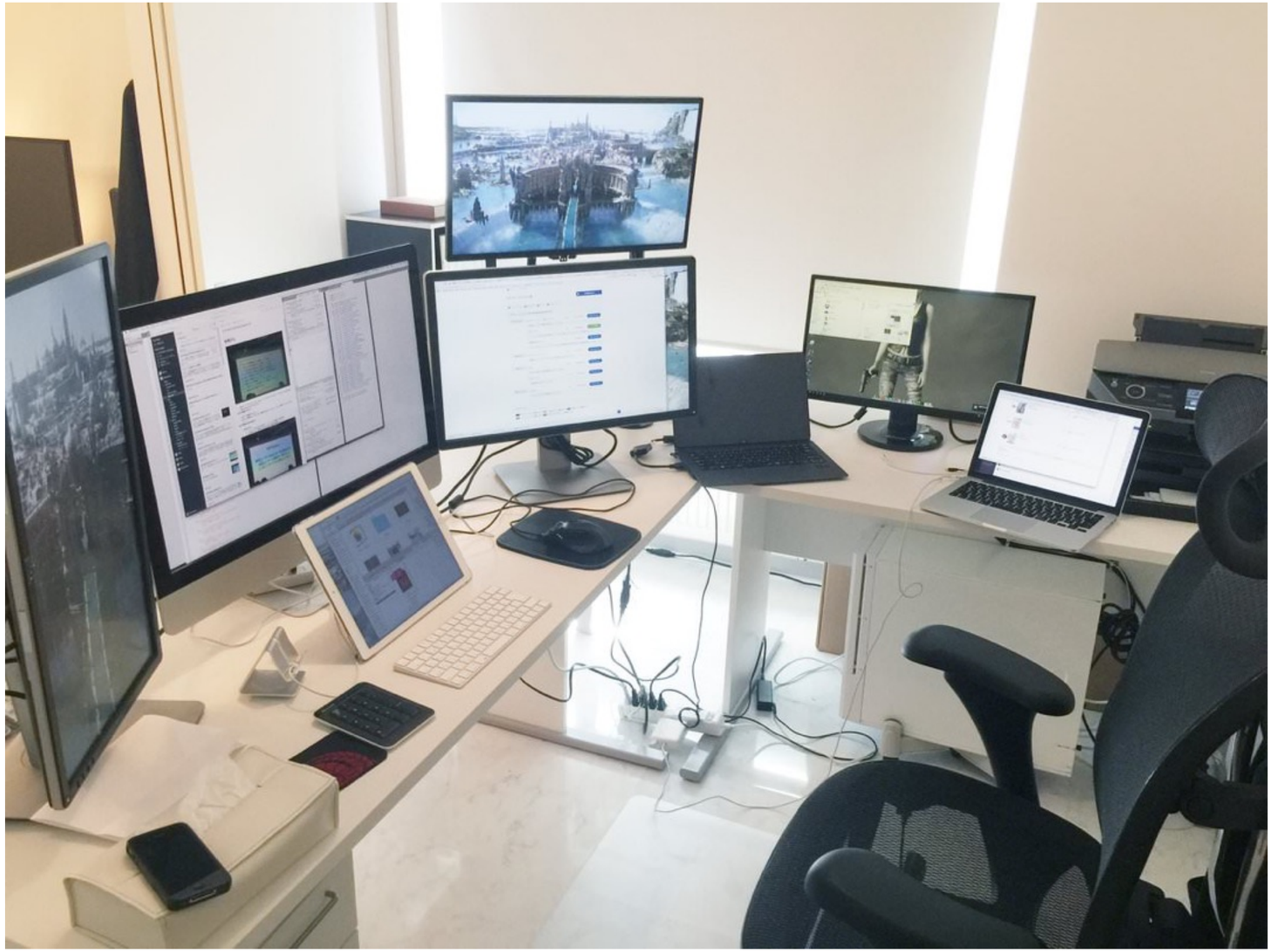
3ヶ月後の工数を劇的に減らす、
将来のための効率的なマークアップ設計

2017/7/1 SaCSS Special12
SaCSS SUMMER FESTIVAL 第1弾 HTML＆CSS

半田 惇志

- 会社：24-7/パンセ 出向中٩('ω')و
  - テクニカルディレクター/シニアエンジニア
  - 主に数百P案件担当(なぜか昔から)
- ブログ：Thinking Salad
- ちょうしょ：せがたかい
- たんしょ：おかねの計算ができない
- 借金：150万٩('ω')و

引っ越しまして、快適です。



Bracketsが好き過ぎて本を出しました。

Kindleストアで絶賛発売中！

税込780円！安い！

お得なキャンペーン実施中！

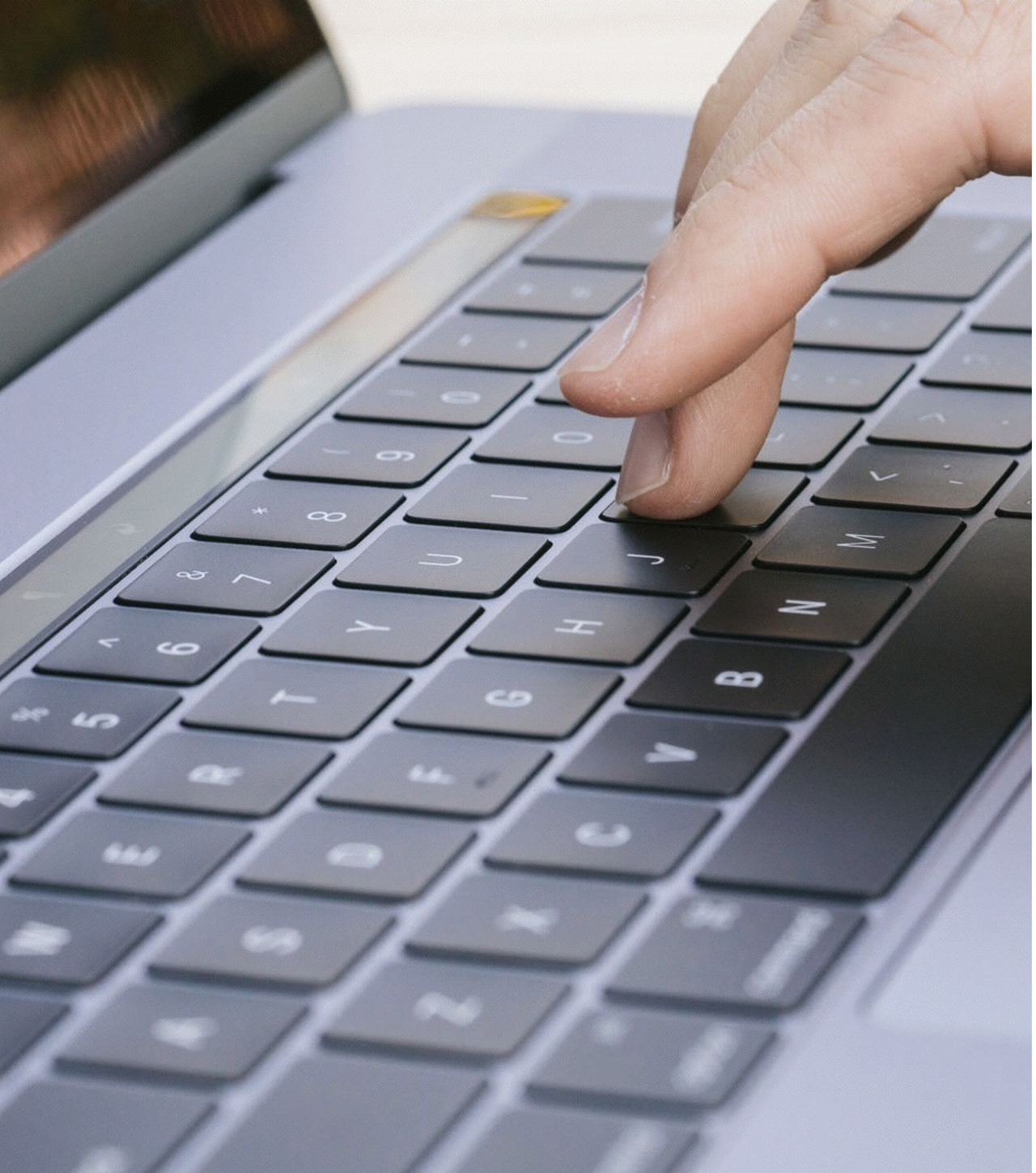
PRECSSも作りました。

OOCSS、SMACSS、BEMをベースにした
中規模以上向けCSSアーキテクチャ

http://precss.io/ja/

INDEX

マルチクラス設計



マルチクラス設計とは

- OOCSSにより普及した概念
- 目的とするスタイルを実現・拡張するにあたり、HTMLに複数のクラスを付与する

シングルクラスの例

Login

Submit

```
.btn-orange{
  display: inline-block;
  border-radius: 3px;
  background-color: orange;
}
.btn-blue{
  display: inline-block;
  border-radius: 3px;
  background-color: blue;
}
```

シングルクラスの例

Login

Submit

```
.btn-orange{
    display: inline-block;
    border-radius: 3px;
    background-color: orange;
}
.btn-blue{
    display: inline-block;
    border-radius: 3px;
    background-color: blue;
}
```

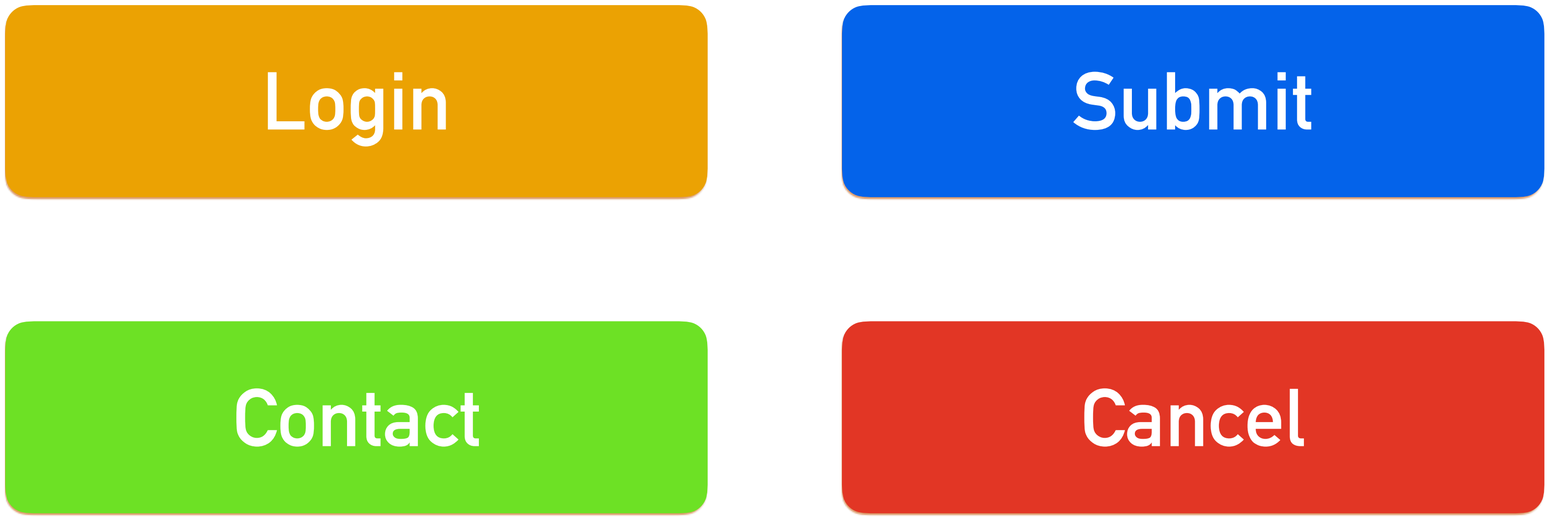
シングルクラスの例

Login

Submit

```
.btn-orange{
    display: inline-block;
    border-radius: 3px
    background-color
}
.btn-blue{
    display: inline-block;
    border-radius: 3px;
    background-color: blue;
}
```

同じコード

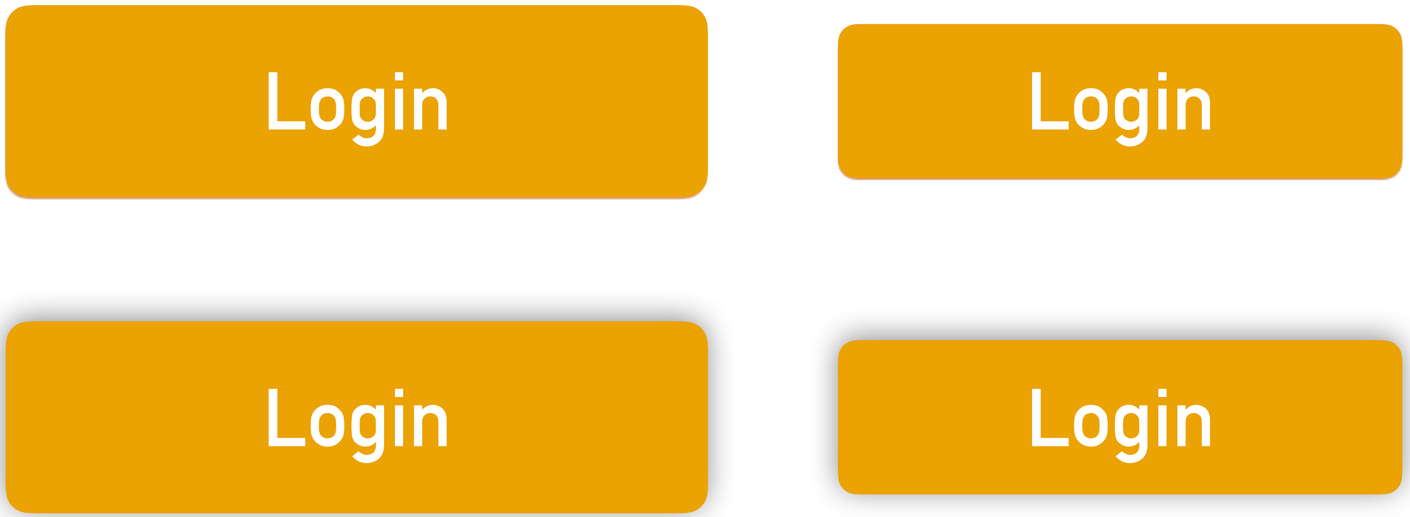

Login
Submit
Contact
Cancel

```css
.btn-orange{
  display: inline-block;
  border-radius: 3px;
  background-color: orange;
}
.btn-green{
  display: inline-block;
  border-radius: 3px;
  background-color: green;
}
```

```css
.btn-blue{
  display: inline-block;
  border-radius: 3px;
  background-color: blue;
}
.btn-red{
  display: inline-block;
  border-radius: 3px;
  background-color: red;
}
```

```css
.btn-orange{
    display: inline-block;
    border-radius: 3px;
    background-color: orange;
}
.btn-green{
    display: inline-block;
    border-radius: 3px;
    background-color: green;
}
```

```css
.btn-blue{
    display: inline-block;
    border-radius: 3px;
    background-color: blue;
}
.btn-red{
    display: inline-block;
    border-radius: 3px;
    background-color: red;
}
```


Login
Login
Login
Login

```css
.btn-orange{
  display: inline-block;
  border-radius: 3px;
  background-color: orange;
}

.btn-orange-shadow{
  display: inline-block;
  border-radius: 3px;
  background-color: orange;
  box-shadow: 0 0 3px #000;
}
```

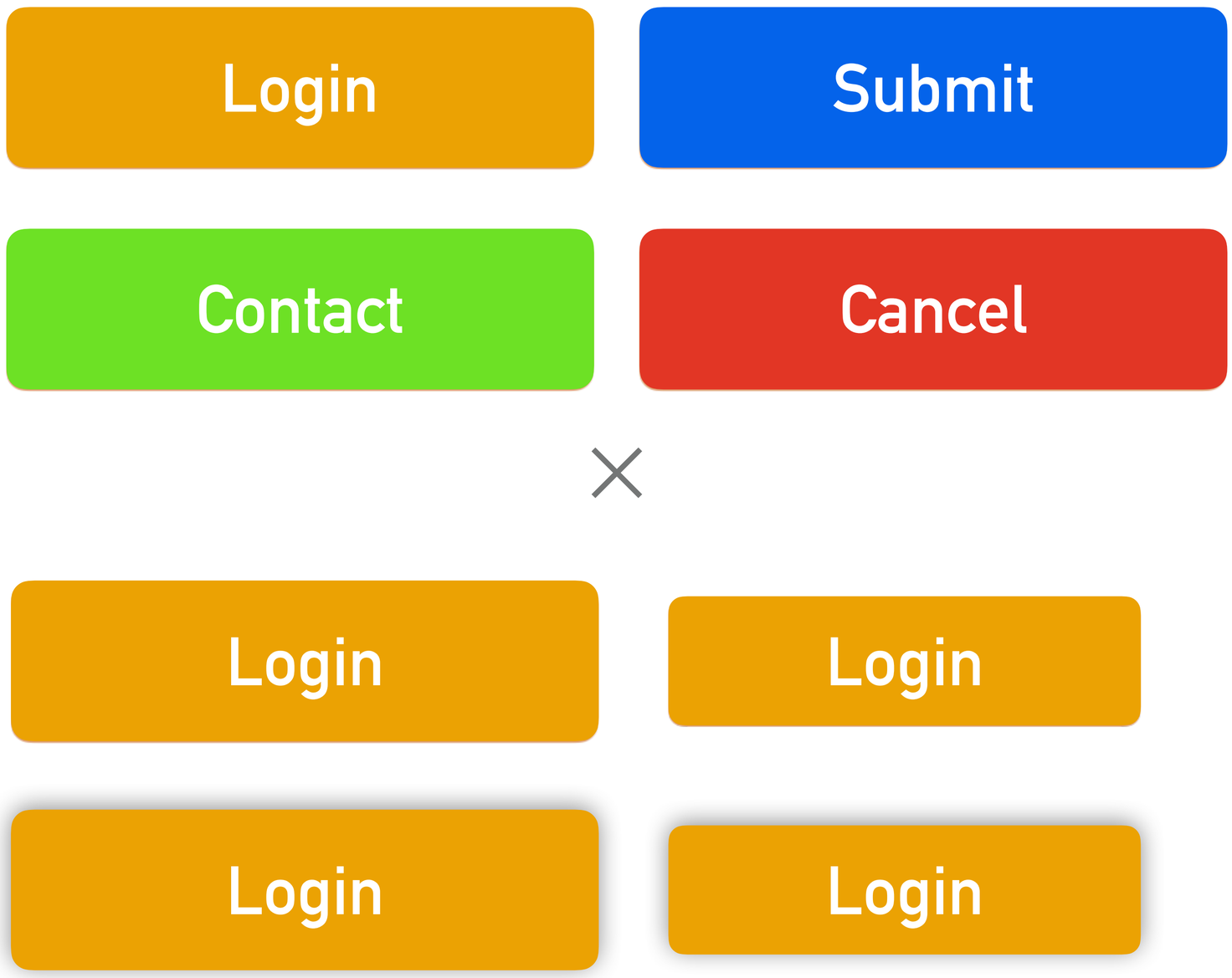
```css
.btn-orange-small{
  display: inline-block;
  border-radius: 3px;
  background-color: orange;
  width: 50%;
}
.btn-orange-small-shadow{
  display: inline-block;
  border-radius: 3px;
  background-color: orange;
  width: 50%;
  box-shadow: 0 0 3px #000;
}
```

```css
.btn-orange{
  display: inline-block;
  border-radius: 3px;
  background-color: orange;
}

.btn-orange-shadow{
  display: inline-block;
  border-radius: 3px;
  background-color: orange;
  box-shadow: 0 0 3px #000;
}
```

```css
.btn-orange-small{
  display: inline-block;
  border-radius: 3px;
  background-color: orange;
  width: 50%;
}

.btn-orange-small-shadow{
  display: inline-block;
  border-radius: 3px;
  background-color: orange;
  width: 50%;
  box-shadow: 0 0 3px #000;
}
```


Login
Submit
Contact
Cancel
Login
Login
Login
Login

つらい。

バリエーションが増えれば増えるほどつらい。

マルチクラスの例

Login

Submit

```
.btn{
    display: inline-block;
    border-radius: 3px;
}
.orange{
    background-color: orange;
}
.blue{
    background-color: blue;
}
```

マルチクラスの例

Login

Submit

```
.btn{
    display: inline-block;
    border-radius: 3px;
}
.orange{
    background-col
}
.blue{
    background-color: blue;
}
```

必要最低限

ただし、この書き方には難点が。

```
<button class="btn orange">Login</button>
```

アイコンを付与するクラス

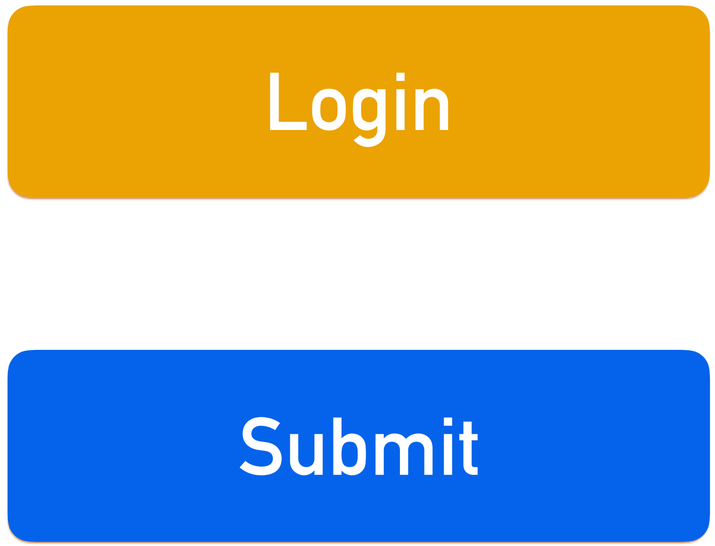
```
<a class="icon">hogehoge</a>
```

こんなときどうする？

赤になるのはどっち？オレンジになるのはどっち？

```
<button class="btn icon orange red">Login</button>
```

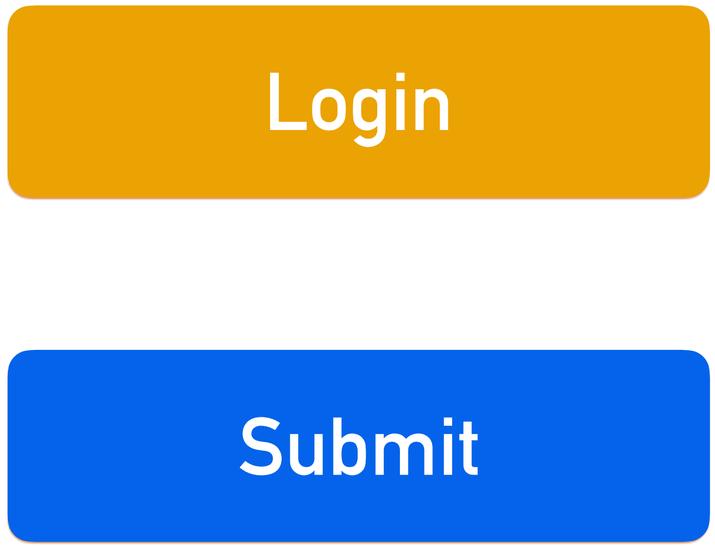
マルチクラスの例




```
<button class="btn btn-orange">Login</button>
```

```
.btn{
  display: inline-block;
  border-radius: 3px;
}
.btn-orange{
  background-color: orange;
}
.btn-blue{
  background-color: blue;
}
```

マルチクラスの例

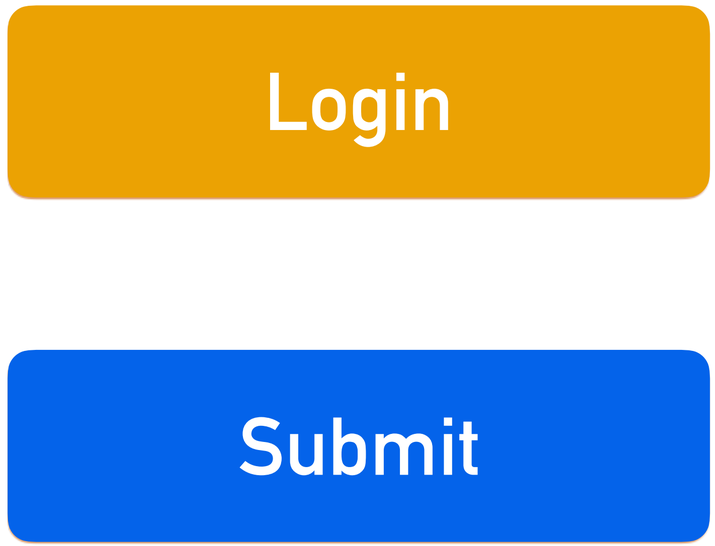



```
<button class="btn btn-orange">Login</button>
```

```
.btn{
  display: inline-block;
  border-radius: 3px;
}
.btn-orange{
  background-color: orange;
}
.btn-blue{
  background-color: blue;
}
```

もう一歩。

マルチクラスの例




```scss
.btn{
  display: inline-block;
  border-radius: 3px;
  background-color: $baseColor;
}
.btn.btn-orange{
  background-color: orange;
}
.btn.btn-blue{
  background-color: blue;
}
```

マルチクラスの例

Login

Submit

```
.btn{
  display: inline-block;
  border-radius: 3px;
  background-color: $baseColor;
}
.btn.btn-orang
  background-c          ge;
}
.btn.btn-blue{
  background-color: blue;
}
```

有事のための
デフォルト定義

詳細度を
高める

バリエーションに対応する。

Login

Submit

```
<button class="btn btn-boxShadow btn-orange">Login</button>

<button class="btn btn-boxShadow btn-blue btn-small">Submit</button>
```

シングルクラス設計と比較したメリット

- **後からバリエーションを増やす際、記述量が減る**
  - 3ヶ月後の予想外の追加要件に対応しやすいのはこっち
  - ただしバランス良くベースクラスを設計するには慣れとセンスが必要
- **モディファイア毎に機能を分離することができる**
  - 拡張性がかなり高く、後から追加仕様が来ても割と対応できる
- **JSやCMS等でクラスを制御する場合、処理が少し減る**

  (既存クラスの削除の必要がない)

シングルクラス設計と比較したデメリット

- **HTMLのクラスが冗長になりがち**
- **あまりに機能を分離し過ぎると、後からプロジェクトに入った人には分かりづらい**
  - モディファイアが増えたり複雑になった場合は、なんらかのドキュメントを
    - Sass内コメント
    - スタイルガイド

クラス命名

大きく分けて2つある

- セマンティックな名前
- （その中間な名前）
- 汎用的な名前


Title Here
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui
More

Login

命名サンプル - セマンティック

- news-block・product-block
  - ニュースや商品にしか使えない
- company-block
  - カンパニーのコンテキストでしか使えない

- セマンティックにしてわざと分離する、スコープを区切るのもあり
- その際はmixin等を上手く用いて、内部的になるべく共通化をしておく

命名サンプル - 中間

- post-block
  - 「ポスト」つまりブログから出力され一覧されるようなものなら何でもOK

命名サンプル - 汎用

- **block/item**
  - シンプル過ぎて分かりづらい。コンフリクトも起きやすい
- **item-block**
  - どんなレイアウトのものでも「item-block」に成り得るので、コンフリクトが起きやすい
- **postit-block・pareto-block**
  - どこでも汎用的に使える
    - ただし大きくなり過ぎるリスクもある
  - ちなみに命名の由来は付箋・パレートの法則(2:8)から

命名サンプル - セマンティック

- company-btn
  - 先ほどと同じく、カンパニーのコンテキストでしか使えない
- btn-login
  - テキストがLoginでないといけない
    - Loginじゃなく、でもオレンジにしたかったらどうするの？

命名サンプル - 中間

- btn-themeColor
  - セマンティックだが、取り回しがしやすい
    - 「オレンジであること」ではなく、

      「テーマカラーであること」が重要
    - テーマカラーがたまたまオレンジだっただけ
- btn-primary/btn-success
  - これらも同じく、「たまたまオレンジだった」だけ
  - ただし、「オレンジだから」という理由でprimaryやsuccessの意味を持たないボタンに使ってはいけない

命名サンプル - 汎用

- btn-orange
  - この名前に期待することは
    - 「ボタンであること」
    - 「オレンジであること」

  なので、中に何が入っても良い。すごく取り回しがしやすい

3つの◯◯！

こういうのは結構難しい

- 「特徴」が入ったりする → feature-block?
- 「オススメ」が入ったりする → recommend-block?
- 「ポイント」が入ったりする → point-block?

- ではimg-blockではどうか？
  - img-blockなんて他と幾らでもバッティングしそうやんけ……

半田がよくやる命名

3つの◯◯！

- card(-block)
  - 世界標準的
- focal-block
  - 目を引くから。また、「特徴・オススメ・ポイント」等「目を引きたい」コンテンツがよく入るから
- vertThumb-block
  - vertical（垂直）であり、かつサムネイルがあるから
- pile-block
  - 画像・タイトル・テキスト…と積み重ねていくから

命名まとめ

- **セマンティック名**
  - 限られたコンテキストでのみ使用され、名前自体に意味が必要な場合
    - 多くはIA・デザイン設計等、上流のレイヤーと絡むことが多い
- **汎用名**
  - 様々なコンテキストで使用される場合(他のページでも使う、他のブロック内でも使う)
  - 命名のポイントは、それ特有の見た目を表すこと。**名は体を表す**
  - **3**ヶ月後の追加要件に対応しやすいのはこっち
    - ただし、規模が大きければ大きいほど、きちんとしたドキュメントや

      スタイルガイドが無いとつらいです

何でもそうですが、結局は案件の特性や
サイトの規模、モジュールが使用される状況に応じて
都度セマンティック・汎用のバランスを
考える必要があります。

モジュール設計

先日、弊社でとある実験をしました。

Section Title1

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Section Title2

Youtube

- Coporate name 1
- Coporate name 2

Facebook

- Coporate name 1
- Coporate name 2

どれが「モジュール」でしょうか？

Aさん

Section Title1

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Section Title2

Youtube

- Coporate name 1
- Coporate name 2

Facebook

- Coporate name 1
- Coporate name 2

Aさん

Section Title1

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Section Title2

Youtube

- Coporate name 1
- Coporate name 2

Facebook

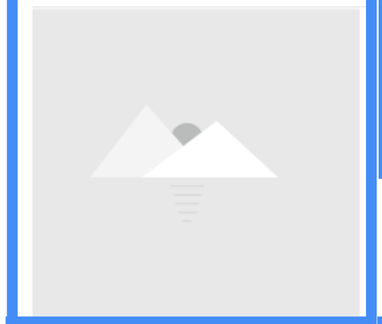
- Coporate name 1
- Coporate name 2

Bさん


Section Title1
Title Here
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui
More
Title Here
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui
More
Section Title2
Youtube
・Coporate name 1
・Coporate name 2
Facebook
・Coporate name 1
・Coporate name 2

Cさん

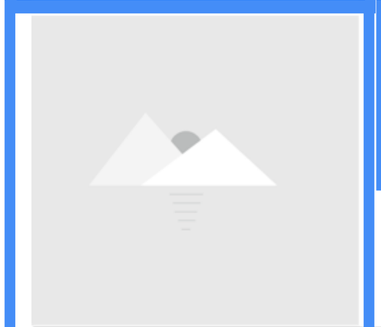
Section Title1



Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More



Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Section Title2

Youtube

- Coporate name 1
- Coporate name 2

Facebook

- Coporate name 1
- Coporate name 2

人によって「モジュール」の粒度・認識が異なります。

これに関しても命名同様、絶対的な正解はなく

案件ごとの情報設計やデザインに強く影響されます。

現状半田がベストプラクティスと考えている設計

- 見出し・ボタン・タグ・ラベル等は最小単位のモジュールとし、全体で使い回す
- 複数の要素からなるモジュールは、子要素に親のモジュール名を継承させる
- モジュール自体がカラムを形成することは避ける

現状半田がベストプラクティスと考えている設計

- 見出し・ボタン・タグ・ラベル等は最小単位のモジュールとし、全体で使い回す
- 複数の要素からなるモジュールは、子要素に親のモジュール名を継承させる
- モジュール自体がカラムを形成することは避ける

Section Title1

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Section Title2

Youtube

- Coporate name 1
- Coporate name 2

Facebook

- Coporate name 1
- Coporate name 2

Section Title1

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Section Title2

Youtube

- Coporate name 1
- Coporate name 2

Facebook

- Coporate name 1
- Coporate name 2

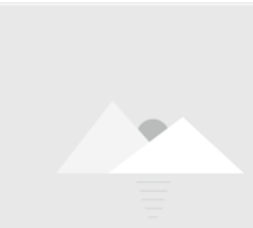
現状半田がベストプラクティスと考えている設計

- **見出し・ボタン・タグ・ラベル等は最小単位のモジュールとし、全体で使い回す**
  - これらのあしらいがページによって変わることはあまりない
    - これらはサイト内ビジュアルにおいて「**一貫性を保つ**」という役割もあるため
    - つまり修正した際、修正が全ページに反映されなければならない
  - ※レイアウトや幅の変化はコンテキストによって起こり得る
    - 「あしらい」＝border, shadow, color, font等

現状半田がベストプラクティスと考えている設計

- 見出し・ボタン・タグ・ラベル等は最小単位のモジュールとし、全体で使い回す
- **複数の要素からなるモジュールは、子要素に親のモジュール名を継承させる**
- モジュール自体がカラムを形成することは避ける

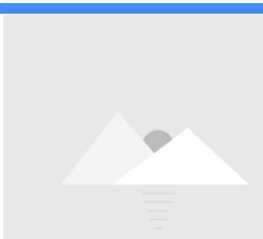
Section Title1



Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More



Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Section Title2

Youtube

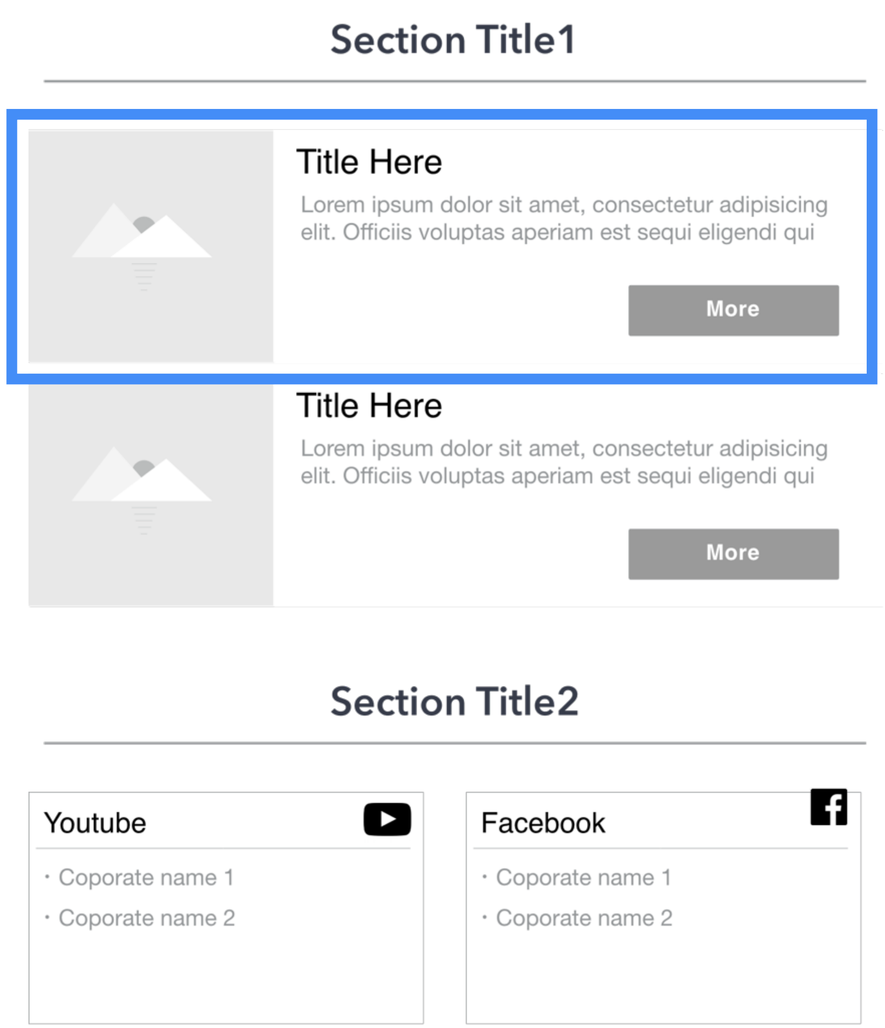
- Coporate name 1
- Coporate name 2

Facebook

- Coporate name 1
- Coporate name 2

子要素に親の

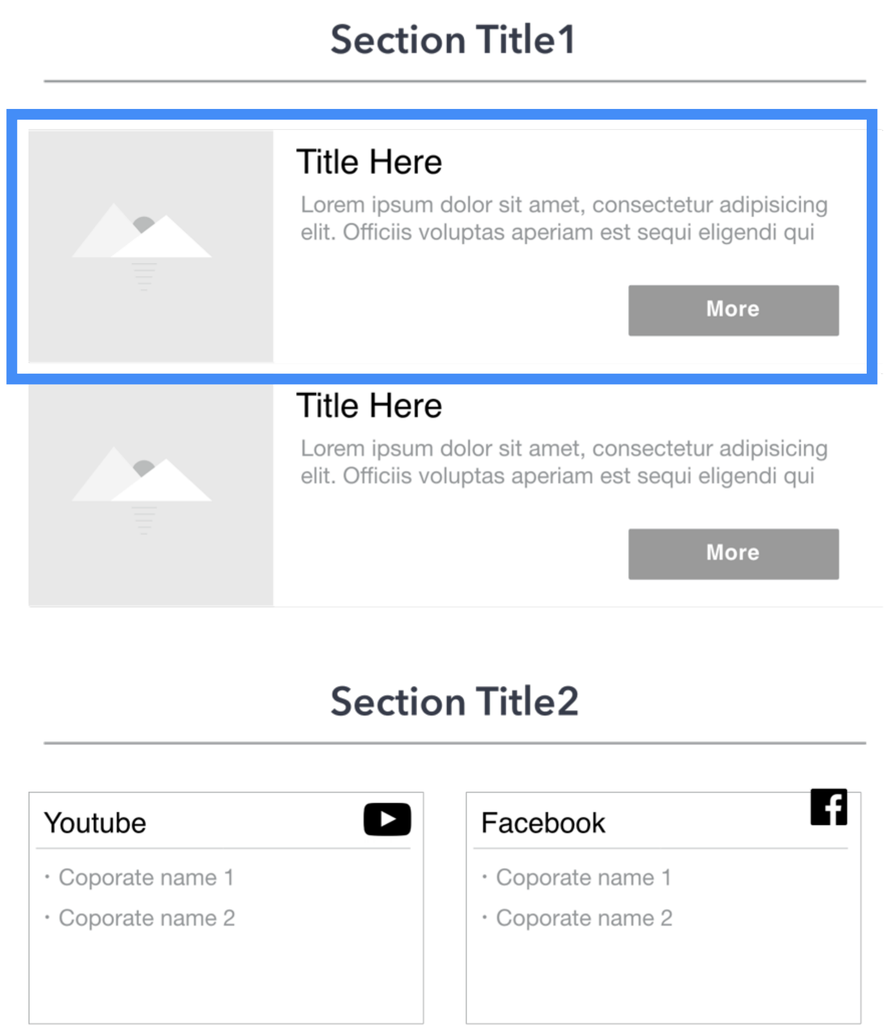
モジュール名を継承させる




```
<div class="pareto-block">
  <figure class="pareto-block-img">
    <img src="...">
  </figure>
  <div class="pareto-block-desc">
    <p class="pareto-block-title">...</p>
    <p class="pareto-block-text">...</p>
    <p class="pareto-block-btn">
      <button class="btn">More</button>
    </p>
  </div>
</div>
```

子要素に親の

モジュール名を継承させる




```
<div class="pareto-block">
  <figure class="pareto-block-img">
    <img src="...">
  </figure>
  <div class="pareto-block-desc">
    <p class="pareto-block-title">...</p>
    <p class="pareto-block-text">...</p>
    <p class="pareto-block-btn">
      <button class="btn">More</button>
    </p>
  </div>
</div>
```

現状半田がベストプラクティスと考えている設計

- **複数の要素からなるモジュールは、子要素に親のモジュール名を継承させる**
  - モジュールの「**責任範囲**」を明確にする
  - 言い換えると、親の名前を継承した子要素は

    「そのモジュール外に持っていくことを想定していない」

    ということを表明している
  - 定義済みのモジュールを子要素として持ち、かつスタイルやレイアウトを

    変更する必要がある場合は子(孫)セレクタを使用する。

```
.pareto-block-btn > .btn{
  float: right;
}
```

現状半田がベストプラクティスと考えている設計

- 見出し・ボタン・ラベル（カテゴリタグ）等は最小単位のモジュールとして定義する
- 複数の要素からなるモジュールは、子要素に親のモジュール名を継承させる
- モジュール自体がカラムを形成することは避ける

Section Title1

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Section Title2

Youtube

- Coporate name 1
- Coporate name 2

Facebook

- Coporate name 1
- Coporate name 2

Section Title1

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Section Title2

Youtube

- Coporate name 1
- Coporate name 2

Facebook

- Coporate name 1
- Coporate name 2

Section Title1

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Title Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Officiis voluptas aperiam est sequi eligendi qui

More

Section Title2

Youtube

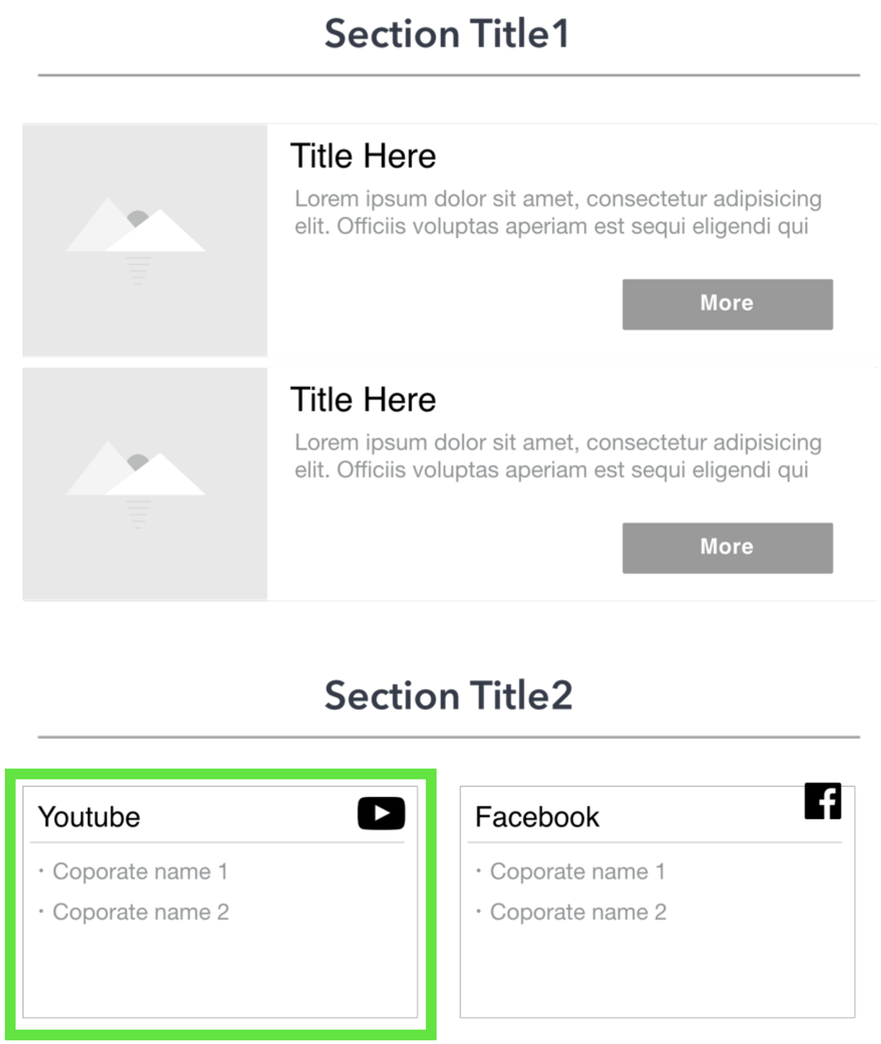
- Coporate name 1
- Coporate name 2

Facebook

- Coporate name 1
- Coporate name 2

モジュール自体が

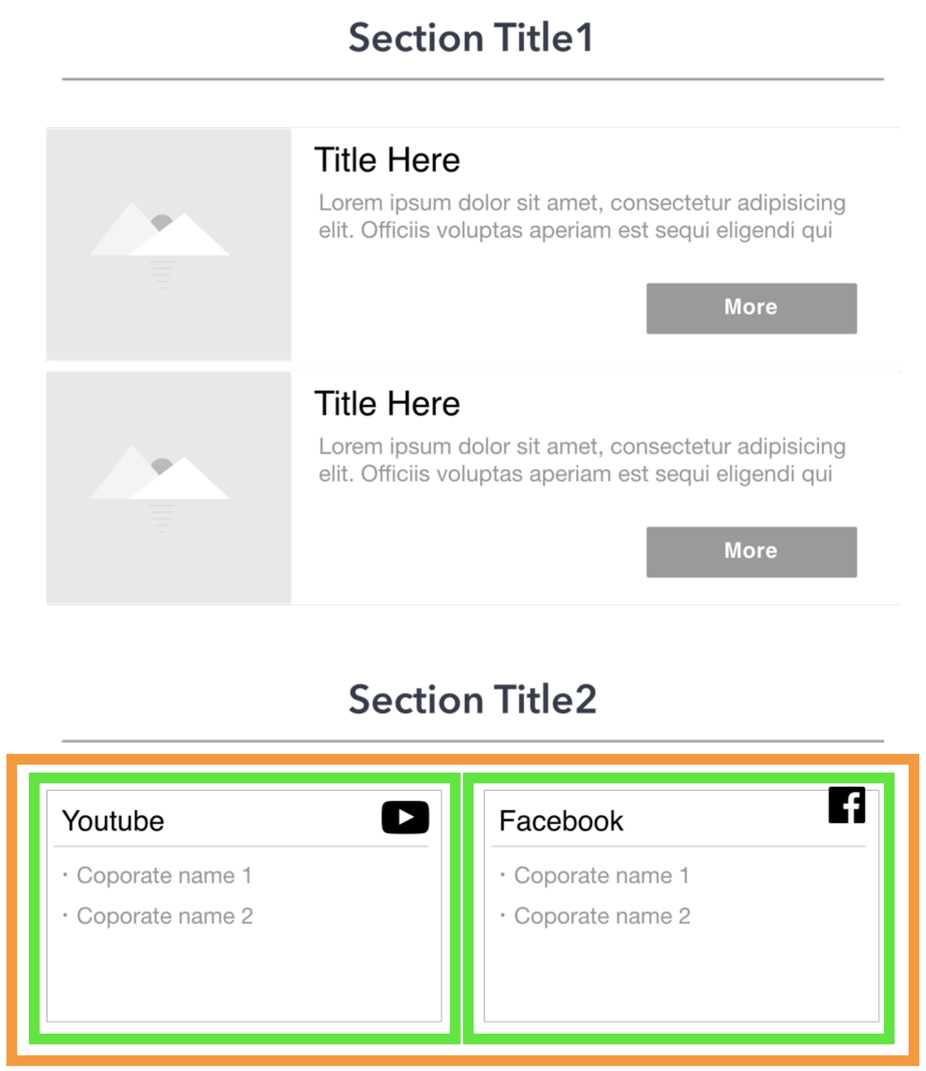
カラムを形成することは避ける




```
<div class="snslist-block">
  <p class="snslist-block-title">...</p>
  <ul class="snslist-block-list">
    <li>...</li>
  </ul>
</div>
```

モジュール自体が

カラムを形成することは避ける




```
<div class="snslist-wrapper snslist-wrapper-2col">
  <div class="snslist-block">
    ...
  </div>
  <div class="snslist-block">
    ...
  </div>
</div>
```

```
.snslist-wrapper > .snslist-block{
  float: left;
}
.snslist-wrapper-2col > .snslist-block{
  width: 50%;
}
```

現状半田がベストプラクティスと考えている設計

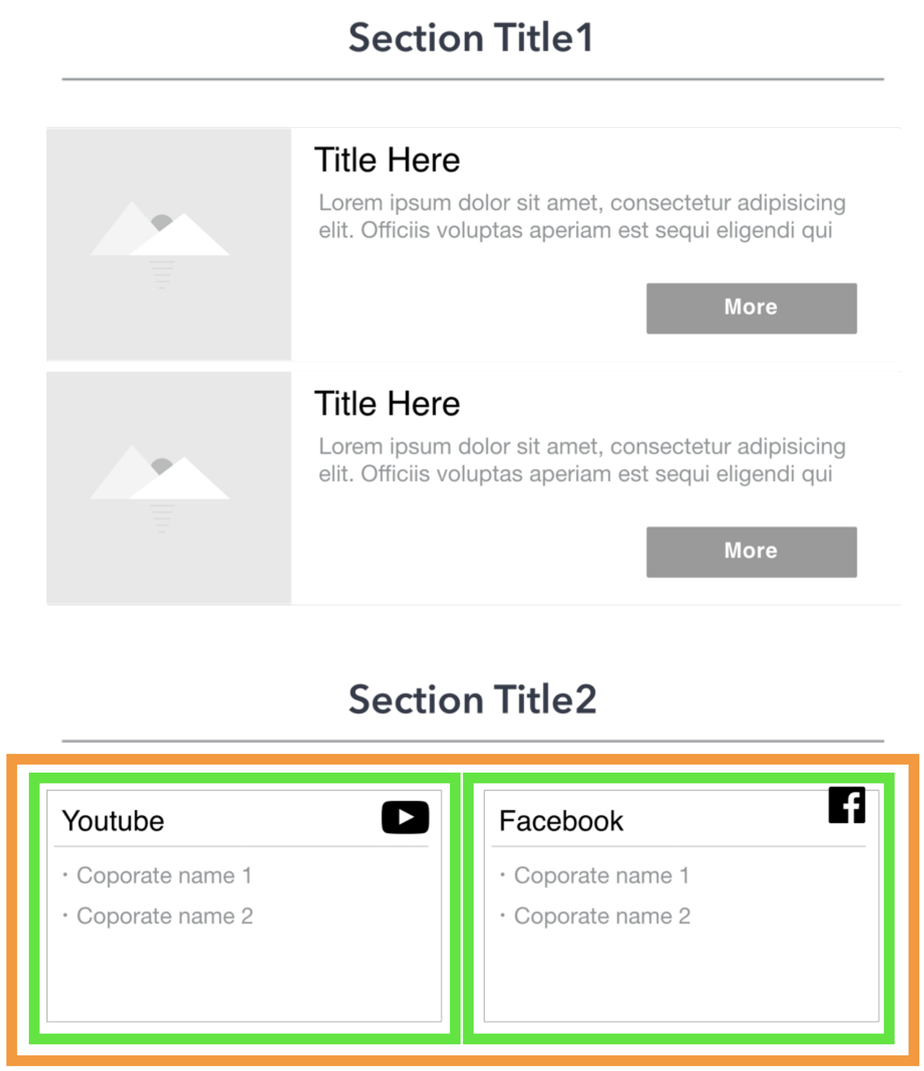
- **モジュール自体がカラムを形成することは避ける**
  - カラムを形成する場合は、そのためのラッパーモジュールを新たに作る
  - CSSフレームワークのグリッドシステムがあれば、そちらを利用することも可能。とても相性がいい
  - モジュール内のカラム形成は可能。(↓のこと)

（3ヶ月後）

ク「やっぱり3カラムにしてもらっていいですか^^」

モジュール自体が

カラムを形成することは避ける

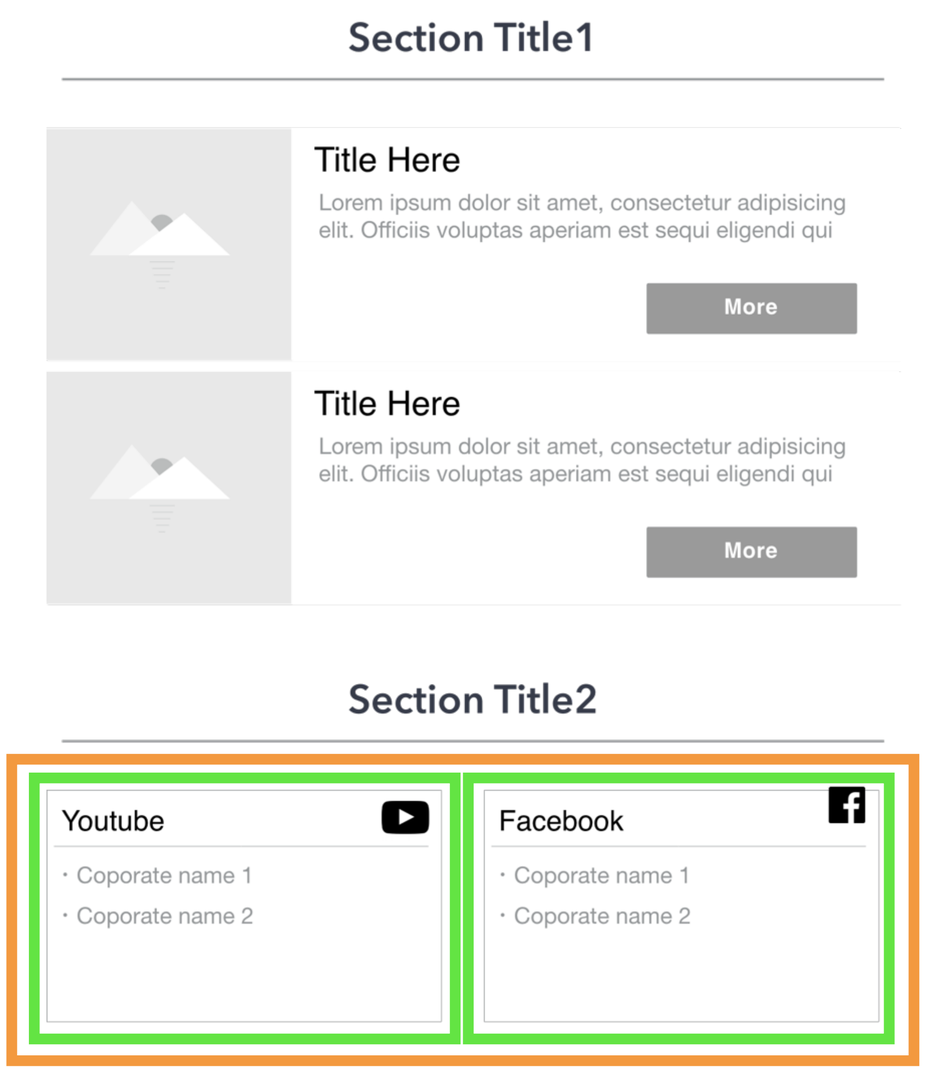



```html
<div class="snslist-wrapper snslist-wrapper-2col">
  <div class="snslist-block">
    ...
  </div>
  <div class="snslist-block">
    ...
  </div>
</div>
```

```scss
.snslist-wrapper > .snslist-block{
  float: left;
}
.snslist-wrapper-2col > .snslist-block{
  width: 50%;
}
```

モジュール自体が

カラムを形成することは避ける




```
.snslist-wrapper > .snslist-block{
  float: left;
}
.snslist-wrapper-2col > .snslist-block{
  width: 50%;
}
.snslist-wrapper-3col > .sns-list-block{
  width: 33%;
}
```

その他気をつけること

- **モジュールの粒度をきちんと意識すること**
  - 前述の通り、私の現状の最適解は「カラムを形成しない」です
    - フレームワークだけでなく、ブロックレイアウト型のCMSとも相性がよい
    - ただし縦方向に長かったり、ページによって要素が増減するモジュールは今も悩ましい
- **名前と責任範囲をきちんと明確にすること**

  名前の責任範囲外の機能を持っているモジュールをよく見かけます

```scss
.btn{
  background-color: $baseColor;
  padding: 10px;
  border-radius: 3px;
}

.btn-orange{
  background-color: orange; //Yep
  padding: 20px; //?!
  border-radius: 5px; //?!?!
}
//君の責任は「オレンジになること」だけだ。越権行為だ。
```

```scss
.btn{
  background-color: $baseColor;
  padding: 10px;
  border-radius: 3px;
}

.btn-orange{
  background-color: orange; //Yep
  padding: 20px; //?!
  border-radius: 5px; //?!?!
}
//君の責任は「オレンジになること」だけだ。越権行為だ。
```

そんな汎用的とか、セマンティックとか、
名前の責任とか、全てにおいていちいち考えるとか
めっちゃ大変なんですが…

ぶっちゃけ慣れしかないです…

後は命名でよく使う単語を公開している方も

いらっしゃるので、それを参考にするのも手です。

「CSS 命名」でおググりください。

それと「絶対このページしか使わない」というものは、
専用の接頭辞を付けて適当に命名してもOKです。

一回しか使わないものに悩まないでください。

（PRECSSではユニークグループとして用意しています）

テンプレートエンジン

テンプレートエンジンについて

- そもそもテンプレートエンジンとは
- EJS、またはNunjucks推奨
- HTMLコードを共通化する
- CMSの構造に合わせる

そもそもテンプレートエンジンとは

- テンプレートとデータを合成し、成果ドキュメントを出力するソフトウェアまたはソフトウェアコンポーネント。
- HTMLの主なテンプレートエンジン
  - EJS
  - Nunjucks
  - Handlebars
  - Pug(Jade)
  - 他にもたくさんありますが、メンテされていないものも多いです

参考：https://ja.wikipedia.org/wiki/テンプレートエンジン

EJS、またはNunjucks推奨

- **EJS**
  - <% %>のデリミタ内にJavaScriptを記述する
  - 使用ユーザー・情報が多い

EJS、またはNunjucks推奨

- Nunjucks
  - Jinja2ベース
  - {% %}のデリミタ内に独自構文を記述する
  - TwigもJinja2とかなり近い構文のため、DrupalやCraft CMS、EC CUBEにそのままロジックを持っていける。とても相性がいい。
  - Mustache記法自体様々な言語で使用されるので、迷ったらこれにしておくと後々幸せになれるかも？
  - Mozilla製

HTMLコードを共通化する

- **SassのmixinとHTMLのコードを共通管理する**
- **後からの直しにかなり強くなる**
- **あまり一般的ではありませんが、マークアップ言語のHTMLに対し、**

  テンプレートエンジン/CMSはプログラムです

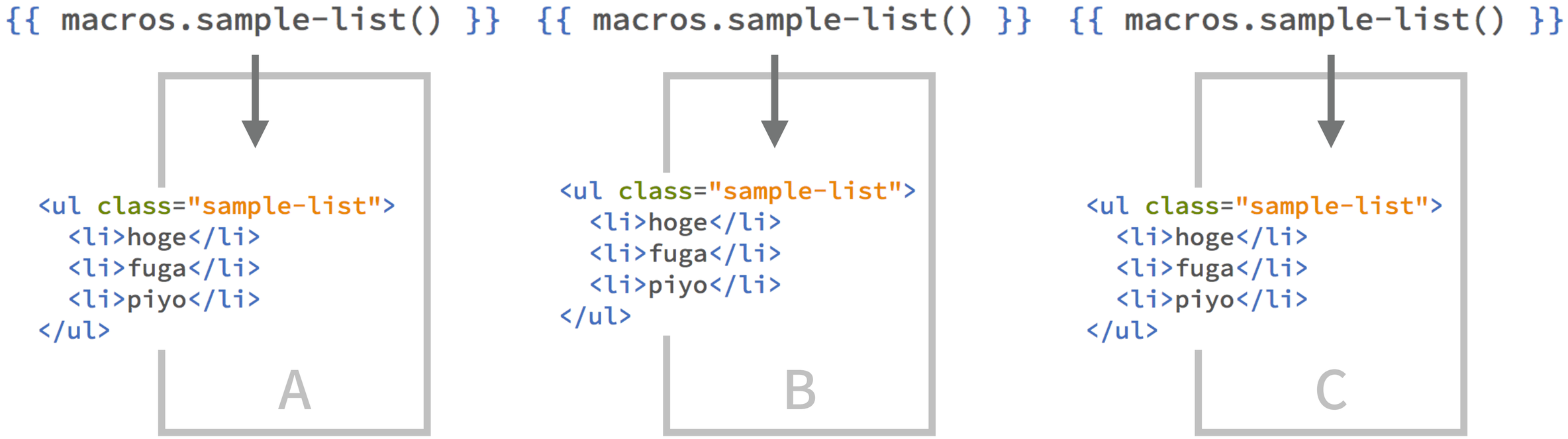
  - であれば、DRYの原則にある程度従うべきと考えています
  - SassよりはよっぽどDRYしやすいです
- **ポイントは構造を共通化し、コンテンツは変更できるようにすること**

HTMLでもコードを共通化する

```
<ul class="sample-list">
  <li>hoge</li>
  <li>fuga</li>
  <li>piyo</li>
</ul>
```

HTMLでもコードを共通化する

```
{%- macro sample-list() %}
  macroの定義
{%- endmacro %}
```



HTMLでもコードを共通化する

①macro定義

```
{%- macro sample-list(item1, item2, item3) %}
  <ul class="sample-list">
    <li>{{ items1 }}</li>
    <li>{{ items2 }}</li>
    <li>{{ items3 }}</li>
  </ul>
{%- endmacro %}
```

②呼び出し

```
{{ macros.sample-list('hoge', 'fuga', 'piyo') }}
```

③出力結果

```
<ul class="sample-list">
  <li>hoge</li>
  <li>fuga</li>
  <li>piyo</li>
</ul>
```

HTMLでもコードを共通化する

①macro定義

```
{%- macro sample-list(item1, item2, item3) %}
  <ul class="sample-list">
    <li>{{ items1 }}</li>
    <li>{{ items2 }}</li>
    <li>{{ items3 }}</li>
  </ul>
{%- endmacro %}
```

改善していきます

②呼び出し

```
{{ macros.sample-list('hoge', 'fuga', 'piyo') }}
```

③出力結果

```
<ul class="sample-list">
  <li>hoge</li>
  <li>fuga</li>
  <li>piyo</li>
</ul>
```

forループ・配列化(改善前)

```
{%- macro sample-list(item1, item2, item3) %}
  <ul class="sample-list">
    <li>{{ items1 }}</li>
    <li>{{ items2 }}</li>
    <li>{{ items3 }}</li>
  </ul>
{%- endmacro %}
```

```
{{ macros.sample-list('hoge', 'fuga', 'piyo') }}
```

forループ・配列化(改善後)

```
{%- macro sample-list(items) %}
  <ul class="sample-list">
    {% for item in items %}
      <li>{{ item }}</li>
    {% endfor %}
  </ul>
{%- endmacro %}
```

```
{{ macros.sample-list(['hoge', 'fuga', 'piyo']) }}
```

forループ・配列化(改善後)

for
ループ化

```
{%- macro sample-list(items) %}
  <ul class="sample-list">
    {% for item in items %}
      <li>{{ item }}</li>
    {% endfor %}
  </ul>
{%- endmacro %}
```

配列化

```
{{ macros.sample-list(['hoge', 'fuga', 'piyo']) }}
```

配列の外部化(改善後)

```
{%- macro sample-list(items) %}
  <ul class="sample-list">
    {% for item in items %}
      <li>{{ item }}</li>
    {% endfor %}
  </ul>
{%- endmacro %}
```

```
{%- set sample-list-items = [
  'hoge', 'fuga', 'piyo'
] %}
{{ macros.sample-list(sample-list-items) }}
```

配列の外部化(改善後)

```
{%- macro sample-list(items) %}
  <ul class="sample-list">
    {% for item in items %}
      <li>{{ item }}</li>
    {% endfor %}
  </ul>
{%- endmacro %}
```

配列の外部化

```
{%- set sample-list-items = [
  'hoge', 'fuga', 'piyo'
] %}
{{ macros.sample-list(sample-list-items) }}
```

引数について

```
<ul class="sample-list sample-list-red">
  <li>hoge</li>
  <li>fuga</li>
  <li>piyo</li>
</ul>
```

引数について

ページによってはクラスを追加したい

```
<ul class="sample-list sample-list-red">
  <li>hoge</li>
  <li>fuga</li>
  <li>piyo</li>
</ul>
```

引数について(改善前)

```
{%- macro sample-list(ext_class, items) %}
  <ul class="sample-list {{ ext_class }}">
    {% for item in items %}
      <li>{{ item }}</li>
    {% endfor %}
  </ul>
{%- endmacro %}
```

```
{%- set sample-list-items = [
  'hoge', 'fuga', 'piyo'
] %}
{{ macros.sample-list('sample-list-red', sample-list-items) }}
```

引数について(改善前)

```
{%- macro sample-list(ext_class, items) %}
  <ul class="sample-list {{ ext_class }}">
    {% for item in items %}
      <li>{{ item }}</li>
    {% endfor %}
  </ul>
{%- endmacro %}
```

```
{%- set sample-list-items = [
  'hoge', 'fuga', 'piyo'
] %}
{{ macros.sample-list('sample-list-red', sample-list-items) }}
```

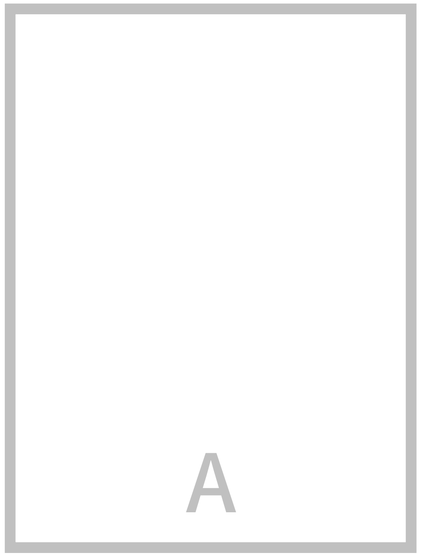
引数について(改善前)

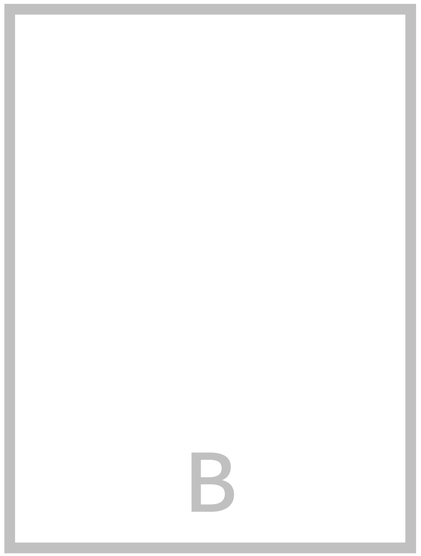
```
{%- macro sample-list(ext_class, items) %}
  <ul class="sample-list {{ ext_class }}">
    {% for item in items %}
      <li>{{ item }}</li>
    {% endfor %}
  </ul>
{%- endmacro %}
```

```
{%- set sample-list-items = [
  'hoge', 'fuga', 'piyo'
] %}
{{ macros.sample-list('sample-list-red', sample-list-items) }}
```

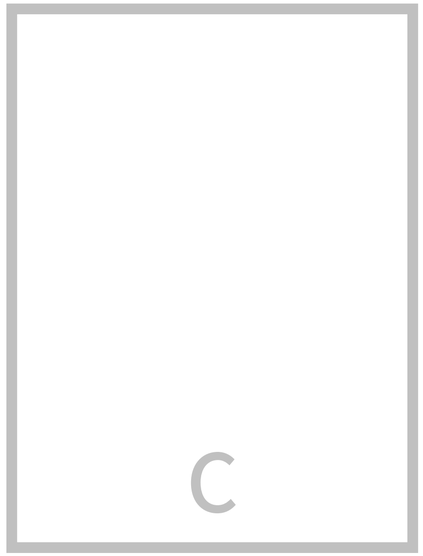
引数について

```
{{ macros.sample-list(sample-list-items) }}
```






```
{{ macros.sample-list() }}
```




```
{{ macros.sample-list('sample-list-red', sample-list-items) }}
```

引数について

- **安易に引数を増やしていくのは好ましくない**
  - テンプレートエンジンによっては、渡す引数と受ける引数が一致していないとコンパイルできない
    - そうするとmacroの引数を増やす度に、全ページのmacro呼び出しを修正しなければならない
  - いちいち順番を覚えるのが面倒
  - いちいち引数宣言を書き換えるのも面倒

↓

- **引数はオブジェクト(連想配列)にして渡す**

引数について(改善前)

```
{%- macro sample-list(ext_class, items) %}
  <ul class="sample-list {{ ext_class }}">
    {% for item in items %}
      <li>{{ item }}</li>
    {% endfor %}
  </ul>
{%- endmacro %}
```

```
{%- set sample-list-items = [
  'hoge', 'fuga', 'piyo'
] %}
{{ macros.sample-list('sample-list-red', sample-list-items) }}
```

引数について(改善後)

```
{%- macro sample-list(data) %}
  <ul class="sample-list {{ data.ext_class }}">
    {% for item in data.items %}
      <li>{{ item }}</li>
    {% endfor %}
  </ul>
{%- endmacro %}
```

```
{%- set sample-list-data = {
  ext_class: 'sample-list-red',
  items: ['hoge', 'fuga', 'piyo']
} %}
{{ macros.sample-list(sample-list-data) }}
```

引数について(改善後)

```
{%- macro sample-list(data) %}
  <ul class="sample-list {{ data.ext_class }}">
    {% for item in data.items %}
      <li>{{ item }}</li>
    {% endfor %}
  </ul>
{%- endmacro %}
```

```
{%- set sample-list-data = {
  ext_class: 'sample-list-red',
  items: ['hoge', 'fuga', 'piyo']
} %}
{{ macros.sample-list(sample-list-data) }}
```

引数について(改善後)

受ける
引数1つ

```
{%- macro sample-list(data) %}
  <ul class="sample-list {{ data.ext_class }}">
    {% for item in data.items %}
      <li>{{ item }}</li>
    {% endfor %}
  </ul>
{%- endmacro %}
```

```
{%- set sample-list-data = {
  ext_class: 'sample-list-red',
  items: ['hoge', 'fuga', 'piyo']
} %}
{{ macros.sample-list(sample-list-data) }}
```

投げる
引数1つ

そこまでするの大変。

そこまでする必要あるのか？

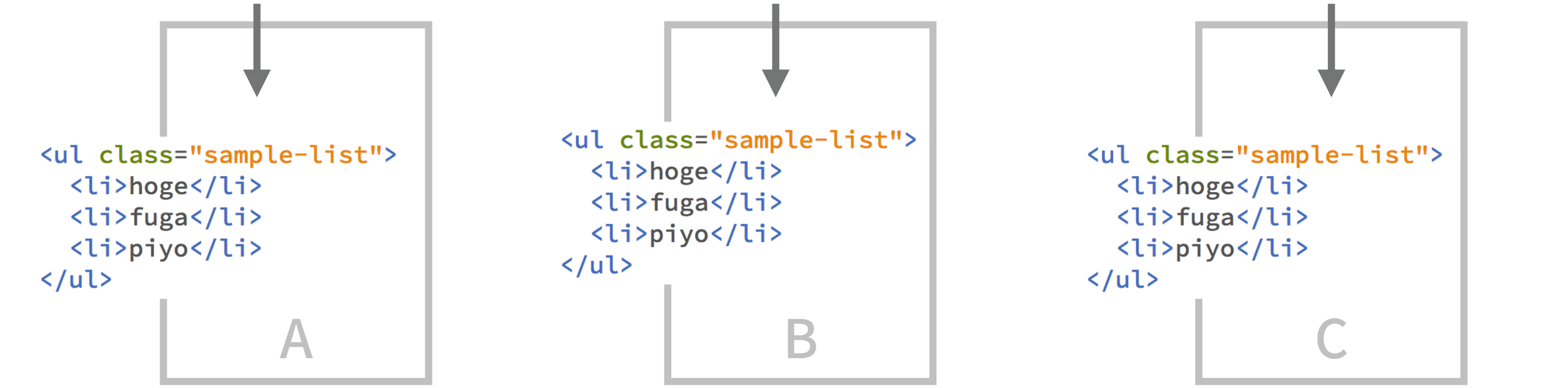
と、思われるかもしれませんが、

これはページ数が増えれば増えるほど

威力を増していきます。

例えばolリストに変更したい

```
{%- macro sample-list() %}
  macroの定義
{%- endmacro %}
```



例えばolリストに変更したい

ここを
変えるだけ

```
{%- macro sample-list() %}
  macroの定義
{%- endmacro %}
```

```
{{ macros.sample-list() }}
```

```
<ul class="sample-list">
  <li>hoge</li>
  <li>fuga</li>
  <li>piyo</li>
</ul>
```

A

```
{{ macros.sample-list() }}
```

```
<ul class="sample-list">
  <li>hoge</li>
  <li>fuga</li>
  <li>piyo</li>
</ul>
```

B

```
{{ macros.sample-list() }}
```

```
<ul class="sample-list">
  <li>hoge</li>
  <li>fuga</li>
  <li>piyo</li>
</ul>
```

C

例えばolリストに変更したい

ここを
変えるだけ

```
{%- macro sample-list() %}
  macroの定義
{%- endmacro %}
```

```
{{ macros.sample-list() }}
```

```
<ol class="sample-list">
  <li>hoge</li>
  <li>fuga</li>
  <li>piyo</li>
</ol>
```

A

```
{{ macros.sample-list() }}
```

```
<ol class="sample-list">
  <li>hoge</li>
  <li>fuga</li>
  <li>piyo</li>
</ol>
```

B

```
{{ macros.sample-list() }}
```

```
<ol class="sample-list">
  <li>hoge</li>
  <li>fuga</li>
  <li>piyo</li>
</ol>
```

C

コンテンツと構造を分離できていれば、

仕様変更があっても最低限の労力で修正できます。

あえて感覚的な言い方をすれば、手綱を握れていないコードが

そこらじゅうに散らばっているのは半田としては非常に落ち着かない

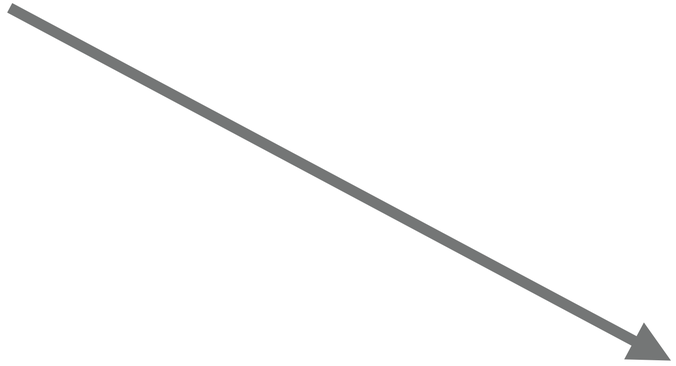
また、応用としてこれをCMSでも再現すると

・クライアントにHTMLをいじって欲しくない

・クライアントがHTMLを怖がる

といったケースにも活用できます。

```
# クライアント用.njk
{%- set prof = {
  img: 'http://....',
  text: 'プロフィールが入ります'
} %}
```



```
# macros.njk
{% import "クライアント用.njk" as cl_data %}

{%- macro prof-block(cl_data.prof) %}
  <img src="{{ data.img }}">
  <p>{{ data.text }}</p>
{%- endmacro %}
```

```
# クライアント用.njk
{%- set prof = {
  img: 'http://....',
  text: 'プロフィールが入ります'
} %}
```

クライアントが
目にするのは
シンプルな形式

```
# macros.njk
{% import "クライアント用.njk" as cl_data %}

{%- macro prof-block(cl_data.prof) %}
  <img src="{{ data.img }}">
  <p>{{ data.text }}</p>
{%- endmacro %}
```

HTMLコードを共通化することの難しさ

- **とはいっても、ベースコーティングとCMSのチーム/会社が別だと現実的には厳しい**
- **構造をそのままCMSに載せられないと威力は半減する(どころか逆に邪魔)**
  - 一応、メジャーなCMSではほとんど可能ですが……
  - 当然処理が走るので、動的生成CMSはmacroがあまりに多いとパフォーマンスにも影響するかも
  - その辺りはソースコードの管理性を重視するか、その他のものをとるか、バランスを見ながら
  - 中〜大規模案件での威力は絶大

テンプレートエンジンについて

- そもそもテンプレートエンジンとは
- EJS、またはNunjucks推奨
- HTMLコードを共通化する
- CMSの構造に合わせる

CMSに載せる際の構造を強く意識する

Nunjucksと互換性のある人達

- **せっかくローカル開発にシステムを入れているので**
  - この段階からCMSを意識して構築すると、テンプレート化する際の工数がかなり減ります
  - 特にNunjucks×Twigの場合は、ローカル開発の変数名等もCMSの予約変数名に合わせると楽

- **基本的には以下を合わせる形**
  - ヘッダー・フッター等のインクルード
  - 変数セットやmacroセット等のインポート

Tips: インクルードパスの書き方について

```
# index.njk
{% include "_inc/_header.njk" %}

# about/index.njk
{% include "../_inc/_header.njk" %}
```

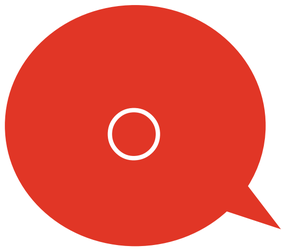
```
# index.njk
{% set inc_path = '_inc/' %}
{% include inc_path + "_header.njk" %}

# about/index.njk
{% set inc_path = '../_inc/' %}
{% include inc_path + "_header.njk" %}
```

Tips: インクルードパスの書き方について

```njk
# index.njk
{% include "_inc/_header.njk" %}

# about/index.njk
{% include "../_inc/_header.njk" %}
```



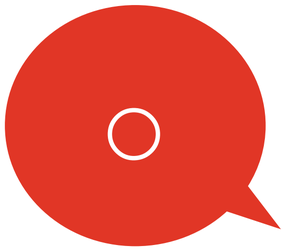
```njk
# index.njk
{% set inc_path = '_inc/' %}
{% include inc_path + "_header.njk" %}

# about/index.njk
{% set inc_path = '../_inc/' %}
{% include inc_path + "_header.njk" %}
```

Tips: インクルードパスの書き方について

```njk
# index.njk
{% include "_inc/_header.njk" %}

# about/index.njk
{% include "../_inc/_header.njk" %}
```



```njk
# index.njk
{% set inc_path = '_inc/' %}
{% include inc_path + "_header.njk" %}

# about/index.njk
{% set inc_path = '../_inc/' %}
{% include inc_path + "_header.njk" %}
```

Tips: インクルードパスの書き方について

- ディレクトリ構造によって「変化する部分」を変数化することにより、

  include文はディレクトリに関わらず他のページでもコピペして使いまわせます

```
# index.njk
{% set inc_path = '_inc/' %}
{% include inc_path + "_header.njk" %}

# about/index.njk
{% set inc_path = '../_inc/' %}
{% include inc_path + "_header.njk" %}
```

Tips: インクルードパスの書き方について

- ディレクトリ構造によって「変化する部分」を変数化することにより、

  include文はディレクトリに関わらず他のページでもコピペして使いまわせます

```
# index.njk
{% set inc_path = '_inc/' %}
{% include inc_path + "_header.njk" %}

# about/index.njk
{% set inc_path = '../_inc/' %}
{% include inc_path + "_header.njk" %}
```

Tips: Pugの使いどころ

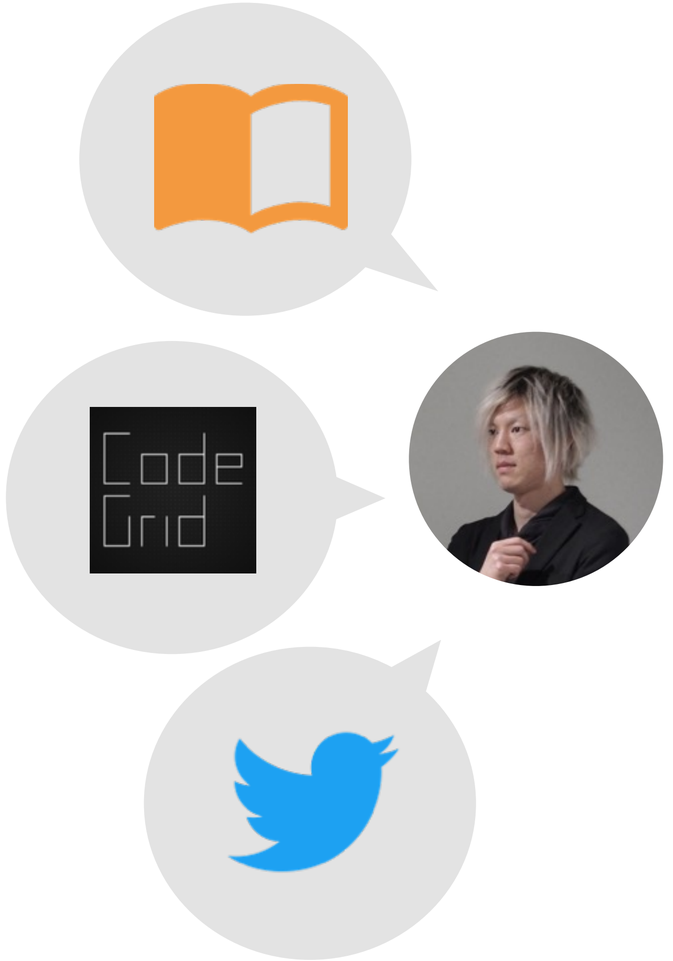
- **EJS/NunjucksはCMSテンプレート化しやすい**
- **対してPugのインデント記法は多くのCMSテンプレートと互換性が無いため、ベースコーディングには向いていません**
  - また.htmlを.ejs/.njk化することは簡単ですが、.pugにするのは大変です
  - 一応コンバーターはありますが…
    - http://html2jade.org/
  - 弊社ではよく、メルマガでPugを使ったりします
    - 案件としてスケールせずページ数も限られるため、インデント記法の弱点が影響しづらい

その他効率化のためのTips

入力支援の使用

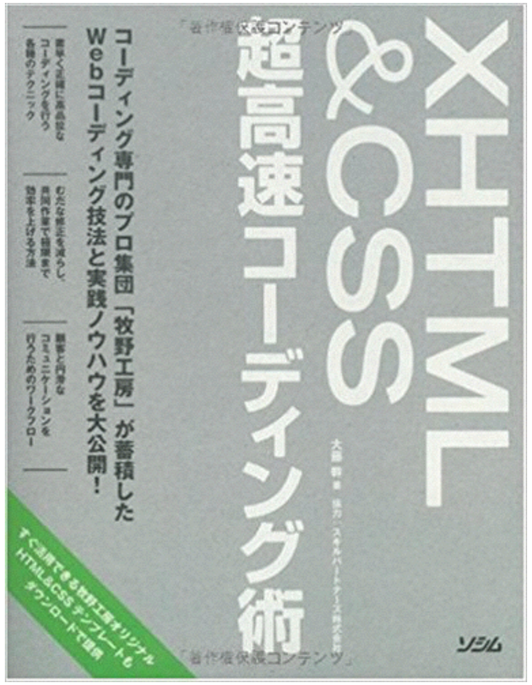
- Emmetはもはや当然
- クラス名・変数名は手で打たない
  - どうせタイポする
- エディタのスニペットの使用
- Clibor/Clipyの使用
- PhraseExpress/Dashの使用

どうやって勉強した？

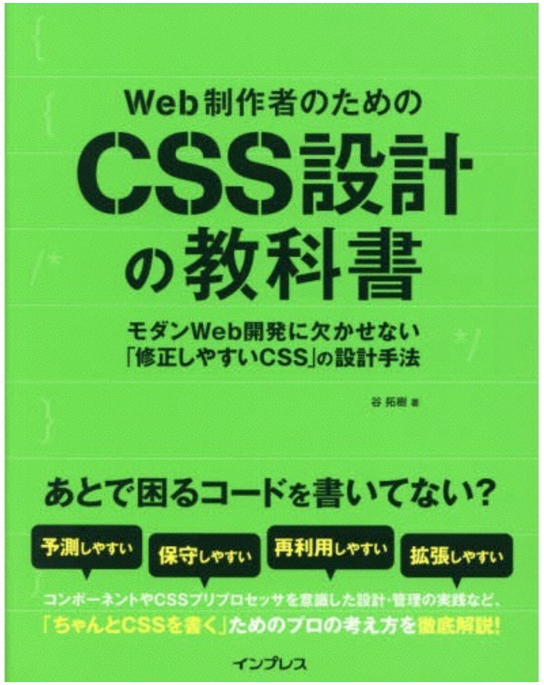


オススメの本

ググるのもいいけど、本は情報がまとまってるから早いよ！疲れないよ！



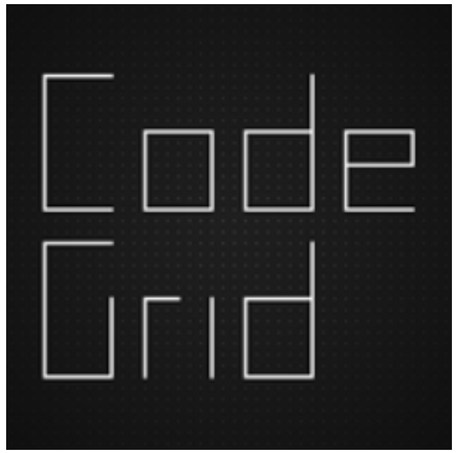
コーディングや、
それにまつわるフローの勉強に！
http://amzn.to/2tsCyCS



CSS設計ならコレ！
http://amzn.to/2spUQzn

最近の事情を俯瞰するならコレ！
http://amzn.to/2u1PHjj

CodeGrid



- 株式会社ピクセルグリッドさん発行のメディア
  - カッチリ基礎を固められる記事から、最新技術まで
  - 全部読むのが大変なほどに充実しています
  - 800円/月
  - とりあえずこれを読んでおけば間違いない

カジュアルゆえに、1番早く最新情報をキャッチできる

著名人を複数フォローしていると、話題のトピックも自ずと掴めます

つまり半田がTwitterばっかりやっているのは、仕事のためだったのだ。

あとは実践あるのみ

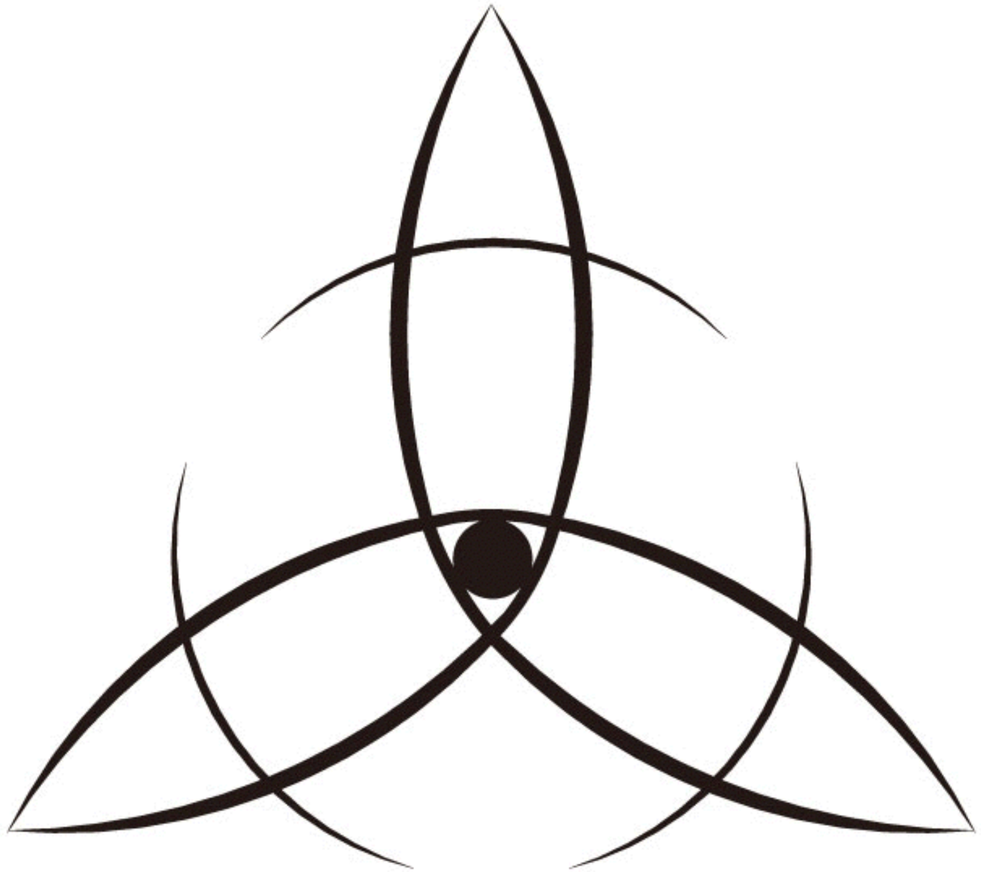
- **特に設計や命名なんかは、実際に問題に直面して初めて重要性を理解します**
  - あとは実践あるのみです
  - ちなみに半田は以下のようにして実践を積みました

これやっといてｗ　後はよろしくーｗｗｗ

元上司S氏

(#^ω^)

まとめ



考えることがたくさんあって大変です。

クラスの分離

名前が汎用・セマンティック

モジュールが小さい・大きい

後工程を考えた設計

いきなり全てやろうとすると大変ですし、

案件によってそこまで必要がない場合もあります。

結局大事なのは「バランス」と、
「そのやり方が今の案件にふさわしいか？」
を考えられることです。

ただし、必要に迫られていきなり超疎結合の
大規模設計をしようとしても、すぐにはできません。

多少オーバースペックでも、日頃からチャレンジして
実際に手を動かすと色々なものが見えてきます。

小規模設計 → 大規模案件　←これは無理

大規模設計 → 小規模案件　←これは可能

その上で「ふさわしいか？」を判断するために、

今日の話がお役に立てれば幸いです。