Webデザイン実習4B

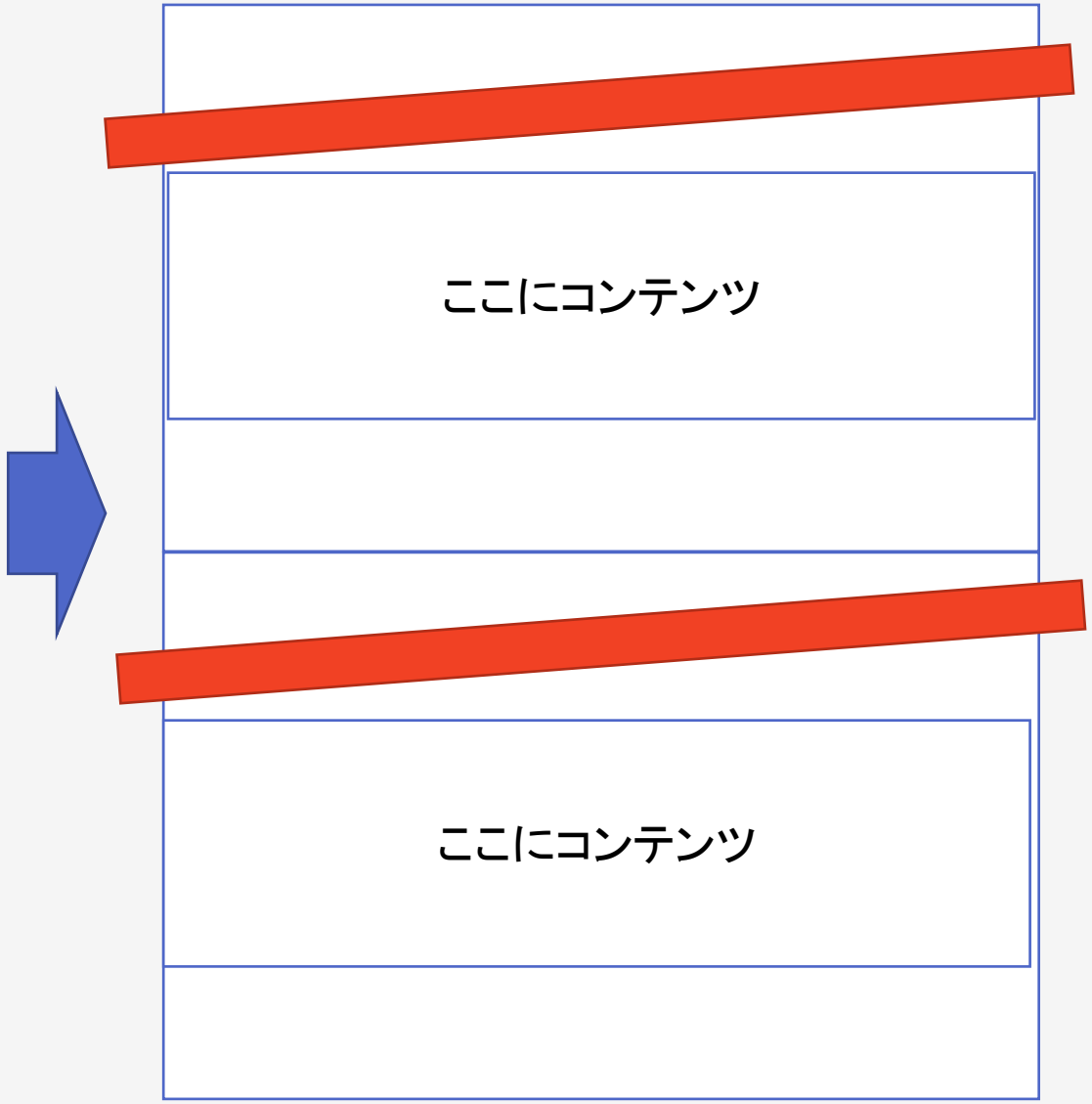
2019/10/29

Kazuma Sekiguchi
class@cieds.jp

斜め背景

```
<section class="contents">
  <div class="contents_inner">
    <!-- ここはコンテンツ -->
</div>
</section>
```

```
.contents{
position: relative;
overflow: hidden;
 background-color: #BE2527;
}
.contents:before {
content: '';
position: absolute;
top: 0;
left: 0;
width: 120%;
height: 80%;
margin: 2% -10% 0;
background: #fff;
transform-origin: right center;
transform: rotate(-2deg);
}
.contents_inner{
padding: 10% 0 0px;
position: relative;
}
```



jQueryを使わない要素の指定

- document.getElementsClassName()やdocument.querySelectorAll()を利用する
  - ほとんどの場合、複数の要素が同時に取得できるので、ループ処理が必要
  - ループで展開して、必要な処理を行う
- CSSによるスタイルを変えたい場合は、.styleで指定が可能だが、複数を変更するときはclassの適用、外しを利用する方が楽
- jQueryの場合、vue.jsなどのライブラリと相性が悪い
  - jQueryだけを使うとは限らない
  - 現在はVue.jsなどの比重が高くなってきている

classの適用

- classList.add（クラス名）でクラスの適用が可能

```
var pelem = document.querySelectorAll(‘p’);
        pelem.forEach(function(value){
        value.classList.add(‘text’);
});
```

- classList.remove（）でクラスを外す

```
var pelem = document.querySelectorAll(‘p’);
        pelem.forEach(function(value){
        value.classList.remove(‘text’);
});
```

classの適用

- classList.contains（クラス名）でクラスを保持しているかどうか確認可能

```
var pelem = document.querySelectorAll('p');
        pelem.forEach(function(value){
        if(value.classList.contains('text')){
                value.classList.add('hasclass');
        }
});
```

- classList.toggle（）でクラスを付け外しする

```
var pelem = document.querySelectorAll('p');
        pelem.forEach(function(value){
        value.classList.toggle('text');
});
```

他の指定方法

- 親要素の指定

```
element.parentNode;
```

- 子要素

```
element.firstElementChild; //最初の子要素
element.lastElementChild;  //最後の子要素
element.children; //子要素リスト
```

- 順番で指定

```
element.previousElementSibling; //1つ前の要素
element.nextElementSibling;//1つ後ろの要素
```

要素の作成

- 要素を作成することが可能

```
var div = document.createElement(‘div’);//divタグを作成
div.textContent = 'hoge';
```

- 要素の挿入

```
element.appendChild(div);//指定した要素の最後の子要素として追加
element.insertBefore(div, element.firstChild);//指定した要素の最初の子要素として追加
element.parentNode.insertBefore(div, element); //指定した要素の直前に追加
element.parentNode.insertBefore(div, element.nextSibling);//指定した要素の直後に追加
```

- 要素の削除

```
element.removeChild(child); //特定の子要素削除
element.parentNode.removeChild(element); //自分を削除
```

イベントの連動

- clickなどのイベントと要素を紐付ける
  - jQueryではonで可能
  - JSでもさまざまな方法で紐付けることが可能
  - ほとんどの場合は、addEventListenerの方が今は良い
  - 後から動的に作成された要素に対しても適用可能

```
element.addEventListener(“イベント名”,function(){
//実行する内容
});
```

  - イベントの紐付けを外すときは、removeEventListenerを利用する

ES6(ES2015),ES2017

- JSはもともとECMAScriptという規格に基づく
  - ブラウザーの進化に併せてJSのバージョンも変化
  - 現在はES6で記述するのが普通（今まではES5で記述）
  - IE11だけ一部動かない、またブラウザーによってはES2017などは動作しない
    - 新しいJSで記述した方が楽
    - 新しいJSで動かない場合は、下位バージョンに変換させる
    - 今のところは変換して利用した方が良い
  - Babelで変換
    - JSを下位バージョンに変換するための仕組み
    - Node上で動作する

Babel

- 通常は、Node上のNPMScriptやGulpから利用する
  - https://babeljs.io/repl
    でオンラインの利用をすることが可能
  - ES6やES2017のものをES5形式に変換可能
  - 一部ダメ

ES6での記述

- 変数が変更
  - varだけでなく、letとconstが登場
    - let：ブロックスコープを持つ変数宣言
    - const：後から置き換えが不可能な変数宣言。初期化時に値が必要

```
let a = “text”;
const b = “text”;
a = 123;
b = 123;//こちらはerrorになる
```

  - 基本的にvarではなくletを利用する
    - できる限りconstを使った方が大人数開発時にエラーが少なくなる

変数スコープ

- スコープは変数の有効の範囲のこと
- letはブロックスコープ
  - {}内だけ有効となる変数を作成可能
  - 通常のJSではvarによる変数の場合、function(){}内だけで有効にすることが可能
    - 関数スコープ

```
for(let i = 0; i < 1; i++){
    console.log(i);//0が表示される
}
console.log(i);//表示されない
```

```
function(){
    let a = “text”;
}
console.log(a);//表示されない
```

無名関数の記述

- イベントに紐付けたりするときに利用する

```
element.addEventListener(“click”,function(){
  //中身
});
```

- ES6から記述を省略可能

```
element.addEventListener(“click”,()=>{
  //中身
});
```

```
(a,b)=>{//引数を指定可能
  return a+b;
}
```

- {}も省略可能（但し1行で中身を記述する）

```
element.addEventListener(“click”,()=>//中身);
```

テンプレートリテラル

- JSで文字と変数を結合するときに「＋」の記号で結合することが必要

```
let a = “Programming”;
let b = “JavaScript”;
console.log(a+” of ”+b);//Programming of JavaScript
```

- ${}とバッククォートを使うことで、文字列内に変数を展開可能
  - 変数を${}に収める
  - 文字列全体を「`」で括る

```
let a = “Programming”;
let b = “JavaScript”;
console.log(`${a} of ${b}`);//Programming of JavaScript
```

for ofによるループ

- 配列を展開する際に利用するのがループ
  - for ofを利用することで楽に展開が可能

```
let arr = ["blue","red","green","yellow"];
for(let value of arr){
    console.log(value);
}
```

  - forEachでも展開可能

```
arr.forEach(function(value){
    console.log(value);
});
```

reduce

- 隣り合う2つの配列要素に対して左から右へ同時に関数を適用し、単一の値にするもの
  - 配列の合計値を算出する場合などに利用可能
  - 単一の文字列を生成することも可能

```
let result = arr.reduce((accumulator, currentValue)=>{
//accumulatorが配列の最初の値（合計の値）
//currentValueが配列の次の値
    return accumulator+currentValue;
  })
console.log(result);
```

map

- Mapデータ形式
  - ある種の配列
  - キーと値の組み合わせを保持することができるオブジェクト

```
let vmap = new Map([[“price", 1000]]);
```

  - newで作成可能
    - 作成と同時に初期化が可能。初期化時に[[]]とする必要あり
  - getで指定したキーの値を取得可能

```
let vmap = new Map([[“price", 1000]]);
console.log(vmap.get(“price”));//1000
```

map

- setで新しい要素を追加可能

```
vmap.set(“name”,”apple”);
```

- keysでインデックスを取得可能
  - valuesで値を取得可能

```
let keys = vmap.keys();
for(key of keys){
    console.log(key);
}
```

- hasでインデックスが存在するかどうか判別可能

```
if(vmap.has(“name”){
```