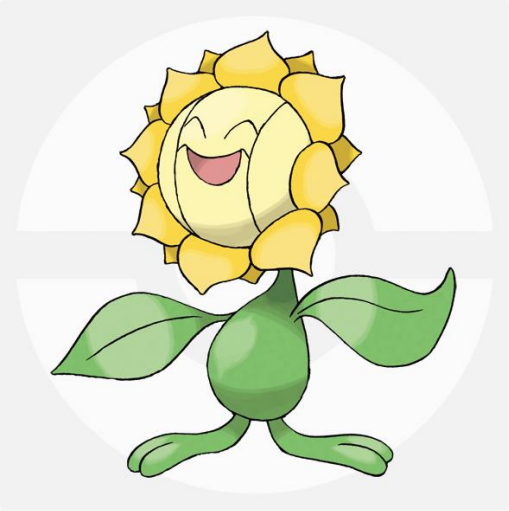
実践課題制作

2019/04/15

Kazuma Sekiguchi
class@cieds.jp

担当



- 関口　和真（Kazuma Sekiguchi）
  見飽きたでしょうけど、顔はやっぱり変わらない
  ←手足はもう少ししっかりしている
- 弱点
  いつでも眠い

  - Comcent,Inc（今年も誰か採用したい。求人出した）
  - Webプログラマ兼コーダーとかいろいろ
  - デザインもごく希にやります
  - たまにWebに記事を書いていたり本を書いていたりします
  - Adobeの多少なりとも関係者
  - 絵は全く描けないので聞かないように

やること

やること

- XDによるプロトタイプ制作方法
- Sassの記述方法、トランスパイルの方法
- Node.jsの使い方
- ConstraintsをNativeAppのレイアウト
- PHPとAjax、REST概念
- データベースの扱い
- カスタマージャーニーマップ（UI/UX）
- マーケティング的知識

やること

- 技術（テクノロジー）とデザインの高度な融合
- 既存の知識を活かしつつ、足りない部分を補う
  - 少なくともWeb屋を名乗るならこの程度は知っておこう
  - SassとかPHPとかNodeとか
- UIとデザイン、マーケティング
  - デザインはマーケティング
  - マーケティング的な知識を有することでデザインに意味が出る

最近の話

- HTML5の仕様策定がW3CからWHATWGに一本化
  - 元々並列で策定されていたが、W3Cが追認する方式に変更
  - 最新版のみ利用可能と変更される（つまり今はHTML5.2以外は利用できない）
- HTML5の仕様上いくつかのタグの扱いが異なっていたが、HTML5.3でWHATWGのものになるはず
  - hgroupが使えるようになるということに
  - 属性の違いが結構多い

いろいろと聞くに

- デザイナーは問題解決能力が必要になってきている
  - 見た目の良いものが作れる、、ではなく何が問題でどう見せて解決するか
- 良いデザイン＝見た目ではない
  - 使い勝手、アクセシビリティ、ターゲットに合わせた色調、拡張性、メンテナンス性などさまざまな要因を考慮する
  - デザイン部分だけでは解決できない＝仕組みを考える
  - 動きによってカバーする

Sass

- やっているはずなんですけどね

- 現在はほぼSass/SCSSで記述するのが一般的
  - 慣れましょう

CSSメタ言語

- CSSを効率的に記述するための記述方法
- CSSを効率的に書くためのものなので、新しい言語という訳では無い

- いくつかの種類が存在
  - Sass ，Scss ，Less辺りが有名で良く使われる

今はまずこれ

CSSの問題

- CSSの仕様上、継承を多く利用してスタイルを適用する
  - 継承の記述が煩雑
  - 継承がCSSファイルだけを見ているだけでは把握しづらい（一覧性に乏しい）
- クラスの動的な適用、排除によるスタイル定義
  - 状態にあわせた（DOM構造の変化に耐える）クラスを多数作成する必要性の向上
  - 多重クラスの利用がしやすくなったことで、ますますCSSに記述する量が増大

高速化の壁

- CSSは複数ファイルに分散して記述が可能
  - CSSファイルに限らず、ファイルの数を減らした方が表示は速くなる
  - できればCSSファイルは1つにしたい
    - 1つにしたらCSSの行数が膨大になり、管理、メンテナンスが面倒になる
  - 最終出力で結合して1つにできれば、メンテナンスが容易なまま高速化が可能
  - 圧縮もついでにしてくれたら良い

SCSS

- CSSにはスコープ概念が存在しない
  - 間違えて他のスタイルを上書きすることもある
  - 命名規則（BEMなど）がさまざま登場してきた理由でもある
- CSSファイルを分離して作成し、最後に必要なCSSを結合していくのが読み出しにも効率的
  - モジュール単位でCSSスタイルを作成
  - いくつかのモジュールでSCSSファイルを作成して結合
  - 最終的に1つのCSSファイルのみを生成する

```
@import “_base”
```

CSSを楽に記述

- 入れ子を簡単に
- 変数を利用可能に
- 同じ表現の使い回しを楽に
- 複数ファイルを結合したい
- できれば圧縮もしたい
- ベンダープレフィックスとか面倒

SassとかLessとか

- Sass（サス）、Less（レス）はともにCSSを簡単に書くためのCSSメタ言語
  - Sassのバージョンアップ版でScss（サス）というのもある
  - 通常サスと呼んだときはSCSSを指すことが多い
- バージョンによって機能に違いがある点は注意
  - 変換ソフトによっては、最新版が使えるとは限らない
- どちらも機能的には変わらない
  - 書きやすい方を選ぶと良い
  - 授業ではSCSSを利用

SassとかLessとか

- どちらもCSSとは違う記述を行う
  - SassはCSSと同じ記述方法も可能（意味無いけど）
  - CSSとして利用する場合は変換が必要
    - 変換をコンパイルと呼ぶ（実際のコンパイルはC言語などで記述したプログラムを実行できるように変換すること。単なる変換とはちょっと違う）
  - 変換せずにJSを通して利用する手もあるが遅くなるためお薦めしない
  - 変換は通常専用のソフトを用いる
    - 今はNode-sassを利用するのが流行り
    - Ruby-scssは開発が止まった

SCSS

- 現状のさまざまなCSSに対する一つの解として使われているのがLESSやSass/SCSS
  - 今後はCSS自体に変数などが入ってくるため、中長期的にはなくなる可能性もある
- SCSSなどを利用することで、ある程度CSSの抱える問題を克服することが可能
  - 全ての問題は克服できない→CSSのそもそもの設計的問題
- Autoprefixerなどを併用することで、ベンダープレフィックスを自動的に付与することも可能

変換

- オンラインでも変換可能
  - http://c2c.briangonzalez.org/など
- インストールして動作するタイプも種類豊富
  - Prepros
  - Koala
  - CodeKit(Mac)
  - Scout
- DreamweaverでもCC2017以降は変換可能

- 変換ソフトによっては、Sassなどを保存したら自動的に変換して、ブラウザーをリロードしてくれるものも
  - 表示確認が楽

出力

- Scssで記述した場合、最終的に利用するのはCSS
  - ScssからCSSへトランスパイルするときに形式を選択可能
  - nested，expanded，compact，compressed
  - 通常はCompressedを選択（コメントや改行、余分な文字など全て排除される）
- Scssの場合、CSSの変更、修正はSCSSファイルで行う
  - CSSを直接編集しないこと（怒られます）
  - CSSが読みづらくても何ら問題ない（Scssが読めれば問題ない）

Scssの記述

Scss

```
$bgcolor:#EEE;
$mainSize:980px;
body{
        background-color:$bgcolor;
        h1{
                text-align:center;
        }
}
#wrapper{
        width:$mainSize;
        margin:0 auto;
        #main{
                width:($mainSize / 2);
        }
}
```

CSS

```
body {
        background-color: #eeeeee;
}
body h1 {
        text-align: center;
}

#wrapper {
        width: 980px;
        margin: 0 auto;
}
#wrapper #main {
        width: 490px;
}
```

Scss

- {の使い方次第で入れ子を表現可能
  - {を閉じないでそのままタグを記述すると親セクレターを付与したセレクターになる
- 変数を利用可能
  - 「$変数名:変数の値」で記述しておけば、どこからでも利用可能
  - 上部にまとめて記述しておけば、メンテナンスしやすい
  - 変数の値を変更すれば、一気に変更が可能
- 計算式が利用可能
  - 四則演算が可能
  - 引き算と割り算は（）でくくること
  - 単位は自動で付与される（割られる対象についている単位が使われる）

SCSSの注意点

- 入れ子が簡単に掛けるため、入れ子が複雑になりやすい
- Sass自体のバージョンがあるため、新しい機能だと利用できないこともある
  - Sass自体をバージョンアップすればOK
- SCSSであまり複雑な関数などを作ると直感的に把握しづらくなるため、分かりやすくなるように心掛ける
  - 関数化して、別ファイルにしておく（隠蔽しておく）

複数のファイルに分ける

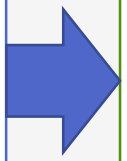
- SCSSでは複数のファイルに分け、読み込むことが可能
- アンダーバーをファイル名先頭に付与することで、パーシャルファイルとして利用できる
  - SCSSを保存してもCSSとしてトランスパイルされない
  - 単独で使用しないファイル（読み込まれて使用する）に使う
  - _base.scss
- 用途にあわせてファイルを分割することで、管理がしやすくなる
  - 要らないCSSファイルを読み込まない、なども可能
- 読み込む際は、@importの後にアンダーバーと拡張子を外して指定する

```
@import “base”
body{
p{
```

Scss

- @import
  - 指定したCSSファイルまたはScssファイルを読み込んでくる
  - 結合が可能
  - CSSの@importとは異なり、記述した位置にファイルが読み込まれるので注意
  - ScssではCSSの@importは多分使えないはず

```
@import base.css
@import layout.scss
```



```
body{/*base.css*/
        margin:0;
        padding:0;
}
.box{/*layout.scss*/
        width:600px;
}
```

Scss

- プロパティも入れ子が可能
  - background-colorなどのハイフン区切りのプロパティはbackgroundの子としてcolorなどを指定可能

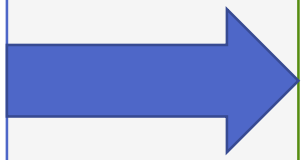
```
#main{
background:{
        color:#EEE;
        image:url(“back.jpg”);
        }
}
```

```
#main {
        background-color: #EEE;
        background-image: url(“back.jpg”);
}
```

Scss

- &セレクター
  - 親セレクターを指したい場合に利用

```
a{
        color:#FFF;
        &:hover{
                color:#F00;
        }
}
```



```
a {
        color: #FFF;
}
a:hover {
        color: #F00;
}
```

Scss

- mixin
  - mixinとは表現の集合体
  - 何度も使うような表現をまとめて記述しておける
    - 修正する場合もmixinを修正すれば全部修正される

```
@mixin normalBox{
        background-color:#FFF;
        font-size:12px;
        box-shadow:1px 1px 1px #000;
        border-radius:3px;
}
#main{
        @include normalBox;
        width:980px;
}
```

```
#main {
        background-color: #FFF;
        font-size: 12px;
        box-shadow: 1px 1px 1px black;
        border-radius: 3px;
        width: 980px;
}
```

SCSS制御文的なもの

- @mixin ，@include

```
@mixin base{
background-color:#ccc;
width:800px;
}
main{
        @include base;
        margin:0 auto;
}
```

SCSS制御文的なもの

- @mixin ，@include
- 外部から値を与えることが可能
- 初期値を設定すれば、@includeで値を与えなくても初期値が使われる

```
@mixin base($color:#F00){
background-color:#ccc;
color:$color;
}
@mixin baseBox($size:900px){
width:$size;
margin:0 auto;
}
body{
        @include base(#999);
        @include baseBox(1160px);
}
```

似たようなもので@extend

- @mixinと似ているが使い道が違う
  - 既存のclass表現などを変更するときに使用する
  - @mixinは既存のclassに表現を追加するときに使用する
  - @mixinはそもそも読み込まれるという前提がある（@extendは普通のクラスなどを読み込むので、読み込まれる前提がない）

```
. extendTableHead {
    @extend .tableHead
    font-size: 10px;
}
```

別に指定したclass定義

それの文字サイズを変更して、新しいクラスを定義

```
@mixin tablehead{
  background-color:#f90;
}
.tableheadline{
  @include tablehead;
  font-size:14px;
}
```

Scss

- 関数を利用可能
  - 使わなくても十分機能的に高機能なため、無理に使うことも無い
  - 変化に強いCSSにはなるので、フレームを作成する際には便利かも

```
$totalWidth: 940px;
$columnCount: 10;

@function getColumnWidth($width, $count) {
  // $columnWidthを計算
  $padding: 10px;
  $columnWidth: floor(($width - ($padding * ($count - 1))) / $count);
  @debug $columnWidth;
  @return $columnWidth;
}

.grid {
  float: left;
  width: getColumnWidth($totalWidth, $columnCount);
}
```

SCSSの関数利用

- SCSSではさまざまな関数を利用することが可能
  - ビルドイン関数
  - ユーザー定義関数
  - JavaScriptに近いがより簡単に記述が可能
- トランスパイラ実行時に実行されてCSSとして展開される
  - JSなどと異なりCSS時に動的に値を変えるような動きをするわけではない
  - CSSのcalc()を使えばある程度の計算はCSS上で可能になっている

SCSS関数（ビルトイン）

- 色関連の関数が豊富
  - rgb関数:rgb(255,200,100)→HEX指定に変換する
  - mix関数:mix(#000,#FFF)→混ぜた色ができる
  - hsl関数：hsl(127,86%,20%)→HEX指定に変換する
  - lighten関数：lighten(#663355,15%)→15%明るくしたHEX値を返す
  - darken関数：darken(#446373,10%) →10%暗くしたHEX値を返す
  - complement(#552638)→補色の色のHEX値を返す

SCSS関数（ビルトイン）

- round関数：round(4.5)→数値を四捨五入
- ceil関数：ceil(7.21)→数値を切り上げ
- floor関数：floor(4.752)→数値を切り捨て
- min関数（20px,53px,10px,$base)→最小の値を返す
- max関数（20px,53px,10px,$base)→最大の値を返す

SCSS制御文的なもの

- @function

```
$mainSize:800px;
@function wrapperSize($size){
	@return $size / 2;
}
main{
	width:wrapperSize($mainSize);
	margin:0 auto;
}
```

SCSS制御文的なもの

- @functionを利用することで、独自の関数を作成することが可能
  - 利用する時は関数名を指定すれば利用可能
  - 関数内では@returnで必ず返り値を指定する
  - 通常戻り値はプロパティの値として利用することが多い
    - セレクターとして利用もできる
  - 通常は、()で引数を受け取るようにして、関数内部の値を変更させる
    - 変更ができない関数は利用する意味が無い

@mixinと@functionの使い分け

- @mixinはCSSの固まりをそのまま持ってくることが可能
- @functionはそのときに値を与えて単一の値だけを計算させることが可能
  - 何らかのCSSプロパティの値として利用する
  - 他のビルトイン関数と組み合わせて利用する
- CSS自体のスタイルをそのまま再利用するのであれば@mixinを利用する
  - @mixin内にも@functionは利用できるため組み合わせることももちろん可能

ループ

- 繰り返して処理をする仕組み
- #{変数}みたいに記述する（インターポレーション）とセレクター側で変数を利用可能

```
@for $i from 1 through 5{      //5回繰り返す。
        .item_#{$i}{
                margin:$i*10px;//$iは1回ずつ増えていく
        }
}
```

配列

- 配列を作成可能
  - 単一の変数に複数の値（文字列）を入れることができる仕組み
  - 通常は@eachと組み合わせて利用する

```
$items = ‘apple’,’orange’,’berry’;
@function urlimgbase($image){
        @return url(“../images/”+$image);
}
@each $item in $items{
.#{$item}{
        background-image:urlimgbase($item+’.jpg);
}
```

分岐

- @if,@elseを利用した分岐が可能
  - 条件に合わせて利用する関数を分けたり、書き出すCSSを変えることが可能

```
for $i from 1 through 10{
.box_#{$i}{
    width:100px;
    @if($i == 10){
        margin-bottom:10px;
    }
    @else{
        margin-bottom:5px;
    }
}
```

Scss

- Scssの保存時、拡張子は、scssとすること（sassではないので注意）
- Scssのコメントは「/* */」のほか、「//」タイプの1行コメント文も可能
  - //タイプの場合は、/* */に変換される（コメントを削除する変換もある）
- 通常Scssファイルの保存先フォルダとCSSファイル保存先フォルダは分けるので、パスに注意