HTML・CSSの基本

染谷

やること

- HTMLとは
- CSSとは
- レスポンシブWebデザインとは

1 HTMLとは

- HTMLとは
  - Hyper Text Markup Language
    - ハイパーテキストで文書を記述するための言語
    - 厳密にはマークアップ言語でありプログラミング言語とは異なる
    - ブラウザやクローラーロボットなどがファイルを解析し、何が書かれているかを理解できるように、「タグ」を使用してテキストや画像を囲む形で記述していくことが特徴
      - HTMLを記述することをマークアップと言う。
    - Webページはすべて基本的にHTMLで作られているといっても過言ではない
    - 現在の最新版はHTML5 (かつてはHTML4系やXHTMLなど)

なにが“ハイパー”なのか？

テキストや画像が表示されているだけでなく、異なる文書(ファイル)や画像を関連付け、即座に表示（遷移）することができる様になっていたり、音声や動画を再生したり、音声で読み上げを行うことも等可能。

通常のテキストファイルではこのようなことはできない。

1-2 HTMLの書式

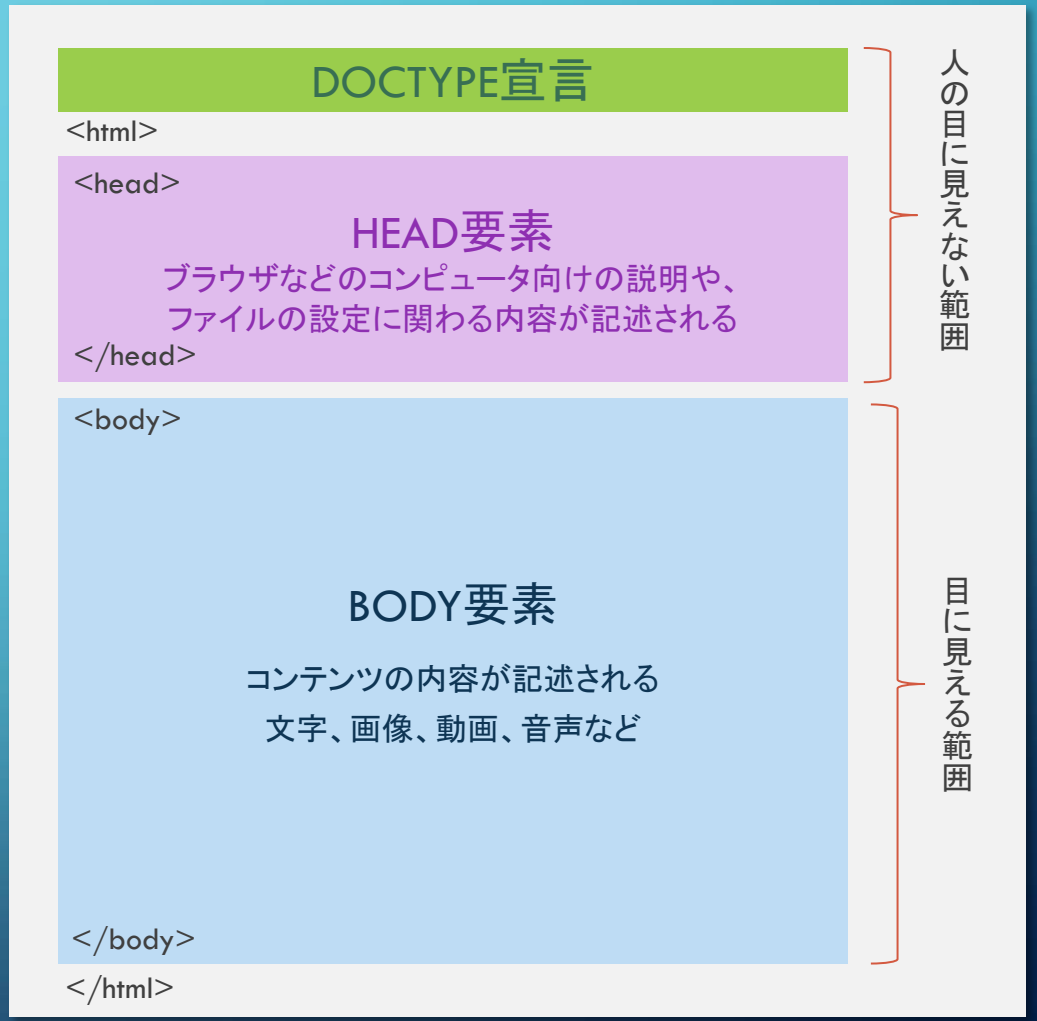
- HTMLには何が書かれているのか？

基本的に、3つの領域に分かれている

1. 「この文書はHTMLで書きます」と宣言する領域
   → DOCTYPE宣言

2. ブラウザなどのコンピュターに説明したり、ページを表示する準備をする領域
   → HEAD要素内

3. ブラウザで表示する領域
   → BODY要素内



DOCTYPE宣言

「以下に記述される文章はHTML5で書きます！」という宣言をするのがルール

書式

```
<!DOCTYPE HTML>
```
※小文字でもいい

HEAD内に記述される内容 (概要)

1.文字コードの設定

2.ページタイトル(ブラウザ・クローラー向け)

3.ページの説明(ブラウザ・クローラー向け)

4.検索用キーワードの設定(ブラウザ・クローラー向け)*

5.CSSやJavascriptなどのサイトのデザイン、動作に必要な外部ファイルの設定

その他にも、head内に記述できる要素は多岐にわたる。
製作者、運営者などの目的により記述内容は様々。

(例) SNS向けの情報設定等

HEAD内に記述される内容 (具体例)

1.文字コードの設定

```
<meta charset="UTF-8">
```
※HTML5はUTF-8推奨（特別な理由がない限りUTF-8を使う）

※HTMLを作成するときの文字コードとブラウザがHTMLをレンダリング(解析して表示する)ときの文字コードが違うと文字化けをおこす。

2.ページタイトル(ブラウザ・クローラー向け)

```
<title>au 占い</title>
```

3.ページの説明(ブラウザ・クローラー向け)

```
<meta name="description" content="占い総合サイト。">
```

4.検索用キーワードの設定(ブラウザ・クローラー向け)*

```
<meta name="keywords" content="au占い,12星座,九星気学,無料占い,手相,運勢,姓名判断,タロット">
```

5.CSSやJavascriptなどのサイトのデザイン、動作に必要な外部ファイルの設定

```
<link rel="stylesheet" href="../css/style.css">
<script src="../js/sidemenu.js"></script>
```

• BODY内に記述される内容

・Bodyタグ(<body>から</body>までの範囲)内に記述されるものは基本的にブラウザに表示されるコンテンツとなる。

・タグと呼ばれる目印のようなものでコンテンツの内容を囲った要素と呼ばれる単位を組み合わせて、コンテンツを構造化していく。

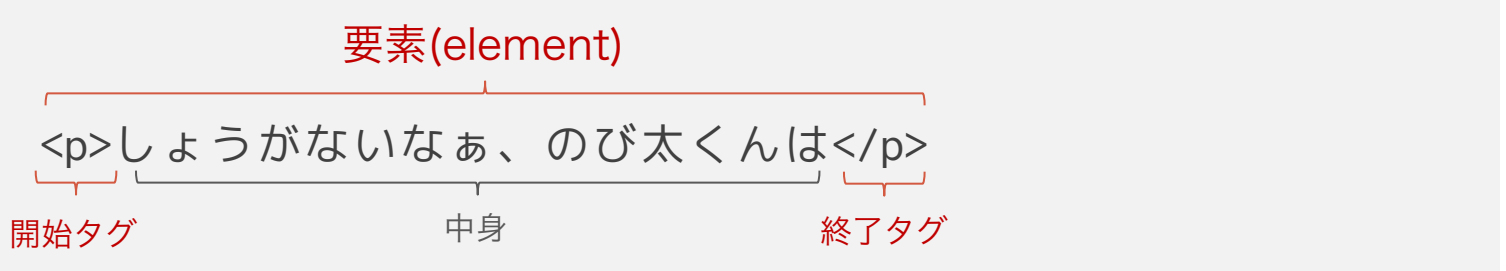
※構造化 => ブラウザが「どのような内容が記述されているか理解できるようにする」、レイアウトしやすいようにグループ化していく

• タグと要素

・タグは開始タグと終了タグの組み合わせて記述する。

・開始タグ + 中身 + 終了タグのまとまりを要素という。

例)



・ところが、中身と終了タグを持たない空要素というものもある。

```
<img src="../images/main_icon.png" alt="アイコン">
```

※他に、改行の
タグやhead要素内に記述される<link>タグや<meta>タグなどが代表的。

Block要素とinline要素

・要素にはブロック要素とインライン要素に分けられる。

・ブロック要素は縦に重なる

・ブロック要素は幅、高さの指定できる。

・インライン要素は横に並ぶ

という性質が特徴的である。

代表的なブロック要素を作るタグ

h1、h2…h6、p、div、ul、ol、tableなど

代表的なインライン要素を作るタグ

span、strong、a、img、buttonなど

インライン要素は、ブロック要素の中に差し込んで使うことが基本的。

ブロック要素のイメージ

```
<h1>
    見出しだよ
</h1>
<p>
    段落だよ1
</p>
<div>
    <p>
        段落だよ2
    </p>
    <p>
        段落だよ3
    </p>
</div>
```

入れ子にすることもでき、ブロック要素のなかでもブロック要素が重なる

インライン要素のイメージ

・インライン要素だけでのイメージ

テキスト1 テキスト1

・ブロック要素との関連イメージ

```
<p>春はあけぼの。やうやう白くなりゆく山際、少し明かりて、<strong>紫だちたる雲</strong>の細くたなびきたる。</p>
```

```
<div>
  こちらを<a href="../next.html">クリック</a>
</div>
```

• 使用頻度の高いタグ 1

1. pタグ (paragraph) => 段落

```
<p>テキストテキスト</p>
```

2.hタグ(heading) => 見出し

```
<h1>記事タイトル</h1>
```

h1～h6まであり、h1がもっとも重要性が高い(大見出し、小見出しなどの関係)。

※イメージ

```
<h1>サイトタイトル</h1>
  <h2>記事タイトル</h2>
    <h3>1章</h3>
    <h3>2章</h3>
    <h3>3章</h3>
```

3. divタグ (division:区分) => div自体に意味はない。

複数の要素をグループ化するために使用されることのが主な使い方。

```
<div>
  <h1>ドラえもんとは</h1>
  <p>22世紀に開発された猫型ロボットである。当初耳があったが、ネズミにかじられ喪失。ロボットなのに？
  </p>
</div>
<div>
 <h1>ドラミちゃん</h1>
  <p>ドラえもんの妹。ドラえもんの耳がないことを気遣って、耳はつけずにリボンにした。</p>
</div>
```

4. spanタグ => span自体に意味はない。

spanで囲んだ部分をインライン要素としてグループ化する。
テキストなどにcssでデザインを当てる範囲を限定するためなどに使用。

```
<p>今日だけ<span>100円引き</span>です！</p>
```

• 使用頻度の高いタグ 2

5. aタグ (anchor) => リンクを作成できる。

```
<a href="リンク先のURLなど">こちらをクリック</a>
```

6.imgタグ(image) => 画像を表示する ※空要素

```
<img src="画像までのパス(URLなど)" >
```

7. リスト関連のタグ

・ulタグ(unorderd list) => 箇条書きタイプのリスト

・olタグ(orderd list) => 順序のふられたリスト。ランキングとか。

・liタグ (list item) => リストの項目。ulタグかolタグの中で使う。

```
<p>買うものリスト</p>
<ul>
  <li>ボールペン</li>
  <li>ノート</li>
  <li>消しゴム</li>
</ul>
```

```
<p>好きな食べ物ランキング</p>
<ul>
  <li>カレー</li>
  <li>からあげ</li>
  <li>ラーメン</li>
</ul>
```

Ex パスPATHについて

パス(PATH)とは

リソース(ファイルや画像などのデータ)までの道筋を表すもの。
絶対パスと相対パスがある。

絶対パス

URLをすべて指定する。インターネット上に存在していれば、どこからでもリソースの場所までたどり着ける。

例)
https://cdn.vie.auone.jp/asset/ft-market/sp/img/howto/hwt_basic_02.png

相対パス

パスが記述されるファイルから見たリソースの位置を記述する。
＊事項参照

※ 相対パスについて

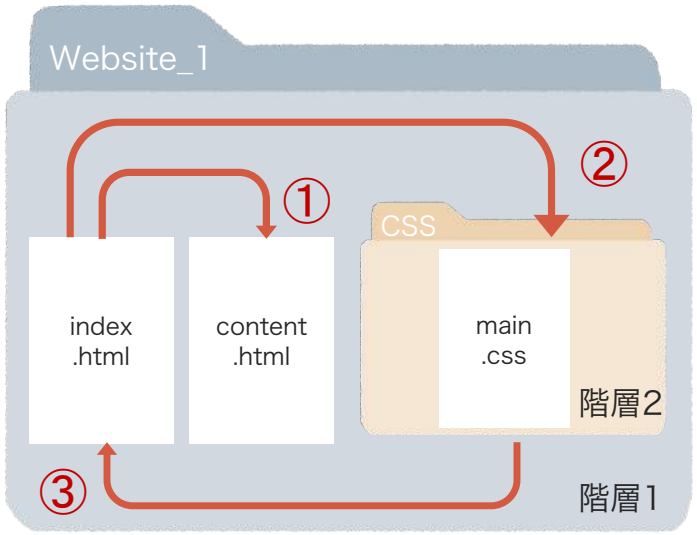
相対パスとはパスが記述されるファイルから見たリソースの位置を記述する。

そのため「https://〜」とは違い、「../img/logo.png」のようにURLの途中から記述されるような書式になる。

相対パスで使用する記述方法

- ./ => パスを記述するファイルと同じ場所(階層)にあるファイルやフォルダを指定する場合
- ../ => パスを記述するファイルからみて、1階層上にあるファイルやフォルダを指定する場合。

例) 相対パスとファイル、フォルダの位置関係



前提

- Website_1というフォルダの中にすべてのファイルとフォルダが入っている
- Website_1のなかには「index.html」「content.html」というファイルと、「CSS」というフォルダが入っている。
- CSSというの中に「main.css」というファイルが入っている。
- Website_1というフォルダの中を階層1、さらにCSSのフォルダの中を階層2とする。

相対パスでの表記

①index.htmlからcontent.htmlへの相対パス

```
./content.html
```

②index.htmlからmain.cssへの相対パス

```
./css/main.css
```

③main.cssからindex.htmlへの相対パス

```
../index.html
```

• 使用頻度の高いタグ 3

8. tableタグ関連 => 表を作成する

tableを構成する基本的なタグ、要素

1. table => 表の大外の枠。
2. tr => 表の1行(table row)
3. th => 行、または列の見出しとなるセル(table heading)
3. td => 表の1つ1つのセル(table data)

table関連のの要素の配置イメージ

```
<table>
  <tr> <th></th> <th></th> <th></th> <th></th> </tr>
  <tr> <td></td> <td></td> <td></td> <td></td> </tr>
  <tr> <td></td> <td></td> <td></td> <td></td> </tr>
  <tr> <td></td> <td></td> <td></td> <td></td> </tr>
</table>
```

1-3 セマンティクスコーディングのために重要なタグ

- セマンティクスコーディングとは?

  その要素(単体でもグループでも)がどのような意味や目的持っているのかをブラウザやクローラーなどに伝えるための記述方法。SEO等によい影響を与える。(と言われている。)

  ⇒ 一言でいうとHTMLに意味付をしましょうということ。

  セマンティクスを意識してコーディングすることで、構造的にも整理することができ、人間(コードを書く人)がHTMLファイルを見たときにも理解しやすくなる。

  ⇒ 「見た目ができていれば良い」というわけではないということ。

- 具体的にはどういうことか？

  ヘッダーやフッター、グローバルナビゲーション(グローバルメニュー)、主要コンテンツ領域などを明確にしてグループ化(セクショニング)していく。

  => HTML4.xまでは、主にdivで特定の領域にclass名(*後述)などをつけることでセクショニングをしていたが、HTML5以降ではセマンティックにコーディングしやすいように新たにタグが追加されている。

1-4 セマンティックのために追加されたタグ(一部)

1. **headerタグ** => ページ内でのヘッダーを構成する要素に対して使用する
2. **footerタグ** =>ページ内でのフッターを構成する要素に対してに使用する
3. **navタグ** => ナビゲーションメニューを構成する要素に使用
    - 原則はグローバルナビに対して使用する
4. **mainタグ** => ページ内でメインとなるコンテンツに対して使用する。
5. **sectionタグ** => コンテンツを構成する要素をグループ化する
6. **articleタグ** => ブログ記事やニュースなどページ内で完結している要素のグループに使用する。
7. **asideタグ** => ページのコンテンツとは関係のない要素をグループ化する。
    - 広告領域や「おすすめコンテンツ」などの補助的なものに対して使用する。

※sample参照

1-5 IDとCLASS

- HTMLの各要素(開始タグ)には属性を付与することができる。

  種類は様々あるが、すべての要素に対して使用できる属性(グローバル属性)のなかでも、もっとも基本的なものが「id」と「class」。

- Idとclassの基本的な使い方と違い
  - 基本的な使い方
    1. 要素にidやclassをつけることで、cssの指定を限定した要素に絞ることができる。
    2. JavascriptなどでHTMLを操作する、CSSの適用を変更する際の目印やトリガ(スイッチ)として使う事ができる。

  - 違い
    1. id名は各htmlファイルのなかで1回しか使用できない。(一意でなければならない。)
    2. class名は繰り返し使い回すことができる。
    3. id名、class名を使ってcssを指定する場合、idの方が優先して適応される。

    => CSSのコードの再利用や汎用性、CSSの適応の優先度の差による書き換えの手間などを考えると、
    CSSの指定にidを使うことは明確な理由がない限りは避けるというのが基本的な考え方。

2 CSSとは

- 2-1 CSSとは
  - Cascading Style Sheet
    - HTMLを装飾するための言語
    - 現在の最新版はCSS3(CSSレベル3)
    - かつてはHTML内にスタイルを指定していたが、HTMLの構造が複雑、かつ汚くなる、またCSSで表現するデザインの複雑化に対応するなどの目的のため、HTMLとCSSは分離させるのが基本の考え方

  - ユーザーエージェント(ブラウザ)、サイト制作者、ユーザー（開発ツール）の3つのCSSがあり、それらの効果を同時に反映させることができる。(カスケーディング)
    - 各ブラウザはデフォルトで独自のCSSを内蔵しており、Web制作者がCSSを作成していなくてもHTMLタグに対応した最低限のデザインを反映するようになっている。

- 2-2 リセットCSSとノーマライズCSS
  - 各ブラウザのデフォルトのCSSは独自のCSSをHTMLに反映していくにあたっては、基本的にじゃまになる。
    - サイトの作成時には、リセットCSSやノーマライズCSSを使用してデフォルトのCSSの効果を無効化、または制限することが基本。
      - リセットCSS
        - すべてのデフォルトCSSを無効化する。
      - ノーマライズCSS
        - 一部のデフォルトCSSを除いて、無効化する。（一部は活用する）
      - リセットCSS・ノーマライズCSSは、一番最初に読み込む、または最初に記述しなければならない。

        => 理由は以降の項目で。

- 2-3 CSSはどこに書くべきか
  - CSSを記述する場所は
    1. HTMLタグに直接記述。
    2. head要素内に<style>タグを使って記述。
    3. Head内から<link>を使って外部(htmlファイルとは別のところから)読み込む。
  - CSS適応の基本ルール
    1. インライン > head内 > 外部ファイル の順でCSSの指定が優先される。
    2. 同じ箇所へのCSSの指定はより後ろに記述した方が適応される。
    3. 同じ箇所への指定が重複した場合、より厳密に指定してある方が優先される。

    上記を考慮すると、予期しないスタイルの適用や、CSSの指定方法をむやみに複雑にしないために、外部ファイル(〇〇.css)に記述することを第一に考えるべきである。

- 2-4 CSSの基本文法
  - CSSファイルの書式
    - 拡張子は「.css」

      例) style.css、top.pageなど
    - 文字コードはUTF-8を使用し、ファイル先頭に記述

ファイルの書き出し例

```
@charset ‘UTF-8’;

* {
  margin: 0;
  padding: 0;
  border: 0;
}

html {
  font-size: 100%;
  height: 100%;
}

～ 省略 ～
```

- CSS指定方法の書式

基本の形

セレクタ

```
h1 {
    font-size: 24px;
}
```

プロパティ　値

1. セレクタ　=> どこの
2. プロパティ => 何を
3. 値　=> どのように

具体例

```
h1 {
  padding: 4px;
  font-size: 24px;
  border-bottom: 2px solid #ccc;
}
```

1行で書くことも可能ではある。

```
h1 {font-size: 24px;}
```

※読みにくくなるのでよほどの理由がない限りは改行する。

- セレクタについて

1. **タイプセレクタ**
   h1やpなどのタグ名を直接指定する。
2. **Idセレクタ**
   指定したid属性をもつ要素を指定する。
3. **クラスセレクタ**
   指定したclass属性をもつ要素を指定する。
4. **全照セレクタ**
   すべての要素に対して指定する。
5. **子孫セレクタ**
   要素の親子関係で要素を指定する。
6. **擬似クラス**
   特定の状態の要素に対して指定する。
7. **疑似要素**
   指定した要素に対して、cssで作成した要素を挿入する。

例

2.
```
#pageTop {
  プロパティ:値;
}
```

3.
```
.astro_rank {
  プロパティ:値;
}
```

4.
```
* {
  プロパティ:値;
}
```

5.
```
div p {
  プロパティ:値;
}
```

6.
```
.btn:hover {
  プロパティ:値;
}
```

```
li:last-child {
 プロパティ:値;
}
```

7.
```
.heaing::before {
  プロパティ:値;
}
```

※1.は前ページの例のような形式
詳しくは解説します。

- セレクタの詳細度

詳細度とは？

・CSSのセレクタの種類によって決められているポイント(点数)のようなもの。
この詳細度が高いほど、優先してCSSが適用されていく。
つまり、同一の要素の同一プロパティに対して複数の指定があった場合、詳細度が高い方CSSだけが適用される。

セレクタごとの詳細度

- idセレクタ ⇒ 1.0.0
- classセレクタ ⇒ 0.1.0
- タイプセレクタ ⇒ 0.0.1
- 全照セレクタ ⇒ 0.0.0

※セレクタのポイントはバージョン番号のように考える。
※詳細度を比較するときは、バージョンがより新しい方が詳細度が高いイメージ。

具体例

```
#top-page .side-menu {
  プロパティ: 値;
}
```

=> 詳細度 1.10.0

```
#top-page .side-menu p {
  プロパティ: 値;
}
```

=> 詳細度 1.10.1

```
div h2 span {
  プロパティ: 値;
}
```

=> 詳細度 0.0.3

```
#top-page .side-menu p {
  プロパティ: 値 !important;
}
```

あらゆるセレクタの詳細度を無視し、強制的に適用される。乱用は危険。

- よく使われるプロパティ

レイアウト系

block要素用

- width
- height
- display
- margin
- padding
- padding
- float
- position
- z-index
- box-sizing

inline要素用

- text-align
- vertical-align
- line-height

装飾系

text関連

- font-family
- font-size
- font-weight
- color
- text-decoration

box関連

- background
- border
- border-radius

ベンダープレフィックスについて

- ベンダープレフィックスとは？

CSSには将来公式で実装予定だが、草案段階(試験期間のようなも)のプロパティもある。
これらは、ベンダー(ブラウザの開発元)が独自にブラウザに先行実装している場合があり、そのようなプロパティを使う時にベンダープレフィックスを使用する。
また、公式に実装された最新のプロパティでも、ブラウザが最新のCSSに対応しきれていない場合に使用することもある。

- 主なベンダープレフィックス
  - -webkit- => GoogleChrome、Safari、Edge
  - -moz- => FireFox
  - -ms- => IE

3 レスポンシブWEBデザイン

- 3-1 レスポンシブWEBデザインとは
  - デバイスの幅に合わせてCSSを切り替える技術
    - Googleご推奨の技術
    - スマートフォン・タブレット・PCのブラウザ幅の変化に対応してCSSを切り替えて最適なデザインを反映させる。
    - それぞれのデバイスに合わせてHTMLを用意しなくて良いため、更新・メンテナンス性が高いなどのメリットがある。
    - 一方で、初期設計に時間がかかる、HTML自体の構造などが複雑化する、PC対応の場合ある程度デザインに妥協が必要な場合もあるなどのデメリットはある。

- 3-2 メディアクエリ
  - デバイスの幅に応じてCSSを切り替える技術。
  - CSSファイル内に記述する場合と、<link>タグ内に記述する場合などがある。
  - 切り替える幅のことをブレイクポイントという。

• メディアクエリの書式

```
@media screen and (max-width: 480px) {
     #contents #l_contents {
          width: auto;
          float: none;
     }
     #contents #r_contents {
          width: auto;
          float: none;
     }
}

@media screen and (min-width: 786px) {
     #contents #l_contents {
          width: auto;
          float: left;
     }
     #contents #r_contents {
          width: auto;
          float: right;
     }
}
```

ここでCSSが切り替わるデバイスの幅を指定している

`max-width: 480px` の場合、「最大で480pxまでの幅の場合(480px以下の場合)」、{}の中のスタイルが適用される。

`min-width: 786px` の場合、「最小で786pxまでの幅の場合(786px以上の場合)」、{}の中のスタイルが適用される。

※ブレイクポイントは複数設定可能。

- スマホ・タブレット・PCといった切り分けや、スマホ縦・スマホ横・タブレット縦・タブレット横・標準PC・大型PCと細かく分けることも可能。
- ブレイクポイントは、その時々でのシェアの高い端末のデバイス幅を参考に決められることが多い。

- 3-3 viewport (ビューポート)
  - デバイスのスクリーンの幅を設定するために記述する<meta>要素。
  - <head>内のできる限り上部に記述する。
  - この記述がないとデバイスのスクリーンの幅に合わせて、サイトが表示されてしまう。

    例)

    サイトの幅が980pxのサイトがあり、スクリーン幅が320pxのiPhoneでアクセスする。

    ↓

    320pxの領域に、980pxのサイトが縮小されて表示される。

    ↓

    見づらいし、mediaクエリを設定していても関係なくなってしまう。

    ↓

    Viewportを適切に書くことで、スクリーン幅が320pxであることを認識させ、mediaクエリなどを想定通りに適用させることができる。

```
<meta name="viewport" content="width=device-width,initial-scale=1">
```

ここで表示領域はデバイスの幅であると指定。

Be Careful