

第6回

JavaScriptから始める プログラミング2016

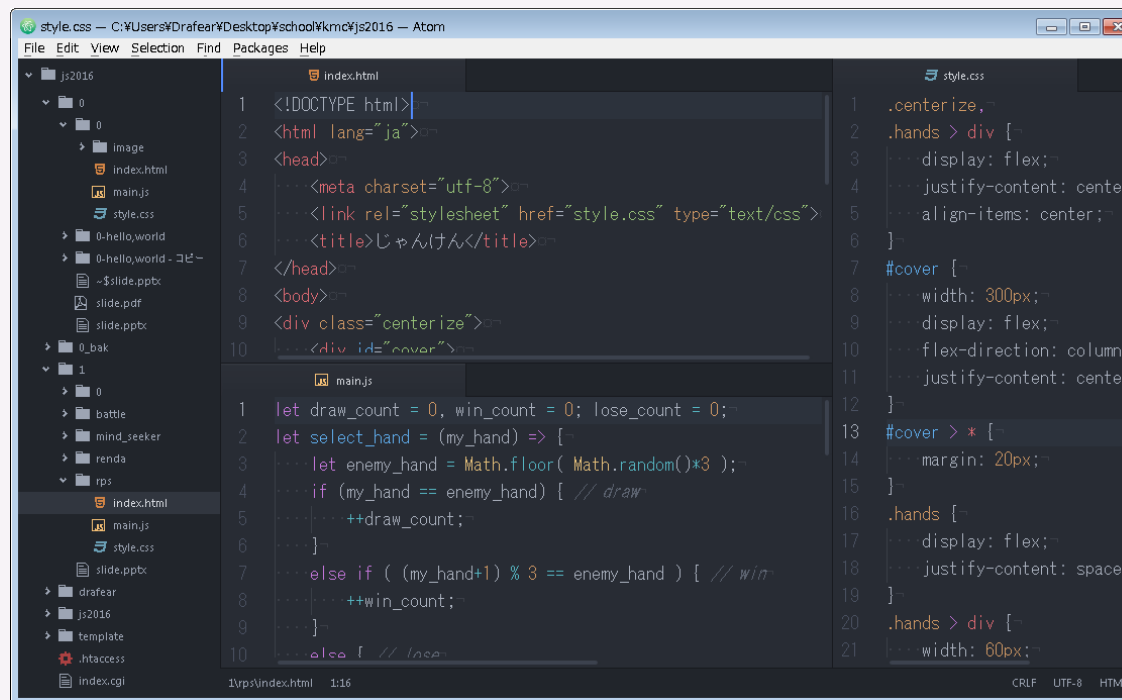
京都大学工学部情報学科
計算機科学コース3回
KMC2回 drafear



@drafear

この講座で使用するブラウザとエディタ

- Google Chrome 
 - <https://chrome.google.com>
- Atom 
 - <https://atom.io/>



重要度

- 今回はたくさんやるので, 重要度の項目を設けます
 - 重要度 ★★★★★ : 必須. 自然と直面するかもしれない.
 - 重要度 ★★★★★☆ : 知らないとその機能の実現が厳しい.
 - 重要度 ★★★☆☆ : 知ってたら便利. 綺麗なコードを書くのに重要かも.
 - 重要度 ★★☆☆☆ : たまに使うかもしれない.
 - 重要度 ★☆☆☆☆ : ほとんど使わない. おまけ. 興味があれば.
 - 重要度 ☆☆☆☆☆ : マニアック. ライブラリ作るなら必要かも.

知見を蓄える回

- 今回のことができなくても十分やっていきます
- なので, 「おもしろい!!」 と思った項目だけ拾って帰って下さい

今回の目標

- JavaScriptの表現の幅を広げる
- CSSのデザインの幅を広げる
- トップページを作る

本日の内容

- JavaScript
 - null, undefined
 - 型, typeof, 型変換, 異なる型同士の演算, NaN, Infinity
 - truthy, falsy, ショートサーキット評価
 - デフォルト仮引数, 残余仮引数
 - switch文
 - namespace(名前空間)
 - クロージャ
 - 分割代入

本日の内容

- HTML
 - <!DOCTYPE html>
 - html, head, body
 - meta(charsetだけ), title
 - css読み込み, js読み込み
 - p, main, a, i, b, strong, code, var, data
 - ul, ol, li, dl, dt, dd
 - div, span
 - article, nav, aside, header, footer, section, address, hgroup
 - h1, ..., h6
 - br, input, label

本日の内容

- CSS
 - box-shadow
 - text-shadow
 - 擬似クラス
 - hover, active, focus, link, visited, empty, checked, disabled, enabled, target
 - not
 - nth-child, nth-last-child, nth-of-type, nth-last-of-type
 - first-child, last-child, first-of-type, last-of-type
 - 擬似要素
 - before, after
 - 色んなデザインを試してみる

てんぷれ

- 以下から雛形などをダウンロードしてください
 - <https://github.com/kmc-jp/js2016>



1. 復習

復習

- 前回のスライドを見てサッと復習する



2. JavaScript

null と undefined

- undefined
 - 未定義であることを表す
 - let hogehoge; と初期値を与えずに書いたときが主.
 - hogehoge = undefined; とはしない.
- null
 - 空. 何もないこと.
 - 戻り値があるはずの関数の戻り値がないことを表すとき.

main.js

```
'use strict'  
let x;  
console.log(x); // undefined  
const obj = null;  
console.log(obj); // null
```

null と undefined

- 探して見つからなかったときに null
- 本来自然数を返す場合は null ではなく -1 で表すこともある

main.js

```
'use strict'
const search = (ary, val) => {
  for (let i = 0; i < ary.length; ++i) {
    if (ary[i] === val) return i;
  }
  return null;
};
const a = [1, 5, 4, 8, 10, 2];
console.log( search(a, 2) ); // 5
console.log( search(a, 100) ); // null
```

型

重要度 ★★★★★

- 真偽値 (Boolean)
 - true, false
- 数値 (Number)
 - 0, -1, 100.4
- 文字列 (String)
 - "hello!!"
- シンボル (Symbol)
 - まだ扱ってませんmm
- オブジェクト (Object)
 - [1, 2, 3], { x: 10, y: 3 },
new ClassName(), () => { ... }
- null
 - null
- 未定義
 - undefined

型判別

重要度 ★★☆☆☆

- `typeof variable`
 - `variable` の型を 文字列 として得る

main.js

```
'use strict'  
let a = "hogehoge";  
console.log( typeof a ); // "string"
```


型判別

重要度 ★★☆☆☆

- `typeof variable`
 - `variable` の型を 文字列 として得る
- 関数の動作を引数の型によって分けたいときなどに使う

main.js

```
'use strict'
const setXY = (elem, x, y) => {
  if (typeof elem === "string") {
    elem = document.getElementById(elem);
  }
  elem.style.left = `${x}px`;
  elem.style.top = `${y}px`;
}
setXY("btn", 100, 10);
```

型判別

重要度 ★★☆☆☆

型	戻り値
未定義	"undefined"
Null	"object"
真偽値	"boolean"
数値	"number"
文字列	"string"
シンボル	"symbol"
ホストオブジェクト	実装に委ねる
関数オブジェクト	"function"
E4X XML オブジェクト	"xml"
E4X XMLList オブジェクト	"xml"
他のオブジェクト	"object"

<https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Operators/typeof>

型変換

- 文字列 ⇔ 数値 の変換はしたい
- 型一覧 (再掲)
 - 真偽値 (Boolean)
 - 数値 (Number)
 - 文字列 (String)
 - シンボル (Symbol)
 - オブジェクト (Object)
 - null
 - undefined

型变换

重要度 ★★★★★☆

- 文字列 ⇔ 数值

String(num), Number(str)

数值 → 文字列

```
'use strict'  
let num = 100;  
let str = String(num);  
console.log(str); // 100  
console.log(typeof str); // string
```

文字列 → 数值

```
'use strict'  
let str = "10";  
let num = Number(str);  
console.log(num); // 10  
console.log(typeof num); // number
```

異なる型同士の演算

重要度 ★★☆☆☆

一例

```
'use strict'
let num = 100;
let str = "10";
// "10010" "string"
console.log(num + str, typeof (num + str));
// 90 "number"
console.log(num - str, typeof (num - str));
// 1000 "number"
console.log(num * str, typeof (num * str));
// 10 "number"
console.log(num / str, typeof (num / str));
// NaN "number" ...って何や？
console.log(num * { x: 10, y: 3 }, typeof (num * { x: 10, y: 3 }));
```

NaN

- NaN (Not a Number)
 - 数値型 ではあるが 数値ではない (哲学)
 - 数値型 との演算で正しく演算されなかったときなどに出てくる

main.js

```
'use strict'  
console.log( Math.sqrt(-1) ); // NaN  
console.log( Number("hoge") ); // NaN
```

NaN判定

- NaN かどうか判定するのに `===` や `!==` は使えない
 - `NaN === NaN` が `false` になる
 - `isNaN(num)` を使

main.js

```
'use strict'  
console.log( NaN === NaN ); // false  
console.log( isNaN(NaN) ); // true
```

Infinity

重要度 ★★★★★

- 無限大を表す

```
main.js
```

```
'use strict'  
console.log( 1/0 ); // Infinity
```


手軽な型変換方法

- 文字列 ⇔ 数値

`num + "", +str`

数値 → 文字列

```
'use strict'
let num = 100;
let str = num + "";
console.log(str); // 100
console.log(typeof str); // string
```

文字列 → 数値

```
'use strict'
let str = "10";
let num = +str;
console.log(num); // 10
console.log(typeof num); // number
```

truthy, falsy

- 論理値として見たときに true として見られるものを truthy
そうでないものを falsy という
- falsy でなければ truthy である
- truthy, falsy といった名前はどうでも良いが、
どういったものがtrueとして見られるかは重要

main.js

```
'use strict'
const check = (v) => {
  if (a) {
    console.log("truthy");
  }
  else {
    console.log("falsy");
  }
}
```

実行例

```
check(100); // truthy
check(0); // falsy
check([]); // truthy
check({}); // truthy
check("kmc"); // truthy
check(null); // falsy
```

truthy, falsy

- falsy であるもの
 - false
 - null
 - undefined
 - 0
 - NaN
 - ""
- これ以外は truthy
 - 0以外の数値や全てのオブジェクトなど

ショートサーキット評価

- $e_1 \ \&\& \ e_2$
 - e_1 を評価して falsy ならそれを返し
そうでなければ e_2 を評価して返す
 - まずは評価されるか否かに注目してみよう

main.js

```
'use strict'
const t = () => {
  console.log("T");
  return true;
}
const f = () => {
  console.log("F");
  return false;
}
```

&& の 動作例

```
> t() && t()
T
T
true
> f() && t()
F
false
> t() && f() && t()
T
F
false
```

ショートサーキット評価

- $e_1 \parallel e_2$
 - e_1 を評価して `truthy` ならそれを返し
そうでなければ e_2 を評価して返す

main.js

```
'use strict'
const t = () => {
  console.log("T");
  return true;
}
const f = () => {
  console.log("F");
  return false;
}
```

|| の動作例

```
> t() || f()
T
true
> f() || t()
F
T
true
> f() || t() || f()
F
T
true
```

ショートサーキット評価

- $e_1 \ \&\& \ e_2$ の e_1 や e_2 は必ずしも論理値である必要はない
- どういった場合に有用かを見てみましょう
 - $\&\&$ は前提を加えるときなど
 - $\|\$ はデフォルト値など

main.js

```
'use strict'  
class Counter {  
  constructor(initValue) {  
    initValue = initValue || 0;  
    this._cnt = initValue;  
  }  
  next() {  
    ++this._cnt;  
    return this._cnt;  
  }  
}
```

動作例

```
const cnt1 = new Counter();  
console.log( cnt1.next() ); // 1  
console.log( cnt1.next() ); // 2  
  
const cnt2 = new Counter(5);  
console.log( cnt2.next() ); // 6  
console.log( cnt2.next() ); // 7
```

ショートサーキット評価

重要度 ★★★★★

- どれかあったらそれを使おうというときにも
 - 新しい機能はブラウザによって名前が違ったりする
 - 注) そういったことは後でいい感じにできるので
我々がプログラムを書く際に気にしなくて良い

main.js

```
const requestAnimationFrame = window.requestAnimationFrame  
                             || window.mozRequestAnimationFrame  
                             || window.webkitRequestAnimationFrame;
```

ショートサーキット評価

重要度 ★★★★★

- && は前提を加えるときなど

main.js

```
const m = [  
  [1, 2, 6],  
  [0, 4, 8],  
  [9, 5, 3],  
];  
if (m.length > 0 && m[0].length > 0) {  
  console.log("not empty"); // 実行される  
}  
else {  
  console.log("empty");  
}
```


ショートサーキット評価

重要度 ★★★★★

- 順番を入れ替えるとエラーが起きるかもしれない

main.js

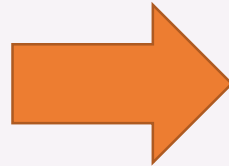
```
const m = [];  
if (m[0].length > 0 && m.length > 0) { // エラー  
  console.log("not empty");  
}  
else {  
  console.log("empty");  
}
```

ショートサーキット評価

重要度 ★★★★★

- 成功したら順に実行する場合にも
 - step1, step2, step3 は true(成功) か false(失敗) を返す

```
if ( step1() ) {  
  if ( step2() ) {  
    step3();  
  }  
}
```



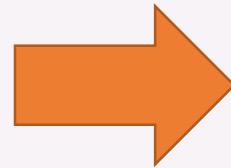
```
step1() && step2() && step3();
```

デフォルト仮引数

- 引数が渡されなかった場合のデフォルト値を指定できる
- 指定しなかった場合は undefined
- (a, b, c = c0) => { ... }

main.js (先ほどの例)

```
'use strict'
class Counter {
  constructor(initValue) {
    initValue = initValue || 0;
    this._cnt = initValue;
  }
  next() {
    ++this._cnt;
    return this._cnt;
  }
}
```



main.js

```
'use strict'
class Counter {
  constructor(initValue = 0) {
    this._cnt = initValue;
  }
  next() {
    ++this._cnt;
    return this._cnt;
  }
}
```

残余仮引数

重要度 ★★★★★☆

- 可変長引数が実現できる
- (a, b, ...args) => { ... }
 - 第三引数以降に与えられた引数を配列として args で受け取る

main.js

```
'use strict'  
const sum = (...args) => {  
  let res = 0;  
  for (const val of args) {  
    res += val;  
  }  
  return res;  
};  
console.log( sum(1, 5, 8, 2) ); // 16
```

残余仮引数

重要度 ★★★★★☆

- 第一引数で与えられた配列 `ary` に
第二引数以降で与えられた要素を順に追加していく関数の例

main.js

```
'use strict'  
const pushToArray = (ary, ...args) => {  
  for (const val of args) {  
    ary.push(val);  
  }  
};  
const a = [10, 20, 30];  
pushToArray(a, 1, 5, 4);  
console.log(a); // [10, 20, 30, 1, 5, 4]
```

1. 任意長引数のminをつくってみよう
 - `min(a0, a1, ..., aN)` : `a0, a1, ..., aN` 中の最小値を返す
 - 何も与えられなかったら `Infinity` を返す
2. 足し算だけをする計算機を作ってみよう
 - `input` タグは初登場ですが, `elem.value` で値が取れます
 - 「=」 ボタンで `left + right` を計算して `result` に表示する

index.html

```
<input type="text" id="left" value="0">  
+  
<input type="text" id="right" value="0">  
<button id="btnCalc"> = </button>  
<span id="result">0</span>
```

演習

1. 任意長引数のminをつくってみよう

main.js

```
const min = (...args) => {  
  let res = Infinity;  
  for (const val of args) {  
    res = Math.min(res, val);  
  }  
  return res;  
}
```

演習

- 実は, `Math.min`, `Math.max` もそうになっている(3つ以上渡せる)

演習

2. 足し算だけをする計算機を作ってみよう

index.html

```
<input type="text" id="left" value="0">  
+  
<input type="text" id="right" value="0">  
<button id="btnCalc"> = </button>  
<span id="result">0</span>
```

main.js (正しくない例)

```
const calc = () => {  
  const left = document.getElementById("left").value;  
  const right = document.getElementById("right").value;  
  document.getElementById("result").textContent = left + right;  
};  
document.getElementById("btnCalc").addEventListener("click", (e) => {  
  calc();  
});
```

演習

- 前スライドのコードを動かすと...

+ = 1215

- どうして上手くうごかないのか？

演習

elem.value は文字列だから！！ 分かった人はすごい！！
つまり↓のようにすればおk！

main.js (正しい例)

```
const calc = () => {  
  const left = +document.getElementById("left").value;  
  const right = +document.getElementById("right").value;  
  document.getElementById("result").textContent = left + right;  
};  
document.getElementById("btnCalc").addEventListener("click", (e) => {  
  calc();  
});
```

+

=

27

switch文

- `switch (exp) {`
 `case c1: ...`
 `case c2: ...`
 `case c3: ...`
 `default: ...`
}
- `exp === c1` のとき `c1` に
 そうでなくて `exp === c2` のときは `c2`
 そうでなくて `exp === c3` のときは `c3`
 そうでない場合は `default` にとぶ.
- `break` があったら `switch` を抜ける

main.js

```
const func = (str) => {  
  switch (str) {  
    case "hoge":  
      console.log("a");  
      break;  
    case "fuga":  
      console.log("b");  
    case "hoge":  
    case "pi"+"yo":  
      console.log("c");  
      break;  
    default:  
      console.log("d");  
  }  
};
```

動作例

```
> func("hoge")  
a  
> func("fuga")  
b  
c  
> func("piyo")  
c  
> func("hage")  
d
```

namespace (名前空間)

- グローバルに変数をとると被るかもしれない
- そして普段エラーが出るところで出なかったりするかもしれない

main.js

```
let val = 10;  
class Class1 {  
  constructor(v0) {  
    this.val = v0;  
  }  
  getValue() {  
    return val;  
  }  
}  
const c1 = new Class1(30);  
console.log( c1.getValue() ); // 10
```

namespace (名前空間)

- そこで namespace (ただのオブジェクト) を使う

main.js

```
'use strict'
const Global = {};
Global.val = 10;
class Class1 {
  constructor(v0) {
    this.val = v0;
  }
  getValue() {
    return val; // エラー
  }
}
const c1 = new Class1(30);
console.log( c1.getValue() );
```

クロージャ

重要度 ★★★★★

- 変数はその {} 内からなら参照できる
- 変数を完全に隠蔽できる

main.js

```
'use strict'
const makeCounter = () => {
  let cnt = 0;
  return () => {
    const res = cnt;
    ++cnt;
    return res;
  };
};
```

動作例

```
const counter1 = makeCounter();
console.log( counter1() ); // 0
console.log( counter1() ); // 1
console.log( counter1() ); // 2
console.log( counter1() ); // 3

const counter2 = makeCounter();
console.log( counter2() ); // 0
console.log( counter2() ); // 1
```

分割代入

重要度 ★★★★★☆

- 配列の要素を分割して変数に代入できる

main.js

```
'use strict'  
const ary = [100, 200, 300];  
const [v1, v2, v3] = ary;  
console.log(v1, v2, v3); // 100 200 300
```


分割代入

重要度 ★★★★★☆

- 1行で swap(入れ替え) できる！！

main.js

```
'use strict'  
let a = 10;  
let b = 20;  
[a, b] = [b, a];  
console.log(a, b); // 20 10
```

分割代入

重要度 ★★★★★☆

- 様々な分割代入

残りを配列として

```
let [a, b, ...c] = [1, 2, 3, 4, 5];  
console.log(a, b, c); // 20 10 [3, 4, 5]
```

デフォルト値の設定

```
let [a = 10, b = 20, c = 30] = [5];  
console.log(a, b, c); // 5 20 30
```

不要な部分を飛ばす

```
let [a, , b] = [5, 10, 15];  
console.log(a, b); // 5 15
```

分割代入

重要度 ★★★★★☆

- オブジェクトの分割代入

基本

```
let { x: x, y: y } = { x: 10, y: 2 };  
console.log(x, y); // 10 2
```

デフォルト値

```
let { x: x=0, y: y, z: z=0 } = { x: 10, y: 2 };  
console.log(x, y, z); // 10 2 0
```

分割代入

重要度 ★★★★★☆

- キーと変数名が一致する場合は `x: x` を省略して `x` と書ける
 - 順番は関係ない

省略記法

```
let {x, y} = { x: 10, y: 2 };  
console.log(x, y); // 10 2
```

分割代入

重要度 ★★★★★☆

- オブジェクトのオブジェクトなども
 - もちろん配列が絡んでいても

sugoi.js

```
let human = {  
  name: "drafear",  
  family: {  
    mother: "base64",  
    father: "asi1024",  
    sister: "wass80",  
  },  
  age: 21,  
};  
let { name, age, family: { sister } } = human;  
console.log(name, age, sister); // drafear 21 wass80
```

分割代入

重要度 ★★★★★☆

- 分割代入 と for-of

sugoi.js

```
'use strict'
let circles = [
  { x: 10, y: 3, r: 20 },
  { x: 20, y: 4, r: 11 },
  { x: 30, y: 5, r: 2 },
];
// 10 3 20
// 20 4 11
// 30 5 2
for (const { x, y, r } of circles) {
  console.log(x, y, r);
}
```



3. HTML

divとはお別れ！...なんてできない

これまでdivを多用してきましたが、
今回は意味に合ったタグを選んでいきます！！

！！注意！！

装飾したりレイアウトを組んだりするためだけにHTML要素の
デフォルトの見栄えを利用しないで下さい。

例えば表を表す table タグでレイアウトを組むなど。

(人間がソースを読む時の)可読性が落ちるだけでなく、
Googleなどのロボットが誤った解釈をしてしまいます！

装飾・レイアウトは CSS で！！

...の前に

その前に今までテンプレとして使ってきたものを説明します！

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css" type="text/css">
  <title>タイトル</title>
</head>
<body>
<script src="main.js"></script>
</body>
</html>
```

<!DOCTYPE html>

タグではありません. html文書であることを表します.

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css" type="text/css">
  <title>タイトル</title>
</head>
<body>
<script src="main.js"></script>
</body>
</html>
```

html, head, body

<html>	基点(ルート)となる要素. 他の全ての要素はこれの子孫としなければならない.
<head>	ドキュメント全般の情報(タイトルなど)を内包し, ユーザに見せない部分.
<body>	文書コンテンツを表す部分. ユーザに見せる部分.

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css" type="text/css">
  <title>タイトル</title>
</head>
<body>
  <script src="main.js"></script>
</body>
</html>
```

<html lang="ja">

lang="ja"

要素の内容が日本語であることを示す

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css" type="text/css">
  <title>タイトル</title>
</head>
<body>
<script src="main.js"></script>
</body>
</html>
```

<meta charset="utf-8">

文字コードが utf-8 であることを示す
全角文字を表す方式のこと.

他にも shift-jis や euc-jp などがある.

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css" type="text/css">
  <title>タイトル</title>
</head>
<body>
<script src="main.js"></script>
</body>
</html>
```

```
<link rel="stylesheet" href="style.css" type="text/css">
```

style.css を読み込んでいる

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css" type="text/css">
  <title>タイトル</title>
</head>
<body>
<script src="main.js"></script>
</body>
</html>
```

<title>タイトル</title>

ページのタイトル.

ブラウザで開いた時のタイトルバーの文字列がこれになる.

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css" type="text/css">
  <title>タイトル</title>
</head>
<body>
<script src="main.js"></script>
</body>
</html>
```

```
<script src="main.js"></script>
```

main.js を読み込んでいる。
最後に読み込むことで、各要素にアクセスできている。

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css" type="text/css">
  <title>タイトル</title>
</head>
<body>
<script src="main.js"></script>
</body>
</html>
```


というわけで

この構造が分かってもらえたかな？
次は div に替わるものを紹介していくよ.

```
<!DOCTYPE html>  
<html>  
<head>...</head>  
<body>...</body>  
</html>
```

テキストコンテンツ要素

- pタグ
 - 段落を表す
 - display: block

```
<p>内容</p>
```

テキストコンテンツ要素

- mainタグ
 - body要素内の主な内容を表す
 - body内に1つ

テキストコンテンツ要素

- ul
 - 順不同リスト
- ol
 - 順序付きリスト
- li
 - ul または ol の直下の要素
 - 逆に ul または ol の直下に li 以外を置いてはいけない

※見た目はCSSでいじれるので
飽くまでも意味だけを捉えて下さい。

```
<ul>  
  <li>要素1</li>  
  <li>要素2</li>  
  <li>要素3</li>  
  <li>要素4</li>  
</ul>
```

- 要素1
- 要素2
- 要素3
- 要素4

```
<ol>  
  <li>要素1</li>  
  <li>要素2</li>  
  <li>要素3</li>  
  <li>要素4</li>  
</ol>
```

1. 要素1
2. 要素2
3. 要素3
4. 要素4

テキストコンテンツ要素

- dl
 - 説明リスト要素
 - 用語と説明のセット
キーと値のセットにも使える
 - 内容は0個以上の<dt>要素と
それに続く1個以上の<dd>要素
- dt
 - 用語
- dd
 - 説明

```
<dl>  
  <dt>Name</dt>  
  <dd>drafear</dd>  
  <dt>Age</dt>  
  <dd>21</dd>  
  <dt>Birthplace</dt>  
  <dd>Japan</dd>  
</dl>
```

Name	drafear
Age	21
Birthplace	Japan

インラインテキストセマンティクス要素

- br
 - 改行 (ただし段落が変わる場合はpタグを用いましょう)
 - imgタグと同じく閉じタグがないタイプ

インラインテキストセマンティクス要素

- `i`
 - イタリック
 - 何らかの理由で他のテキストと意味を切り離したいとき
- `b`
 - 特別な重要性や関連性を伴わないで通常と文体の異なるテキスト
 - 要約中のキーワードなど
- `strong`
 - 重要なテキスト

インラインテキストセマンティクス要素

- data
 - 内容を機械可読な形式で示す
 - 例えば 商品名 である内容を 商品コード として示す

```
<p>New Products</p>
<ul>
  <li><data value="3967381398">Mini Ketchup</data></li>
  <li><data value="3967381399">Jumbo Ketchup</data></li>
  <li><data value="3967381400">Mega Jumbo Ketchup</data></li>
</ul>
```

<https://developer.mozilla.org/ja/docs/Web/HTML/Element/data>

インラインテキストセマンティクス要素

- code
 - プログラムのコードであることを示す
- var
 - プログラムや数式中の変数であることを示す

改めて div / span

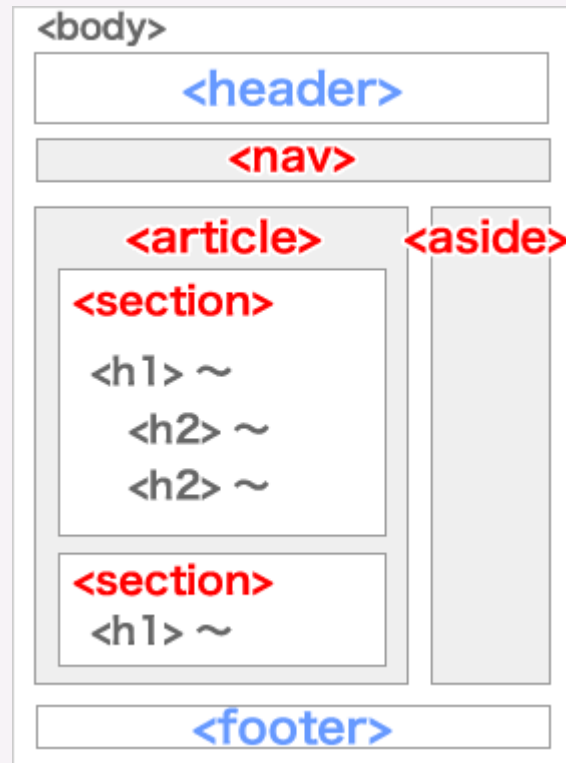
- div
 - ブロックレベル要素の汎用コンテナ
 - 特に意味は無い
- span
 - インラインレベル要素の汎用コンテナ
 - 特に意味は無い

コンテンツセクショニング要素

<article>	自己完結した独立した要素を表し、それ単体で1つのコンテンツとなる。
<nav>	メニューなど、他のページやページ内の他の部分へのリンクであるナビゲーション要素のうち主要なもの。
<aside>	サイドバーなど、メインコンテンツと切り離せるセクション。
<header>	イントロダクションやナビゲーション等のグループを表す。 1つ以上の見出しやロゴ、検索フォーム、 セクションのヘッダ部分を含むことができる。
<footer>	<article>, <aside>, <nav>, <section>, <blockquote>, <body>, <details>, <fieldset>, <figure>, <td> の中で最も近い祖先要素のフッターを表す
<section>	見出しを付けられるセクション
<address>	<article>, <body> の中で最も近い祖先要素の著作者の連絡先情報を表す
<hgroup>	セクションのヘッダを表す
<h1>, ..., <h6>	見出しを表す。h1を最上位, h6を最下位のレベルとする。

コンテンツセクショニング要素

- ... わかりましたか？？ 難しいと思うので, 例を挙げます.



<http://honttoni.blog74.fc2.com/blog-entry-61.html>

コンテンツセクショニング要素

index.html

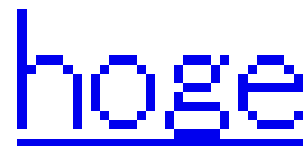
```
<header>
  <h1>タイトル</h1>
  <h2>サブタイトル</h2>
  <p>内容</p>
</header>
<nav>
  <h1>ナビゲーションタイトル</h1>
  <ul>
    <li><a>メニュー1</a></li>
    <li><a>メニュー2</a></li>
    <li><a>メニュー3</a></li>
  </ul>
</nav>
<article>
  <section>
    <h1>セクションタイトル</h1>
```

```
    <p>内容</p>
  </section>
</article>
<aside>
  <section>
    <h1>タイトル</h1>
    <ul>
      <li><a>サブメニュー1</a></li>
      <li><a>サブメニュー2</a></li>
      <li><a>サブメニュー3</a></li>
    </ul>
  </section>
</aside>
<footer>
  <p>内容</p>
  <address>著作者の連絡先</address>
</footer>
```

そしてついに登場

- a
 - リンクを張る

```
<a href="./hoge.html">hoge</a>
```



最後に

- input

未入力時に薄く
表示される文字列

```
<dl>
  <dt>名前</dt>
  <dd><input type="text" id="tbName" placeholder="京大 次郎"></dd>
  <dt>性別</dt>
  <dd>
    <input type="radio" id="rbSexMale" name="sex"> 男
    <input type="radio" id="rbSexFemale" name="sex"> 女
  </dd>
  <dt>趣味</dt>
  <dd>
    <input type="checkbox" id="cbHobbyProgramming1" name="hobby" checked> プログラミング
    <input type="checkbox" id="cbHobbyProgramming2" name="hobby"> プログラミング
    <input type="checkbox" id="cbHobbyProgramming3" name="hobby" disabled> プログラミング
  </dd>
</dl>
```

デフォでチェック入り

無効 (チェックを入れたり
外したりできない)

最後に

- 下のような構造にするとお良い
 - label タグで囲むと, テキストをクリックしてもチェックが入る
 - radio button だけでなく checkbox も同じ

```
<ul>  
  <li><label><input type="radio" id="rbSexMale" name="sex"> 男</label></li>  
  <li><label><input type="radio" id="rbSexFemale" name="sex"> 女</label></li>  
</ul>
```


演習

トップページがなくて寂しいのでトップページを作ろう！！

ついでにコンテンツ増やしたり改良・拡張したり？
今更かもしれないが、自己紹介ページ作っても良いかも？

下のディレクトリと並べて index.html を設置！

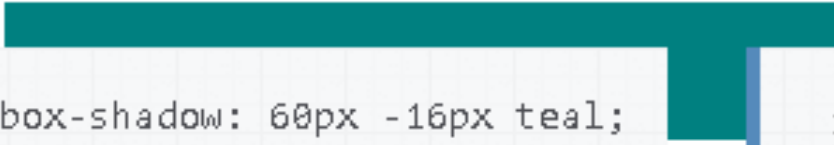
<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 Parent Directory		-	
 Jyankenn/	15-May-2016 18:00	-	
 template/	08-May-2016 16:05	-	



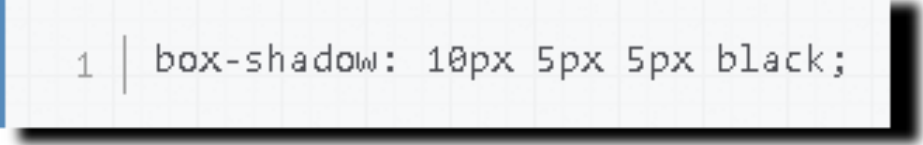
4. CSS

box-shadow

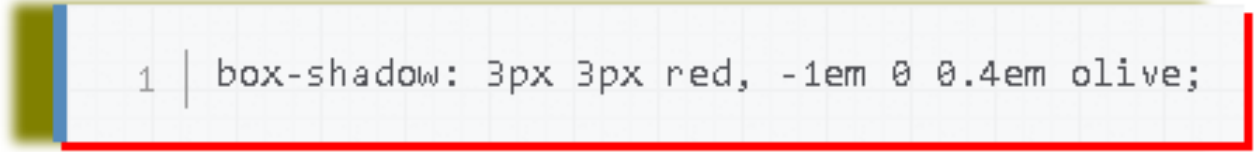
- box-shadow [inset] *dx dy* ぼかし距離 広がり距離 *color*



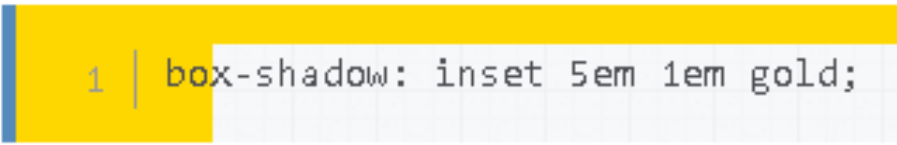
```
1 | box-shadow: 60px -16px teal;
```




```
1 | box-shadow: 10px 5px 5px black;
```



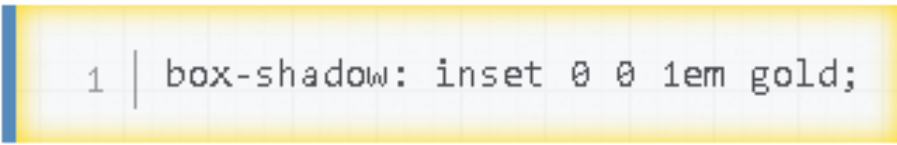
```
1 | box-shadow: 3px 3px red, -1em 0 0.4em olive;
```



```
1 | box-shadow: inset 5em 1em gold;
```



```
1 | box-shadow: 0 0 1em gold;
```



```
1 | box-shadow: inset 0 0 1em gold;
```

box-shadow

- box-shadow [inset] *dx dy* [(ぼかし距離) [広がり距離] [color]
 - inset: 内側に影
 - dx, dy: x方向, y方向のずれ(offset)

```
1 | box-shadow: 60px -16px teal;
```

```
1 | box-shadow: 10px 5px 5px black;
```

```
1 | box-shadow: 3px 3px red, -1em 0 0.4em olive;
```

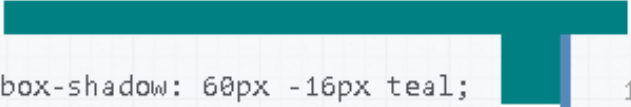
```
1 | box-shadow: inset 5em 1em gold;
```

```
1 | box-shadow: 0 0 1em gold;
```


```
1 | box-shadow: inset 0 0 1em gold;
```

box-shadow

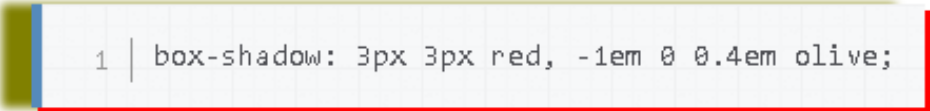
- `box-shadow [inset] dx dy [ぼかし距離] [広がり距離] [color]`
 - ぼかし距離: グラデーションをかけていく距離 (default: 0)
 - 広がり距離: 同じ色のまま広がっていく距離 (default: 0)
 - color: 色 (default: ブラウザによって異なる)



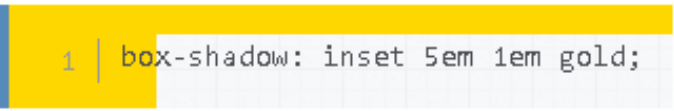
```
1 | box-shadow: 60px -16px teal;
```



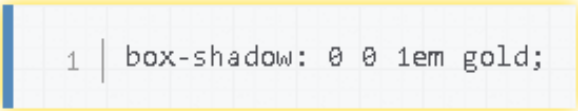
```
1 | box-shadow: 10px 5px 5px black;
```



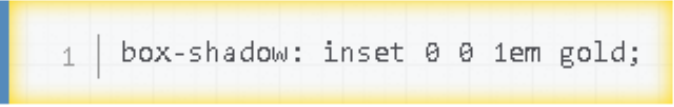
```
1 | box-shadow: 3px 3px red, -1em 0 0.4em olive;
```



```
1 | box-shadow: inset 5em 1em gold;
```



```
1 | box-shadow: 0 0 1em gold;
```



```
1 | box-shadow: inset 0 0 1em gold;
```

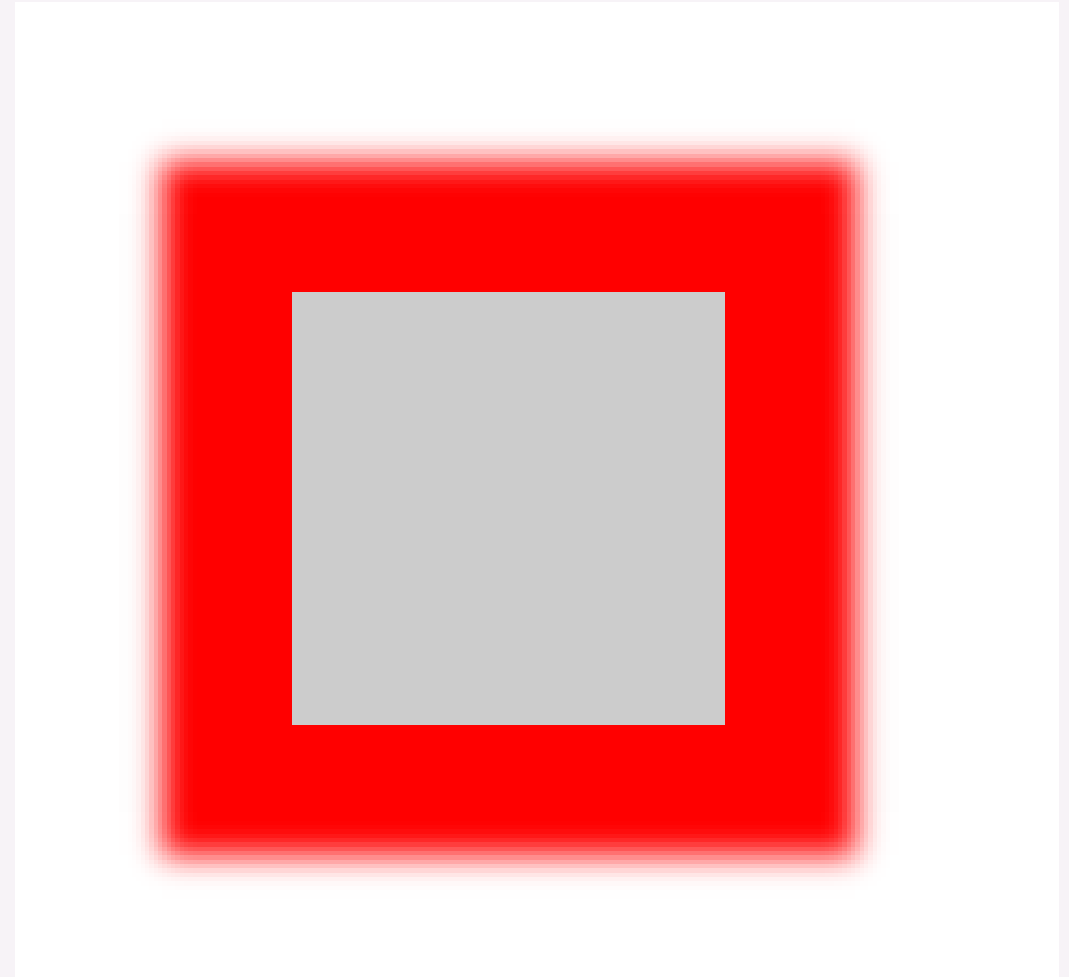
box-shadow

index.html

```
<div class="box"></div>
```

style.css

```
.box {  
  width: 100px; height: 100px;  
  background-color: #ccc;  
  box-shadow: 0 0 10px 30px red;  
}
```



box-shadow

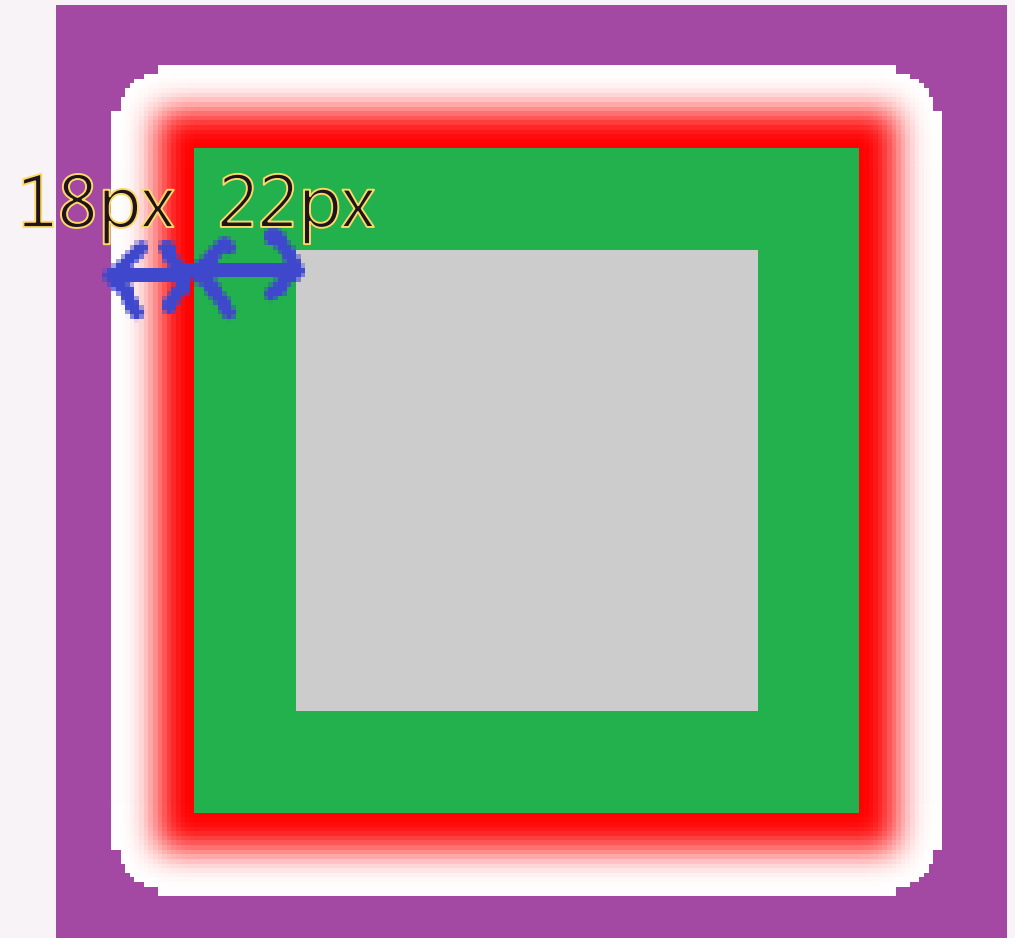
- 全く同じ色のところをペイントで塗りつぶして測ってみた

index.html

```
<div class="box"></div>
```

style.css

```
.box {  
  width: 100px; height: 100px;  
  background-color: #ccc;  
  box-shadow: 0 0 10px 30px red;  
}
```



text-shadow

- text-shadow *dx dy* [(ぼかし半径)] [*color*]

```
1 .white-with-blue-shadow {  
2   text-shadow: 1px 1px 2px black, 0 0 1em blue, 0 0 0.2em blue;  
3   color: white;  
4   font: 1.5em Georgia, "Bitstream Charter", "URW Bookman L", "Century Schoolbook L", serif;  
5 }
```

```
1 <p class="white-with-blue-shadow">  
2   Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudan  
3   veritatis et quasi architecto beatae vitae dicta sunt explicabo.  
4 </p>
```

Sed ut perspiciatis unde omnis iste natus error sit voluptatem
accusantium doloremque laudantium, totam rem aperiam,
eaque ipsa quae ab illo inventore veritatis et quasi architecto
beatae vitae dicta sunt explicabo.

擬似クラス

- :hover のように, セレクタに :ほげほげ と書くものを擬似クラスという

:hover	マウスオーバーされている間
:active	マウスをクリックしてから離すまで
:focus	テキストボックスなど, 現在選択中であるとき
:link	そのリンクが未訪問であるとき
:visited	そのリンクが訪問済みであるとき
:empty	その要素の内容が空であるとき (半角スペース1つさえ含まない)
:checked	その checkbox や radio button がチェックされているとき
:disabled	その要素の disabled 属性値が与えられているとき
:enabled	その要素の disabled 属性値が与えられていないとき
:target	その要素がURLで #id と参照されているとき

擬似クラス :not

- selector1:not(selector2)
 - セレクタ1 かつ (セレクタ2でない)

擬似クラス

- ...番目の要素
 - ... には a , $a+n$, $a-n$, a , even, odd のいずれかが書ける
 - a , b は整数. 1-indexed. $n = 0, 1, \dots$

<code>selector:nth-child(...)</code>	親の直下...番目の要素であってselectorにマッチするもの
<code>selector:nth-last-child(...)</code>	後ろから数える点以外は↑と同じ
<code>selector:nth-of-type(...)</code>	同じタグ名の兄弟の中で...番目の要素であってselectorにマッチするもの
<code>selector:nth-last-of-type</code>	後ろから数える点以外は↑と同じ
<code>:first-child</code>	親の直下1番目の要素
<code>:last-child</code>	親の直下最後の要素
<code>:first-of-type</code>	親の直下で最初に現れた当該要素
<code>:last-of-type</code>	親の直下で最後に現れた当該要素

擬似クラス

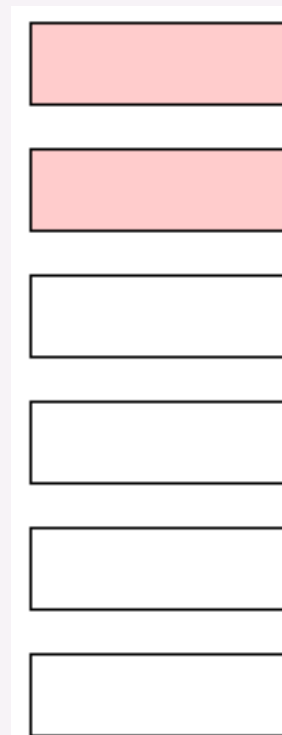
:~-child と :~-of-type の違い
タグ名毎で別個に考えるかそうでないか

index.html

```
<div id="hoge">  
  <p></p>  
  <div></div>  
  <p></p>  
  <div></div>  
  <p></p>  
  <div></div>  
</div>
```

style.css

```
#hoge > * {  
  width: 100px;  
  height: 30px;  
  border: 1px solid black;  
}  
#hoge > *:first-of-type {  
  background-color: #ffcccc;  
}
```



擬似クラス

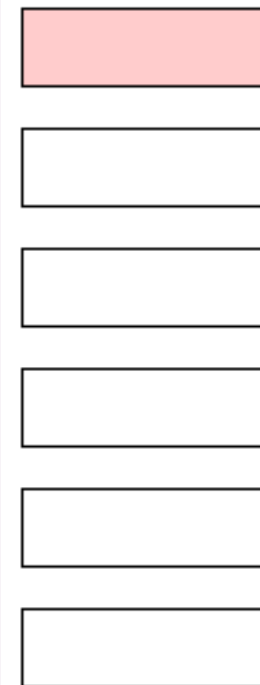
:~-child と :~-of-type の違い
タグ名毎で別個に考えるかそうでないか

index.html

```
<div id="hoge">  
  <p></p>  
  <div></div>  
  <p></p>  
  <div></div>  
  <p></p>  
  <div></div>  
</div>
```

style.css

```
#hoge > * {  
  width: 100px;  
  height: 30px;  
  border: 1px solid black;  
}  
#hoge > *:first-child {  
  background-color: #ffcccc;  
}
```



擬似要素

要素の前や後ろに擬似的な要素を作れる

selector::before, selector::after

index.html

```
<span id="score">60</span>
```

style.css

```
#score::before {  
  content: "score: ";  
}
```

score: 60

擬似要素

list-style: none; で「・」がなくなる

index.html

```
<ul class="mylist">
  <li>要素1</li>
  <li>要素2</li>
  <li>要素3</li>
</ul>
```

style.css

```
.mylist {
  list-style: none;
  padding-left: 0;
}
```

要素1
要素2
要素3

擬似要素

要素の前や後ろに擬似的な要素を作る (::before, ::after)

index.html

```
<ul class="mylist">
  <li>要素1</li>
  <li>要素2</li>
  <li>要素3</li>
</ul>
```

◆ 要素1

◆ 要素2

◆ 要素3

style.css

```
.mylist {
  list-style: none;
  padding-left: 0;
}
.mylist > li {
  position: relative;
  padding-left: 18px;
  margin: 15px 0;
  cursor: pointer;
  transition: all .3s;
}
.mylist > li:hover {
  color: #aa2288;
}
```

style.css

```
.mylist > li::before {
  display: block;
  content: "";
  width: 8px; height: 8px;
  background-color: #0000ff;
  position: absolute;
  top: calc(50% - 4px); left: 0;
  transform: rotate(45deg);
  box-shadow: 0 0 8px 0 #0000ff;
  transition: all .3s;
}
.mylist > li:hover::before {
  background-color: #ff00aa;
  box-shadow: 0 0 8px 0 #ff00aa;
}
```


擬似要素

要素の前や後ろに擬似的な要素を作る (::before, ::after)

index.html

```
<ul class="mylist">
  <li>要素1</li>
  <li>要素2</li>
  <li>要素3</li>
</ul>
```

◆ 要素1

◆ 要素2

◆ 要素3

style.css

```
.mylist {
  list-style: none;
  padding-left: 0;
}
.mylist > li {
  position: relative;
  padding-left: 18px;
  margin: 15px 0;
  cursor: pointer;
  transition: all .3s;
}
.mylist > li:hover {
  color: #aa2288;
}
```

style.css

```
.mylist > li::before {
  display: block;
  content: "";
  width: 10px;
  background-color: #0000ff;
  position: absolute;
  top: calc(50% - 4px); left: 0;
  transform: rotate(45deg);
  box-shadow: 0 0 8px 0 #0000ff;
  transition: all .3s;
}
.mylist > li:hover::before {
  background-color: #ff00aa;
  box-shadow: 0 0 8px 0 #ff00aa;
}
```

計算する

回転させる

CSSによる様々なデザイン

- 見出し
 - <http://www.nxworld.net/tips/50-css-heading-styling.html>

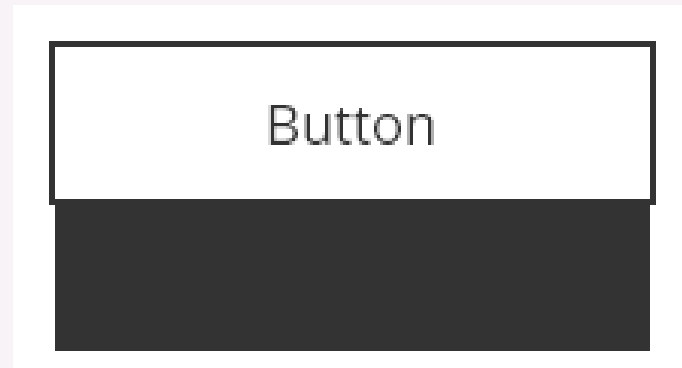
Lorem ipsum dolor sit amet.

あのイーハトーヴォのすきとおった風

CSSによる様々なデザイン

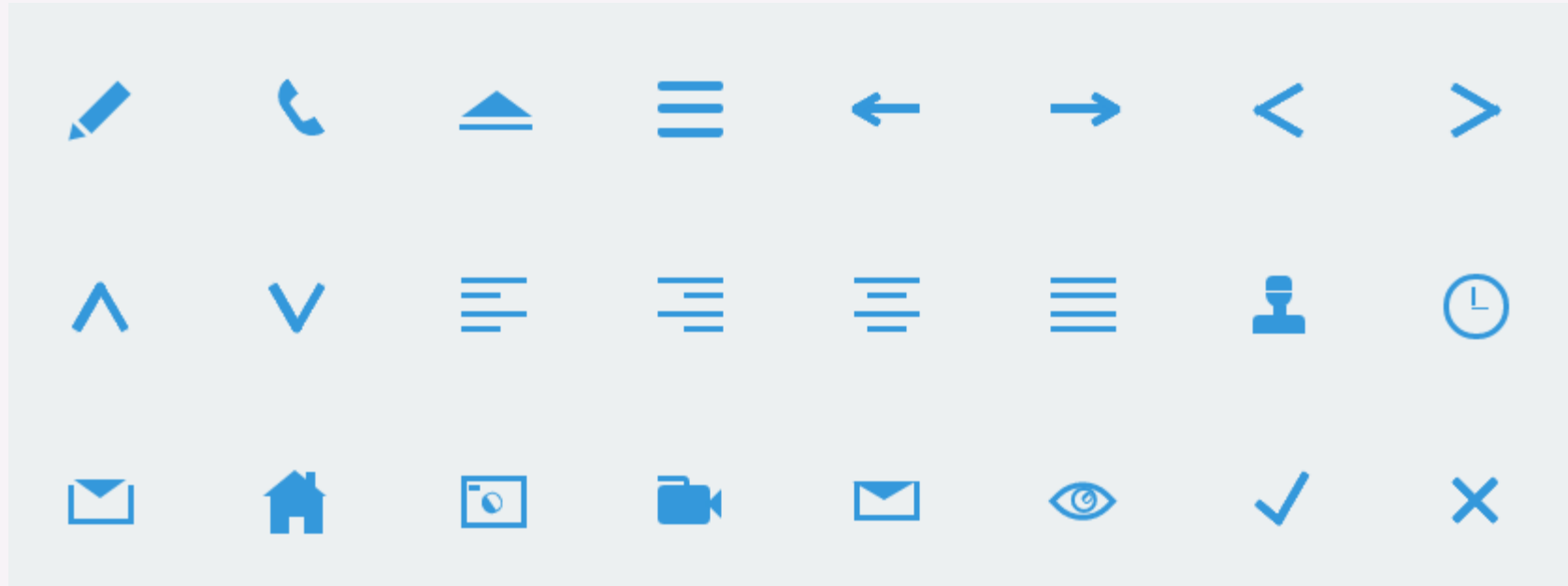
- ボタン

- <http://www.nxworld.net/tips/css-only-button-design-and-hover-effects.html>



CSSによる様々なデザイン

- アイコン
 - <http://littlebox.cabmaddux.com/>








5. トップページ作る

トップページを作ろう！

残りはトップページの続きを行おう！！

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 Parent Directory		-	
 Jyankenn/	15-May-2016 18:00	-	
 template/	08-May-2016 16:05	-	