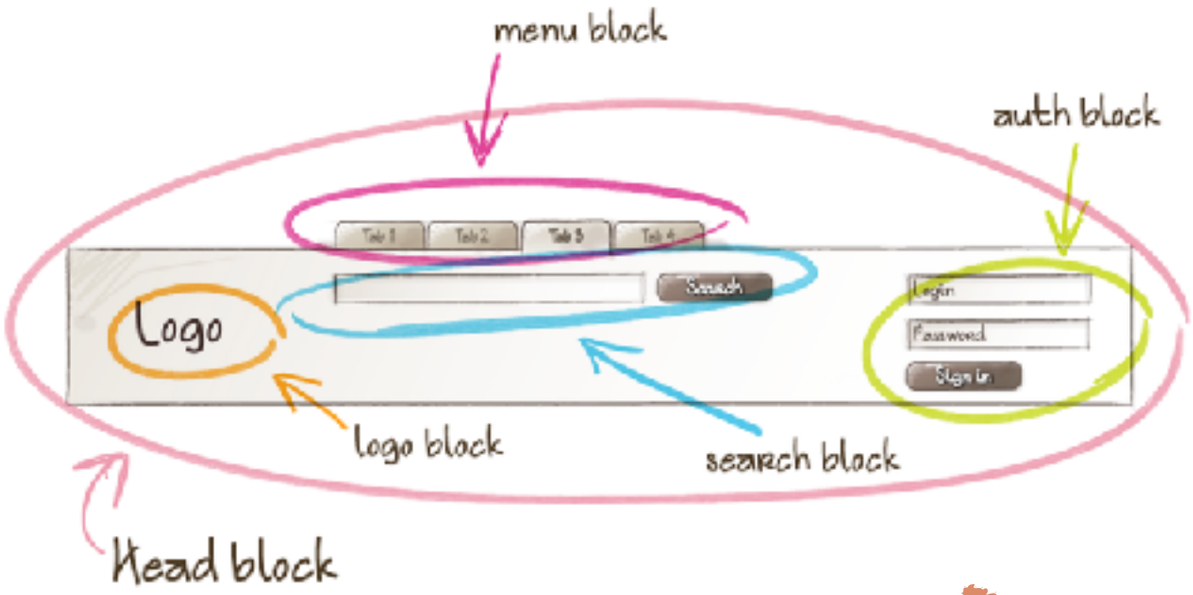
メリット

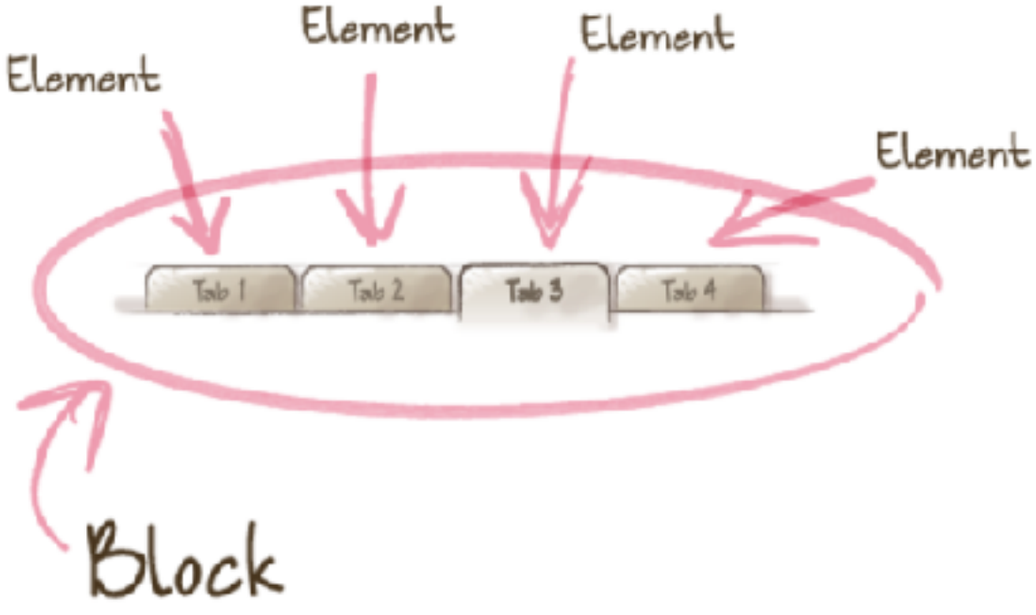
- **CSSを見ただけでどこで使われるものなのか分かる**
- **入れ子にならないので優先度で悩まない**
- **採用してるところが多く共有しやすい**

デメリット

- **class名が長くなってしまう**
- **名前をつけるときにBlockとElement迷う**
- **HTML側が見づらい（css側はsassで書けば大丈夫）**
- **Bootstrapと相性悪い**

3. BEM





Block ： .menu__tab

Element ： .menu_tab__item

Modifier ： .menu__tab__item—active

https://morizyun.github.io/web/css-bem-basic-rule.html

HTML

```html
<div class="menu">
  <ul class="menu__tab">
    <li class="menu__tab__item"><a class="menu__tab__link" href="#">タブ1</a></li>
    <li class="menu__tab__item"><a class="menu__tab__link" href="#">タブ2</a></li>
    <li class="menu__tab__item--active"><a class="menu__tab__link" href="#">タブ3</a></li>
    <li class="menu__tab__item"><a class="menu__tab__link" href="#">タブ4</a></li>
    <li class="menu__tab__item"><a class="menu__tab__link" href="#">タブ5</a></li>
  </ul>
</div>
```

CSS

```css
.menu__tab{
  list-style: none;
  padding:0;
  text-align: center;
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
}
.menu__tab__item{
  flex:1;
  flex-basis: 120px;
  background-color:#65c6ba;
  border: 1px solid white;
}
.menu__tab__item--active{
  flex:1;
  flex-basis: 120px;
  background-color:#b1e2dc;
  border: 1px solid white;
}
.menu__tab__link{
  display: block;
  text-decoration: none;
  color: #fff;
}
```

タブ1 | タブ2 | タブ3 | タブ4 | タブ5

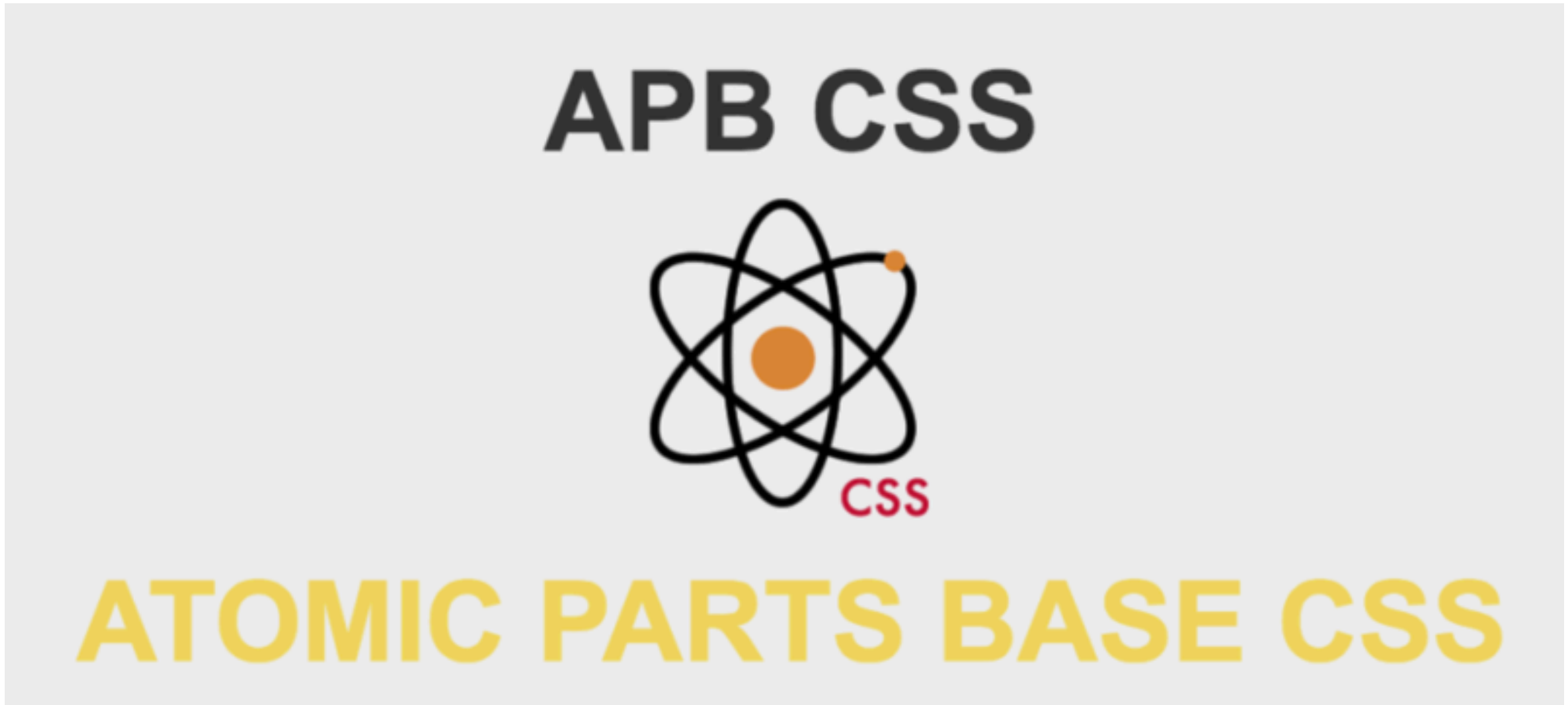
Block ： .menu__tab

Element ： .menu_tab__item

Modifier ： .menu__tab__item—active

4. APBCSS

- Atomic Parts Base CSS
- AtomicDesignの考え方を元にしたCSS



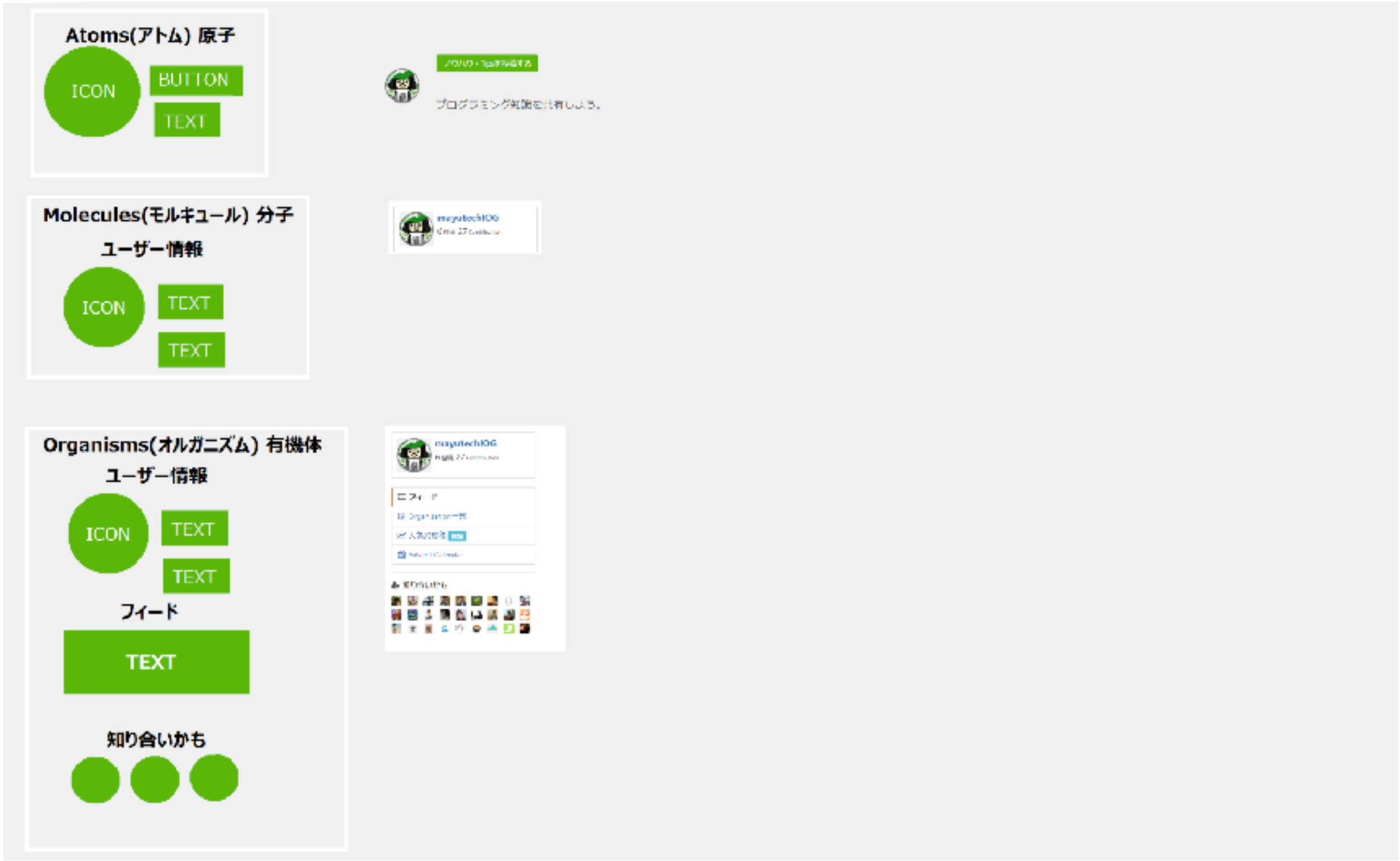
AtomicDesignとは

- AtomicDesignはデザイン設計（CSS設計ではない）
- AtomicDesign(原子のデザイン）という名前のとおり、ボタンやロゴやアイコンなどそれ以上分解できない要素を原子として考え、そこから色々な要素を組み合わせてページを作っていく考え方
- Webページを構成する要素を５つのステージで構成・管理

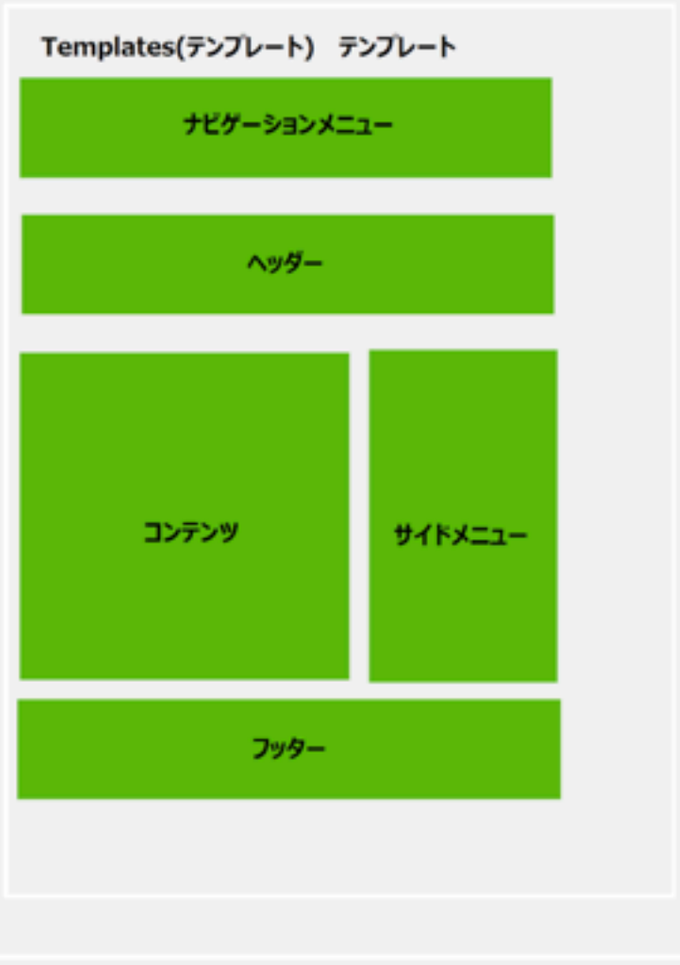
AtomicDesignとは

Atoms(アトム) 原子	それ以上分解できない要素	ボタン
Molecules(モルキュール) 分子	原子同士を組み合わせた要素	フォーム+ボタン
Organisms(オルガニズム) 有機体	分子同士を組み合わせた要素	ナビゲーションメニューやヘッダー
Templates(テンプレート) テンプレート	有機体を組み合わせた要素	ナビゲーション、ヘッダー、コンテンツ、サイドメニュー、フッター
Pages(ページ) ページ	テンプレートに具体的な画像、文字を入れたもの	

AtomicDesignとは



https://qiita.com/yuxio/items/4db19395835f84d3ef15



https://qiita.com/yuxio/items/4db19395835f84d3ef15

4. APBCSS

特徴

- AtomicDesignが採用しているそれ以上分解できない構成要素をAtomic Partsとして定義していく
- 多くのCSSアーキテクチャでも用いられている、Layoutといった概念はなく、 構成の考え方もその逆で、細部化出来ないUIパーツから、定義していく
- OOCSS + SMACSS を取り入れた、マルチクラスを採用
- ６つのクラスタイプで構成・管理

メリット

- メンテナンスしやすい
- デザイン設計と紐付いた考え方なのでデザイナーと共有できる

デメリット

- 説明コストがかかる
- 細かく命名ルールを決めて共有しておかないと破綻する可能性が高い

APBCSS ルール

APBCSS ルール

Atomic	原子パーツ名となるクラス名	text / icon / btn / thumbnail / label / input
Module	UI を包括するモジュール名となるクラス名	header / footer / contents mainLogo / title /column thumbnailList / textList
Skin	装飾などのクラス名	red / blue / wide high / low /stripe
Number	ナンバリング用のクラス名	one / two / no[nth] first / error / checked
State	状態を表すクラス名	one / two / no[nth] first / error / checked
Other	その他のクラス名。パーツのセマンティックネームもここに含む。	wrap / Service name Page name / Namespace

APBCSS

分子


```
<p class="btn primary”>
<a href="#"><span class="icon social github"></span>Button</a></p>
```

```
.btn {
    .icon {
        &.social {
            margin: 0px 6px -11px -34px;
        }
    }
}
```

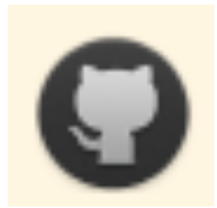
http://apbcss.com/

APBCSS

原子


```
<p class="btn primary"><a href="#">Button</a></p>
```

```
.btn {
  position: relative;
  width: 200px;
  height: 40px;
  border-radius: 6px;
  a {
    display: block;
    line-height: 40px;
    text-align: center;
    text-decoration: none;
    color: #fff;
    font-size: 18px;
  }
  &.primary {
    background: #404040;
    @include background(linear-gradient(top, #404040, #282828));
    &:hover { background: #303030; }
  }
}
```



```
<p class="icon social github"></p>
```

```
.icon {
  display: inline-block;
  &.social {
    width: 34px;
    height: 34px;
    background: url(SpriteImagePath) no-repeat;
    @include background-size(205px auto);
    &.github { background-position: -171px top; }
  }
}
```

APBCSS

分子


```
<p class="btn primary”>
<a href="#"><span class="icon social github"></span>Button</a></p>
```

```
.btn {
    .icon {
        &.social {
            margin: 0px 6px -11px -34px;
        }
    }
}
```

http://apbcss.com/

代表的な4つのCSS設計アイデア まとめ

1. **OOCSS**
   → シンプルで取り入れやすい
2. **SMACSS**
   → カテゴリ分けすることで、パターン抽出でき無駄なコードが生まれない
3. **BEM**
   → 命名規則がわかりやすく、共有しやすい
4. **APBCSS**
   → デザインモックを作る段階から同じ方向の設計ができるためCSSコードの破綻しにくくなる

参考資料

1. Web制作者のためのCSS設計の教科書 (谷 拓樹 著)
2. はじめてのCSS設計 (田辺 丈士 著)
3. 現場のプロが本気で教える HTML/CSSデザイン講義 (森本 恭平 著)
4. OOCSS　http://oocss.org/
5. SMACSS　http://smacss.com/
6. BEM　https://en.bem.info/
7. APBCSS　http://apbcss.com/