

# 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によって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、自由に書いてしまう
- 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

## セレクトに詳細度がある問題

---

## # HTML

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

## # CSS

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

この文字は何色でしょう？

- Idとclassが併用されると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

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

## CSS

```
1 body{ font-size: 16px; }
2 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

```
1 <p class="text">何色かな？</p>
```

CSS

```
1 .text{ color: black; }  
2 .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

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

CSS

```
1 .cls1{ font-size: 16px; }  
2 .cls2{ color: red; }  
3 .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

## Object-Oriented CSS

### Hosted

Play with these in Firebug to learn the basics.

Template  
Grids  
Module  
Content (very alpha)

### Welcome, Velocity Conference participants!

All the resources you need to get started are linked from the left navigation. Start by downloading the base files. Exercises one and two can be completed in Firebug if you are comfortable with it. Then you can download the finished file at the beginning of Exercise 3.

### Downloads on Github

Velocity download  
Alternate download

### Stuck?

If you don't keep up with any of the exercises you can view (and download) the finished examples here.

Starting Template  
Exercise 1: Template  
Exercise 2: Grids  
Exercise 3: Module Manipulation  
Exercise 4: Module Creation

Stubbornella, Github, FAQ



# 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

```
1 <div>
2   <a href="#" class="button">SUBMIT</a>
3 </div>
```

## CSS

```
1 .button {
2   display: inline-block;
3   padding: 0.5rem 1rem;
4   text-decoration: none;
5   background: #668ad8;
6   color: #FFF;
7   border-bottom: solid 4px #627295;
8   border-radius: 3px;
9   font-size: 1rem;
10 }
```

SUBMIT

- 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

```
1 <div>
2   <a href="#" class="button button-submit">SUBMIT</a>
3 </div>
4 <br>
5 <div>
6   <a href="#" class="button button-cancel">CANCEL</a>
7 </div>
```

## CSS

```
1 .button {
2   display: inline-block;
3   padding: 0.5rem 1rem;
4   text-decoration: none;
5   border-bottom: solid 4px #627295;
6   border-radius: 3px;
7 }
8 .button-submit {
9   background: #668ad8;
10  color: #FFF;
11  font-size: 1rem;
12 }
13 .button-cancel {
14  background: #ff6666;
15  color: #FFF;
16  font-size: 1rem;
17 }
```

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

```
1 <header class="header">
2   <h1 class="logo">logo</h1>
3 </header>
```

## CSS

```
1 .header .logo {
2   background-color: pink;
3   color: white;
4   padding: 1rem;
5   width: 10rem;
6   text-align: center;
7 }
```

logo

- 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

```
1 <header class="header">
2   <h1 class="logo">logo</h1>
3 </header>
4 <br>
5 <footer class="footer">
6   <h1 class="logo logo-small">logo</h1>
7 </footer>
```

## CSS

```
1 .logo {
2   background-color: pink;
3   color: white;
4   padding: 1rem;
5   width: 10rem;
6   text-align: center;
7 }
8
9 .logo-small {
10  width: 5rem;
11 }
```

logo

- 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

logo

## 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/>



Workshops Sign in

# Scalable and Modular Architecture for CSS

A flexible guide to developing sites small and large.



**"SMACSS is becoming one of the most useful contributions to front-end discussions in years"** ☺

I've been analyzing my process (and the process of those around me) and figuring out how best to structure code for projects on a larger scale. What I've found is a process that works equally well for sites small and large.

Learn how to structure your CSS to allow for flexibility and maintainability as your project and your team grow.

### What is it?

SMACSS (pronounced "swacks") is more style guide than rigid framework. There is no library within here for you to download or install. There is no git repository for you to clone. SMACSS is a way to examine your design process and as a way to fit these rigid frameworks into a flexible thought process. It is an attempt to document a consistent approach to site development when using CSS. And really, who isn't building a site with CSS these days?!

Get to know Scalable and Modular Architecture for CSS:

#### Read it Online

SMACSS started out as a free online book and that continues to be true. You can always [read the book online](#).

#### Download the Book

The e-book comes in PDF, ePub, and mobi formats for easy installation on almost any e-reader. [Download the zip!](#)

[Download the book!](#)

### What's in SMACSS?

#### Preface

1. About the Author
2. Introduction

#### Core

3. Categorizing CSS Rules
4. Base Rules
5. Layout Rules
6. Module Rules
7. State Rules
8. Theme Rules
9. Changing State

#### Aspects of SMACSS

10. Depth of Applicability
11. Selector Performance
12. HTML5 and SMACSS
13. Prototyping
14. Preprocessors
15. Deep the Base
16. The Icon Module

## 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

<b>base</b>	デフォルトのスタイル	body, html
<b>layout</b>	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
<b>module</b>	再利用可能なパーツの指定	*-box, nav, logo
<b>state</b>	moduleやlayoutの状態を指定	is-active, is-hidden
<b>theme</b>	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

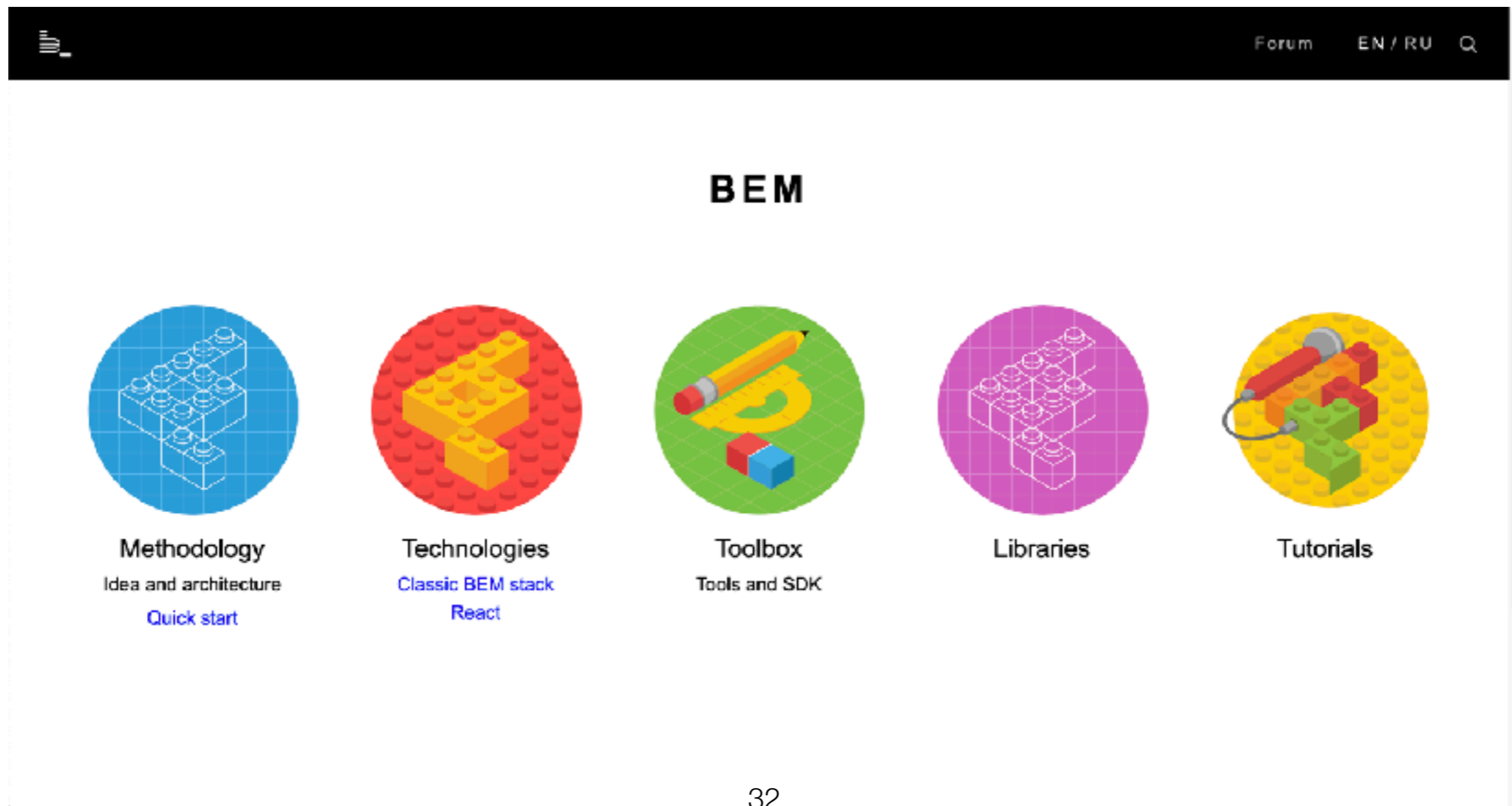
---

```
|— 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.scss
```

## 3. BEM

---

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



The image shows a screenshot of the BEM website's navigation menu. At the top right, there are links for "Forum", "EN / RU", and a search icon. The main content area is titled "BEM" and features five circular icons representing different sections:

- Methodology**: Idea and architecture, Quick start
- Technologies**: Classic BEM stack, React
- Toolbox**: Tools and SDK
- Libraries**
- Tutorials**



## 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

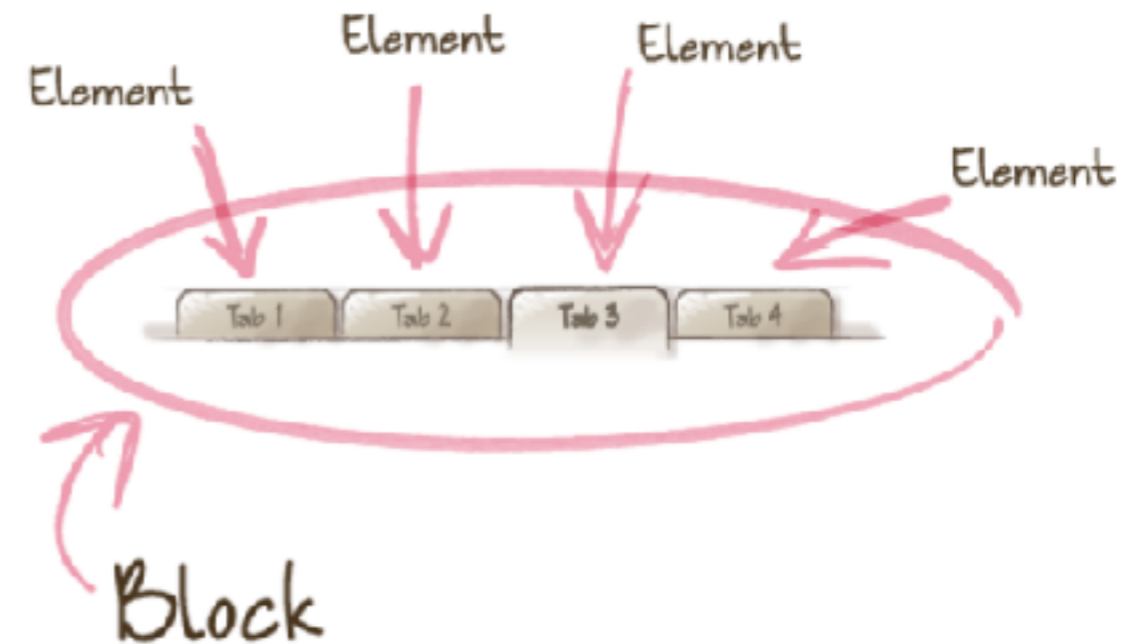
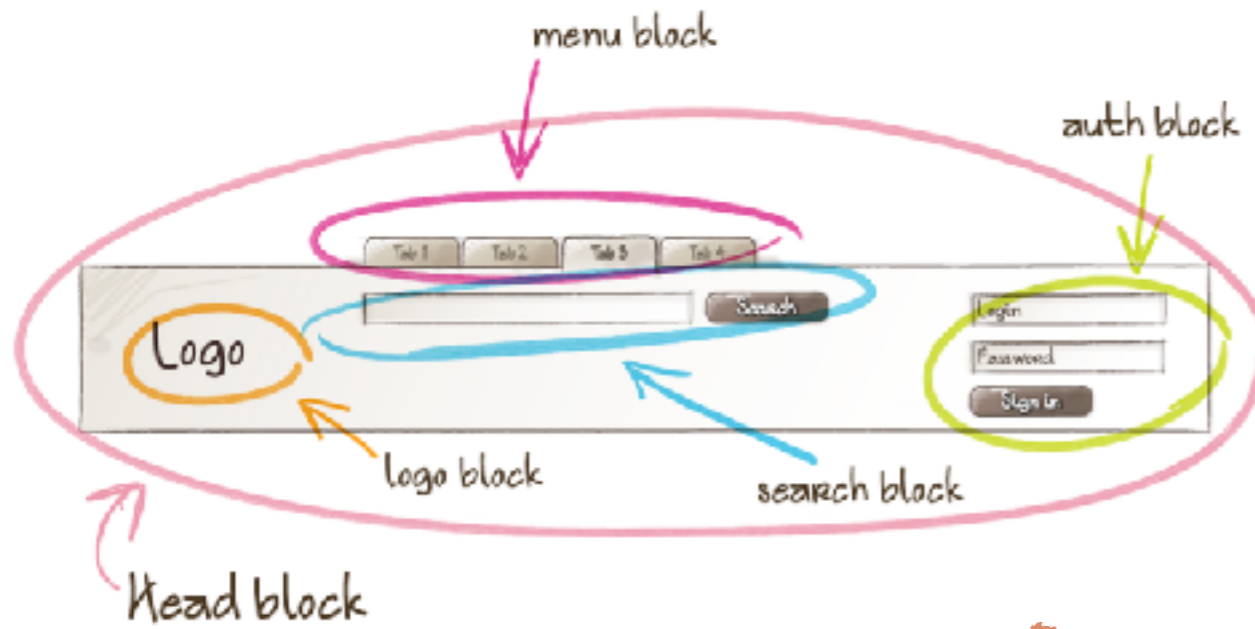
- CSSを見ただけでどこで使われるものなのか分かる
- 入れ子にならないので優先度で悩まない
- 採用してるところが多く共有しやすい

### デメリット

- class名が長くなってしまふ
- 名前をつけるときにBlockとElement迷ふ
- 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`

HTML

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

CSS

```
1- .menu__tab{
2-   list-style: none;
3-   padding:0;
4-   text-align: center;
5-   display: flex;
6-   flex-direction: row;
7-   flex-wrap: wrap;
8- }
9- .menu__tab__item{
10-  flex:1;
11-  flex-basis: 120px;
12-  background-color:#65c6ba;
13-  border: 1px solid white;
14- }
15- .menu__tab__item--active{
16-  flex:1;
17-  flex-basis: 120px;
18-  background-color:#b1e2dc;
19-  border: 1px solid white;
20- }
21- .menu__tab__link{
22-  display: block;
23-  text-decoration: none;
24-  color: #fff;
25- }
```

タブ1

タブ2

タブ3

タブ4

タブ5

**Block** : .menu\_\_tab

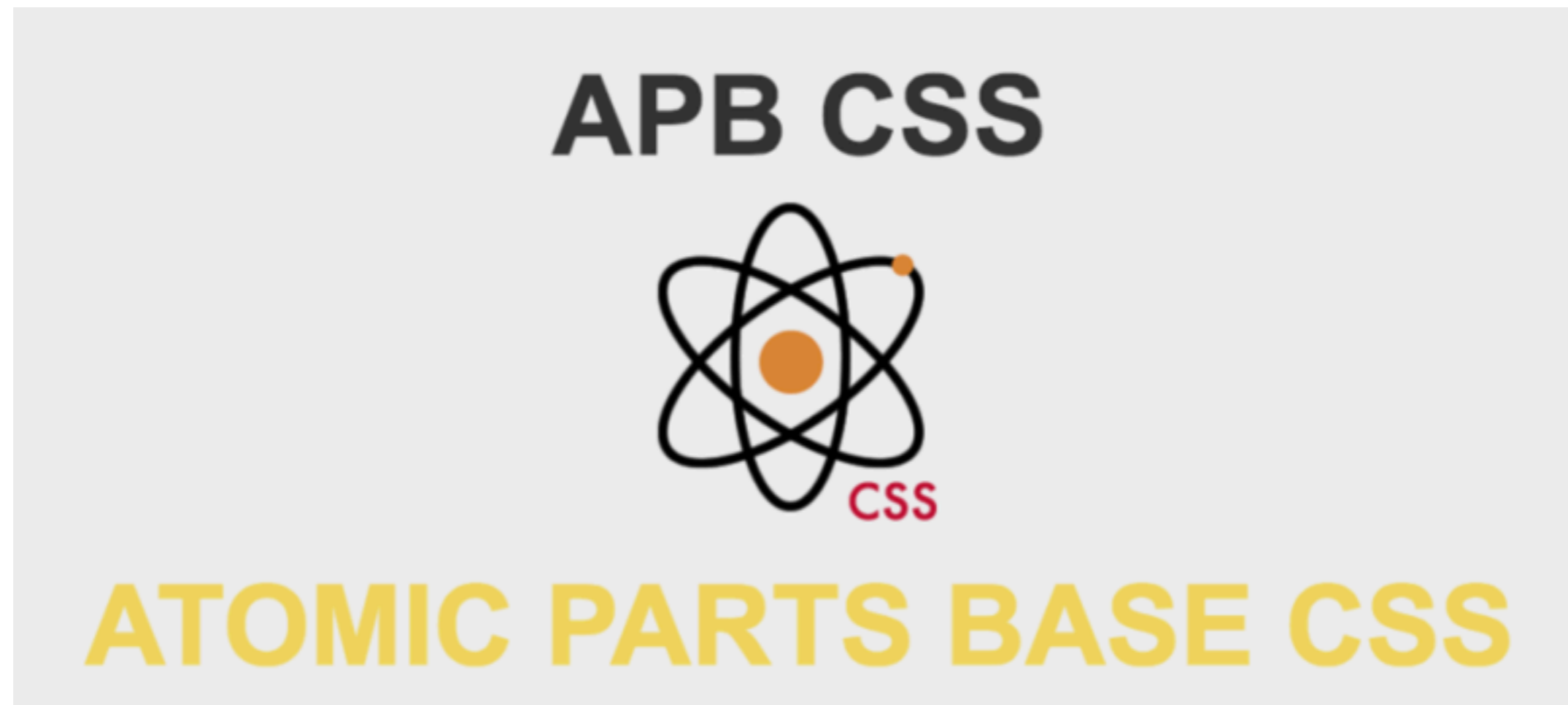
**Element** : .menu\_\_tab\_\_item

**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

---

<b>Atoms(アトム)</b> 原子	それ以上分解できない要素	ボタン
<b>Molecules(モルキュール)</b> 分子	原子同士を組み合わせた要素	フォーム+ボタン
<b>Organisms(オルガニズム)</b> 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
<b>Templates(テンプレート)</b> 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
<b>Pages(ページ)</b> 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

The diagram illustrates Atomic Design through three levels of abstraction, each shown in a white box on the left and a corresponding UI screenshot on the right.

- Atoms (原子):** Labeled "Atoms(アトム) 原子", this level consists of basic UI elements: a green circle labeled "ICON", a green rounded rectangle labeled "BUTTON", and a green rectangle labeled "TEXT". The corresponding screenshot shows a small profile header with a green button labeled "フォロー + 通知を解除する" and a text element "プログラミング知識を共有しよう。".
- Molecules (分子):** Labeled "Molecules(モルキュール) 分子", this level combines atoms into a "ユーザー情報" (User Information) block. It includes an "ICON" circle, two "TEXT" rectangles, and another "TEXT" rectangle. The screenshot shows a profile card for "megatech106" with "177 followers".
- Organisms (有機体):** Labeled "Organisms(オルガニズム) 有機体", this level represents a more complex component like a "ユーザー情報" (User Information) section. It features an "ICON" circle, two "TEXT" rectangles, a "フィード" (Feed) section with a large "TEXT" rectangle, and a "知り合いかも" (Maybe you know) section with three "ICON" circles. The screenshot shows a full profile page for "megatech106" with a bio, a list of organizations, and a grid of avatars.





## 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

---

CSS Class Type



**Atomic**



**Module**



**Skin**



**Number**



**State**



**Other**

# APBCSS ルール

---

<b>Atomic</b>	原子パーツ名となるクラス名	text / icon / btn / thumbnail / label / input
<b>Module</b>	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
<b>Skin</b>	装飾などのクラス名	red / blue / wide high / low /stripe
<b>Number</b>	ナンバリング用のクラス名	one / two / no[nth] first / error / checked
<b>State</b>	状態を表すクラス名	one / two / no[nth] first / error / checked
<b>Other</b>	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;  
    }  
  }  
}
```

# APBCSS

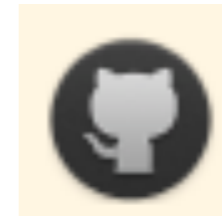
---

## 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;  
    }  
  }  
}
```

# 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/>