Webページ 作成公開 勉強会

概要

Webページを作って公開してみよう！

【使用技術】

- Git
- HTML/CSS
- JavaScript/TypeScript
- Vue.jsフレームワーク
- Node.js
- Tailwind CSS
- GitHub Pages

Webページ　仕組み編

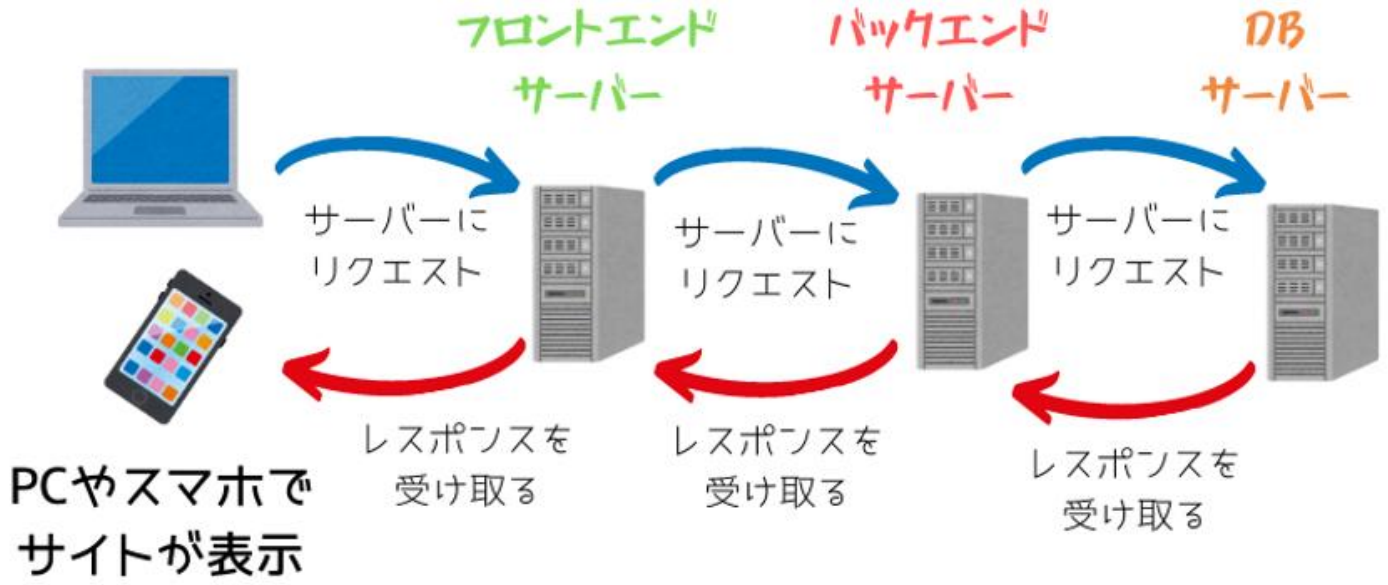
Webの仕組み

フロントエンド：ユーザーが触れる部分
バックエンド：ユーザーが触れれない部分（サーバサイド・データベース）

使用言語

フロントエンド
- HTML/CSS
- JavaScript
- TypeScript

バックエンド
- Java
- PHP
- JavaScript
- Python
- Kotlin
- Go など



※DNSサーバ、プロキシサーバ等は省略

引用：https://easy-life-tips.com/post/atyfxaccl

静的サイト・動的サイト

静的サイト：ページを事前に生成したサイト
動的サイト：ユーザーごとに表示を変えるサイト　例）ログインなどの機能

静的サイト

- 『閲覧目的』に主に使用
- 動的サイトと比べて、セキュリティ的に安全
- 使用言語が少ない（覚える量が少ない）

フロントエンド
- HTML/CSS
- JavaScript
- TypeScript

※HTML/CSSはプログラミング言語ではなく、マークアップ言語

GitHubに登録していれば
静的サイトを公開可能にするサービス

IPアドレス・ドメイン

IPアドレス： インターネットにおける住所（数値のみ）
ドメイン： インターネットにおける住所（数値、文字列）

→ IPアドレスとドメインを紐づけるためにDNS(Domain Name System)サーバーがある

IPアドレスを問い合わせる

名前解決

DNSサーバー

IPアドレスでアクセス

Webサーバー

URL

引用：https://www.asobou.co.jp/blog/web/url-optimisation

Webページ 作成編

本格的なWebページ作成

【手順】

1.どういうサイトを作るか決定

2.サイトマップ作成　・・・ ページの階層構造を決定

3.ワイヤーフレーム作成　・・・ Webページの要素の配置を決定

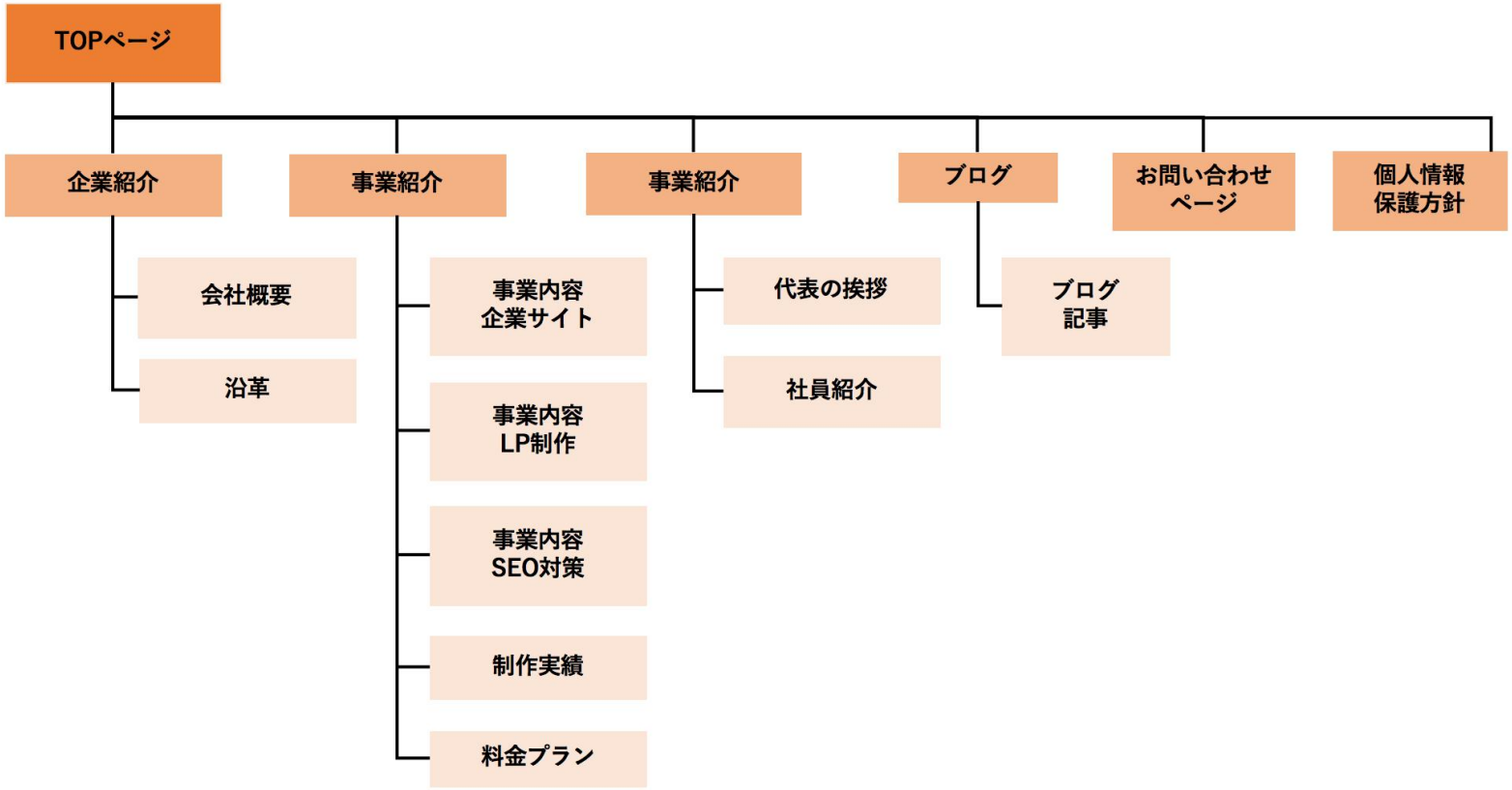
4.トーン＆マナー作成　・・・ 色、表記の統一を決定

5.デザイン作成　・・・ ファビコンなどのデザインを決定

6.実装

サイトマップ

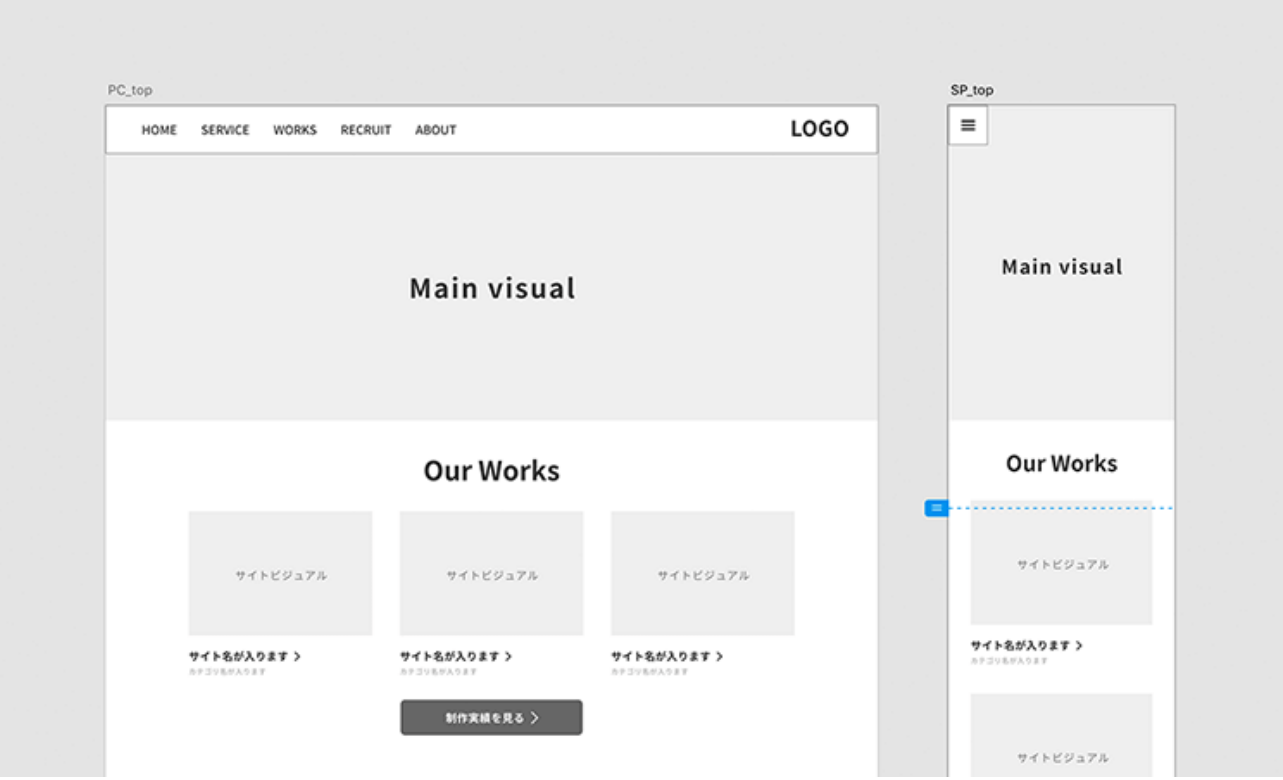
ページの階層構造を決定



引用：https://chot.design/web-director/e2277491ddc2/

ワイヤーフレーム

Webページの要素の配置を決定



引用：https://coosy.co.jp/blog/article06/

ワイヤーフレーム作成ソフトウェア

Figma

Xd

トーン&マナー

色、表記の統一を決定

引用：https://xn--3kq3hlnz1 3dlw7bzic.jp/tone-and-manner/

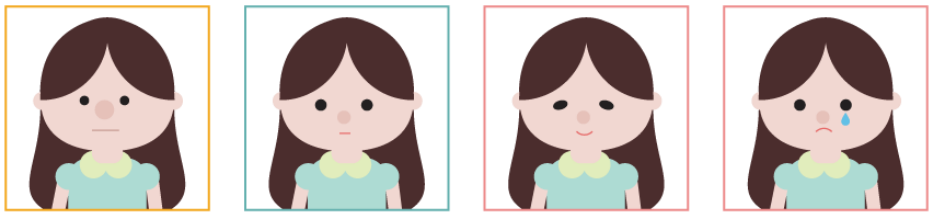
例）文章表記を決定

- User　→ ○ ユーザー　× ユーザ
- 個数 → ○ 1つ　× 一つ　× ひとつ
- 「です・ます」調で統一

【色選びの参考】
配色を制する者はフラットデザインを制す！配色選びのコツを徹底解説
https://pecopla.net/web-column/flatdesign-color

実装（使用する言語、フレームワークの紹介）

Git	変更履歴
HTML	Webページの要素の配置
CSS	デザイン、配置の微調整
JavaScript(JS)	Webページの動作（ボタンを押すとページ遷移など）
TypeScript(TS)	JSの型指定版。型指定できないJSより安全
Vue.js	JSのフレームワークの1つ。覚えやすい。
Node.js	JS実行環境
Tailwind CSS	CSS フレームワーク。より簡単にデザイン可能



引用：https://fastcoding.jp/blog/all/jquery/html-css-javascript/

実装環境

Visual Studio Code(VS Code)

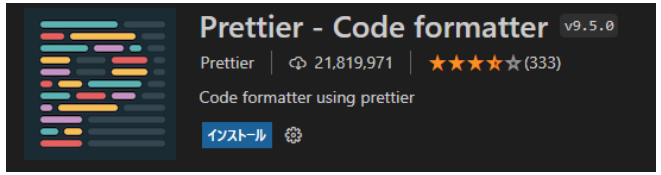
Webページ作成の際は、これがベスト

➡ 拡張機能によってWebページ制作が便利

Google Chrome

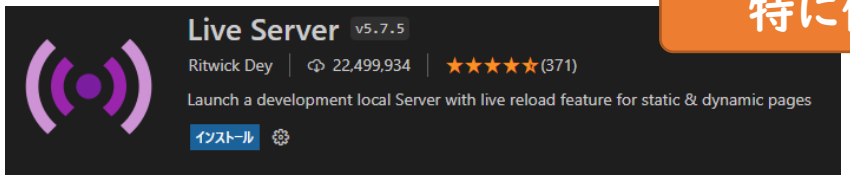
Webページの実行環境

VS Code 拡張機能

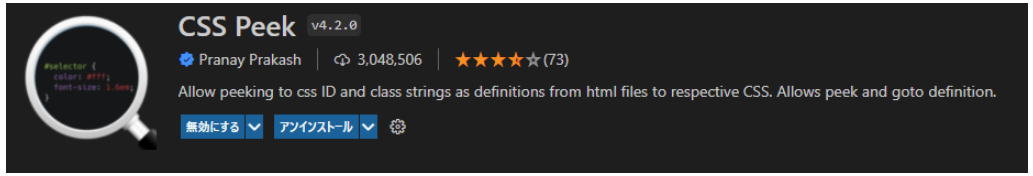


コードフォーマット

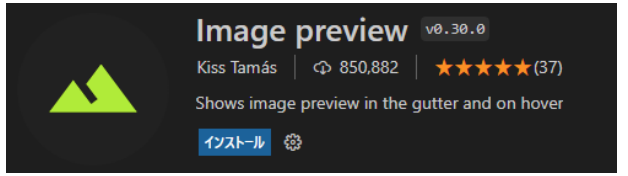
特に便利！



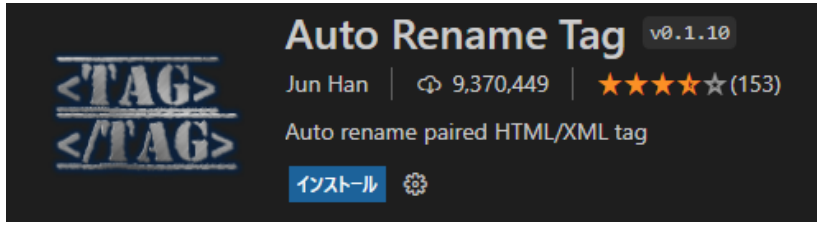
HTMLファイルをリアルタイムプレビュー
変更があると自動リロード可能



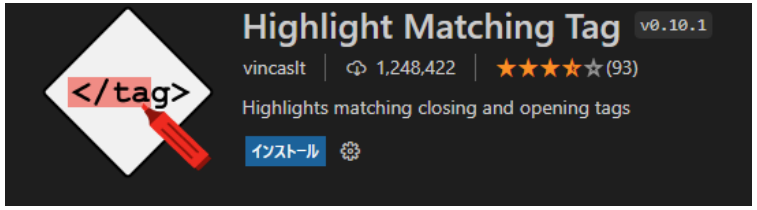
タグ、クラス、IDに対して
どのようなスタイルが適用されているか確認



<img>タグに書かれた画像を表示



タグ名変更時にペアで自動変換



タグをハイライトして見やすくする

HTML

HTML

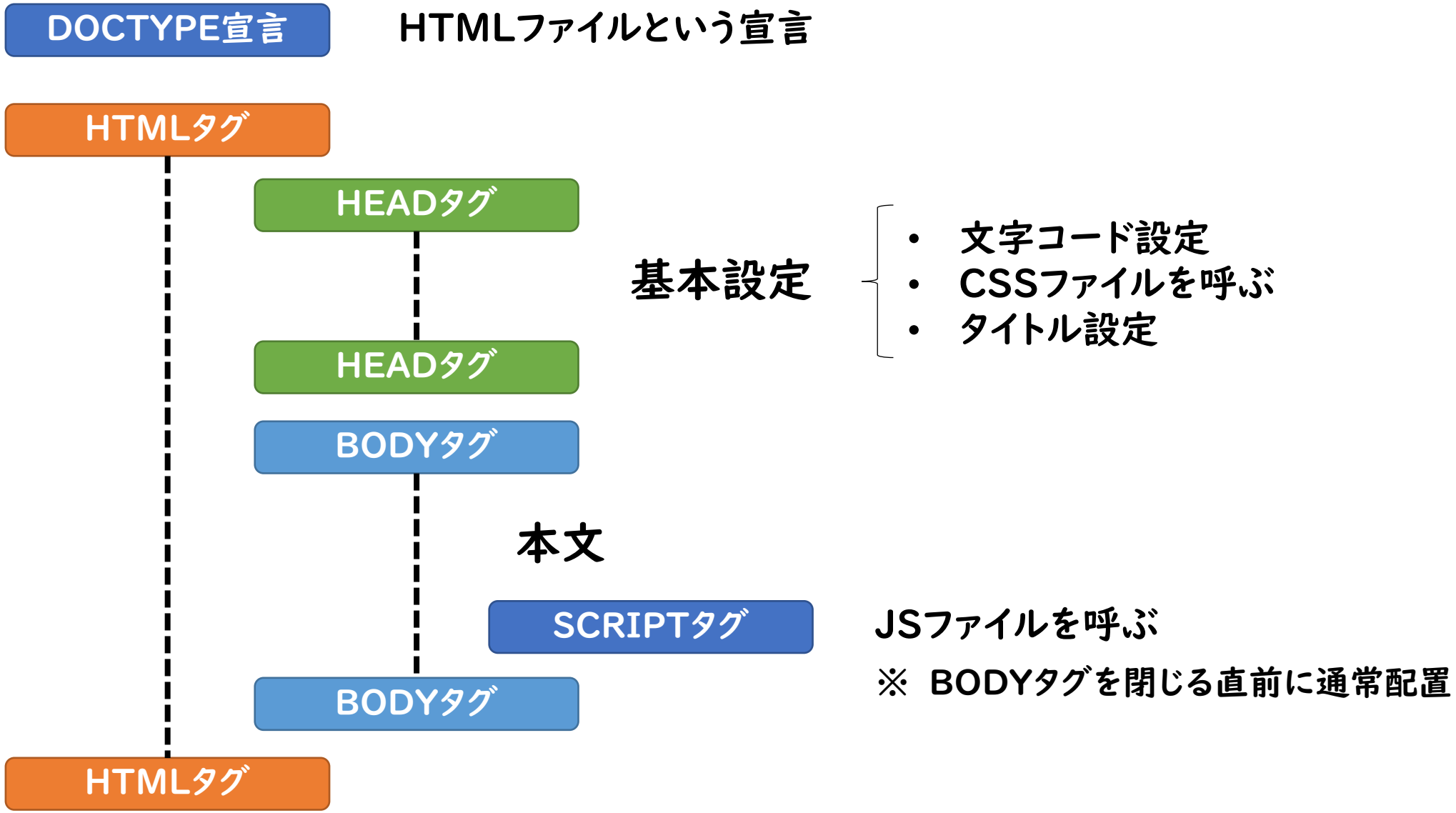
HTML（Hyper Text Markup Language）

- マークアップ言語
- タグで要素を構成

習得するためには、

- 基本構成を把握
- 複数のタグを把握

HTML 基本構成


DOCTYPE宣言
HTMLファイルという宣言
HTMLタグ
HEADタグ
基本設定
・ 文字コード設定
・ CSSファイルを呼ぶ
・ タイトル設定
HEADタグ
BODYタグ
本文
SCRIPTタグ
JSファイルを呼ぶ
※ BODYタグを閉じる直前に通常配置
BODYタグ
HTMLタグ

HTML 基本構成 例

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <link rel="stylesheet" href="stylesheet.css">
        <title>ページタイトル</title>
    </head>
    <body>
        <a>本文</a>
        <script src="1st.js" type="text/javascript"></script>
    </body>
</html>
```

DOCTYPE宣言

文字コード設定

CSSファイル呼ぶ

タイトル設定

JSファイル呼ぶ

HTML HEADタグ内のタグ

title	Webページのタイトル決定
link	基本CSSファイルを呼ぶために使用
meta	Webページの諸情報（文字コードなど）決定 ※「metaタグ」は『ファビコン設定』など数多くあるので知りたい人は検索

```
<meta charset="utf-8">
<link rel="stylesheet" href="stylesheet.css">
<title>ページタイトル</title>
```

HTML BODYタグ内のタグ

タグは大きく3種類存在

- ブロックレベル要素 ・・・ 1行全体を対象

<h1> ～ <h6>,<li>,<p>,<table>, など

- インラインレベル要素 ・・・ 1行の一部を対象

,
,<img>,<button>,<input>,<script>, など

- HTML5で新規追加された特殊タグ(可読性向上が目的)

<header>,<footer>,<main>,
<nav>,<canvas>,<video>,<article> など

HTML　ブロックレベルとインラインレベル

- ブロックレベル要素 ・・・　1行全体を対象
- インラインレベル要素 ・・・　1行の一部を対象

※　その他、高さや幅の設定等にも違いが出る
【詳細はWeb検索で】

ブロック要素の例

これはブロック要素です。div要素は独立した行で表示されます。

これもブロック要素です。新しいdiv要素は新しい行で表示されます。

インライン要素の例

これは インライン要素 です。span要素は他のテキストと同じ行に表示されます。 これもインライン要素です。

引用：https://it-biz.online/web-design/inline-block/

HTML クラスとID

各タグに「クラス」と「ID」を付与することで、CSSでデザインを変更可能

- クラス … 複数指定可能
- ID … 単一の要素のみ可能（重複不可）

```
<body>
    <div class=“content” id =“test”>
        <div class="header">入力画像</div>
        <div class="header">出力画像</div>
    </div>
    <script src="1st.js" type="text/javascript"></script>
</body>
```

百聞は一見に如かず～♪

HTML 実践

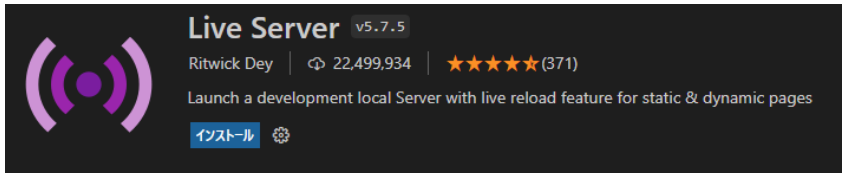
コード・実行方法・結果

ソースコードURL：
https://github.com/textcunma/webpage_study/tree/main/tutorial_codes/1th_html

実行方法

VS Codeの右下のココを押す

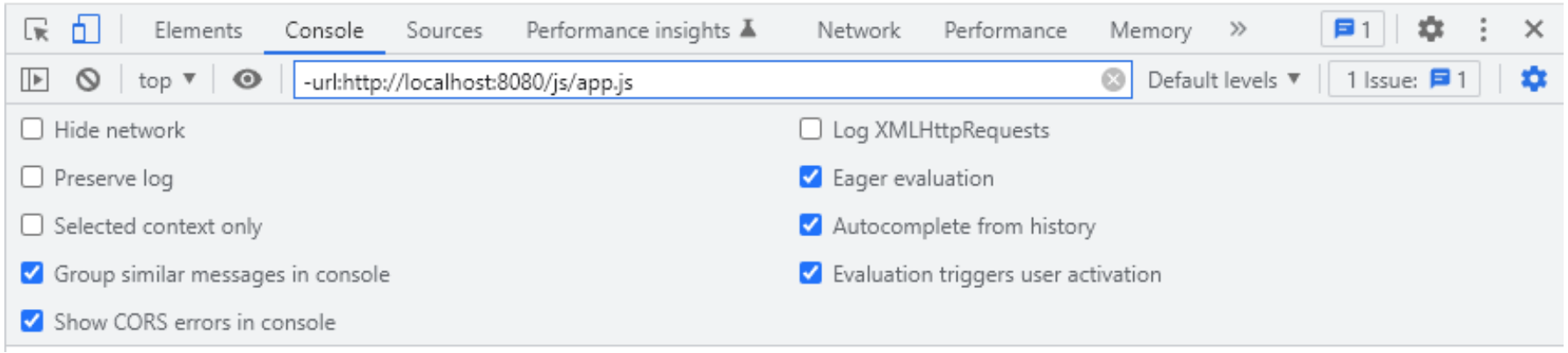
※拡張機能を入れる必要あり



表示結果

Chromeデベロッパーツール

ブラウザを表示した状態で『F12キー』を押す

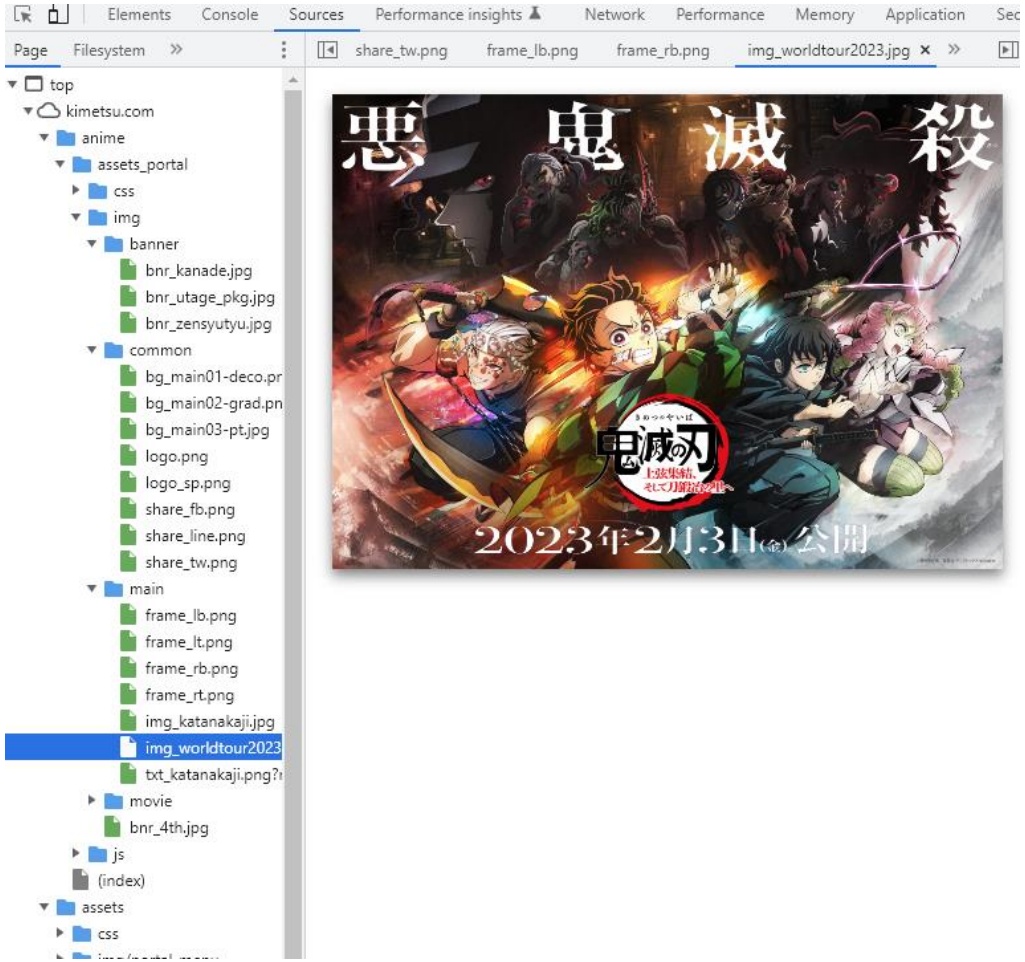


とにかくこれ抜きではWebページ制作は語れないほど重要。
以下、要チェック!!

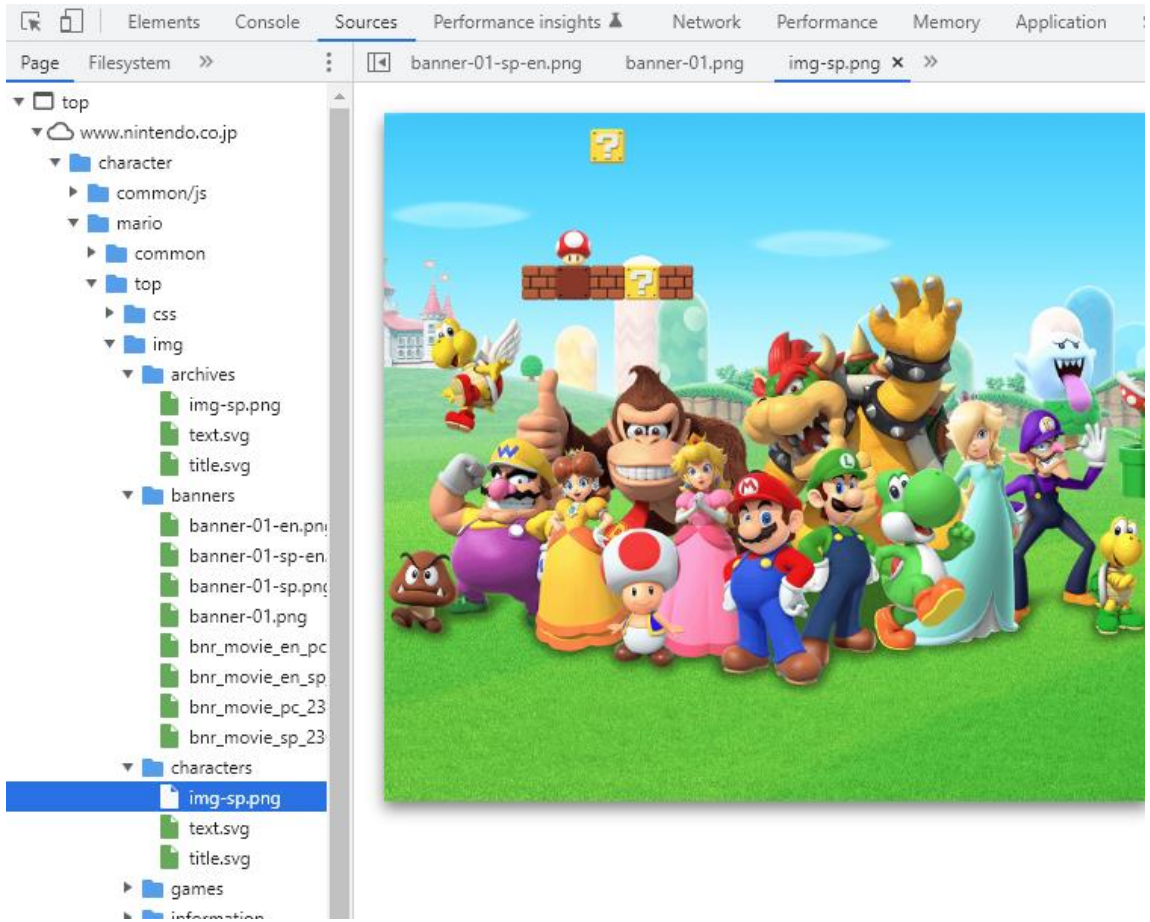
Google Chromeデベロッパーツールの基本的な使い方をわかりやすく解説
https://willcloud.jp/knowhow/dev-tools-01/

Chromeデベロッパーツール　遊び

デベロッパーツールを使うと、色々なサイトの素材や構造がわかる



https://kimetsu.com/anime/



https://www.nintendo.co.jp/character/mario/index.html

CSS

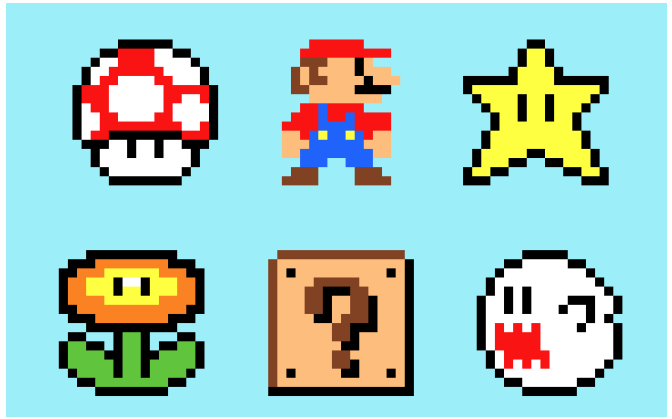
CSS

CSS（Cascading Style Sheets）

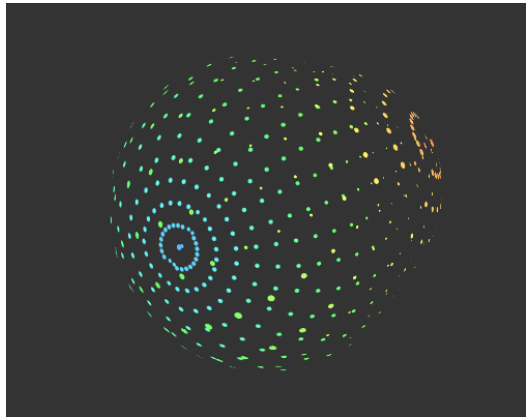
- HTMLで書かれたタグに対してデザインする

簡単なようで、とても奥が深いです
沼にハマらないようにしましょう

例） CSSサンプル



引用：https://codepen.io/una/pen/oXXRgg



引用：
https://codepen.io/iamlark/pen/jYzYJg

CSSルール

CSSを記述するルールは基本的に以下の3つ

タグに対して適用 → そのまま書く

例）　　div{}

クラスに対して適用 → .(ドット)を書く

例）　　.test{}

IDに対して適用 → #(シャープ)を書く

例）　　#test{}

CSS 疑似クラス・疑似要素

疑似クラス　例) hover

特定の条件に対してのみ行う処理

マウスをホバーした際に色が変化する処理によく使う

hover: マウスがホバーした場合の処理を記述

ホバーすると背景色が変化

疑似要素　例) before, after

初心者は知らなく良い

HTMLに記述された要素に追加で付与する要素

セレクタ

直下セレクタ

あるタグの直下にあるタグに対してCSSを適用

初心者は知らなく良い

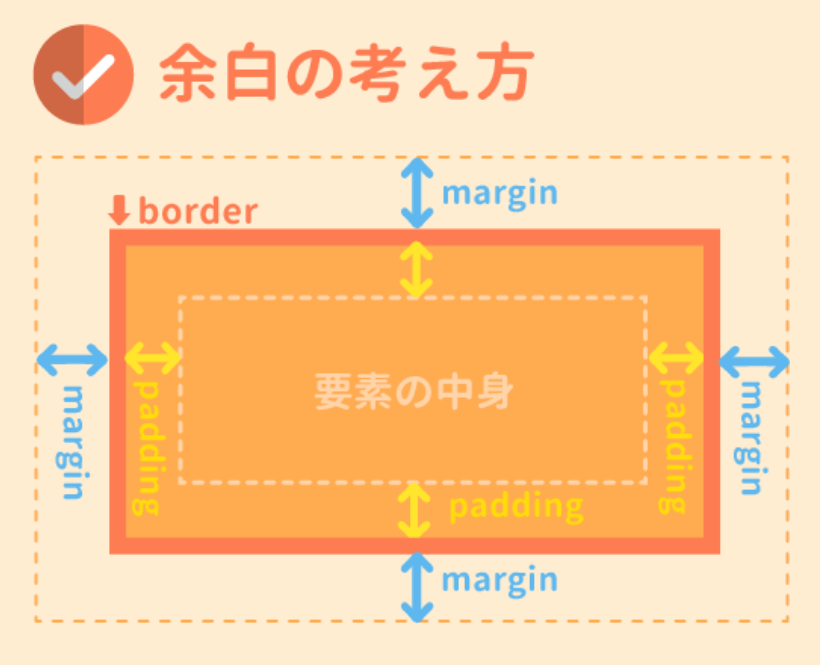
例）boxクラス内にあるpタグの色を変更する

```
/* 文字色の設定 */
.box > p {
    color: #313129;
}
```

その他、隣接セレクタ、間接セレクタなどがある

padding と margin

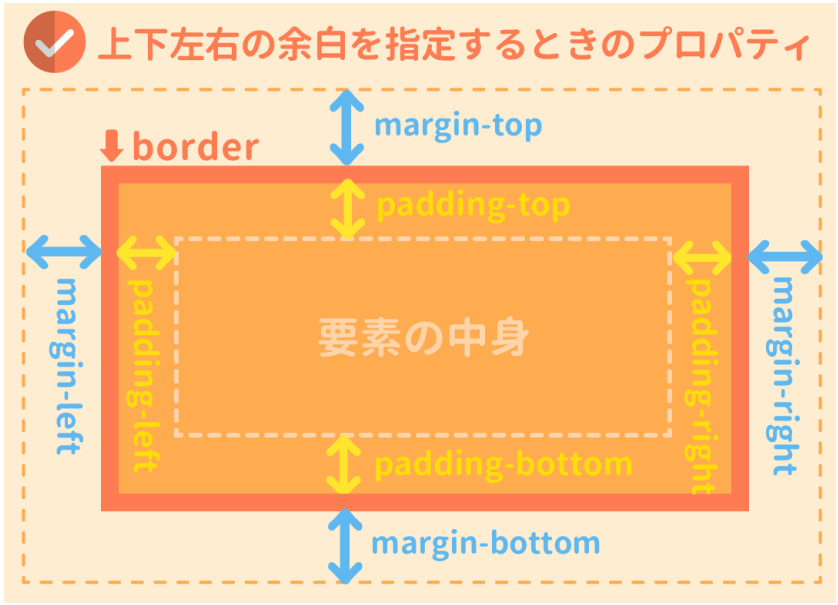
- paddingは「内側」
- marginは「外側」



引用：https://saruwakakun.com/html-css/basic/margin-padding

margin: 10px;　**省略記法**

margin: 10px 10px 10px 10px;
上　右　下　左
時計周りで指定

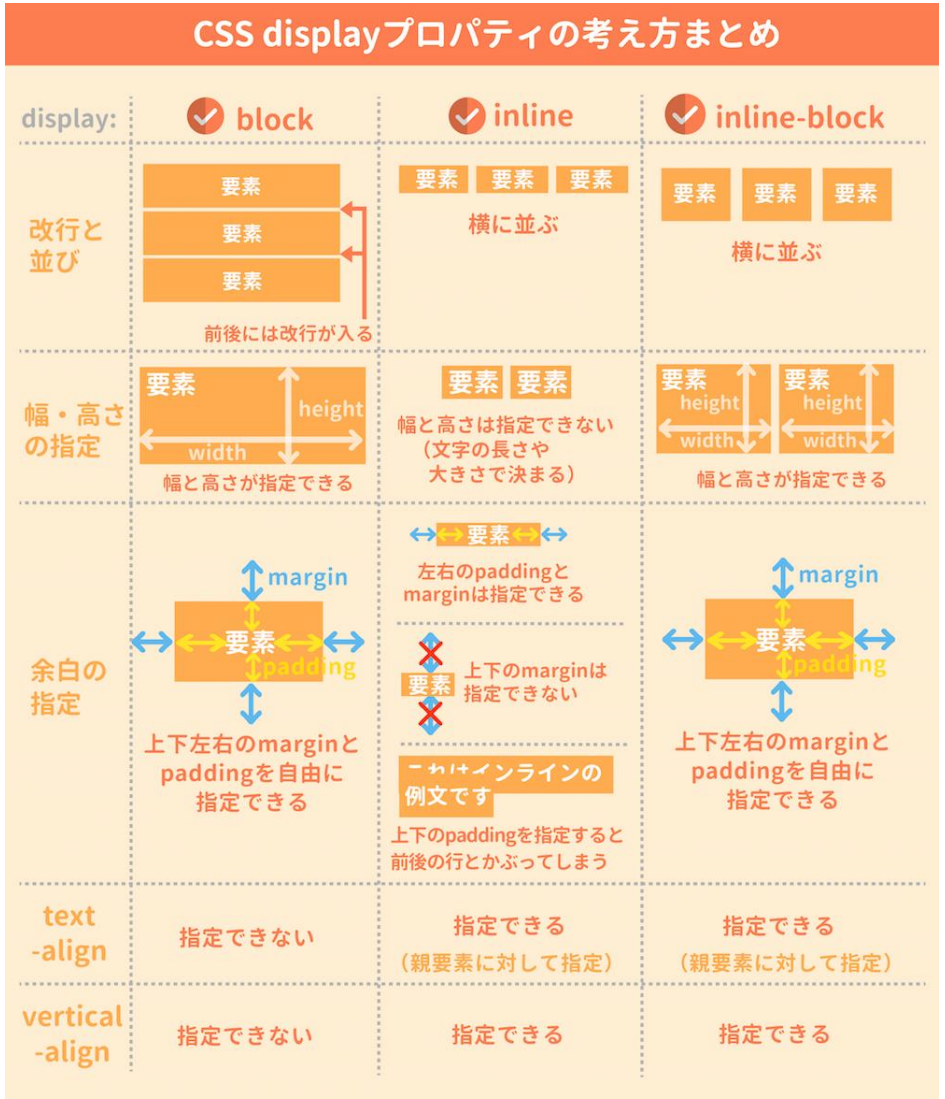


引用：https://saruwakakun.com/html-css/basic/margin-padding

display（要素の表示形式（表示レイアウト）の決定）

block	1行に1つの要素を表示
inline	1行に複数の要素を表示
inline-block	並びはinline、中身はblock
flex	Flexbox表示
none	非表示

CSSの難所の1つ



引用：https://saruwakakun.com/html-css/basic/display

position（要素をどのような位置基準で表示するかを設定）

static	初期値
relative	相対位置
absolute	絶対位置
fixed	固定位置
sticky	スクロールしても画面に表示し続ける

CSSの難所の1つ

位置を指定する流れ

1. `positon` で基準を決める
2. `top` `bottom` `left` `right` で具体的な位置を数字で調整

引用：https://saruwakakun.com/html-css/basic/relative-absolute-fixed

百聞は一見に如かず～♪

CSS 実践

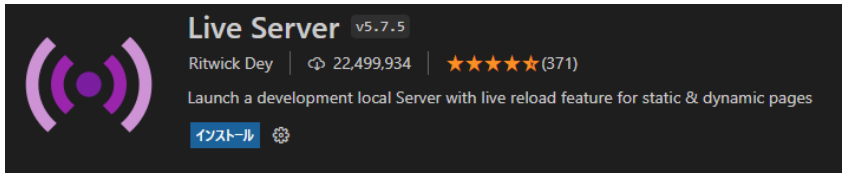
コード・実行方法・結果

ソースコードURL：
https://github.com/textcunma/webpage_study/tree/main/tutorial_codes/2nd_css

実行方法

VS Codeの右下のココを押す

※拡張機能を入れる必要あり



表示結果

CSS変更

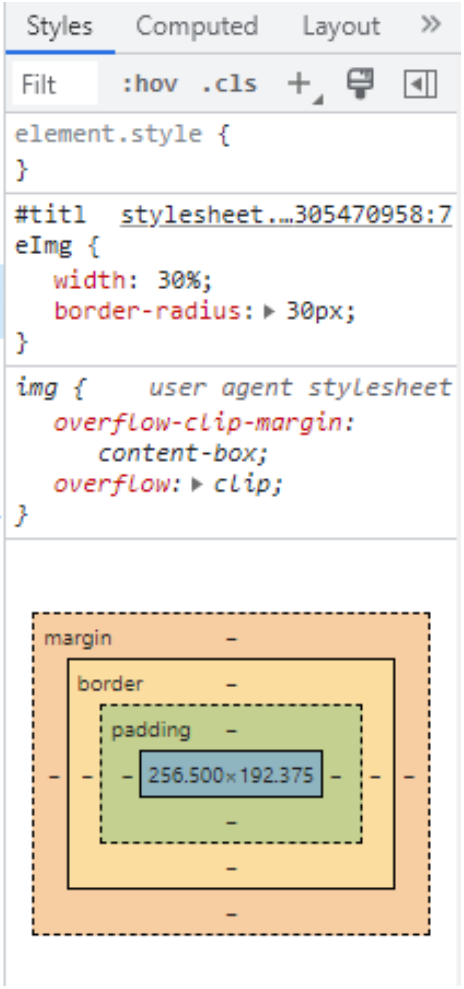
CSSを変更して保存すると動的に変化


```
#titleImg{
      width:30%;
      border-radius: 30px;
}
```

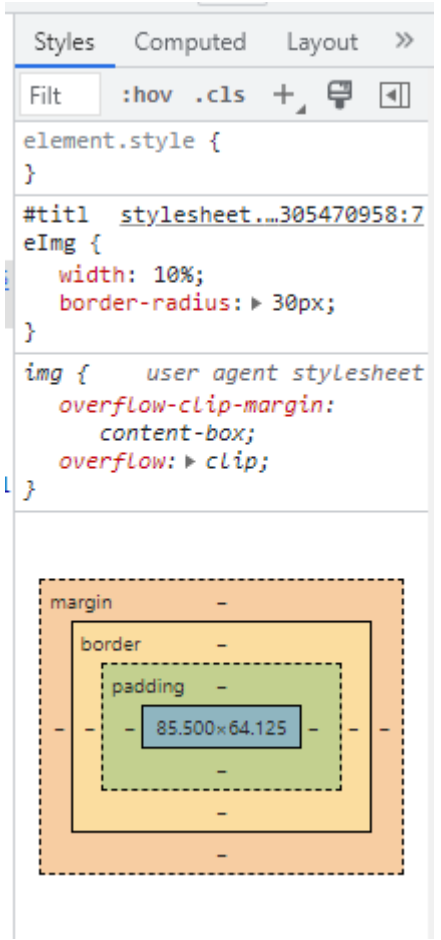

```
#titleImg{
      width:10%;
      border-radius: 30px;
}
```

CSS変更(デベロッパーツール)

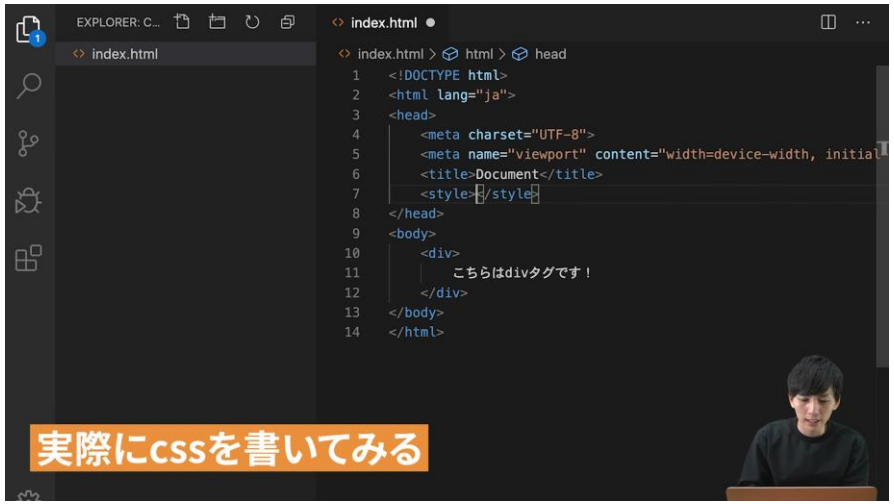
デベロッパーツールでもCSS変更可能



変更したい値の場所を
ダブルクリックで変更可能

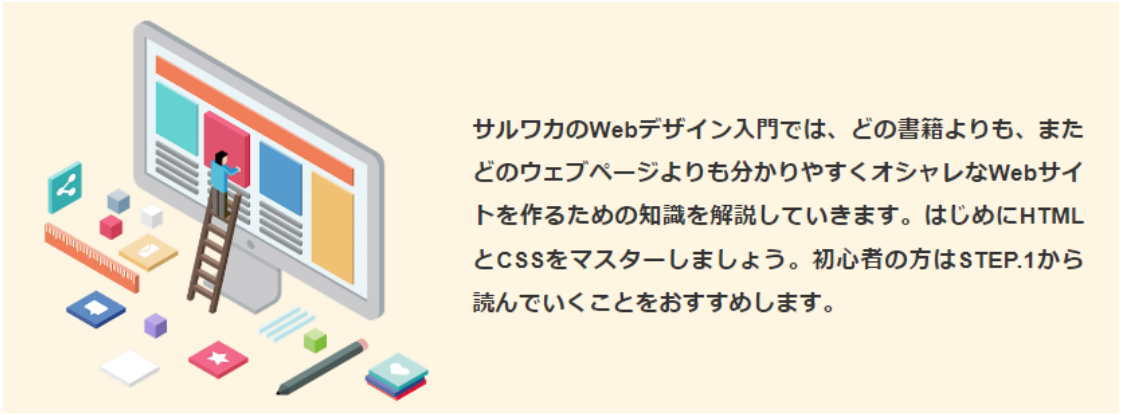


より詳しく知りたい方へ



【CSS #1】基礎からちゃんと学ぶ CSS 入門！基本構文を抑えよう！
【ヤフー出身エンジニアが教える初心者向けプログラミング講座】

https://youtu.be/xBLIzweHYic



HTML&CSS入門
Webデザインをイチから学ぼう

https://saruwakakun.com/html-css/basic

Tailwind CSS

Tailwind CSS

2021年頃から話題のCSSフレームワーク（人気）

指定のクラスにすることでデザインを行う、学習コスト低め

```
<button class="font-bold items-center justify-center inline-flex duration-300 border-solid rounded bg-primary-100 text-white hover:bg-primary-hover" type="button">  ラベル </button>
```

メリット

実際に導入してみてのメリットは以下のとおりです。

- 決められたユーティリティクラスを使うだけなので、クラス名を考えなくて良い
- 様々なJSのフレームワークと相性がよく、他のフレームワークに流用することが簡単
- 既存のクラス名を使い回すので、パーツが増えた時にクラス名を追加してCSSが増えることがない
- 特定のクラスを変えるだけでデザインの調整ができる
- テーマの設定を行うことで、デザインルールから逸脱せずにスタイリングできる
- 多くの開発で採用されているので、情報が得やすい
- アプリ開発ではAtomic Designをよく使うので、相性が良い

引用：https://baigie.me/engineerblog/?p=314

JavaScript (JS)

JavaScript（JS）

- 非同期処理が可能な代表的な言語
- データ型が指定できない　→　バグが発生しやすい

くそったれJavaScript
https://qiita.com/rana_kualu/items/793f0cbdde6a88f86394

- ライブラリ
  - jQuery
- 実行エンジン
  - Node.js(バックエンドでJavaScriptを使用可能に)
- フレームワーク
  - Vue.js、React、Angular など

JS 基礎構文

- 定数: const
- 変数: let (※varは古い)

型指定できない

条件分岐

```
if (year == 2015) {
  alert( "That's correct!" );
  alert( "You're so smart!" );
}
```

繰り返し処理

```
while (condition) {
  // code
  // いわゆる "ループ本体"
}
```

```
for (let i = 0; i < 3; i++) {
  alert(i);
}
```

関数

```
function sayHi() {
  alert( "Hello" );
}
```

引用：https://ja.javascript.info/

JS 出力

文字列を表記するには、以下2つ

- console.log(“”);　→　デベロッパーツールのコンソール画面に文字表記

- alert(“”);　→　モーダルウィンドウに文字表記

引用：https://www.modis.co.jp/candidate/insight/column_101

アロー関数

関数表記を簡略化

```
let sum = function (a, b) {
    return a + b;
};
```

```
let sum = (a, b) => {
    return a + b;
};
```

アロー（矢印）を表記すれば、「function」を省略できる

Strictモード（厳格モード）

- JavaScriptを比較的挙動がおかしくならないようにするモード
- JSファイルの先頭に記述

```
1  'use strict';
2
```

▶ 1.ミスをエラーとして返す

▶ 2.例外を発生させる

▶ 3.削除できないプロパティの削除操作に対してエラーを返す

▶ 4.関数の引数名が一意であることを必須とする

▶ 5.プリミティブ値にプロパティを設定することを禁止する

引用：https://zenn.dev/naberina/books/imoni-manner-book/viewer/3-1_imoni-manner-book

イテレータとジェネレータ

初心者は知らなく良い

イテレータ（反復子）

順番に呼び出すもの

イテラブルなオブジェクト=リスト、配列、文字列　など

```
const obj = ["A", "B", "C"];
const entries = obj.entries();
console.log(entries.next());
```

⟸ イテラブル（反復可能）にする

⟸ next()で逐次的に呼び出せる

ジェネレータ

イテレータの亜種

逐次的に返り値を生成

yieldによって一時停止が可能で占有メモリを減少可能

ジェネレータ関数

初心者は知らなく良い

通常の関数	ジェネレータ関数
function(){}	function*(){}
return	• return • yield • yield*

1万個のデータを生成して返す ⟷ 1万個のデータを生成するが逐次的に出力

占有するメモリ数
が大きく異なる

【より詳しく知りたい方へ】
JavaScript の ジェネレータ を極める!
https://qiita.com/kura07/items/d1a57ea64ef5c3de8528

例外処理

同期エラー処理（非同期のErrorはcatchできない）

try catch 文	tryでエラー判定、catchでエラー処理
throw 演算子	エラーを生成
try catch finally 文	tryでエラー判定、catchでエラー処理、finally内は必ず実行処理

Johnというメンバーなら「エラー」

```
    if (member === "John") {
        throw new Error("John is not allowed");
    }
```

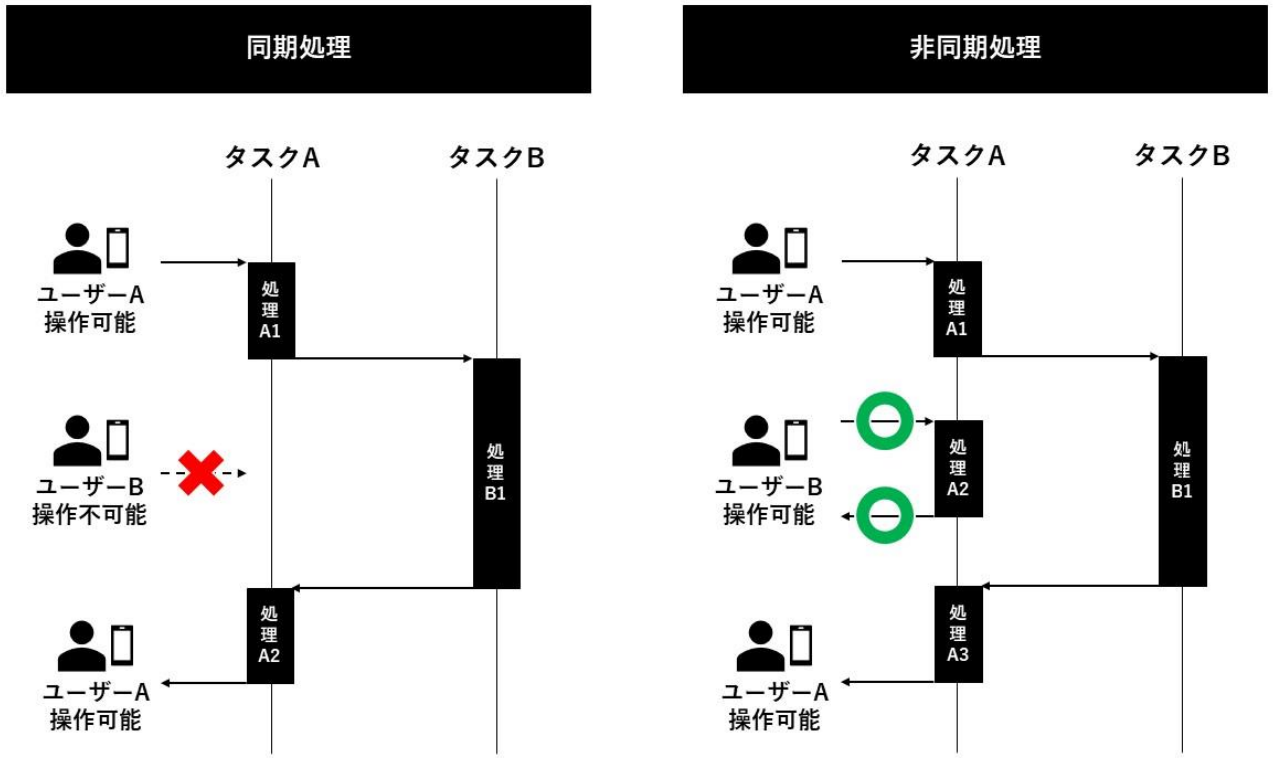
```
try{
    for (let i = 0; i < 5; i++) {
        console.log(array2[i]);
    }
} catch(err) {
        console.log(err);
} finally {
        console.log("終了");
}
```

同期処理と非同期処理

同期処理： 前の処理が完了してから次の処理を行う

非同期処理： 前の処理が完了する前に次の処理を行う

➡ 何等かの処理をしながら、ユーザーの操作を可能に



引用：https://www.cresco.co.jp/blog/entry/17612/

非同期処理の歴史

JSの非同期処理の歴史
https://www.messiahworks.com/archives/21598

非同期処理は記述法がややこしいことを理由に改善していった

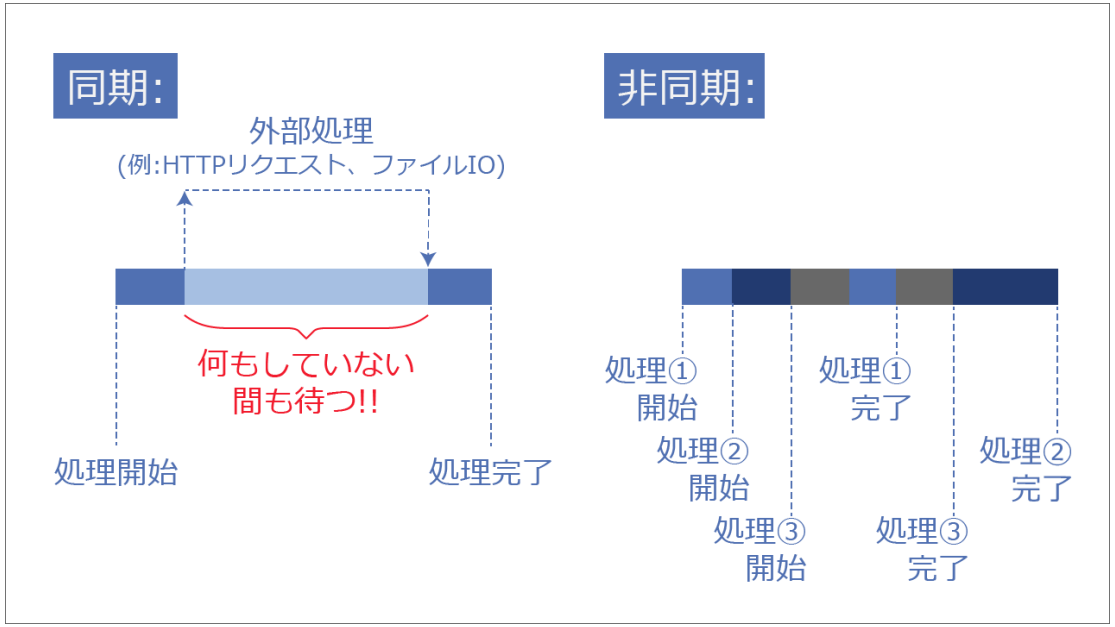
コールバック関数 ➡ Promise ➡ async/ await

2015年　　2017年

非同期処理とコールバック関数

非同期処理　→　複数の処理が同時に実行し、処理の完了順序が保証されない

コールバック関数　→　**処理の完了を通知するため、非同期関数に渡される関数**
これによって、ある処理結果を別の処理にすぐに利用可能



引用:https://codezine.jp/article/detail/11815

コールバック関数

コールバック関数 … 処理の完了を通知するため、非同期関数に渡される関数

エラー発生

```
// サーバのa.txtを非同期関数で読み込む
const xhr = new XMLHttpRequest();
xhr.open('GET', 'http://localhost/a.txt');
xhr.send();
// 非同期関数が終わってない状態なのでエラー
console.log(xhr.responseText);
```

コールバック関数　採用

```
const xhr = new XMLHttpRequest();
xhr.open('GET', 'http://localhost/a.txt');
// addEventListenerは非同期関数
xhr.addEventListener('load', (event) =>
console.log(xhr.responseText));
xhr.send();
```

非同期処理が終わったら、console.logで出力

引用：
https://www.messiahworks.com/archives/21598

コールバック地獄

複数のコールバック関数を実行しようとすると、コードの可読性が低下する問題

```
console.log(0);
setTimeout(function(){
  console.log(1);
  setTimeout(function(){
    console.log(2);
  }, 1000);
}, 1000);
```

1000[ms]ごとに0から2まで出力

出力：
0 → 1 → 2

➡ 非同期処理において、『Promise』や『Async/Await』を使用した方が良い

Promise

Promiseを使用した非同期関数は、処理が完了しなくとも「Promiseオブジェクト」を返す

Promiseオブジェクト … まだ完了していない可能性のある結果を保持するオブジェクト

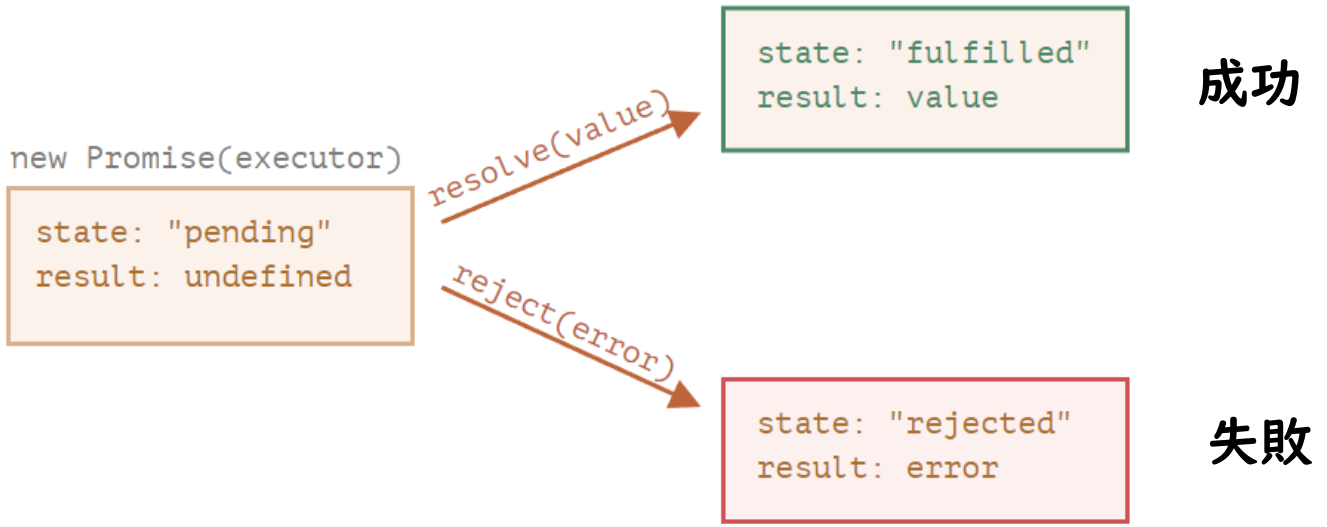
非同期関数の完了後の処理は『thenメソッド』によって記述

完了後の処理が入れ子構造にならず、コールバック地獄を解消

Promiseオブジェクトの3状態

pending	初期状態（保留状態）。処理が実行中の状態
fulfilled	Promiseオブジェクトの処理が正常に終了した状態 成功時に得られた結果が返る
rejected	Promiseオブジェクトの処理がエラーで終了した状態 エラー情報が返る

- resolve → 成功状態に遷移
- reject → 失敗状態に遷移



引用：https://ja.javascript.info/promise-basics

Promise 記述方法

```
const promise = new Promise((resolve, reject) => {
     if (Math.random() > 0.5){
          resolve("正常終了");      // 成功した状態へ遷移
     } else {
          reject("エラー");         // 失敗した状態へ遷移
     }
   });

promise.then(result => {
          console.log(result); // '正常終了'が出力
     }).catch(error => {
          console.error(error); // エラー情報を出力
});
```

定数promiseは、「Promiseオブジェクト」

then()のメソッドチェーン

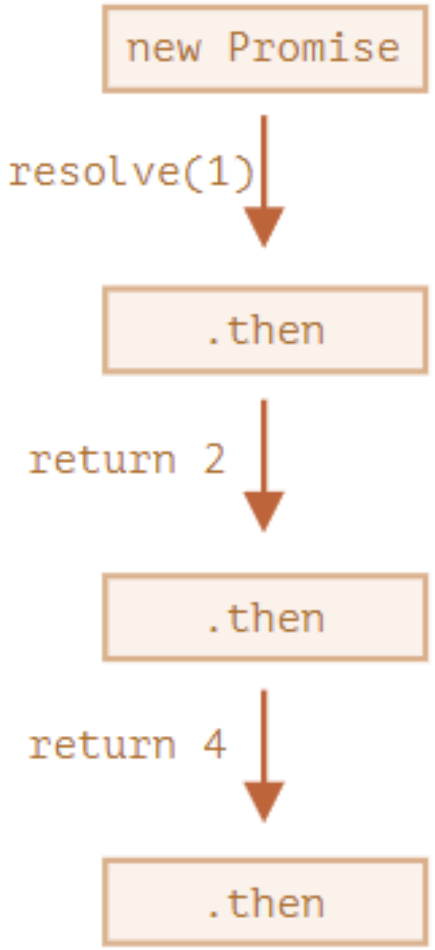
メソッドチェーン
.(ドットで)でオブジェクトを連結する記法

非同期処理で勝手にスキップされてはいけない処理に対して利用

→ .then().then().then()と連鎖的に処理を行う

チェーンメソッドでコードが長くなって読みづらい…

→ async/await で解決



引用：https://ja.javascript.info/promise-chaining

async , await

Promiseをより簡単に記述可能

async 非同期処理を含む関数を定義

await 非同期処理を行う部分
Promiseオブジェクトを受け取り、promise が確定しその結果を返すまで、待機

async , await 例

簡素になった!!

```
//  promiseの書き方
fetch("http://wttr.in/Fukuoka?format=j1")
  .then(function (res) {
    console.log(res); // => Response
    return res.json();
  }).then(function (json) {
    console.log(json); // => json
  })
```

```
async function get_data() {
  const a = await fetch("http://wttr.in/Fukuoka?format=j1");
  console.log(a); // => Response
  const b = await a.json();
  console.log(b); // => json
}

get_data();
```

引用：https://zenn.dev/kawaxumax/articles/0044a0e30536e2

百聞は一見に如かず～♪

JavaScript 実践

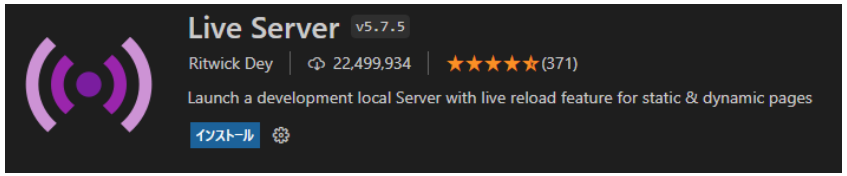
コード・実行方法・結果

ソースコードURL：
https://github.com/textcunma/webpage_study/tree/main/tutorial_codes/3rd_js

実行方法

VS Codeの右下のココを押す

※拡張機能を入れる必要あり



表示結果

TypeScript（TS）

TypeScript（TS）

TypeScript … 型指定可能になったJavaScript、2014年に登場

【利点】
- 型指定によってエラーが減る
- ほとんどJSと同じ記述が可能

【欠点】
- コンパイルする必要 → ブラウザ等の環境ではJSしか動かない
  TSをJSにコンパイルする必要がある

JS：インタープリター型言語
TS：コンパイラ型言語

JavaScript

```
let name = "John";
let age = 30;
```

→

TypeScript

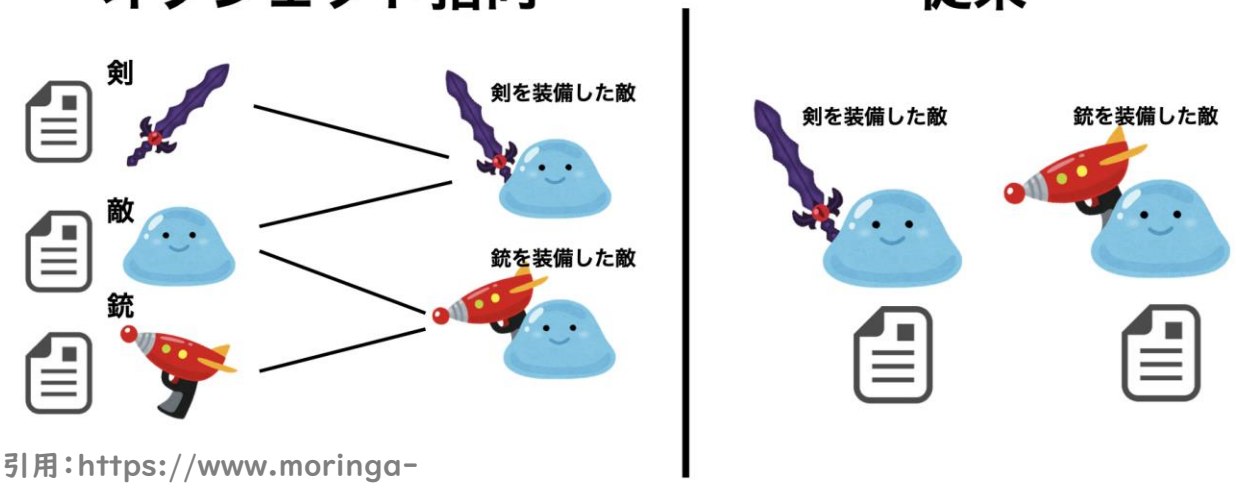
```
let name: string = "John";
let age: number = 30;
```

オブジェクト指向

オブジェクト指向プログラミング

オブジェクト指向 … プログラムを『オブジェクト』として捉える考え方
オブジェクト同士の相互作用によってプログラムを構築

※ オブジェクト → データと処理（関数）をまとめたもの



引用：https://www.moringa-moringu.com/%E3%82%AA%E3%83%96%E3%82%B8%E3%82%A7%E3%82%AF%E3%83%88%E6%8C%87%E5%90%91%E3%83%97%E3%83%AD%E3%82%B0%E3%83%A9%E3%83%9F%E3%83%B3%E3%82%B0%E3%81%A8%E3%81%AF/

例）
Java、Python、JavaScript

【利点】
コードの再利用が可能
集団開発に向いてる

【欠点】
コードが複雑化しやすい

※昨今ではオブジェクト指向は古いという人もいますが…

オブジェクト指向プログラミング　用語

クラス	オブジェクトの設計書。役割ごとに作成。 クラス内には『データ』と『関数』が記述。 • クラス内の変数 → メンバ変数（プロパティ） • クラス内の関数 → メンバ関数 特殊な関数「getter」、「setter」が存在
継承	親クラスの変数や関数を子クラスに引き継ぐこと →プログラムを再利用可能
カプセル化	プログラムの一部のデータを隠蔽して外部から アクセスできないようにすること
ポリモーフィズム	同じ名前の関数を、異なるクラスのオブジェクトで異なる 振る舞いをするようにする特性 この特性を実現するのが『オーバーライド』

クラス　基本構造

```
class Robot {
    constructor(name, owner) {
        this.name = name;
        this.owner = owner;
    }
    talk() {
        console.log(`My name is ${this.name}`);
    }
}

const r = new Robot('ATOM', 'Dr.Ochanomizu');
r.talk();   // My name is ATOM
console.log(r.owner);   // Dr.Ochanomizu
```

メンバ変数

メンバ関数

コンストラクタ（初期化関数）

※『this』は自分自身を表す
Pythonでいう『self』と同じ
（selfの方がわかりやすいよね…）

インスタンス生成

- コンストラクタによってクラスを初期化
- new演算子によってクラスのインスタンスを生成
- インスタンスを用いてメンバ関数を外部から使用

クラス　特殊メソッド

- getter：メンバ変数の値を出力する関数
- setter ： メンバ変数の値を更新する関数

```
class Robot {
    constructor(name, owner) {
        this.name = name;
        this.owner = owner;
    }
    get out_info() {
        return `${this.name} is owned by ${this.owner}`;
    }
    set in_info(new_owner) {
        this.owner = new_owner;
    }
}
```

getter関数

setter関数

クラス　特殊メソッド2

static関数
インスタンス化を行わずに使用できるメソッド

ただし、

- **インスタンスから呼べない**
- **thisで呼べない**

→　オブジェクトの生成や複製などユーティリティ関数として使用

```
class Teacher {
  constructor(props) {
    this.name = props.name;
    this.subject = props.subject;
    this.experience = props.student;
  }
  // staticメソッドの定義、インスタンスに対して作成
  static create(props) {
    return new Teacher(props);
  }
}

const props = {
  name: '山田　花子',
  subject: '理科',
  student: 35
};
// staticメソッドの呼び出し
teacher1 = Teacher.create(props);
console.log(teacher1.name); // "山田　花子"
```

引用：https://tcd-theme.com/2021/09/javascript-class-staticmethod.html

継承

親クラスを子クラスに継承

```
class Robot {
    constructor(name, owner) {
        this.name = name;
        this.owner = owner;
    }
    talk() {
        console.log(`My name is ${this.name}`);
    }
}

class CatRobot extends Robot {
    constructor(name, owner) {
        super(name, owner);
    }
    eat = () => {
        console.log(`${this.name} is eating`);
        super.talk();   //(親クラスの関数を呼ぶ)
    }
}

const c = new CatRobot('Doraemon', 'Fujiko.F');
c.eat();    // Doraemon is eating  My name is Doraemon
```

親クラス「Robot」
子クラス「CatRobot」

継承の記述方法

「extends 親クラス名称」

super()関数
子クラスから親クラスのメソッドを呼び出す
※オーバーライド（後で説明）している

カプセル化

外部からアクセスさせないようにする
「プライベートメンバ変数」、「プライベートメンバ関数」には「#」をつける

```
class Robot {
    constructor(name, owner) {
        this.name = name;
        this.#owner = owner;
    }
    #talk() {   // private method
        console.log(`My name is ${this.name}`);
    }
    get out_info() {
        return this.#talk();
    }
}

const r = new Robot('ATOM', 'Dr.ochanomizu');
// r.talk();   // エラー
// console.log(r.owner);       // エラー
console.log(r.out_info);    // My name is ATOM
```

プライベートメンバ変数

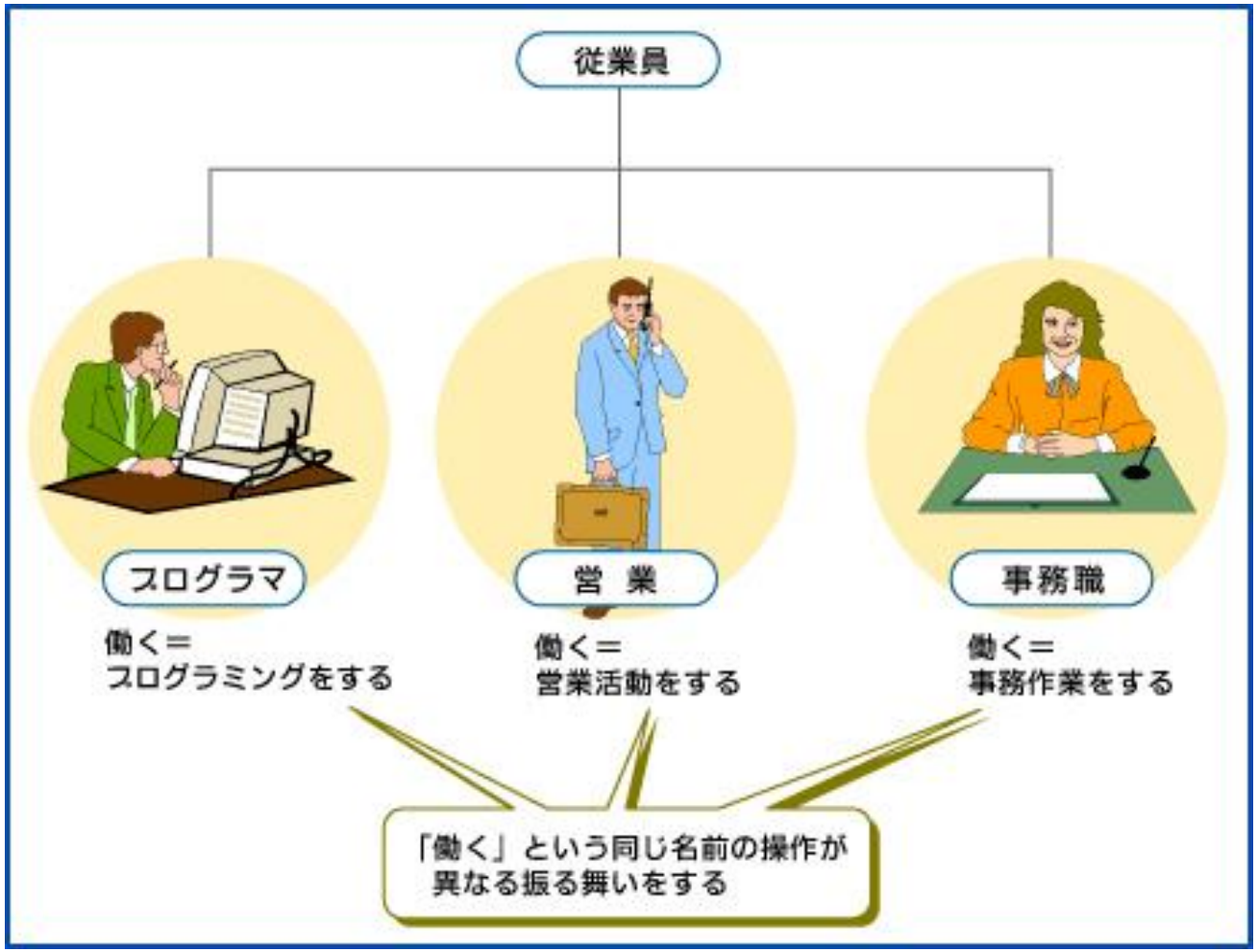
プライベートメンバ関数

プライベートメンバ変数、関数に
アクセスしようとするとエラー

ポリモーフィズム

同じ命令でも異なる振る舞いをする特性

『オーバーライド』によってこの特性を実現
→ 親クラスの関数を上書き



引用：
https://atmarkit.itmedia.co.jp/ait/articles/1403/17/news037_4.html

オーバーライド

```
class Robot {
    constructor(name, owner) {
        this.name = name;
        this.owner = owner;
    }
    talk() {
        console.log(`My name is ${this.name}`);
    }
}

class CatRobot extends Robot {
    constructor(name, owner, year) {
        super(name, owner);
        this.year = year;
    }
    talk() {
        console.log(`${this.year}生まれの僕、${this.name}`);
    }
}

const c = new CatRobot('Doraemon', 'Fujiko.F',2112);
c.talk();   // 2112生まれの僕、Doraemon
```

コンストラクタのオーバーライド
super()関数を使って親クラスのコンストラクタを呼び出す

オーバーライド
親クラスと同じ関数名だが、子クラスが優先して使用

Vue.jsフレームワーク

JavaScriptのフレームワーク

- Vue.js … 初心者向けと言われる
- React … 難易度が高いが業務で使うことが多い

フレームワークによって様々な利点
大きく「SPAが作成可能」、「コンポーネント」

SPA（Single Page Application）

必要な情報だけをリクエスト可能
ヘッダーやフッターは更新せず、中身だけを変更したい場合などに高速化可能

コンポーネント

1つのページを作るために、いくつもの部品にばらして作れる
→　要素をパッチワークして1つのページを作る
各要素に限定してHTML/CSSなどが書ける（可読性が向上）

Vue.js

Vue3

- 2021年に登場
- Vue2とは書き方が多いに変更  ネット検索の際に注意が必要
- TypeScriptで記述可能

Vue Router

- Vueでルーティング機能を実現

※ルーティング
リクエストされたURLに応じてコンポーネントを選択してDOMを動的に変化させて表示
→初アクセスで情報をすべて渡し、ページ遷移をサーバーとやり取りする必要がない

環境構築のためには「Node.js」が必要

Vue.jsの記述方法

要素（components）ごとにファイルを作成
→　要素のパッチワークによってWebページを作成

1つのファイルには「script」「template」「style」の3つのタグがある

script ➡ JSもしくはTSを記述

template ➡ レイアウトを決定（HTML）

style ➡ デザインを決定（CSS）

これら3つのタグによって、1つの要素を作成

v-bind

要素の属性にデータを動的にバインド

```
<template>
     <div>
       <input type="text" :value="message">
       <img :src="imageSrc">
     </div>
   </template>

<script>
export default {
data() {
     return {
     message: 'Hello, Vue!',
     imageSrc: 'https://example.com/images/logo.png'
     }
}
}
</script>
```

CSSの幅などを変更したい場合に使用

v-if

条件に基づいて要素を条件付きでレンダリング

```
<template>
    <div>
      <p v-if="isDisplayed">This is displayed when isDisplayed is true.</p>
      <p v-else>This is displayed when isDisplayed is false.</p>
    </div>
  </template>

  <script>
  export default {
    data() {
      return {
        isDisplayed: true
      }
    }
  }
  </script>
```

v-for

配列やオブジェクトの各要素を繰り返し処理し、動的に要素をレンダリング

```
<template>
    <div>
      <ul>
        <li v-for="(item, index) in items" :key="index">{{ item }}</li>
      </ul>
    </div>
  </template>

  <script>
  export default {
    data() {
      return {
        items: ['apple', 'banana', 'orange']
      }
    }
  }
  </script>
```

Webページ　実践

事前準備

- Node.jsをインストール
  - コマンドプロンプトでnpmコマンドを使用可能にする
- Gitインストール
  - コマンドプロンプトでgitコマンドを使用可能にする
- GitHubアカウント登録
- GitHubに公開鍵登録
  - プライベートリポジトリをダウンロードできるようにする
  - https://yu-report.com/entry/githubssh/
  - https://qiita.com/shizuma/items/2b2f873a0034839e47ce

Git、GitHub

補足資料

Git → コードの変更履歴を残すためのもの

【リポジトリ】
- ローカルリポジトリ ： PC内で変更履歴を保存する場所
- リモートリポジトリ ： ネットサーバー内で変更履歴を保存する場所

GitHub → リモートリポジトリサービスの1つ

公開リポジトリ

全世界に公開ため**注意が必要**

艦これユーザー「さぶれ」氏、SMBCのソースコードを
GitHubに公開して失業&損害賠償700万円

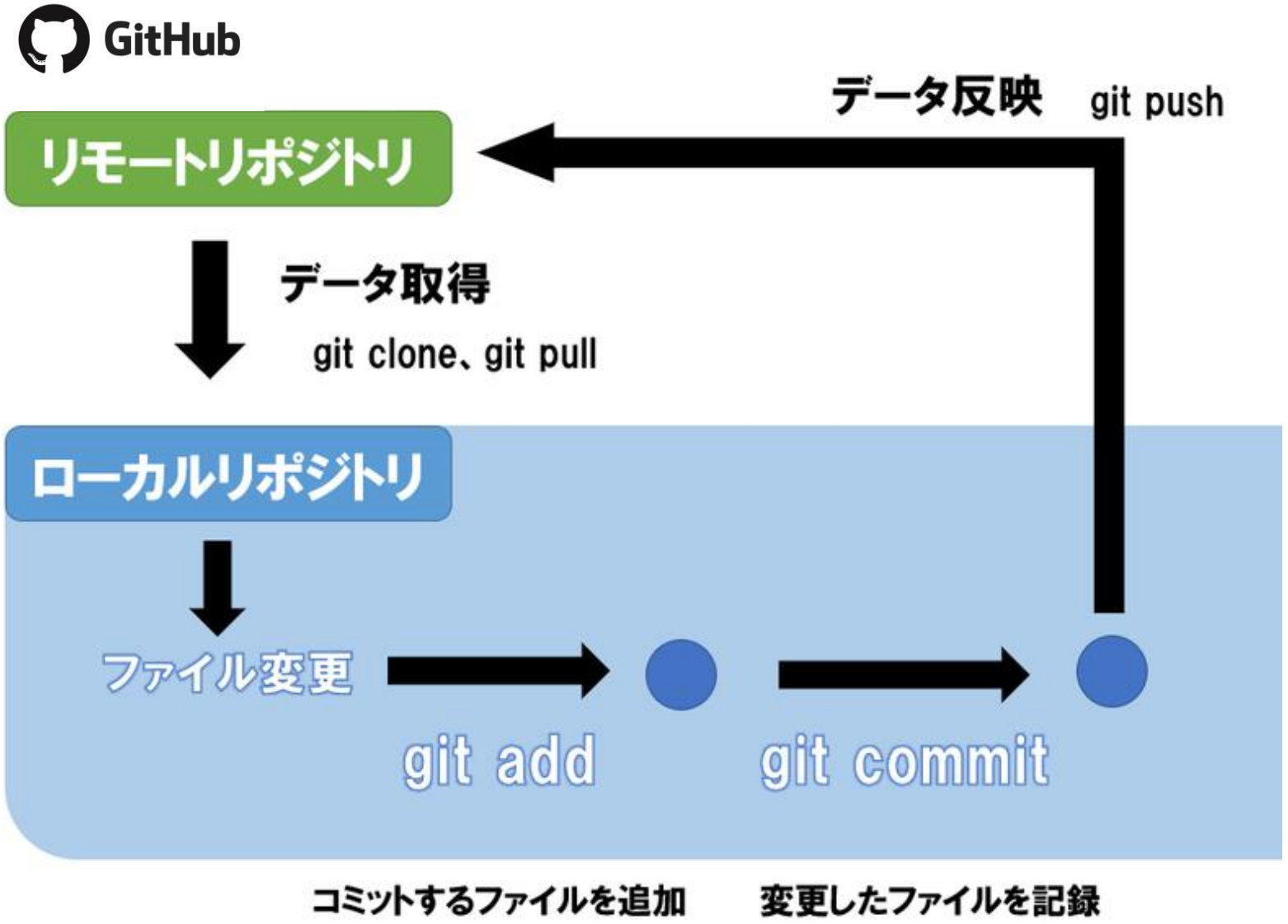
非公開リポジトリ

基本的に自分のみ見れる
(閲覧編集を指定ユーザに権限与えれる機能あり)

バックアップ目的で使える

Gitの流れ

補足資料


GitHub
データ反映
git push
リモートリポジトリ
データ取得
git clone、git pull
ローカルリポジトリ
ファイル変更
git add
git commit
コミットするファイルを追加
変更したファイルを記録

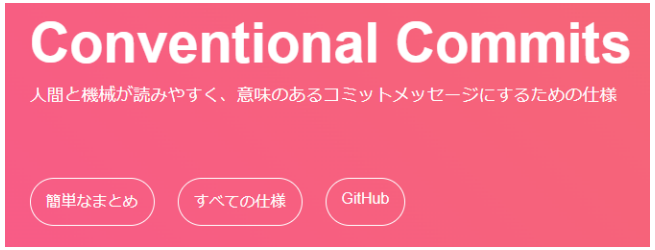
Gitの流れ　コマンド操作

補足資料

＜基本手順＞
1. リモートリポジトリを作成
2. git clone　〇〇
3. コードを編集
4. git add --all
5. git commit -m “メッセージ”
6. git push

※コミットメッセージは、テキトーは×
何を変更したのか分かるものに

Conventional Commitsを参考に
https://www.conventionalcommits.org/ja/v1.0.0/

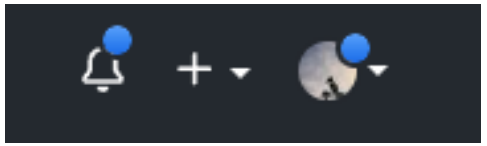


※補足
- git add --all ： 変更があったファイル全てを追加
- git add test.py : test.pyのみを追加

※ミクシィの新卒研修資料は良いらしいです

ミクシィ　Git研修講義【21新卒技術研修】
https://youtu.be/aZ90usArA6g

GitHub 非公開リポジトリ作成



GitHub画面右上の「＋」ボタンから「New Repository」を選択

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
Import a repository.

Owner * / Repository name *

test

Great repository names are short and memorable. Need inspiration? How about urban-happiness?

Description (optional)

test

Public
Anyone on the internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

リポジトリ名や説明は任意

「Private」を選択

Initialize this repository with:
Skip this step if you're importing an existing repository.

Add a README file
This is where you can write a long description for your project. Learn more.

Add .gitignore
Choose which files not to track from a list of templates. Learn more.

.gitignore template: Node

「Node」を選択

Choose a license
A license tells others what they can and can't do with your code. Learn more.

License: None

非公開リポジトリをダウンロード

※windowsだと
特定のフォルダ上で
「Shift+右クリック」で
そのフォルダをカレントディレクトリに
なったPowerShellを開けれる

コピーボタンをクリック

コマンドプロンプトで「git clone _____コピーしたリンク_____」

Viteでプロジェクトを作成

以下、コマンドをコマンドプロンプトに入力

```
npm init vite@latest
```

指示に従って以下の設定を行う（ここの設定は間違わないように…）

- プロジェクト名
- 使用するフレームワーク
- 「Customize with create-vue」を選択 ➡

```
? Select a variant: » - Use arrow-keys.
    TypeScript
    JavaScript
>   Customize with create-vue ↗
    Nuxt ↗
```

- 「TypeScript」、「Vue Router」、「ESLint」、「Prettier」でYesを選択

```
√ Package name: ... samplewebpage
√ Add TypeScript? ... No / Yes
√ Add JSX Support? ... No / Yes
√ Add Vue Router for Single Page Application development? ... No / Yes
√ Add Pinia for state management? ... No / Yes
√ Add Vitest for Unit Testing? ... No / Yes
√ Add an End-to-End Testing Solution? » No
√ Add ESLint for code quality? ... No / Yes
√ Add Prettier for code formatting? ... No / Yes
```

ローカルホスト（ローカル上のWebサーバー）を立ち上げ

先ほど作成したプロジェクトフォルダに移動

```
cd ./[フォルダ名]
```

パッケージをインストール

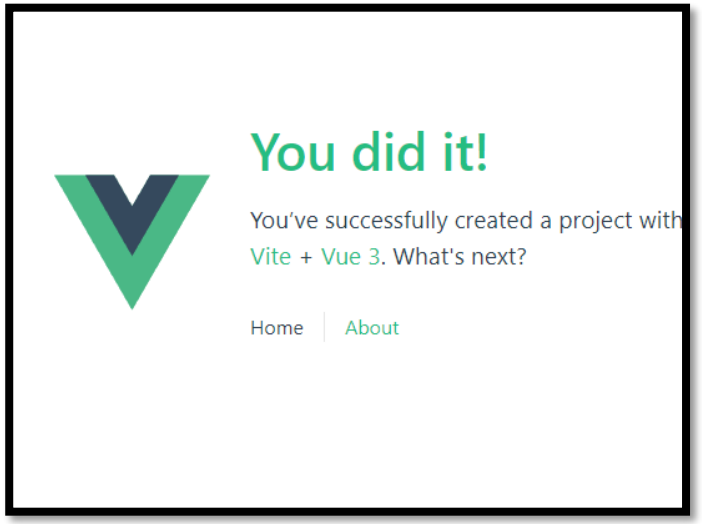
```
npm install
```

ローカルホストを起動

```
npm run dev
```

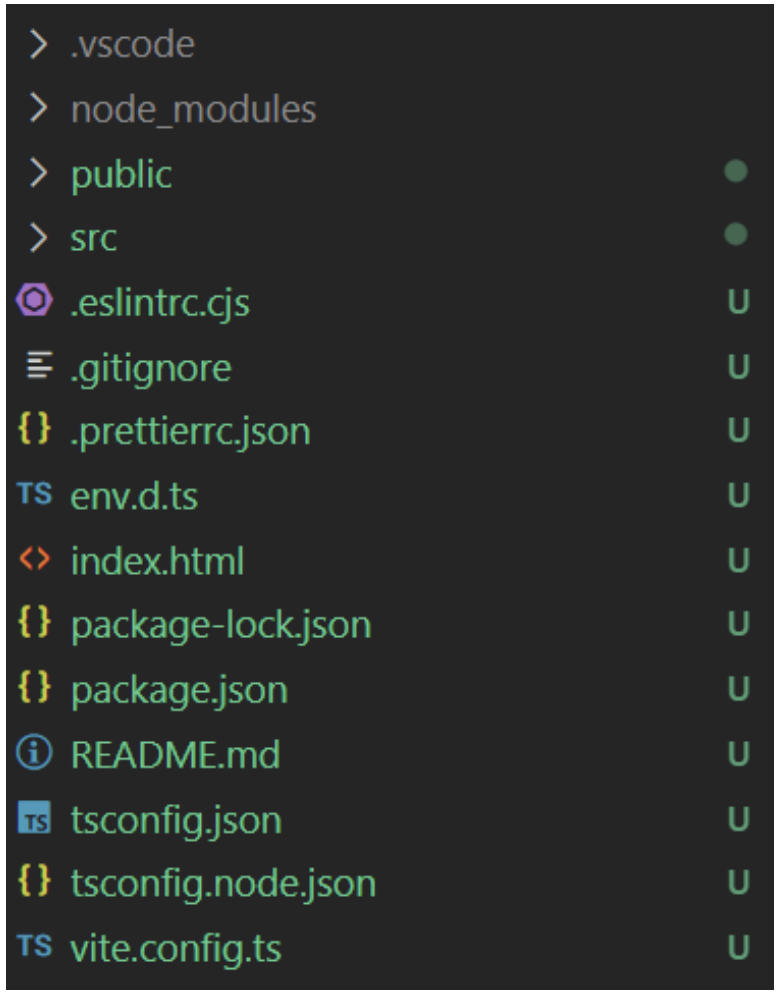
ローカルホストを停止 → Ctrl + c

起動画面



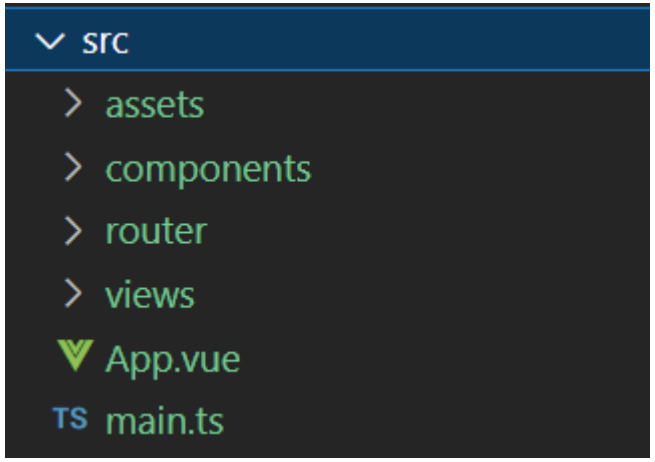
参考：話題の爆速CLIツール「Vite」をVue.jsの定番ツール「Vue CLI」と徹底比較！
https://codezine.jp/article/detail/14939

全体的なディレクトリ構造　説明



- publicディレクトリ
  - ファビコン
- srcディレクトリ
  - メインとなるソースコード
- index.html
  - ファビコンや使用するTSのパス名が記述
  - 基本触りません！
- vite.config.ts
  - ビルドディレクトリなどを設定
- その他
  - ライブラリのバージョン
  - 諸設定

srcディレクトリ構造　説明



- assetsディレクトリ
  - 画像、SVG、CSSファイル
- componentsディレクトリ
  - ページの要素となる情報
    これらの要素を用いてページを作成
- routerディレクトリ
  - ページの名前、パス名を記述
- viewsディレクトリ
  - ページのレイアウトを決定
    componentsをどう配置するか
- App.vue
  - Webページ全体の管理(ページの追加など)
- main.ts
  - App.vueに書かれた内容とindex.htmlのbodyタグ内の内容を繋ぎ合わせる

Tailwind CSS 設定

1. 下記、コマンドを入力してTailwind CSSをインストール

```
npm install -D tailwindcss@latest postcss@latest autoprefixer@latest
```

```
npx tailwindcss init -p
```

2. tailwind.config.jsにおいて、purgeを変更

```
  purge: ['./index.html', './src/**/*.{vue,js,ts,jsx,tsx}'],
```

3. srcフォルダ内に「index.css」を作成
4. src/main.tsにおいて、以下の変更

```
import './assets/main.css'
```

```
import './index.css'
```

5. 【任意】src/assets内のbody要素だけはindex.html内にコピー

6. src/assets内のcssファイルを削除

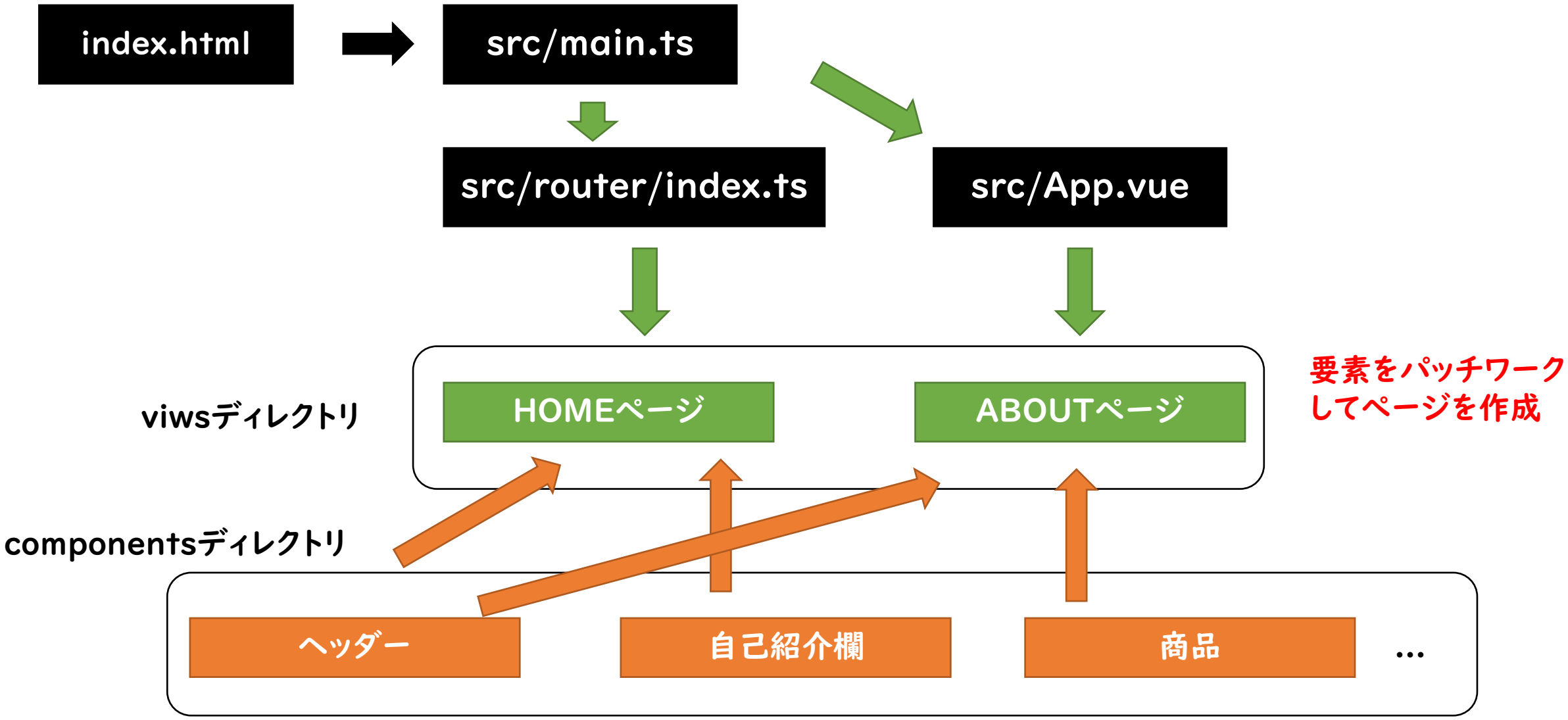
参考：
https://zenn.dev/grimm_marchen/articles/8d297bf7ea1127

App.vueを変更

```
<script setup lang="ts">
import { RouterLink, RouterView } from 'vue-router'
import HelloWorld from './components/HelloWorld.vue'
</script>

<template>
  <header>
    <img alt="Vue logo" class="logo" src="@/assets/logo.svg"
width="125" height="125" />

    <div class="wrapper">
      <HelloWorld msg="You did it!" />

      <nav>
        <RouterLink to="/">Home</RouterLink>
        <RouterLink to="/about">About</RouterLink>
      </nav>
    </div>
  </header>

  <RouterView />
</template>
```

```
<template>
  <router-view/>
</template>
```

ファイルの流れ


index.html
src/main.ts
src/router/index.ts
src/App.vue
viwsディレクトリ
HOMEページ
ABOUTページ
componentsディレクトリ
ヘッダー
自己紹介欄
商品
…
要素をパッチワーク
してページを作成

ページをパッチワーク的に作成

【HomeView.vue】

```vue
<script lang="ts">
    import HeaderView from '@/components/HeaderView.vue'
    import FooterView from '@/components/FooterView.vue'
    import HomeConView from '@/components/HomeConView.vue';

    export default (await import('vue')).defineComponent({
        name: 'HomeView',   // index.tsと名称をリンクさせる
        components: {
            HeaderView,
            HomeConView,
            FooterView
        }
    })
</script>
```

```vue
<template>
    <div>
        <HeaderView />
        <div class="w-80 sm:w-1/2 m-10 mx-auto">
            <HomeConView />
            <AboutConView class="mt-10"/>
        </div>
        <FooterView />
    </div>
</template>
```

コンポーネント（components）にある
各要素を呼び出す

どのような順番で表示させるかを決定

index.tsを変更

```
import { createRouter, createWebHashHistory } from 'vue-router'
import HomeView from '../views/HomeView.vue'
import AboutView from '../views/AboutView.vue'

const router = createRouter({
  history: createWebHashHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'home',
      component: HomeView
    },
    {
      path: '/about',
      name: 'about',
      component: AboutView
    }
  ]
})

export default router
```

HomeView, AboutViewの2ページあるWebページを考える

『createWebHashHistory』にする

- ハッシュ履歴を行う
- historyモードより早い

ページ遷移のためのリンク作成

router-linkタグを使う
index.tsで登録したパス名を使用

【HeaderView.vue】

```
<router-link to="/about" >About</router-link>
```

【index.ts】

```
{
  path: '/about',
  name: 'about',
  component: AboutView
}
```

v-forを使って効率的にリスト作成


```
<div v-for="chara in charas" :key="chara.id" class="chara-display">
    <p class="sub-headline"> {{ chara.name }}</p>
    <p class="text-2xl"> {{ chara.loc }}</p>
</div>
```

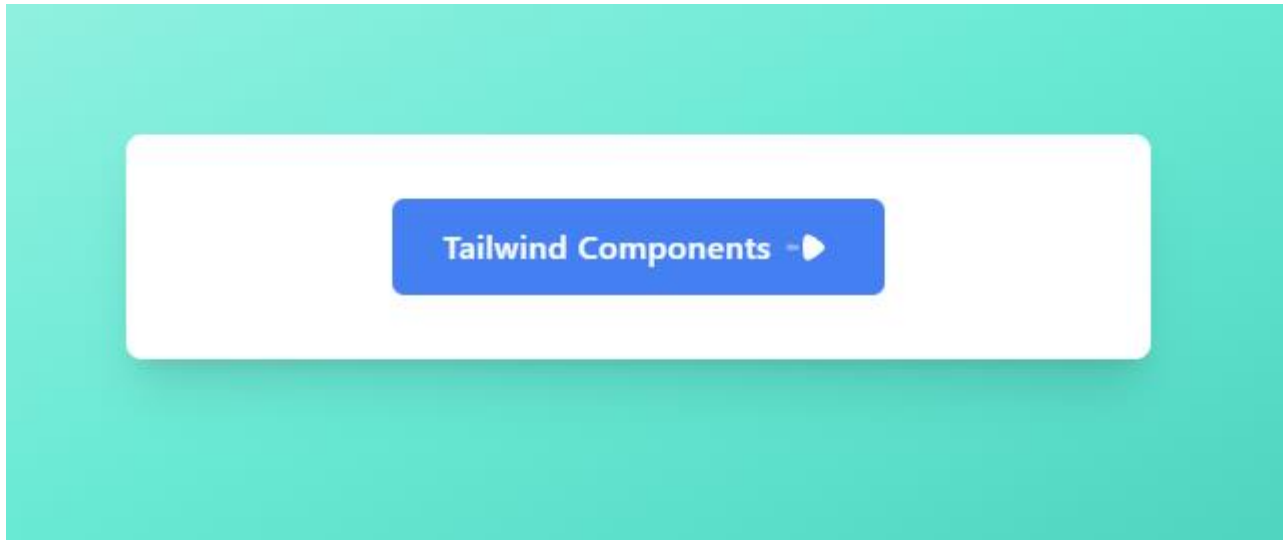
返されたdataを用いて、キャラクターの名前や場所を表示

```
data() {
    return {
        charas: [
            {
                id: 1,
                name: 'くまモン',
                loc: '熊本',
            },
            {
                id: 2,
                name: 'はばたん',
                loc: '兵庫',
            },
            {
                id: 3,
                name: 'オカザえもん',
                loc: '愛知',
            },
        ]
    }
}
```

デザインを考えるのが面倒

テンプレートサイトを見よう

- Tailwind CSS のテンプレートサイトまとめ
  https://qiita.com/Masahiro111/items/f7d6ad8280ae92717f0f
- その他、グーグルで「Tailwind CSS template」と検索してみよう



引用：https://tailwindcomponents.com/component/button-animate-hover

レスポンシブデザイン

PC or スマホ・タブレットで見た場合で、デザインを変更する

→ 現状、スマホでサイトを見る人が多いためスマホ基準でデザイン
（だから、スクロールが長いサイトが多い）

Responsive Design

Using responsive utility variants to build adaptive user interfaces.

Every utility class in Tailwind can be applied conditionally at different breakpoints, which makes it a piece of cake to build complex responsive interfaces without ever leaving your HTML.

There are five breakpoints by default, inspired by common device resolutions:

Breakpoint prefix	Minimum width	CSS
`sm`	640px	`@media (min-width: 640px) { ... }`
`md`	768px	`@media (min-width: 768px) { ... }`
`lg`	1024px	`@media (min-width: 1024px) { ... }`
`xl`	1280px	`@media (min-width: 1280px) { ... }`
`2xl`	1536px	`@media (min-width: 1536px) { ... }`

引用：https://tailwindcss.com/docs/responsive-design

レスポンシブデザインは
『画面の幅』でどの表示すればいいか判定

Tailwind CSSでは全5種の画面幅に対して
対応できるようにしている

Tailwind CSSの独自クラスを作成

@layer components で作成可能 → よく使うデザインは省略して書ける

【index.css】

```
@layer components {
     .chara-display {
          @apply flex flex-col border rounded-lg gap-3 p-4 md:p-6;
     }
}
```

vite.config.tsを書き換え

ページ公開のために、ビルドしたフォルダを「docs」にしたい

```
export default defineConfig({
  base: './',
  build: {
    outDir: '../docs'
  },
  plugins: [vue()],
  resolve: {
    alias: {
      '@': fileURLToPath(new URL('./src',
import.meta.url))
    }
  },
})
```

追加

ビルド

ビルド

```
npm run build
```

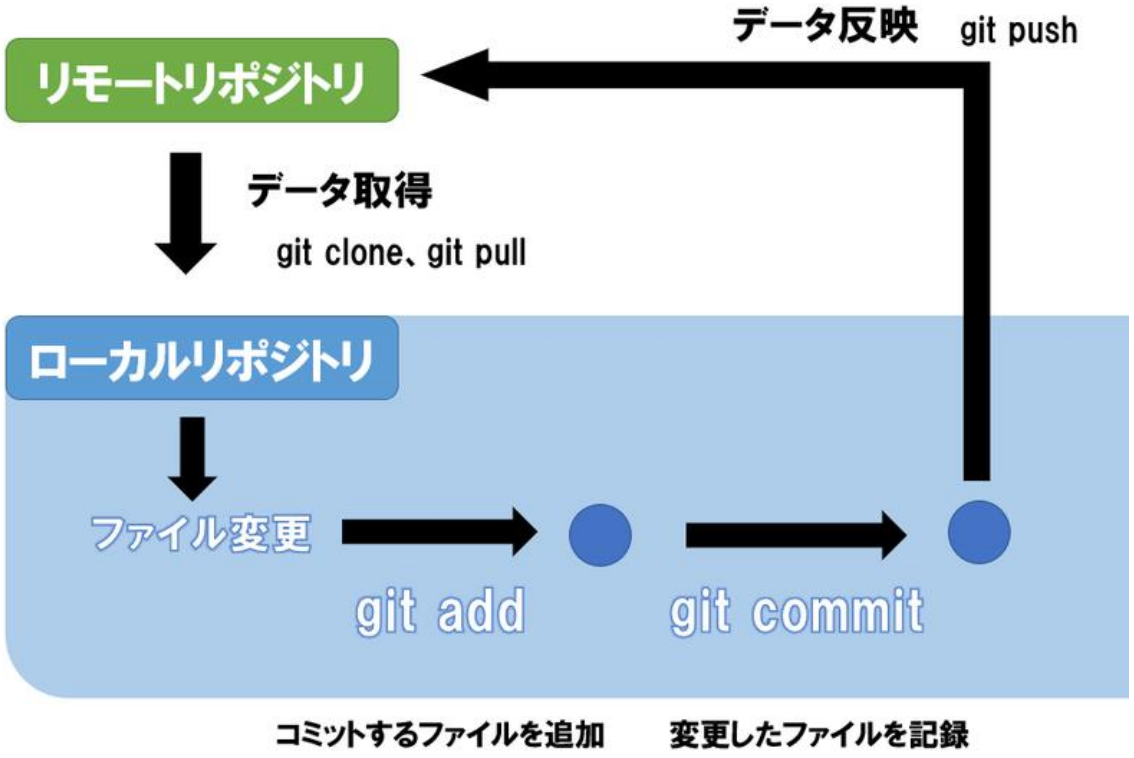
→ docsフォルダを作成

docsフォルダは必ず、「./docs」になるようにする
※「./src/docs」ではいけない

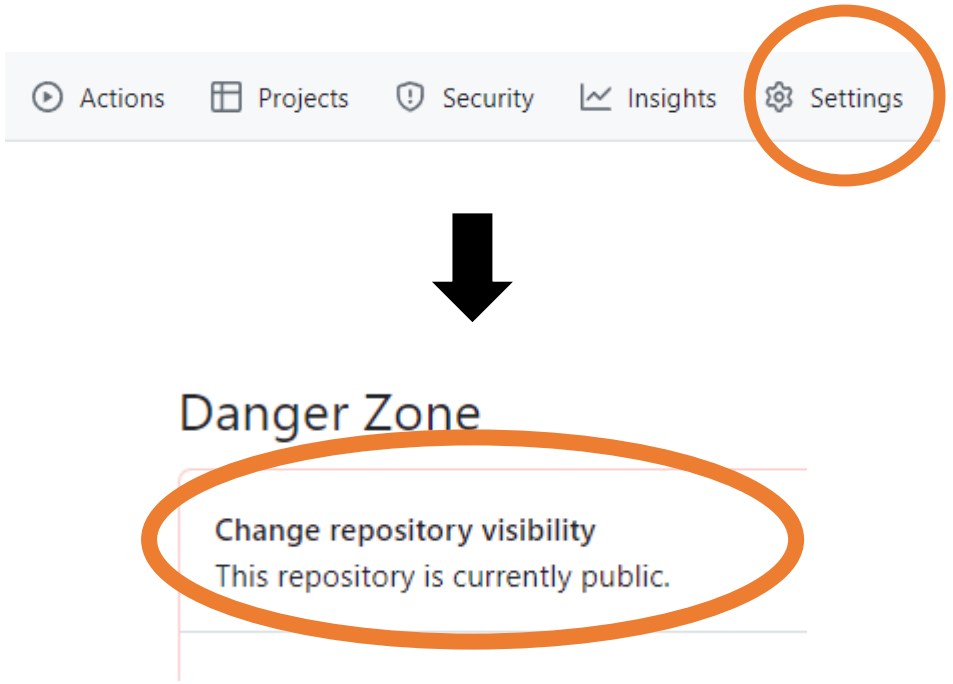
Webページ　公開編

ソースコードを公開

GitHubにpushする

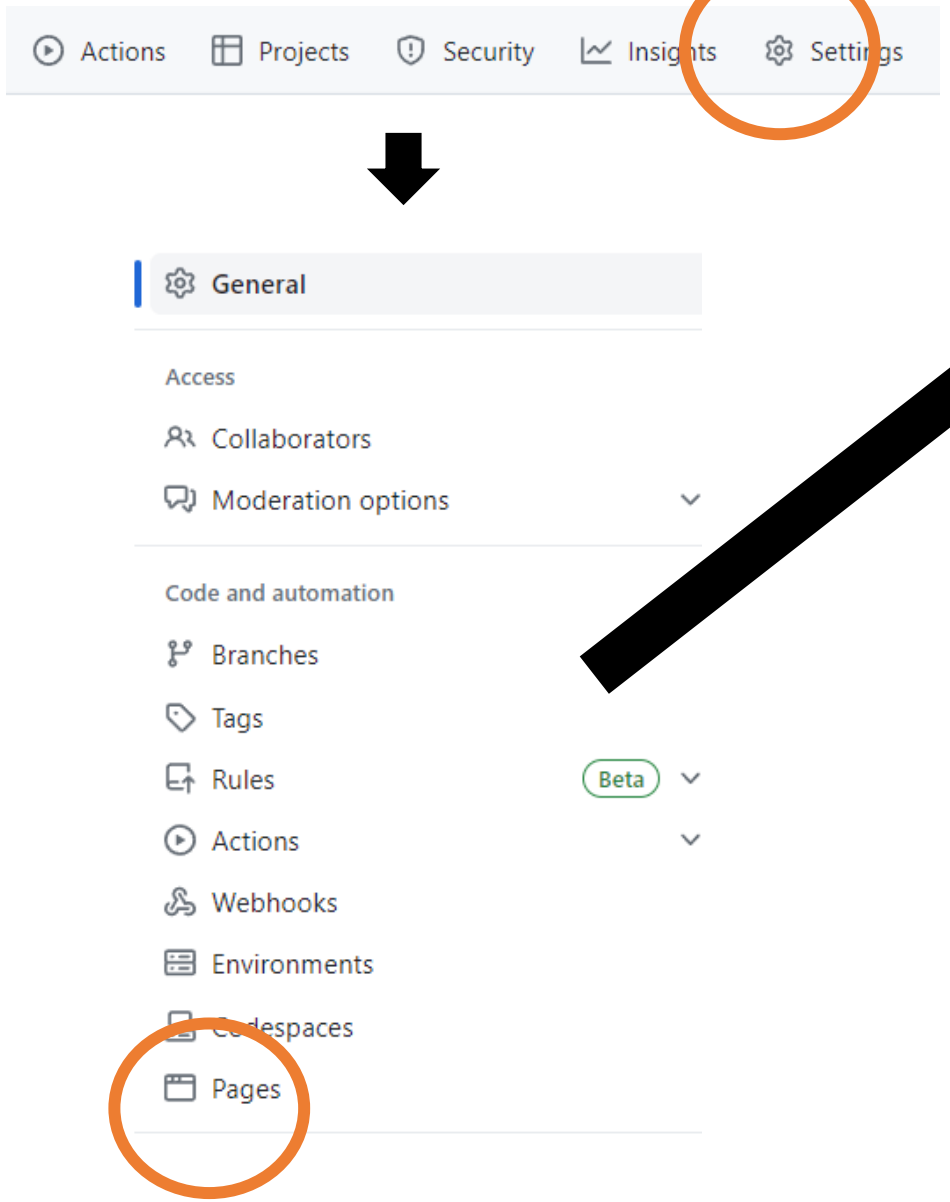


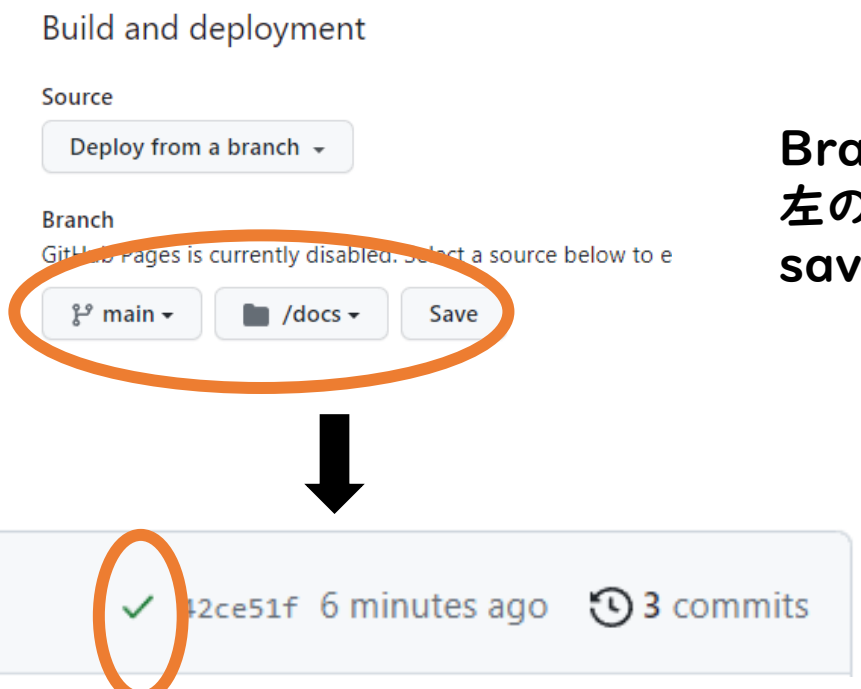
GitHubにpublicにする



privateから、publicにする

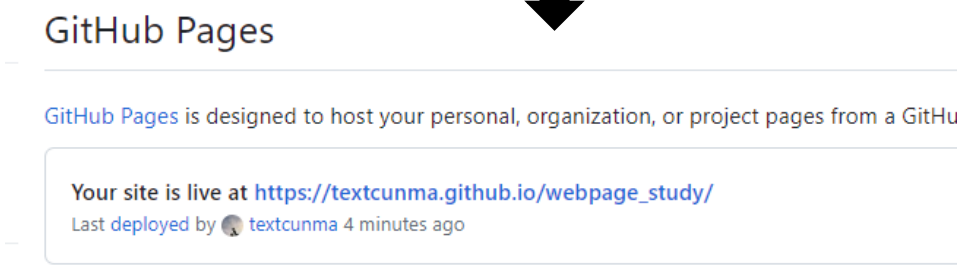
デプロイ（GitHub Pagesを使用）





Branchを
左のように設定して
save

チェックマークがついたら『完了』



設定画面にいくと、
リンクが貼られている

おしまい