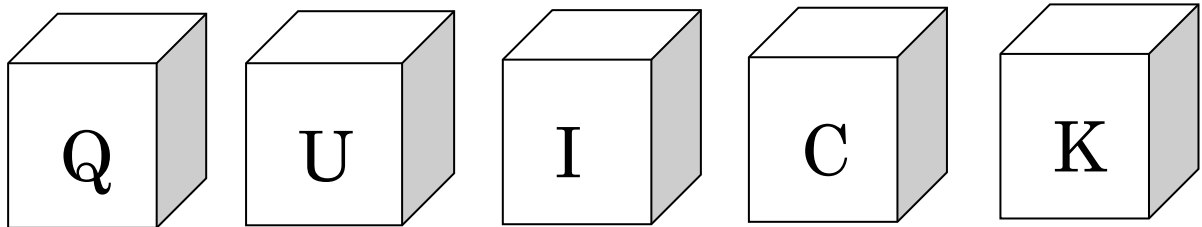


## Quick Check

### ▸ React Series: Basic



## >> React 入門 ～

秒ごとに切り替わるタイマーを表示してみる

(再レンダリングの仕組みと最適化／State 管理 編)

## React 入門 ～ 秒ごとに切り替わるタイマーを表示してみる (再レンダリングの仕組みと最適化 / State 管理 編)

仕事や趣味で Web サイトを使っていて、直接ユーザーの目に触れる UI (フロントエンド) から、こういう UI 格好良いな、使い勝手がよさそう、反応速度が遅い等、UI によってユーザーが Web サイトに対して抱く印象も変わってくるように思います。

「画面操作の際の動きが重たい」などといったフロントエンドのパフォーマンス改善を考える際に、Google が提唱している「Core Web Vitals」と呼ばれる指標があるようです。

- ・ サイト表示の速さ : 「LCP (Largest Contentful Paint)」
- ・ ユーザーアクションへの反応の速さ : 「FID (First Input Delay)」
- ・ レイアウトずれが起きないかどうか : 「CLS (Cumulative Layout Shift)」

こうしたパフォーマンス改善を考えることで、サービスの使いやすさも変わってくるように思います。

フロントエンドのパフォーマンス改善を考えることの題材として、React 基礎の振り返りも含めて、今回、CodeSandbox というオンラインのコードエディタ上で「秒ごとに切り替わるタイマー」を表示することで、React の「再レンダリングと State 管理」について確認をしていきたいと思います。URL での共有、GitHub との連携も行うことができるサービスですので、勉強用として使い勝手も良いと思います。

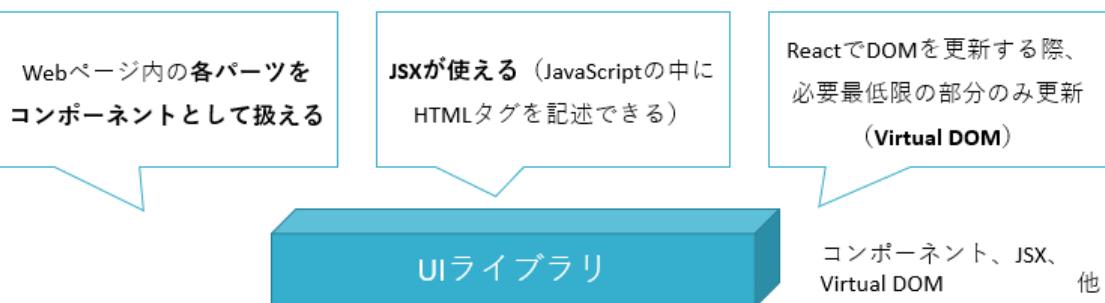
それでは、「React の再レンダリングの仕組みと最適化 / State 管理」についての内容を確認していきたいと思います。CodeSandbox 上で手を動かして React のコードを書いて考える機会にしていければ幸いです。

## ○ モダンな Web 開発のためのライブラリ : React について基本事項レビュー

- ・ React とは？

# React とは？

- ・ Meta社とコミュニティが開発している「**UIライブラリ**」  
～ HTMLをコンポーネントとして定義し、組み合わせて使うことができる
- > 特徴



ここからは始める「React」～基礎事項の確認をしていきます

React で UI コンポーネントを作成する際、JSX 記法というコンテンツを返す (JavaScript に HTML を埋め込む) 関数を定義します。

## JSX記法

- ・ コンポーネントを作るには、「**JSX**」で**コンテンツを返す関数を定義** する

⇒ 関数の返却値として **HTMLのタグを記述し、コンポーネントとして画面を構成**

file: src/index.js

```
import React from "react";
import ReactDOM from "react-dom";

const App = () => {
  return (
    <div>
      <h1>Hello World !</h1>
    </div>
  );
};
```

ReactDOMのrender関数内  
第1引数：render対象  
第2引数：render箇所

関数名をHTMLのようにタグで囲う  
ことでコンポーネントとして扱える

```
ReactDOM.render(<App />,
  document.getElementById("root") );
```

まずは、CodeSandbox で React のプロジェクトを作成し、Hello World を表示させてみましょう。

CodeSandbox のメニュー画面で表示されたテンプレートの中にある [React] をクリック  
一度 src フォルダ内にあるファイルを全て削除し、index.js を新規に作成します。

以下のコードを記載して、「Hello World」を表示してみましょう。

```
JS index.js x
1 import React from "react";
2 import ReactDOM from "react-dom";
3
4 const App = () => {
5   return (
6     <div>
7       <h1>Hello World!</h1>
8     </div>
9   );
10 };
11
12 ReactDOM.render(<App />, document.getElementById("root"));
```

Browser Tests Terminal  
https://swdg01.csb.app/  
Hello World!

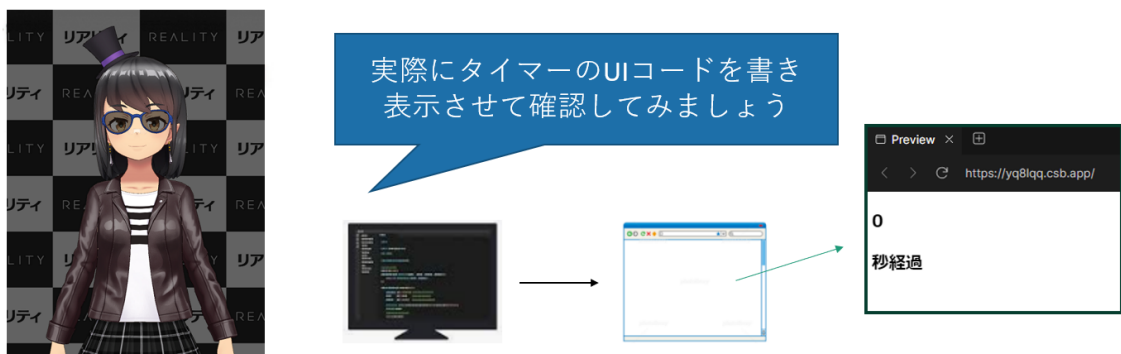
準備が整いましたので

それでは、React で「秒ごとに切り替わるタイマーを作成」していきたいと思います。

○ モダンな Web 開発のためのライブラリ：React について基本事項レビュー

CodeSandBox で、「秒ごとに切り替わるタイマー」を表示してみたいと思います。

## [演習]：Reactで秒ごとに切り替わるタイマーを表示



Point => ☒ コンポーネントの分割

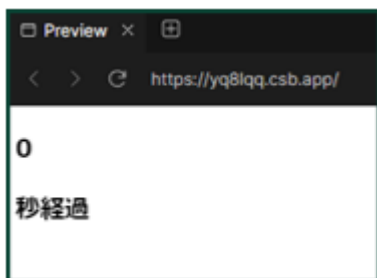
Watch という名前の関数コンポーネントを読み込む

☒ 画面に表示するデータの状態や関数の実行タイミング

React Hook ～ **useState** と **useEffect** の利用

```
src
├── Watch.js ← 追加
└── index.js
```

まずはコンポーネントを分割させて、下のような初期状態を表示させるところまで、トライしましょう。



## Reactで秒ごとに切り替わるタイマーを表示する①

・コンポーネントの分割

Watch という名前の関数コンポーネントを読み込む

src

- Watch.js ← 追加
- index.js

Watch.js

他のファイルでも使えるようにexportする必要がある

```
import { useState } from "react";
export const Watch = () => {
  const [time, setTime] = useState(0);
  return (
    <h3> <time> {time} </time>
      <p> 秒経過 </p>
    </h3>
  );
};
```

index.js

import側で UIコンポーネントを読み込む

```
import ReactDOM from "react-dom";
import { Watch } from "./Watch";

ReactDOM.render( <Watch />,
  document.getElementById("root") );
```

タイマーの処理の書き方としては、

- ・ `setTimeout` ... 一定時間後に一度だけ特定の処理をおこなう
- ・ `setInterval` ... 一定時間ごとに特定の処理を繰り返す

の2つありますが、今回は1秒ごとにカウントしていきますので、「`setInterval`」を利用しましょう。

`window.setInterval` として、このコールバック関数を1秒ごとに実行するようにしましょう。

```
window.setInterval( () => {}, 1000 );
```

```

window.setInterval(() => {
  setTime((prev) => prev + 1);
}, 1000);

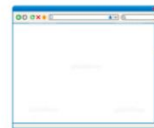
```

この第 1 引数のコールバック関数が 1 秒ごとにこれで実行されますので、この部分で setTime の元々の秒数に+1 して新しいステートとします。

タイマー処理を追加して再度表示させるとどうなるのでしょうか？



実際に先ほど書いたタイマーのUIを表示させて確認してみましょう

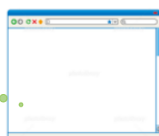


恐らく、いまの状態では、「1秒ごとにカウントされない」と思います  
**useEffect** を利用して「1秒ごとに」カウントされるようにしましょう！

・ useEffect とは？

## useEffect とは

- ・ useEffect は、関数を実行するタイミングをReactレンダリング後に遅らせる  
 ☞ 画面リロード時など再レンダリングが起きて欲しくない タイミング の処理を制御  
 useEffectは、「初回レンダリング後」と「依存配列にある何かに変更された時」に実行される



cf) DOMの変更、ロギング、APIからデータを取得 etc...  
 ⇒ 副作用（関数の実行によって発生する外部の変更）を扱う

▶ useEffect ( ReactHooksと総称される機能群にある関数 )

**useEffect { 実行する関数, [ 依存配列 ] ;**

▶ useEffect の import  
**import { useEffect } from "react"**

1つ目：コールバック変数

2つ目：useEffect が監視する値のリスト

先ほどは、

`setTime` によって再レンダリングされる際に、コンポーネント自体が改めて実行されることになり、  
ので、何個も `setInterval` が実行される状態となり、秒数がおかしくカウントされてしまいます。

そのような状況を解決する為に、`useEffect` を利用します。

初期化時に 1 度だけ実行されれば良いという場合には、`useEffect` の第 2 引数を `{}` 空にして記述しましょう。

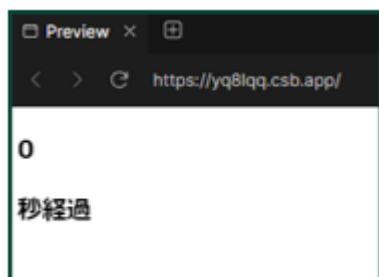
以下のようにタイマー処理を追記して、再度表示してみましょう。

```
import { useEffect, useState } from "react";

export const Watch = () => {
  const [time, setTime] = useState(0);

  useEffect(() => {
    window.setInterval(() => {
      setTime((prev) => prev + 1);
    }, 1000);
  }, {});

  return (
    <h3>
      <time>{time}</time>
      <p>秒経過</p>
    </h3>
  );
};
```



初期画面から 1 秒ごとカウントされていることが確認できると思います。

=====  
今回の内容はいかがだったでしょうか。スクリプトを実行しながら動作確認できると面白いなと感じて頂けた方もいらっしゃるかもしれません。自分にできる範囲のものから少しずつ **JavaScript** にも挑戦してみようかなと思っていただければ幸いです。

以上となります。

参考文献：

- ・クジラ飛行機『いまどきの JS プログラマーのための Node.js と React アプリケーション開発テクニック』（第 5 版） ソシム株式会社（2020 年）
- ・岡田 拓巳『モダン JavaScript の基本から始める **React** 実践の教科書 』Informatics&IDEA（2021 年）