Webデザイン実習3B

2019/7/08

Kazuma Sekiguchi
class@cieds.jp

課題

- 今までにやったJSの技法を活かしてサイトを作成
  - インタラクションのあるサイトを作成
  - インタクションは効果的に利用すること
  - jQueryの使用OK、jQueryプラグインはそれだけを利用する訳では無いならOK（プラグイン以外も利用する）
  - JSの機能は2つ以上あること
- テーマ
  - 夏に楽しめるもの、こと、場所の紹介サイト
  - 自分で作成したキャラクターなどは許可
  - トップページのみ作成すればOK

最終課題

- 画像などは自分で描いたもの、撮ったもの、商用利用可のもののみ利用可
  キャラクター関係は不可（特にネズミー関係）（自分で作ったキャラクターならOK）
- デザイン（ai、psd、sketch、XDなど編集可能なもの）とそれのPDF化したもの、HTML,CSS,js,画像群を提出
- デザインカンプはアウトライン化、ラスタライズ化しないこと
  - 保存バージョンは最新のものでOK

提出日など

- Topページはindex.htmlとすること
  - 直下に配置すること
  - 日本語ファイル名は不可
- 画像はimagesフォルダにまとめて配置すること
- カンプ（aiとかpsdとか）は直下に配置すること
- カンプと最終的に完成したものが多少異なるのはOK
  - 全然違うのはダメだけど

提出日など

- 2019年7月29日までに提出
  - 直接関口に提出してください（メール提出は不可）
  - その前に提出するのはOK
  - 早めに出しておくことをお奨めします

見るところとか

- きちんとタグが使い分けできているか
- CSSが書けているかどうか。妥当なCSSになっているかどうか
- 表示したときに崩れていないか、画像が表示されるか
- ファイル名、フォルダー名が英数字だけで構成されているかどうか
- デザインがWeb的かどうか
- JSがきちんと作成されているか、きちんと動作するか
- scriptタグなどの指定に間違いが無いか。jqueryを複数読み込んだりしていないか
- jqueryの初期化指定が正しくできているか
- jsを効果的に使用できているか

夏休みの課題

- ウェブサイトを30サイト閲覧し、そのウェブサイトに関してレポート
- A4用紙に余白を四方それぞれ20mm取り、ウェブサイトのスクリーンショット（メインなページ）を貼り付け
  - Word等で作成するのが楽
- 下部に「閲覧日」、「URL」、「タイトル」、「良いと思った点や改善点など（3行程度）」を記入
- 1ファイルのPDF形式にして提出（ファイル名を学籍番号＋名前にすること）

WebSitesのまとめサイト

- WebDesignClip（日本のサイト）
  - http://webdesignclip.com/
- WebDesignFile（海外サイト）
  - http://www.webdesignfile.com/
- I/O 3000 | Webデザインギャラリー
  - http://io3000.com/
- Webデザインリンク集
  - http://bm.straightline.jp/

提出日など

- 夏休み明けの最初の授業に提出

段組

- なんだかんだ言ってもPC向けのWebサイトでは良く使うレイアウト
  - メニューを横並びにするのも段組とも言える
- 横並びにするなら~~float~~
  - というのはCSS的にも正しくない
  - floatは本来回り込みを指定するプロパティ
  - 画像の周りなどに文字を流し込む方向を指定するもの
  - そのため高さが算出できなくなりレイアウト崩れの原因に

float

- 今ではレイアウトを組み合わせるために使用する意味は下がっている
  - 画像の回り込みとしての位置指定では使用する
- floatを使うことで他の人が判断しづらいコードになることも
  - float解除するために余分なコードが必要になる
  - clearfixに代用されるためにclear-afterプロパティというものが提案されているが採用ブラウザー無し

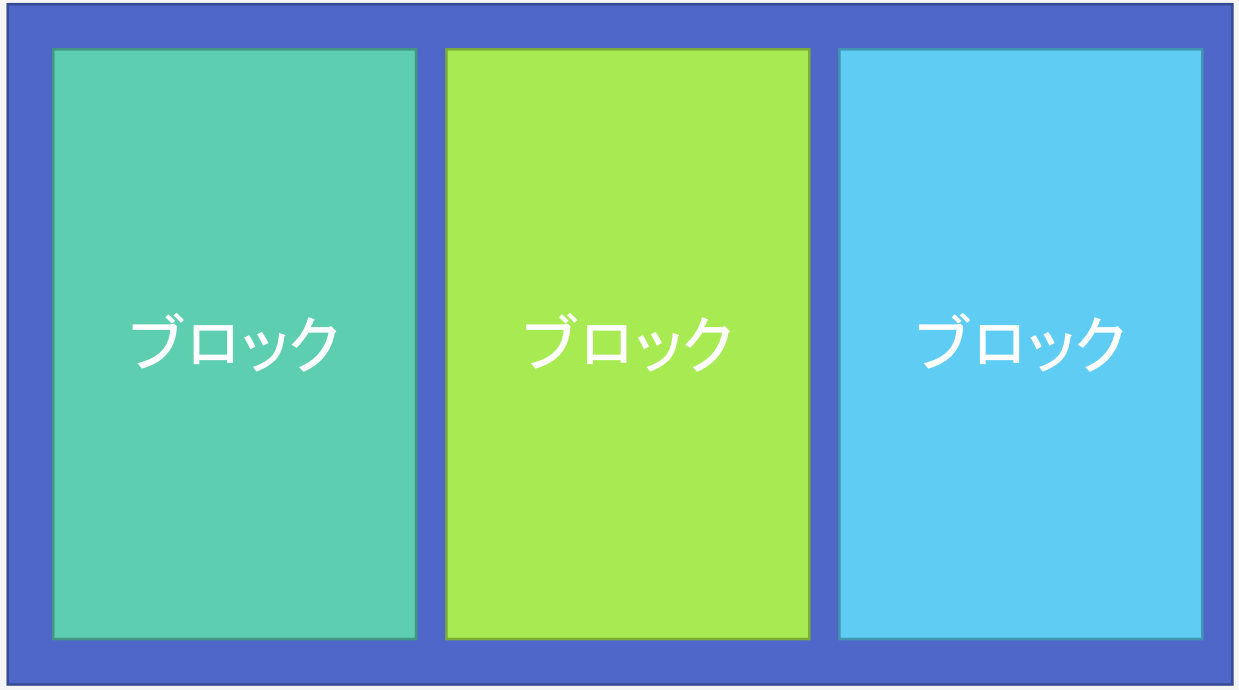
段組をどう実現するか

- flexboxか、CSS Gridを使う
  - どちらもレイアウトのためのCSSプロパティ
  - IEでもある程度利用可能なため、ブラウザー互換性は高い
  - float解除なども不要なため、崩れづらい
  - 細かいプロパティ調整が可能
- ベンダープレフィックスの付与
  - まだ必要なブラウザー対応は必要かも
  - autoprefixerなどを利用すれば自動的に付与してくれる

Autoprefixer

- PostCSSの一部とされる
  - SCSSで記述したCSSをCSSに変換した後で行う処理に存在
  - 必要なベンダープレフィックスを補ってくれる
  - 今のところSCSSをCSSに変換するには、Gulpなどのビルドシステムを使うが、そこに組み込むのが一般的
  - https://autoprefixer.github.io/
    でも使えるので、とりあえず試すなら

flexbox




```
.wrapper{
        display:-webkit-box;
        display:box;
        display:-webkit-flexbox;
        display:flexbox;
        display:-webkit-flex;
        display:flex;
}
```

- ブロック全体を括るブロックにflexプロパティを設定
- 囲われた中にある要素が横並びになる
  - 高さも一番高くなる物に合わせてくれる
  - float解除は不要

flexboxの利用

- flexboxはCSS3からサポート
  - CSS3のサポート状況はブラウザによって異なる
  - InternetExplorerの古いバージョンなどはサポートしていない
  - 古いブラウザを切り捨てるのであれば、問題なし（IE10からはサポート）
- スマートフォンならまず対応しているので、こちらを利用しない手は無い

flexboxで改行

- flexboxのデフォルトは1行で収まるように調整される
  - 内部要素に幅が指定されていても無視されて1行で収まるように調整されてしまう

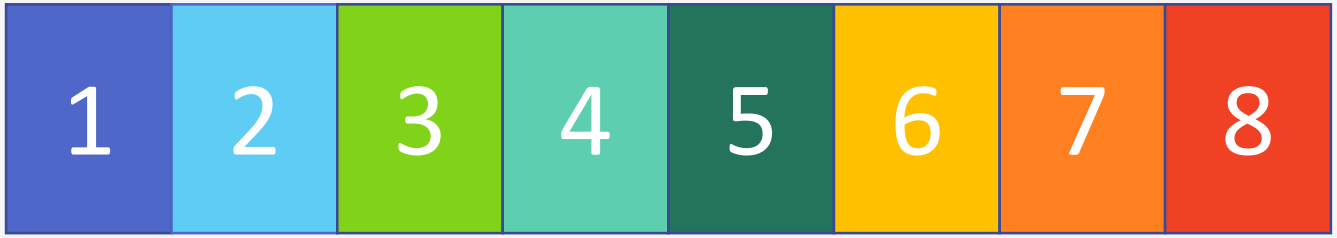
```
-webkit-box-lines:multiple;
-moz-box-lines:multiple;
-webkit-flex-wrap:wrap;
-moz-flex-wrap:wrap;
-ms-flex-wrap:wrap;
flex-wrap:wrap;
```

  - これを記述することで、1行に収まらない場合は改行されるように変更される

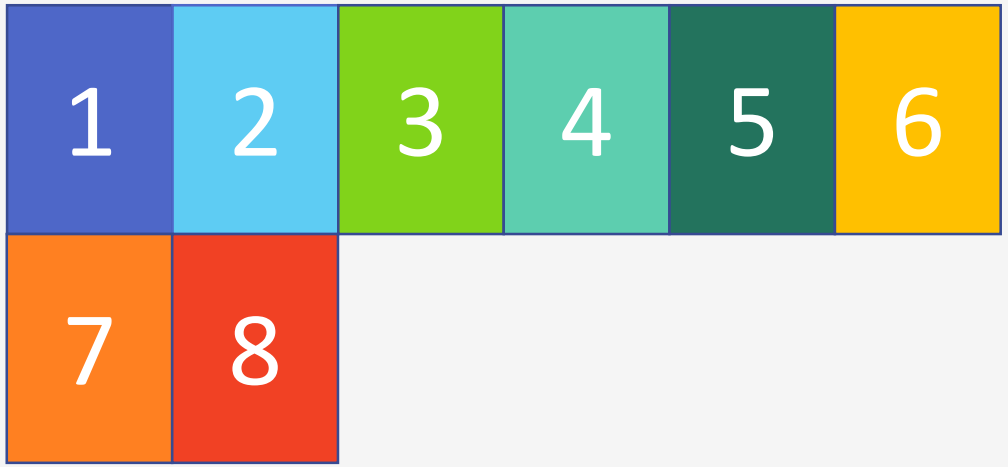
flexboxの利用

- flexboxを利用するとカラムを楽に作成可能
  - display:boxなどを指定
  - MediaQuery内などで解除する場合は、display:blockでOK
  - 崩れにくい、順番を任意に入れ替えることが可能、子要素の大きさで改行有り・なしにすることが可能など多数利点
- display:inline-blockを使う手も
  - inlineだが、blockレベルという意味不明な指定
  - inline＝横並びが可能、block＝幅指定とかが可能
  - 横並びにしたい要素自体に指定し、幅を指定することが可能
    - 幅が収まらなければ勝手に改行される

flexboxのいろいろ



- display:flexで横並び

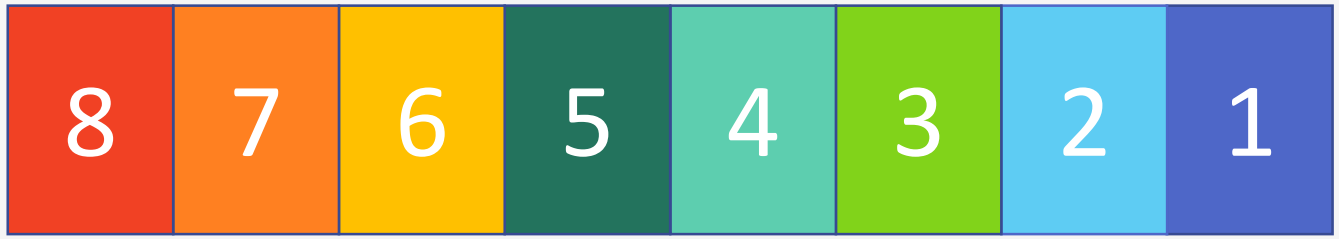


- flex-wrap:no-wrap→改行を許可しない（デフォルト）

：wrap→改行を許可する

：wrap-reverse→改行を許可するが、改行された行を先に表示する

flexboxのいろいろ

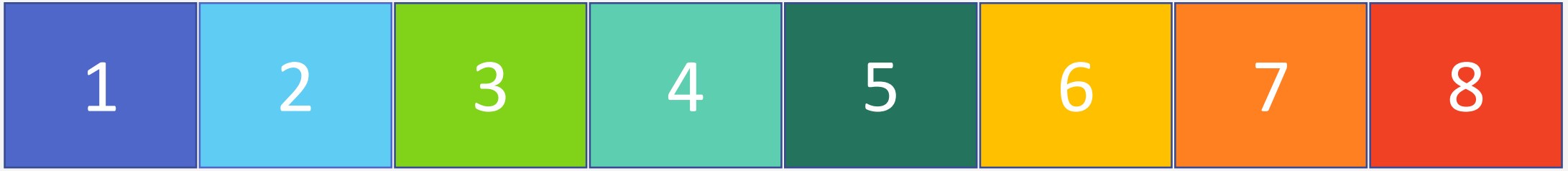


- flex-direction：row→左から右へ配置（デフォルト）
  ：row-reverse→右から左へ配置
  ：column→上から下へ配置
  ：column-reverse→下から上へ配置

flexboxのいろいろ

- justify-content：flex-start→左寄せ（デフォルト）
  ：flex-end→右寄せ（reverse系が入っていると左寄せ）
  ：center→中央寄せ
  ：space-between→最初と最後を両端に配置して残りは均等配置
  ：space-around→すべてを均等配置

flexboxのいろいろ



- align-items→垂直方向の位置指定

：stretch→親要素のサイズに合わせて拡大する

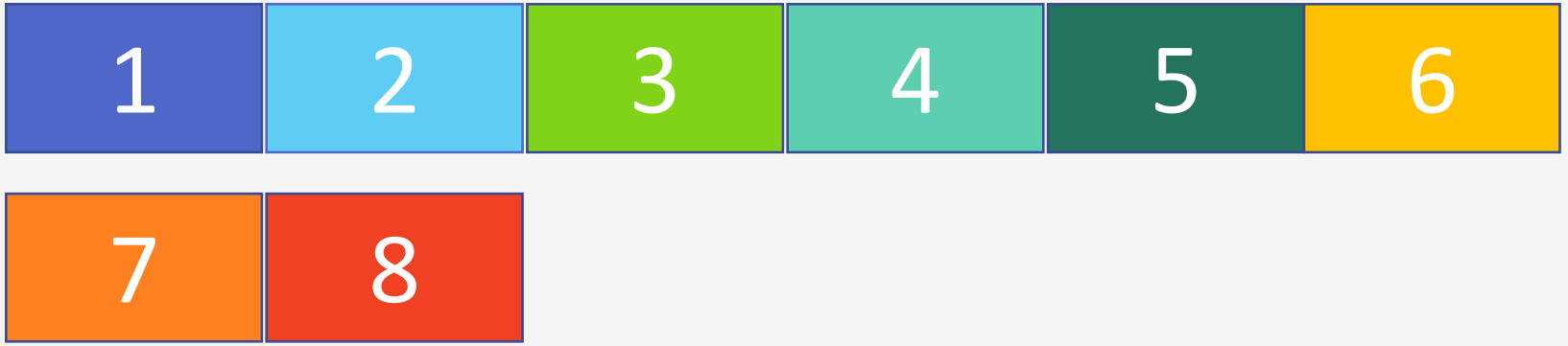
：flex-start→親要素の上側に配置

：flex-end→親要素の下側に配置

：center→親要素の中央寄せ

：baseline→中の文字のベースラインに合わせて配置

flexboxのいろいろ



- align-content→垂直方向の配置位置（flex-wrap:wrapのとき）

：stretch→親要素のサイズに合わせて子要素を拡大し、敷き詰める

：flex-start→親要素の上側に配置

：flex-end→親要素の下側に配置

:center→親要素の中央寄せ

：space-between→最初と最後の子要素を上下の隙間を取らずに配置し、残りの子要素を均等に間隔を開けて配置

：space-around→上と下に隙間をとって均等に間隔を開けて配置する

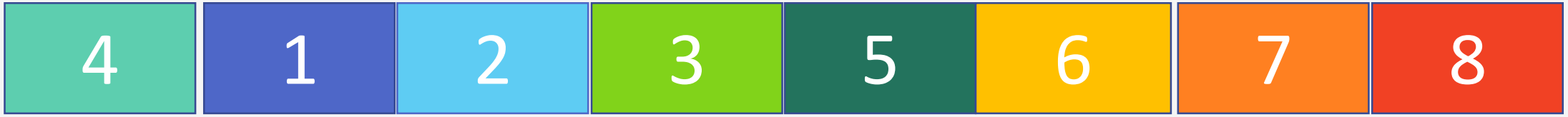
flexboxのいろいろ（子要素に指定）

- flex-grow：子要素の大きさを指定
  ：数字で指定。他の要素と比べての相対的な大きさを指定。0がデフォルト

flexboxのいろいろ（子要素に指定）

- flex-shrink：子要素の小ささを指定
  ：数字で指定。他の要素と比べての相対的な小ささを指定。1がデフォルト、0で縮小しない

flexboxのいろいろ（子要素に指定）



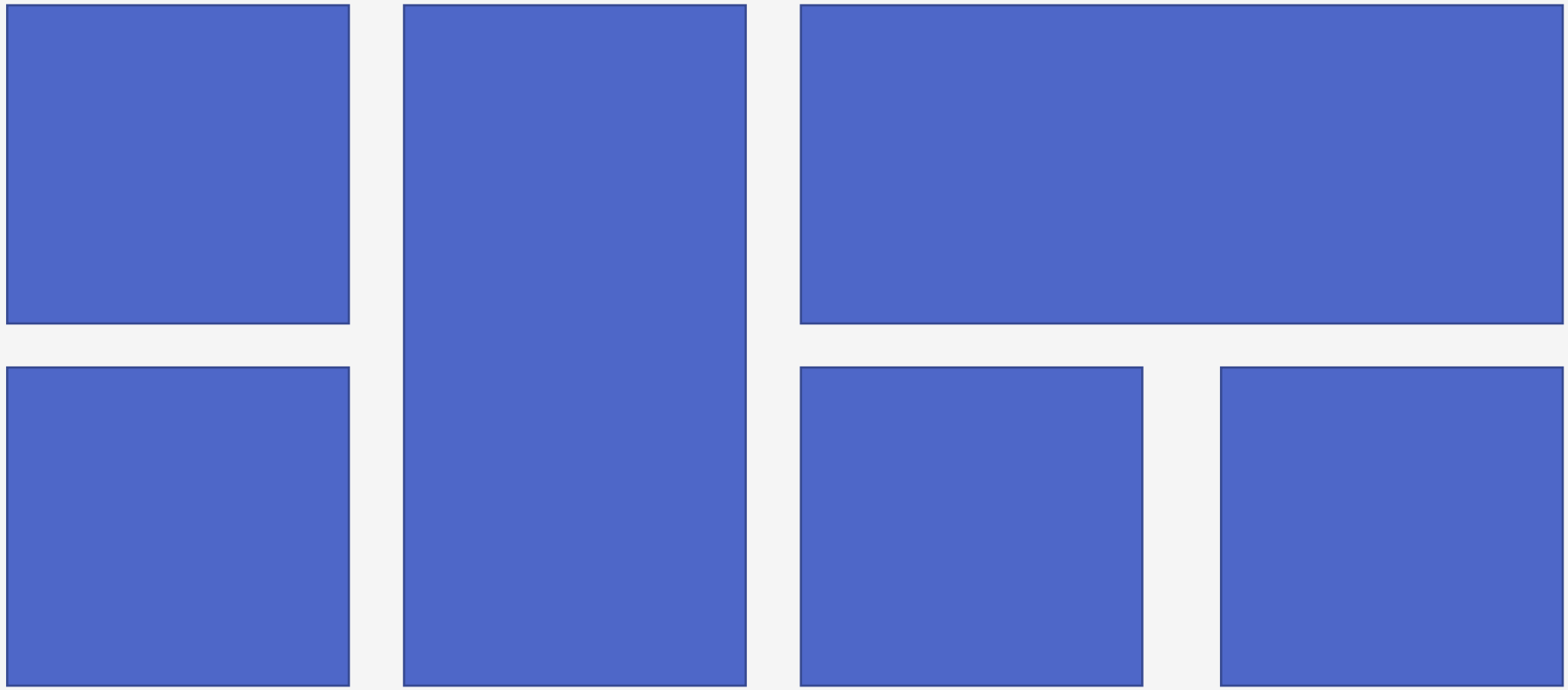
- order：順番を整数で指定（負数を指定可能）
  ０を指定するとデフォルトの位置に配置される。数が低いほど最初の方に指定され、高い物ほど後の方に配置される
  他の子要素にorderプロパティを指定しなければ1を指定すれば後の方に配置される

CSSGrid

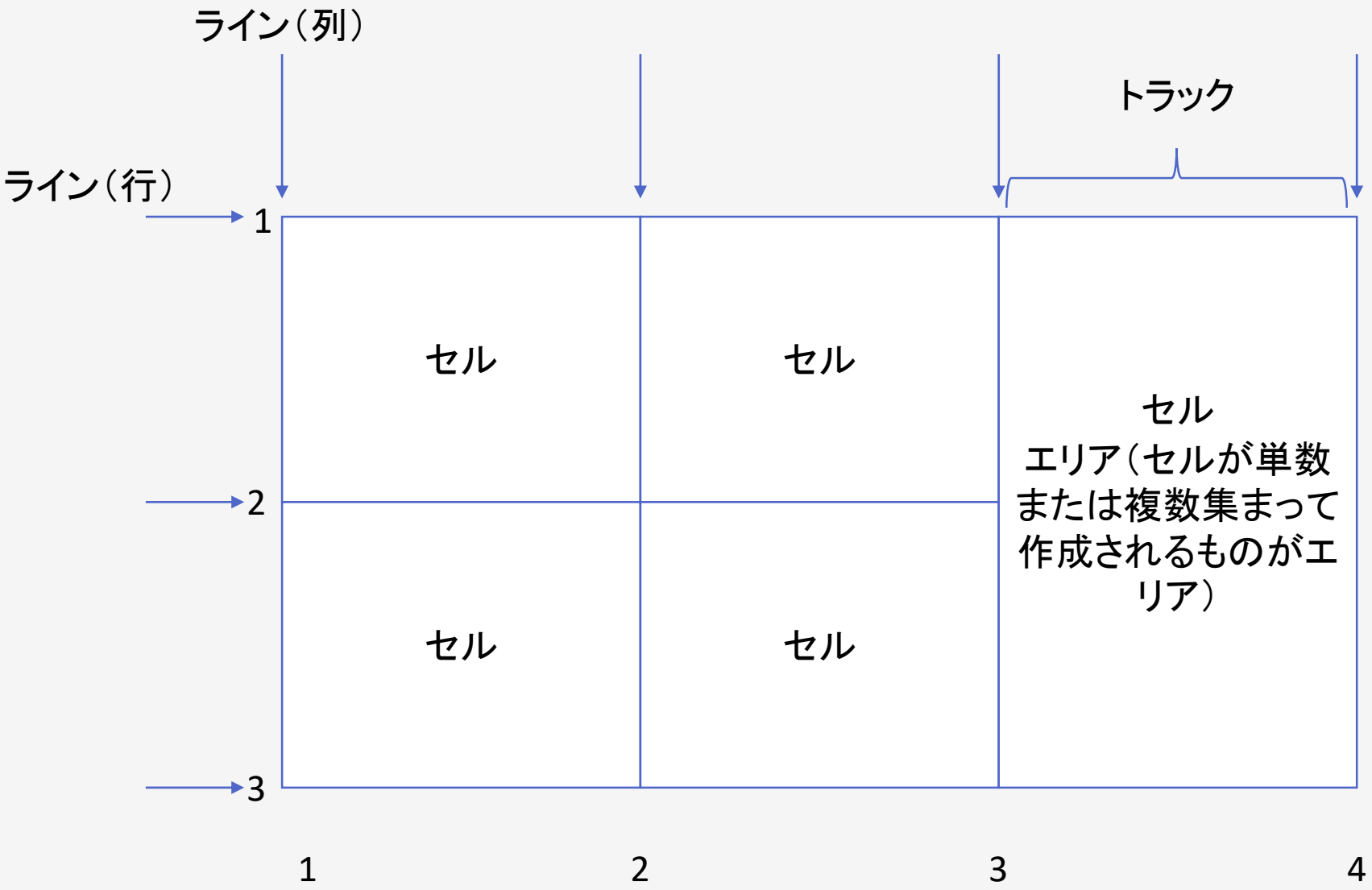
- CSSで2次元のグリッドを描きレイアウトを作成するための仕組み
  - これまで面倒だったレイアウトが簡単になるとされている
  - （慣れるまではこれまでよりもかなり面倒になる）
- マス目を作成してそこに要素を配置していくことが可能
  - 幅、位置も自由に設定ができる
  - グリッドからズレる要素を配置すると面倒な面も出てくる
- 現状サポートしているブラウザーがFirefox52以上、Chrome57以上、Safari10.1以上
  - IE11も一部であれば対応している

CSSGrid

- 下部のようなレイアウト配置がCSSの少ない記述で可能
  - 崩れることが少ない



分かりづらいが言葉



- ラインは行と列が存在
  - 1から数えていく
  - 右から、または下から数える場合は-1,-2のように数えていく

CSSの記述

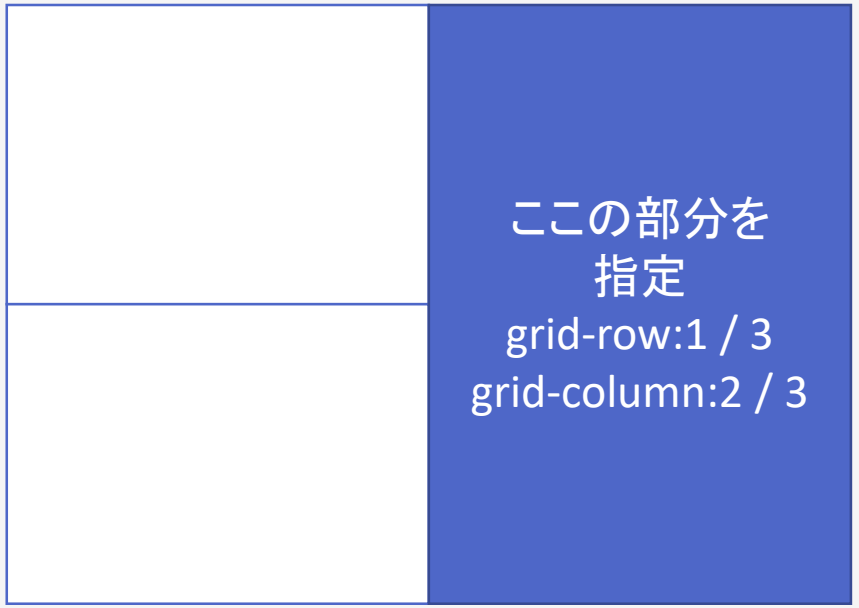
- コンテナを作成し、中に収める要素を指定する
  - コンテナ=グリッド全体を括る要素
  - コンテナにdisplay:gridを指定
  - grid-template-rows:とgrid-template-columns:でトラックサイズを指定する
  - 単位としてfrという単位が利用可能
- 中の要素は直接の子要素がアイテム（セル）として配置される
  - CSSでの明確な指定は不要
  - 位置をgrid-row,grid-columnで指定する

CSSの記述

- grid-template-columsに対して、repeat()記述が可能
  - grid-template-columns:1fr 1fr 1frは
    grid-template-columns:repeat(3,1fr)と記述可能
  - 他の指定と組み合わせることも可能
    grid-template-columns:200px repeat(3,150px) 1fr
- grid-template-rowsで指定された数が足りない場合は、勝手に枠が作成される
  - 但し、サイズは自動的に拡大されるため、コンテンツ内部に依存する
  - 数を設定したい場合は、
    grid-auto-rowsとgrid-auto-columnを設定しておく

CSSの記述

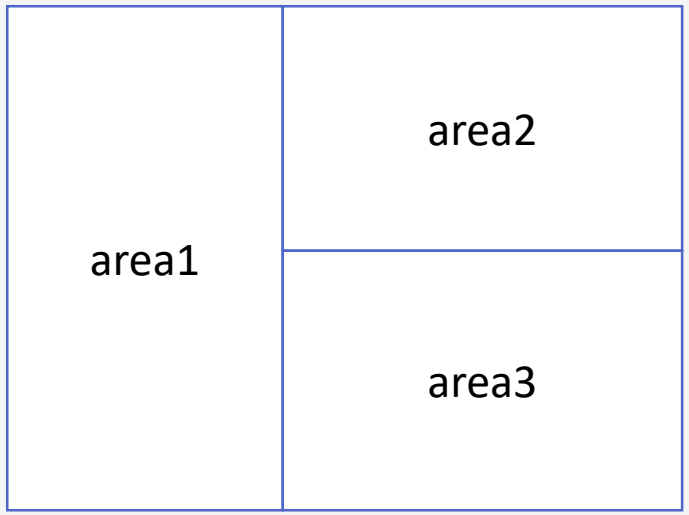
- 位置の指定
  - grid-row:1 / 3;のように記述
    - 行側に3行分高さを取るという意味になる
  - grid-column:2 / 3;
    - 列側に2番目の線から3番目の線の位置まで幅を取るという意味
- 隣り合わせの線を指定する場合は/を省略可能
  - grid-row:1 / 3;
  - grid-column:2;



CSSの記述

- エリアを利用する場合
  - コンテナ要素に対してgrid-template-areas:を記述

```
.container{
display:grid;
grid-template-areas:
“area1 area2”
“area1 area3”
```



  - ダブルクォーテーションで括り、行ごとに名前を指定していく
  - 各線ごとに半角スペースを開けて表現
  - 改行で次の行を示す

CSSの記述

- エリアに配置する場合
  - アイテムに対して

```
.box1{
grid-area: area1;
}
```

と指定すればOK

  - この場合は、エリア名に”　”は付けない

CSS Gridのプロパティ

- grid-auto-rowsとgrid-auto-columnはminmax()による指定が可能
  - 最低の幅（高さ）と最大幅（高さ）を指定可能
  - grid-auto-rows:minmax(200px,auto)にすると、最低高さが200pxで最大値は内部依存と指定できる
- セル間の間隔はcolumn-gap、row-gapで指定可能
  - もともとはgrid-column-gap,grid-row-gapだったので注意
  - 並記した方が無難

IE互換とか

- autoprefixerのバージョン5以降だとかなり上手く補ってくれるためCSS Gridを使う際は通しておくと便利
- IEを無視して、Edgeの古いのを無視するなら通さなくてもOK

CSS filter

- まだきちんとブラウザー実装されていないが、利用できるプロパティ
  - 画像などを加工することが可能
  - CSSでの加工なため、バリエーションを作るときに便利
  - url()でSVGファイルを指定すると、SVGのフィルターが利用可能（IEとEdge以外）

```
filter:適用するフィルター名（適用量）
```

CSS filter

- まだきちんとブラウザー実装されていないが、利用できるプロパティ
  - 画像などを加工することが可能
  - CSSでの加工なため、バリエーションを作るときに便利
  - url()でSVGファイルを指定すると、SVGのフィルターが利用可能（IEとEdge以外）

CSS filter

- blur：ぼかしを適用する。適用量では単位としてpxを付与した値を指定する
- brightness：明るさを増す。適用量で0以上の値を指定し、0または0%で真っ黒な状態に、1または100%でオリジナルと同じ明るさに、2または200%で2倍の明るさになる
- contrast：コントラストを増す。適用量で0以上の値を指定し、0または0%でグレーな状態に、1または100%でオリジナルと同じコントラストに、2または200%で2倍のコントラストを強調したものになる
- grayscale：グレーに近づけていく。適用量で0以上の値を指定し、0または0%でオリジナルと同じ状態に、1または100%で完全なグレーになる

CSS filter

- saturate：彩度を増す。適用量で0以上の値を指定し、0または0%で彩度の無い無彩色な状態に、1または100%でオリジナルと同じ彩度に、2または200%で2倍鮮やかになる
- sepia：セピアに近づけていく。適用量で0以上の値を指定し、0または0%でオリジナルと同じ状態に、1または100%で完全なセピア色になる
- hue-rotate：適用量で、単位degを付与した角度を与えることで、色相を変えることが可能。0degまたは360degでオリジナルと同じ色、180degで色相が反転する
- invert：階調を反転させる。適用量で0以上の値を指定し、0または0%でオリジナルと同じ状態に、1または100%で完全に階調を反転させる

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.scss;”
```

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がScss（サス）
  - 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
- 今はSCSSで記述するのが一般的

Sassとか

- 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Sass/SCSS
- 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

TaskRunner

- Web制作で決まった処理を自動化して行うための仕組み
- 拡張モジュールを入れることでさまざまな処理を自動化することが可能
  - ScssからCSSへの変換
  - CSSの圧縮、JSの圧縮
- gulp.jsが有名だが、npm scriptでも対応可能
  - どちらもNode.js上で動作するアプリ
  - 動作させるには行いたいことを記述したJSONファイルが必要、動作はCUI画面から行う（通称黒い画面）
  - 昔はGruntというのもあった

流れ

Sass,Lessで記述し保存

コンパイラ（トランスパイラ）で変換

CSSを生成

生成されたCSSを利用

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

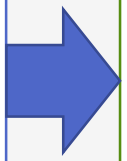
- SCSSでは複数のファイルに分け、読み込むことが可能
- アンダーバーをファイル名先頭に付与することで、パーシャルファイルとして利用できる
  - SCSSを保存してもCSSとしてトランスパイルされない
  - 単独で使用しないファイル（読み込まれて使用する）に使う
  - _base.scss
- 用途にあわせてファイルを分割することで、管理がしやすくなる
  - 要らないCSSファイルを読み込まない、なども可能
- 読み込む際は、@importの後にアンダーバーと拡張子を外して指定する

```
@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

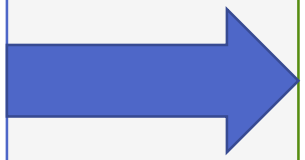
```
#main{
background:{
        color:#EEE;
        image:url(“back.jpg”);
        }
}
```

```
#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

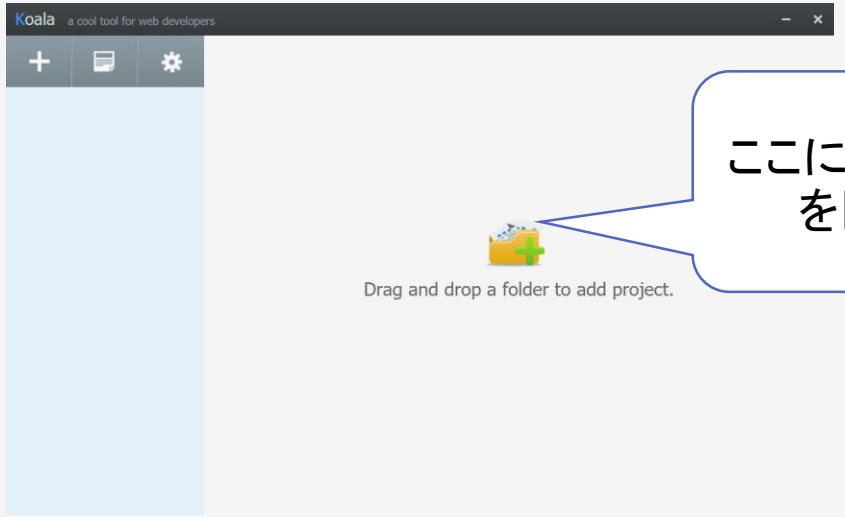
- @if,@elseを利用した分岐が可能
  - 条件に合わせて利用する関数を分けたり、書き出すCSSを変えることが可能

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

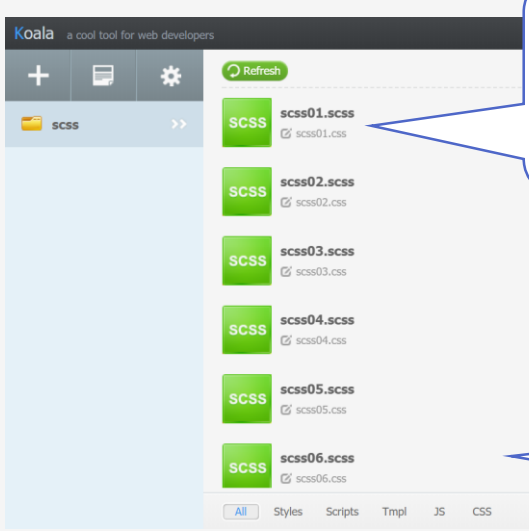
Scss

- 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

Koala

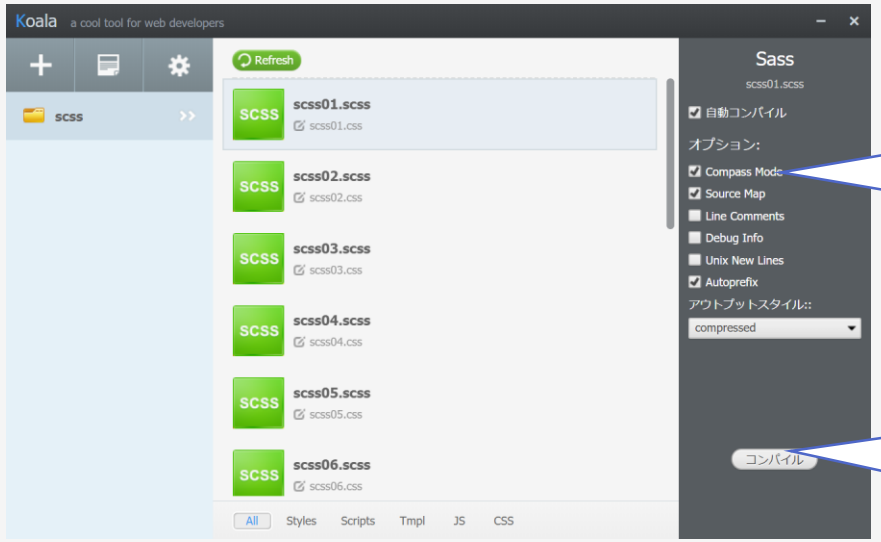


ここにフォルダーをドロップ



対象となるファイルが検出される

以降はファイルを編集して保存すれば自動的にコンパイルされる



変換時の設定を行う「環境設定」で設定したものが反映されている

コンパイルを押せば、変換される