【2024年最新】React(v18)完全入門ガイド｜Hooks、Next14、Redux、TypeScript @Udemy
困った時は ChatGPT 使う！

---

セクション2：

4. ★重要★コースで使用するVSCodeの拡張機能を準備しよう

・Prettier - Code formatter
・Live Server
・Auto Rename Tag
・ES7+ React/Redux/React-Native snippets
→ 一番下に「Snippets」のリンク有。リンク先にReactのスニペット（ショートカット）を使えるようにする

セクション4：

22. Reactを動かしてみよう

```
<head>
  <script src="../../../../libs/react.development.js"></script>
  <script src="../../../../libs/react-dom.development.js"></script>
  <script src="../../../../libs/babel-standalone.js"></script>
</head>
<body>
  <div id="app"></div>

  <script type="text/babel">
    // POINT ReactをHTMLにマウントする書き方
    const appEl = document.querySelector("#app");
    const root = ReactDOM.createRoot(appEl);
    root.render(<h1>こんにちは</h1>);
                    ↑この書き方は JSX （javascript内にhtmlを書くことをJSX）
    // POINT React18以前の書き方
    // ReactDOM.render(<h1>こんにちは</h1>, appEl)
  </script>
</body>
```

23. Reactコンポーネントって何？コンポーネントを定義してみよう

**コンポーネントはJSXを返す**　　**コンポーネントの実行**

★重要★ コンポーネントの先頭は大文字とする
OK: Example / NG: example

---

1:コンポーネントの書き方

```
function Example() {
    return <h1>Hello Components</h1>;
}
```

---

2:コンポーネントの書き方　アロー関数

```
const Example = () => {
    return <h1>Hello Components</h1>
};
```

★中身がreturn文のみ(1行？)なら、以下のように書ける★

```
const Example = () => <h1>Hello Components</h1>;
```

---

▼JSXが複数行の時は( )で括る

```
const Example = () => {
  return (
    <div>
      <h1>Hello Components</h1>
    </div>
  );
}
```

↓↓↓ ★上記はreturn分しかないので、{ } return 消してもOK★

```
const Example = () => (
  <div>
    <h1>Hello Components</h1>
  </div>
);
```

---

注意：以下の書き方だと undefined になる！

```
const a = () => {
  return  ← ;が無いと「return; ←このようになっている」/ ;が隠れている状態になっている
  ("戻り値");
};
```

↓↓↓ ★returnの後に書けばOK★

```
<body>
    <div id="app"></div>
    <script type="text/babel">
        const appEl = document.querySelector('#app');
        const root = ReactDOM.createRoot(appEl);
        function Example() {
            return <h1>Hello React</h1>;
        }
        root.render(<Example />);
    </script>
</body>
```

▼表示結果　※上記の場合、正しくは Hello React です

# Hello Components

```
const a = () => {
  return ("戻り値");
};

console.log(a());
```

25. Reactのプロジェクトの作成方法

★注意★ ※24.のコース
本コースのプロジェクトの雛形ですが、create-react-appからviteに変更
Reactの雛形のプロジェクトの作成はcreate-react-appが非推奨となり、viteで作成することが推奨されています。

<create-react-app のプロジェクトの作成方法>

1. create-react-app コマンドを使用して新しいReactプロジェクトを作成
コマンド：npx create-react-app my-test-app　★ my-test-app がプロジェクト名になる ★
このコマンドにより、my-test-appという名前のディレクトリが作成され、
その中にReactプロジェクトの初期設定が行われます。
↓
2. プロジェクトが作成されたら、そのディレクトリに移動します。
コマンド：cd my-test-app
↓
3. Reactの開発サーバーを起動して、アプリケーションをブラウザで確認します。
コマンド：npm start

※この枠内、https://qiita.com/kimascript/items/c390dc0998f8e81d49ab から引用

<vite のプロジェクトの作成方法>
1. 以下のコマンドを使用して新しいReactプロジェクトを作成
コマンド：npm create vite@latest
↓
上記のコマンドたたくと、以下のように name名はどうする？ frameworkはどうする？ など選択肢が出てくる

```
  create-vite@latest
Ok to proceed? (y) y
✔ Project name: … test-project
✔ Select a framework: › react
```

```
✔ Select a framework: › react
? Select a variant: › - JavaScript
```

↓

2. プロジェクトが作成されたら、そのディレクトリに移動します。

コマンド：cd test-project

↓

3. npm install する

コマンド：npm install

↓

4. npm run start する

コマンド：npm run start

※一部参考サイト： https://www.webcreatorbox.com/blog/vite-react

※上記でプロジェクトが立ち上がる！

↓

コマンド打った下の方にurlが書いてある

例）Local: http://localhost:3000

<プロジェクトの終了方法>　※立ち上がっているサーバー止める方法

Ctrl + C

↓

バッチ ジョブを終了しますか（Y/N）？　←と表示されるが、

↓

もう一回　Ctrl + C　で止まる！

---

補足：package.json の中のscriptsにコマンドが書いてある

```
  "scripts": {
    "dev": "vite",
    "start": "vite",
    "build": "vite build",
    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0",
    "preview": "vite preview"
  },
```

---

26. ★重要★本コースで使用するプロジェクトの使い方

package.json を見つけた階層まで移動

↓

npm install する

04_react_basic> npm install

```
C:\react-udemy\react-guide-material\04_react_basic> npm install
```

↓

npm start する

04_react_basic> npm start

```
C:\react-udemy\react-guide-material\04_react_basic> npm start
```

でサーバーを立ち上げる！

27. コンポーネントにスタイルを当ててみよう

★スタイルの当て方は色々ある★

<一番ベーシックな方法>

★POINT★ クラス名は class ではなく className で指定する

HTML のように見えますが、これは JSX と呼ばれる構文です。class 属性は className を使用しましょう。

```
<div className="component">
  <h3>Hello Component</h3>
</div>
```

↓

★POINT★ CSS ファイルを import する

```
import "./Example.css";
```

▼Example.css ファイル

```
.component {
  padding: 1rem;
  color: blue;
  border: 5px solid blue;
}
```

28. コンポーネントの分割方法

★export 何通りかある★

<名前付きexport> List.jsx 作成 2-1

```
const List = () => {
  return (
    <ul>
      <li>item-1</li>
      <li>item-2</li>
      以下、省略
    </ul>
  );
};
```

★POINT★ List コンポーネントを 名前付きexport

```
export { List };
```

---

補足： 以下のように、定数 a なども追加可能

```
const a = 0;
export { List, a };
```

or

<名前付きexport の別の記法> 2-2

```
export const List = () => {
  return (
    <ul>
      <li>item-1</li>
      <li>item-2</li>
      以下、省略
    </ul>
  );
};
```

**Hello Component**

- item-1
- item-2
- item-3
- item-4
- item-5

↓

▼Example.jsx ファイル / List を import

```
import "./Example.css";
import { List } from "./components/List";

const Example = () => {
  return (
    <div className="component">
      <h3>Hello Component</h3>
      <List />
    </div>
  );
};

export default Example;
```

<以下の赤枠 return () 部分をコンポーネント化してみるver>

```
components
List.jsx
Example.css
Example.jsx
073_practice_compo...
075_fragment
080_expr_in_jsx
085_expr_and_state
087_practice_expr
```

```jsx
  return (
    <div className="component">
      <h3>Hello Component</h3>
      <List />
    </div>
  );
};

export default Example;
```

↓

↓　ファイル作成、名前など変更 ※画像参照

↓

```
react-guide-material
04_react_basic
src
070_component_nest
end
components
Child.css
Child.jsx
List.jsx
Example.jsx
start
073_practice_compon...
075_fragment
080_expr_in_jsx
085_expr_and_state
```

```jsx
const List = () => {
  return (
    <ul>
      <li>item-1</li>
      <li>item-2</li>
      <li>item-3</li>
      <li>item-4</li>
      <li>item-5</li>
    </ul>
  );
};

// POINT List コンポーネントを 名前付き export
export { List };
```

```
070_component_nest
end
components
Child.css
Child.jsx
List.jsx
Example.jsx
```

```
.App-end .component {
  padding: 1rem;
  color: blue;
  border: 5px solid blue;
}
```

```
react-guide-material
04_react_basic
src
070_component_nest
end
components
Child.css
Child.jsx
List.jsx
Example.jsx
start
```

```jsx
// POINT List コンポーネントを名前付き import
import { List } from "./List";
import "./Child.css";

const Child = () => {
  return (
    <div className="component">
      <h3>Hello Component</h3>
      <List />
    </div>
  );
```

```
> 073_practice_compon...
> 075_fragment
> 080_expr_in_jsx
> 085_expr_and_state
```

```
};

// POINT コンポーネントを default export
export default Child;
```

★重要★ export default を読み込む場合は { } は削除する

正: `import Child from "./components/Child";`

誤: `import { Child } from "./components/Child";`

```
04_react_basic
src
070_component_nest
end
components
Child.css
Child.jsx
List.jsx
Example.jsx
start
```

```
import Child from "./components/Child";

const Example = () => <Child />;
// const Example = () => {
//   return <Child />
// };

export default Example;
```

**Hello Component**

- item-1
- item-2
- item-3
- item-4
- item-5

名前付きexport と export default どちらを使えば良いのか？

↓

```
export { List };
```

```
export default Child;
```

基本は export default を使えばよい！

30. 不要なタグを出力しないFragmentの使い方

▼以下の書き方はエラーになる
同じ階層に3つのタグが書かれている！

```
const Child = () => {
  return (
    <div className="component">
      <h3>Hello Component</h3>
    </div>

    <h3>Hello</h3>
    <p>テストテスト</p>
  );
```

▼ルート要素が１つならOK！ で一つにまとめる！
だが、不要の は書きたくない！無駄な 。

```
const Child = () => {
  return (
    <div>
      <div className="component">
        <h3>Hello Component</h3>
      </div>

      <h3>Hello</h3>
      <p>テストテスト</p>
```

```
};
```

```
    </div>
  );
};
```

↓

上記右の場合だと、不要の (ルート要素)がある。無駄な 。

↓

★重要★ 一般的な書き方

(ルート要素) 書きたくない場合、<React.Fragment> を使う！

この <React.Fragment> は html に表示されない！

あと一番上に import を追記する！

```
import React from "react";    ←追記する！

const Child = () => {
  return (
    <React.Fragment>
      <div className="component">
        <h3>Hello Component</h3>
      </div>

      <h3>Hello</h3>
      <p>テストテスト</p>
    </React.Fragment>
  );
};
```

↓

↓

★重要★ 上記の補足 / 以下のようにも書ける

※{ Fragment } と <Fragment> の部分変更

```
import { Fragment } from "react";

const Child = () => {
  return (
    <Fragment>
      <div className="component">
        <h3>Hello Component</h3>
      </div>

      <h3>Hello</h3>
      <p>テストテスト</p>
    </Fragment>
```

→ → → →

★重要★ 左、さらに補足 / 以下のようにも書ける

<Fragment> → <> 省略することも可能！

その場合、以下の import 部分は不要になる！

~~import { Fragment } from "react";~~    ←不要×

```
const Child = () => {
  return (
    <>
      <div className="component">
        <h3>Hello Component</h3>
      </div>

      <h3>Hello</h3>
      <p>テストテスト</p>
    </>
```

```
  );
};
```

```
  );
};
```

★重要★
<Fragment> は属性を付けることは出来ない！<Fragment> は消えてしまうので。
例：<Fragment  className=" ">

ただ1点だけ付けれる属性あり！
key="" は付けること可能！ループの際に必要。次の？セクションで説明。
<Fragment key="">

★重要★
Fragmentがなぜ必要なのか？
Reactのコンポーネントはルート要素が１つでなければならない（独立したツリー構造になっている）というルールがあります。
そのため、Fragmentを使うことで、複数の要素を返すコンポーネントを余分なノードを追加することなくまとめることができます。

31. JSX内でJSコードを実行してみよう

▼Expression.jsx内

{ }の中には javascriptのコードを書き込むことが出来る！

```
import "./Expression.css";

const Expression = () => {
  const title = "Expression";
  return <h3>Hello {title}</h3>
};

export default Expression;
```

▼Example.jsx内

メソッドも実行できる！

```
import "./Expression.css";

const Expression = () => {
  const title = "Expression";
  return (
```

class名が Expression → expression 全て小文字になっている！

```
    <div className={title.toLowerCase()}>
      <h3>Hello {title}</h3>
    </div>
  );
};

export default Expression;
```

→ →

```
▼<div class="expression">
  ▼<h3>
     "Hello "
     "Expression"
   </h3>
 </div>
```

```
const Expression = () => {
  const title = "Expression";
  const arry = ["item1", "item2", "item3"];

  return (
    <div className={title.toLowerCase()}>
      <h3>Hello {title}</h3>
      <h3>{arry}</h3>
    </div>
  );
};
```

配列の中身が展開されて表示される！

関数を呼び出して return した値を埋め込むこともできます。

```
const Expression = () => {
  const hello = (arg) => `${arg} Function`;
  // const hello = (arg) => {
  //   return `${arg} Function`;
  // };

  return (
    <div>
      <h3>{hello("Hello")}</h3>
    </div>
  );
};
```

←こちらは省略しないver

コメント 書き方

```
  return (
    <div>
      <h3>{hello("Hello")}</h3>
      <h3>{/* 画面上に表示されません */}</h3>
    </div>
```

```
▼<div>
   <h3>Hello Function</h3>
   <h3></h3> == $0
 </div>
```

```
  );
```

波括弧内{ } に JSX を記述することもできます。

```
  return (
    <div>
      {<h3>JSXです！</h3>}
    </div>
  );
```

※ javascript のコードとみなされる。

JSXです！

```
▼<div>
  <h3>JSXです！</h3>
</div>
```

変数に代入したJSXも埋め込めます。

```
const Expression = () => {
  const jsx = <h3>Hello JSXだよ</h3>;

  return (
    <div>
      {jsx}
    </div>
  );
};
```

Hello JSXだよ

```
▼<div>
  <h3>Hello JSXだよ</h3>
</div>
```

32. 【TIPS】式と文の違い

式：何らかの値を返すもの（変数に代入できるもの）

文：変数宣言、for文、if文、switch文やセミコロンで区切るもの。

< if文 >

▼return の中に以下のような if文 はエラーになる！ →

```
const Child = () => {
  return (
    <div className="component">
      <h3>式と文</h3>
      {if(true) { 'hello' }}
    </div>     ↑エラーになる
  );
};
```

▼三項演算子 の書き方ならOK！ →

```
const Child = () => {
  return (
    <div className="component">
      <h3>式と文</h3>
      {true ? 'hello' : 'bye'}
    </div>
  );
};
```

▼return の外に書けばOK！

```
const Child = () => {
  if( ) {
  }

  return (
    <div className="component">
      <h3>式と文</h3>
    </div>
  );
};
```

< for文 >

▼return の中に以下のような for文 はエラーになる！

```
const Child = () => {
  return (
    <div className="component">
      <h3>式と文</h3>
      {for(let i = 0; i < 5; i++) {
              ↑エラーになる↓
      }}
    </div>
  );
};
```

▼return の外に書けばOK！ if文も同様！

```
const Child = () => {
  for(let i = 0; i < 5; i++) {
    const a = 0;
  }

  return (
    <div className="component">
      <h3>式と文</h3>
      { a }
    </div>
  );
};
```

34. propsでコンポーネントに値を渡してみよう

▼color="red" 記述

```
const Example = () => <Child color="red" />;
```

▼props 記述

```
const Child = (props) => {
```

▼console.log(props);
color:"red" が渡っている

```
▼ {color: 'red'}
    color: "red"
  ▶ [[Prototype]]: Object
```

★補足★
props → p でもOK

```
# App.css
App.jsx
BaseErrorBoundary.jsx
```

```jsx
export default Child;
```

↓ ----------

color="red" 設定

```
090_props
start
components
# Child.css
Child.jsx
Example.jsx
100_practice_props
```

```jsx
import Child from "./components/Child";

const Example = () => <Child color="red" />;

export default Example;
```

▼className={`component ${props.color}`}　　props.color 設定

```
src
090_props
start
components
# Child.css
Child.jsx
Example.jsx
100_practice_props
110_props_children
120_props_rules
130_whats_jsx
```

```jsx
import "./Child.css";

const Child = (props) => {
  return (
    <div className={`component ${props.color}`}>
      <h3>Hello Component</h3>
    </div>
  );
};

export default Child;
```

```
react-guide-material
04_react_basic
src
090_props
start
components
# Child.css
Child.jsx
Example.jsx
100_practice_props
```

```css
.App-start .component {
  padding: 1rem;
  color: blue;
  border: 5px solid blue;
}

.App-start .component.red {
  color: red;
  border: 5px solid red;
}
```

Hello Component

↓ ----------

▼2個記述することで、以下のような結果になる！

```jsx
<Child color="" />
<Child color="red" />
```

```
react-guide-material
04_react_basic
src
090_props
start
components
# Child.css
Child.jsx
Example.jsx
```

```jsx
import Child from "./components/Child";

const Example = () => {
  return (
    <>
      <Child color="" />
      <Child color="red" />
    </>
  )
```

Hello Component

Hello Component

```
> 100_practice_props
> 110_props_children
> 120_props_rules
> 130_whats_jsx
```

```
10  };
11
12  export default Example;
```

---

▼分割代入（上で行ったことを分割代入でやってみる）　※Propsを分割代入で受け取る！

```
const Child = ({ color }) => {
    ～
  <div className={`component ${color}`}>
```

←★注意★ { color } は javascript の機能(分割代入)！React の機能ではない。

★結果は上記と同じ

```
src
  090_props
    start
      components
        # Child.css
        Child.jsx
      Example.jsx
    100_practice_props
    110_props_children
    120_props_rules
    130_whats_jsx
    140_react_eleme...
```

```
3   import "./Child.css";
4
5   const Child = ({ color }) => {
6
7     return (
8       <div className={`component ${color}`}>
9         <h3>Hello Component</h3>
10      </div>
11    );
12  };
13
14  export default Child;
```

---

※分割代入 / デフォルト値の設定も可能　※Propsのデフォルト値を設定する！

```
const Child = ({ color = 'green' }) => {
```

```
src
  090_props
    start
      components
        # Child.css
        Child.jsx
      Example.jsx
    100_practice_props
    110_props_children
    120_props_rules
    130_whats_jsx
    140_react_eleme...
```

```
3   import "./Child.css";
4
5   const Child = ({ color = 'green' }) => {
6
7     return (
8       <div className={`component ${color}`}>
9         <h3>Hello Component</h3>
10      </div>
11    );
12  };
13
14  export default Child;
```

```
react-guide-material
  04_react_basic
    src
      090_props
        start
          components
            # Child.css
            Child.jsx
          Example.jsx
        100_practice_props
        110_props_children
        120_props_rules
```

```
1   .App-start .component {
2     padding: 1rem;
3     color: blue;
4     border: 5px solid blue;
5   }
6
7   .App-start .component.red {
8     color: red;
9     border: 5px solid red;
10  }
11
12  .App-start .component.green {
```

<Child /> ←colorを設定しないと、上記の green になる！

```
react-guide-material
04_react_basic
src
085_expr_and_sta...
087_practice_expr
090_props
end
start
components
Child.css
Child.jsx
Example.jsx
100_practice_props
```

```
import Child from "./components/Child";

const Example = () => {
  return (
    <>
      <Child />
      <Child color="red" />
    </>
  )
};

export default Example;
```

```
120_props_rules
130_whats_jsx
140_react_eleme...
App.css
```

```
color: green;
border: 5px solid green;
}
```

Hello Component

Hello Component

---

※補足※ 上記を以下のように設定することも可能。color: c = 'green' の箇所。

```
const Child = ({ color: c = 'green' }) => {

  return (
    <div className={`component ${c}`}>
```

```
src
090_props
start
components
Child.css
Child.jsx
Example.jsx
100_practice_props
110_props_children
120_props_rules
130_whats_jsx
140_react_eleme...
```

```
import "./Child.css";

const Child = ({ color: c = 'green' }) => {

  return (
    <div className={`component ${c}`}>
      <h3>Hello Component</h3>
    </div>
  );
};

export default Child;
```

▼結果は上記と同じ

Hello Component

Hello Component

35. propsに色々な値を渡してみよう

▼Props の受け取り方

親コンポーネントで属性値のように記述した各値は、１つのオブジェクトとして子コンポーネントで受け取ることができる。

```
react-guide-material
import Child from "./components/Child";
```

```
04_react_basic
src
090_props
start
components
Child.css
Child.jsx
Example.jsx
100_practice_props
110_props_children
120_props_rules
130_whats_jsx
140_react_eleme...
App.css
App.jsx
BaseErrorBounda...
index.css
lectures.js
```

```jsx
const Example = () => {
  const hello = (arg) => `Hello ${arg}`;

  return (
    <>
      <Child
        num={123}
        fn={hello}
        bool
        obj={{ name: 'Tom', age: 18 }}
      />
    </>
  )
};

export default Example;
```

★ propsには全てのタイプの値を渡すことができます。

```
react-guide-material
04_react_basic
src
090_props
start
components
Child.css
Child.jsx
Example.jsx
100_practice_props
110_props_children
120_props_rules
130_whats_jsx
140_react_eleme...
App.css
App.jsx
BaseErrorBounda...
index.css
```

```jsx
import "./Child.css";

const Child = ({ color: c = 'green', num, fn, bool, obj }) => {
  console.log(bool);

  return (
    <div className={`component ${c}`}>
      <h3>Hello Component</h3>
      <h3>{num}</h3>
      <h3>{fn('Props')}</h3>
      <h3>{ bool ? 'true' : 'false'}</h3>
      <h3>{ obj.name + obj.age}</h3>
    </div>
  );
};

export default Child;
```

**Hello Component**

**123**

**Hello Props**

**true**

**Tom18**

---

★補足★ 上記を以下のようにも書ける
追記：`color: "red",`　変更：`num: 123`

```
react-guide-material
```

```jsx
import Child from "./components/Child";
```

```
04_react_basic
src
090_props
end
start
components
Child.css
Child.jsx
Example.jsx
100_practice_props
110_props_children
120_props_rules
130_whats_jsx
140_react_eleme...
App.css
App.jsx
BaseErrorBounda...
index.css
lectures.js
main.jsx
.eslintrc.cjs
.gitignore
```

```
import Child from "./components/Child";

const Example = () => {
  const o = {
    color: "red",
    num: 123
  }

  return (
    <>
      <Child
        {...o}

        // 以下でもOKだが面倒くさい
        // color={o.color}
        // num={o.num}
      />
    </>
  )
};

export default Example;
```

```
const o = {
  color: "red",
  num: 123
}
```

このように書ける！{...o}　※ ... はスプレッド演算子

▼以下でもOK！
color={o.color}
num={o.num}

123

37. 特別なプロパティ ~ props.children

★children を使う場合、開始タグ、終了タグを分けて間にchildrenに渡したい値を設定

```
<Container title="Childrenとは？" />
```

→

```
<Container title="Childrenとは？">
  {1}
</Container>
```

★POINT★ コンポーネントの子要素は props.children に渡る！
コンポーネントが子要素を持つ場合、props.children という特別なプロパティとして受け渡されます。

★コンポーネントの中に他のコンポーネントが書ける★
※1.props.children
▼Profile(子)コンポーネントが Container(子)コンポーネントの children(<Profile/>) に渡ってくる

react-guide-material
04_react_basic
src

```
import Profile from "./components/Profile";
import Container from "./components/Container";
```

▼1.Profile(子)コンポーネント

```
110_props_children
  start
    components
      Container.css
      Container.jsx
      Profile.css
      Profile.jsx
    Example.jsx
  120_props_rules
  130_whats_jsx
  140_react_elemen...
App.css
```

```jsx
const Example = () => {
  return (
    <div>
      <Container title="Childrenとは？">
        <Profile />
      </Container>
    </div>
  );
};

export default Example;
```

```
react-guide-material
04_react_basic
  src
    110_props_children
      start
        components
          Container.css
          Container.jsx
          Profile.css
          Profile.jsx
        Example.jsx
      120_props_rules
      130_whats_jsx
```

```jsx
import "./Profile.css";

const Profile = ({ name, age, country, color }) => {
  return (
    <div className={`profile ${color}`}>
      <h3>Name: {name}</h3>
      <p>Age: {age} </p>
      <p>From: {country}</p>
    </div>
  );
};

export default Profile;
```

▼上記のProfile(子)コンポーネントを children として受け取る

<u>★POINT★ props.children として子要素を受け取る</u>

<u>ここでは分割代入を使用して、 props.children を取り出しています。</u>

```
react-guide-material
04_react_basic
  src
    110_props_children
      start
        components
          Container.css
          Container.jsx
          Profile.css
          Profile.jsx
        Example.jsx
      120_props_rules
      130_whats_jsx
```

```jsx
import "./Container.css";

const Container = ({ title, children }) => {
  console.log(children);
  return (
    <div className="container">
      <h3>{title}</h3>
      {children}
    </div>
  );
};

export default Container;
```

↓続く ----------

★補足★

<Profile/> 追加すれば2個表示される(赤枠)

```
Profile.css
Profile.jsx
Example.jsx
120_props_rules
130_whats_jsx
140_react_elemen...
```

```jsx
const Example = () => {
  return (
    <div>
      <Container title="Childrenとは？">
        <Profile/>
        <Profile/>
```

```
App.css
App.jsx
BaseErrorBoundar...
index.css
```

```
    </Container>
    </div>
  );
};
```

Age:

From:

↓続く ----------

★ここ重要？★

`const profile = [` ～ の値を渡すことが可能　　※コンポーネントの流れ(ファイル受け渡しは「※1.props.children」に記述)

```
react-guide-material
04_react_basic
src
087_practice_expr
090_props
100_practice_props
110_props_children
end
components
Example.jsx
start
components
Container.css
Container.jsx
Profile.css
Profile.jsx
Example.jsx
120_props_rules
130_whats_jsx
140_react_elemen...
```

```jsx
import Profile from "./components/Profile";
import Container from "./components/Container";

const profile = [
  { name: "Takashi", age: 19, country: "Japan" },
  { name: "Jane", age: 28, country: "UK", color: "red" },
];

const Example = () => {
  return (
    <div>
      <Container title="Childrenとは？">
        <Profile {...profile[0]}/>
        <Profile {...profile[1]}/>
      </Container>
    </div>
  );
};

export default Example;
```

Childrenとは？

Name: Takashi

Age: 19

From: Japan

Name: Jane

Age: 28

From: UK

```
react-guide-material
04_react_basic
src
110_props_children
start
components
Container.css
Container.jsx
Profile.css
Profile.jsx
Example.jsx
```

```jsx
import "./Container.css";

const Container = ({ title, children }) => {
  console.log(children);
  return (
    <div classNameNameName
```

```
export default Container;
```

```
import "./Profile.css";

const Profile = ({ name, age, country, color }) => {
  return (
    <div className={`profile ${color}`}>
      <h3>Name: {name}</h3>
      <p>Age: {age} </p>
      <p>From: {country}</p>
    </div>
  );
};

export default Profile;
```

↓続く ----------

★補足★

上記、以下のように children={[]}配列使うことも可能

ただ、配列で渡す場合だとコンソールエラーが発生。対処法は下のセクションのループの所で書く。

```
Warning: Each child in a list should have a unique "key" prop.
Check the render method of `Example`. See
```

```
const Example = () => {
  return (
    <div>
      <Container title="Childrenとは？" children={
        [
          <Profile {...profile[0]}/>,
          <Profile {...profile[1]}/>
        ]
      } />
    </div>
  );
};
```

```
const Example = () => {
  return (
    <div>
      <Container title="Childrenとは？" children={
        [
          <Profile {...profile[0]}/>,
          <Profile {...profile[1]}/>
        ]
      } />
    </div>
  );
};
```

▼key={}を書けばエラーが消える！対処法は下のセクションのループの所で書く。

```
<Profile key={profile[0].name} {...profile[0]} />,
<Profile key={profile[1].name} {...profile[1]} />
```

```
<Profile key={profile[0].name} {...profile[0]} />,
<Profile key={profile[1].name} {...profile[1]} />
```

↓続く ----------

▼first, second など使って、個別に渡せることも可能(赤枠)

```
react-guide-material
04_react_basic
src
085_expr_and_state
087_practice_expr
090_props
100_practice_props
110_props_children
end
start
components
Container.css
Container.jsx
Profile.css
Profile.jsx
Example.jsx
120_props_rules
130_whats_jsx
140_react_elemen...
App.css
App.jsx
BaseErrorBoundar...
index.css
lectures.js
main.jsx
.eslintrc.cjs
.gitignore
index.html
package-lock.json
package.json
```

```jsx
import Profile from "./components/Profile";
import Container from "./components/Container";

const profile = [
  { name: "Takashi", age: 19, country: "Japan" },
  { name: "Jane", age: 28, country: "UK", color: "red" },
];

const Example = () => {
  return (
    <div>
      <Container title="Childrenとは？" children={
        [
          <Profile key={profile[0].name} {...profile[0]} />,
          <Profile key={profile[1].name} {...profile[1]} />
        ]
      }
      />

      <Container
        title="個別に渡せる"
        first={<Profile key={profile[0].name} {...profile[0]} />}
        second={<Profile key={profile[1].name} {...profile[1]} />}
      />
    </div>
  );
};

export default Example;
```

```
react-guide-material
04_react_basic
src
085_expr_and_state
087_practice_expr
090_props
100_practice_props
110_props_children
end
start
components
```

```jsx
import "./Container.css";

const Container = ({ title, children, first, second }) => {
  console.log(children);
  return (
    <div class="container">
      <h3>{title}</h3>
      {children}
      {first}
      {second}
```

```
Container.css
Container.jsx
Profile.css
Profile.jsx
Example.jsx
```

```jsx
    </div>
  );
};

export default Container;
```

38. propsの重要なルール

★propsの流れは一方通行

親 → 子へ / 子 → 親へ渡すことは出来ない！

```
react-guide-material
04_react_basic
src
120_props_rules
start
components
Bye.jsx
Hello.jsx
Example.jsx
130_whats_jsx
140_react_elemen...
App.css
App.jsx
BaseErrorBoundar...
index.css
lectures.js
main.jsx
.eslintrc.cjs
```

```jsx
import Bye from "./components/Bye"
import Hello from "./components/Hello"

const Example = () => {

  // POINT propsの流れは一方通行
  const name = 'Tom';

  return (
    <>
      <Hello name={name} />
      <Bye name={name} />
    </>
  );
};

export default Example;
```

Hello Tom

Bye Tom

```jsx
const Hello = (props) => {
  return (
    <div>
      <h3>Hello {props.name}</h3>
    </div>
  );
};

export default Hello;
```

```jsx
const Bye = (props) => {
  return (
    <div>
      <h3>Bye {props.name}</h3>
    </div>
  );
};

export default Bye;
```

★propsは読み取り専用

プロパティ(props) を変更することは不可能！ → 変更できる方法は後のセクションで説明！

```jsx
const Hello = (props) => {

  props.name = 'Bob';
  console.log(props.name);
```

▼エラーになる！

TypeError: Cannot assign to read only property 'name' of object '#<Object>'

```
components
  Bye.jsx
  Hello.jsx
Example.jsx
130_whats_jsx
140_react_elemen...
App.css
App.jsx
```

```
6   return (
7     <div>
8       <h3>Hello {props.name}</h3>
9     </div>
10  );
11 };
12
13 export default Hello;
```

```
at Hello (http://localhost:3001/src
```

※補足※ propsが読み取り専用ということをコンソールで確認する方法
`Reflect.getOwnPropertyDescriptor(props, 'name');`

```
react-guide-material
04_react_basic
src
120_props_rules
start
components
  Bye.jsx
  Hello.jsx
Example.jsx
130_whats_jsx
140_react_elemen...
App.css
App.jsx
BaseErrorBoundar...
index.css
lectures.js
main.jsx
```

```
1  const Hello = (props) => {
2
3    // POINT propsは読み取り専用
4    // props.name = 'Bob';
5    // console.log(props.name);
6
7    const desc = Reflect.getOwnPropertyDescriptor(props, 'name');
8    console.log(desc)
9
10   return (
11     <div>
12       <h3>Hello {props.name}</h3>
13     </div>
14   );
15 };
16
17 export default Hello;
```

```
▼ {value: 'Tom', writable: false,
    configurable: false
    enumerable: true
    value: "Tom"
    writable: false
  ▶ [[Prototype]]: Object
```

39. JSXの正体

▼JSX
React による JavaScript の構文を拡張したもの。
JSX は JS のオブジェクトに変換される。
<h1>Hello, world!</h1>; →→ JSオブジェクト(React要素)

▼JSX がオブジェクトに変換される過程

```
const sample1 = (
  <h1 className="greeting">Hello World</h1>
);
```

BABEL

```
const sample1 = React.createElement(
  "h1",
  {className: "greeting"},
  "Hello World"
);
```

```
const sample1 = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello World'
  }
};
```

← React要素

41. セクションまとめ

マウント：利用可能な状態にする

<コンポーネントの定義>

▼コンポーネントは JavaScript の関数として定義する。

```
function Welcome(props) {
  return <h1>Hello {props.name}</h1>;
}
```

```
<Welcome name="Tom" />
```

```
function Welcome(props) {
  return <h1>Hello {props.name}</h1>;
}
```

↑関数で定義されるコンポーネントは、関数コンポーネントと呼ばれる。

```
function App() {
  return (
    <div>
      <Welcome name="Bob" />
      <Welcome name="Tom" />
      <Welcome name="Lisa" />
    </div>
  );
}
```

セクション5: イベントリスナと状態管理（State）：

43. イベントに合わせて関数を実行してみよう

```
const Example = () => {
  const clickHandler = () => {
    alert('ボタンがクリックされました。');
  }
  const clickHandler2 = () => {
    console.log('ボタンがクリックされたーー！。');
  }
  return (
    <>
      <button onClick={clickHandler}>クリックしてね</button>
      <button onClick={clickHandler2}>クリック！</button>
    </>
  );
};

export default Example;
```

★注意★

以下の書き方は誤り(×)！ ()を書くことで関数が実行されてしまう！

```
onClick={clickHandler()}
```

## 44. 開発でよく利用するイベントタイプ

```
const Example = () => {
  return (
    <div>
      <h3>コンソールを確認してください。</h3>
      <label>
        入力値のイベント：
        <input
          type="text"
          onChange={() => console.log("onChange検知")}
          onBlur={() => console.log("onBlur検知")}
          onFocus={() => console.log("onFocus検知")}
        />
      </label>
```

入力値のイベント：

onChange

onFocus検知
onBlur検知
onFocus検知
onChange検知

```
import "./Example.css";

/* POINT イベントリスナの登録方法 */
const Example = () => {
  return (
    <div>
      <h3>コンソールを確認してください。</h3>
      <label>
        入力値のイベント：
        <input
          type="text"
          onChange={() => console.log("onChange検知")}
          onBlur={() => console.log("onBlur検知")}
          onFocus={() => console.log("onFocus検知")}
        />
      </label>
```

react-guide-material
05_state_and_event
src
010_eventlistener
end
start
Example.css
Example.jsx
015_other_events
end
Example.css
Example.jsx
start
020_useState
030_useState_ren...
040_multiple_state

---

```
const Example = () => {
  return (
    <div>
      <h3>コンソールを確認してください。</h3>
      <div>
        <label>
          入力値を取得：
          <input type="text" onChange={(e) => console.log(e.target.value)} />
        </label>
      </div>
```

★e.target.valueで入力値を取得

入力値を取得：

abcdef

件の非表示

a
ab
abc
abcd
abcde
abcdef

react-guide-material
05_state_and_event
src
015_other_events
end

```
import "./Example.css";

/* POINT イベントリスナの登録方法 */
const Example = () => {
  return (
```

```
Example.jsx
start
020_useState
030_useState_ren...
040_multiple_state
050_prev_state
055_practice_state
060_state_object
```

```jsx
<div>
  <h3>コンソールを確認してください。</h3>
  {/* POINT e.target.valueで入力値を取得 */}
  <div>
    <label>
      入力値を取得：
      <input type="text" onChange={(e) => console.log(e.target.value)} />
    </label>
  </div>
```

---

```jsx
<div
  className="hover-event"
  onMouseEnter={() => console.log("カーソルが入ってきました。")}
  onMouseLeave={() => console.log("カーソルが出ていきました。")}
>
  ホバーしてね！
</div>
```

```jsx
<div
  className="hover-event"
  onMouseEnter={() => console.log("カーソルが入ってきました。")}
  onMouseLeave={() => console.log("カーソルが出ていきました。")}
>
  ホバーしてね！
</div>
```

★重要★
上記のイベントタイプ(onChangeなど)、javascript と同じイベントタイプが使える！
注意） React：onChange → → javascript：onchange ※javascriptだと小文字になる！

★補足★
onChange：javascript のイベントで言うとonInput と同じ
React では onInput を使わない？onChange を使う！？

▼onChange さらに補足
javascriptの場合：onchange は値の入力が完了し、フォーカスが外れた時、発火する
Reactの場合：入力値が変更されたタイミングで発火

45. イベントに合わせて画面表示を更新してみよう

▼一見、上手く行きそうだが、上手く行かない…

```jsx
const Example = () => {
  let displayVal;
  return (
```

abcde =

```jsx
const Example = () => {
  let displayVal;
  return (
```

```
    <>
      <input
        type="text"
        onChange={(e) => {
          displayVal = e.target.value
        }}
      /> = {displayVal}
    </>
  );
};
```


```
    <>
      <input
        type="text"
        onChange={(e) => {
          displayVal = e.target.value
        }}
      /> = {displayVal}
    </>
  );
};
```


↓続く ----------

上記の場合、useState を使う！ Reactの関数！

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

const Example = () => {
  let valArry = useState("hello");
  console.log(valArry);
  // return (
  //   <>
```

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

const Example = () => {
  let valArry = useState("hello");
  console.log(valArry);
  // return (
  //   <>
```


★重要★ useState の返す値は、配列になる！

↓続く ----------

↓useStateは[ 値、変更用の関数 ]を返す

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

const Example = () => {
  let valArry = useState();
  console.log(valArry);
  return (
    <>
      <input
        type="text"
        onChange={(e) => {
          const setFn = valArry[1];
          setFn(e.target.value)
        }}
      /> = {valArry[0]}
```

valArry[0] は値がないので、表示無し


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

const Example = () => {
  let valArry = useState();
  console.log(valArry);
  return (
    <>
      <input
        type="text"
        onChange={(e) => {
          const setFn = valArry[1];
          setFn(e.target.value)
        }}
      /> = {valArry[0]}
```

```
    </>
  );
};
```

↓
入力すると↓

abc = abc

★重要★ useState を使うことによって、入力値をリアルタイムで更新できる

```
    </>
  );
};
```

↓続く ----------

★重要★
useState 使う場合、分割代入([ ])で取得することが多い！

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

const Example = () => {
  // useStateは[ 値、変更用の関数 ]を返す
  // 分割代入で取得
  let [val, setVal] = useState();

  return (
    <>
      <input
        type="text"
        onChange={(e) => {
          setVal(e.target.value);
        }}
      />
      = {val}
    </>
  );
};

export default Example;
```

更新関数
let [val, setVal] = useState(0);
現在の値　　初期値

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

const Example = () => {
  // POINT useStateは[ 値、変更用の関数 ]を返す
  // POINT 分割代入で取得
  let [val, setVal] = useState();

  return (
    <>
      <input
        type="text"
        onChange={(e) => {
          setVal(e.target.value);
        }}
      />
      = {val}
    </>
  );
};

export default Example;
```

abc = abc

46. 【重要】ステートとは？

```
      <input
      type="text"
      onChange={(e) => {
        displayVal = e.target.value
      }} />  = {displayVal}
    </>
  );
};

export default Example;
```

値を変更

画面に表示されない！
表示に変化なし！

111111 =

```
const Example = () => {
  let displayVal;
  return (
    <>
      <input
      type="text"
      onChange={(e) => {
        displayVal = e.target.value
      }} />  = {displayVal}
    </>
  );
};

export default Example;
```

onChangeによって赤線のコードは実行される

このコードはExample()が実行されないと変わらない。

オレンジの波枠
⇒アロー関数の本文
⇒すなわち、赤線のコード
のみ、クリックによって実行されます。

```
const Example = () => {
  let displayVal;
  return (
    <>
      <input
      type="text"
      onChange={(e) => {
        displayVal = e.target.value
      }} />  = {displayVal}
    </>
  );
```

Exampleが実行される度に値が空になってしまう。

再実行(再レンダリング)

state に保存

React内部に保持されたコンポーネントに紐づく値を state と呼ぶ

useState は React hooks と呼ばれる

React hooks

state(状態)とは？

47. 【重要】ステートとレンダリングの仕組み

▼useState 使わない場合

入力すると、onChangeは実行されるが、Exampleは実行されない！
※ stateを利用しない場合、再レンダリングされない！

```
> 060_state_object
> 064_state_array
> 068_practice_obj_st...
> 070_state_and_com...
> 080_state_and_props
> 090_practice_state_...
# App.css
App.jsx
BaseErrorBoundary...
# index.css
JS lectures.js
```

```
10          type="text"
11          onChange={(e) => {
12            console.log(e.target.value);
13            displayVal = e.target.value;
14          }}
15        />
16        = {displayVal}
17      </>
18    );
19  };
20
21  export default Example;
```

↓続く ----------

▼useState 使用した場合　★重要★

setVal を読み込んだタイミングで、Example(関数コンポーネント)が再実行されている！

```
react-guide-material
05_state_and_event
src
030_useState_render
start
Example.jsx
> 040_multiple_state
> 050_prev_state
> 055_practice_state
> 060_state_object
> 064_state_array
> 068_practice_obj_st...
> 070_state_and_com...
> 080_state_and_props
> 090_practice_state_...
# App.css
App.jsx
BaseErrorBoundary...
# index.css
JS lectures.js
```

```
1   import { useState } from "react";
2
3   const Example = () => {
4     let [ val, setVal ] = useState();
5     console.log('再レンダリングされました');
6     return (
7       <>
8         <input
9           type="text"
10          onChange={(e) => {
11            console.log(e.target.value);
12            setVal(e.target.value);
13          }}
14        />
15        = {val}
16      </>
17    );
18  };
19
20  export default Example;
```

```
123 = 123
2 件の非表示
再レンダリングされました
1
再レンダリングされました
12
再レンダリングされました
123
再レンダリングされました
```

48. 【複数のステート】ステート使用時の注意点！

ボタンAを0回押しました！

▼useState / ボタン押したら、1づつプラスされる(カウントアップ)

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

const Example = () => {
  const [countA, setCountA] = useState(0);
  const [countB, setCountB] = useState(10);
  const [countC, setCountC] = useState(100);

  return (
    <>
      <p>ボタンAを{countA}回押しました！</p>
      <button onClick={() => {
        setCountA(countA + 1);
        }}>ボタンA</button>

      <p>ボタンBを{countB}回押しました！</p>
      <button onClick={() => {
        setCountB(countB + 1);
        }}>ボタンB</button>

      <p>ボタンCを{countC}回押しました！</p>
      <button onClick={() => setCountC(countC + 1)}>ボタンC</button>
    </>
  );
};

export default Example;
```

ボタンAを0回押しました！

ボタンA

ボタンBを10回押しました！

ボタンB

ボタンCを100回押しました！

ボタンC

ボタンAを5回押しました！

ボタンA

ボタンBを23回押しました！

ボタンB

ボタンCを112回押しました！

ボタンC

react-guide-material
- 05_state_and_event
  - src
    - 040_multiple_state
      - start
        - Example.jsx
    - 050_prev_state
    - 055_practice_state
    - 060_state_object
    - 064_state_array
    - 068_practice_obj_s...
    - 070_state_and_co...
    - 080_state_and_pro...

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

const Example = () => {
  const [countA, setCountA] = useState(0);
  const [countB, setCountB] = useState(10);
  const [countC, setCountC] = useState(100);

  return (
    <>
      <p>ボタンAを{countA}回押しました！</p>
      <button onClick={() => {
        setCountA(countA + 1);
        }}>ボタンA</button>
```

```
<p>ボタンCを{countC}回押しました！</p>
<button onClick={() => setCountC(countC + 1)}>ボタンC</button>
</>
);
};

export default Example;
```

★注意★

useState はコンポーネント内のトップレベル( {ここ} )に記述する。

```
const Example = () => { ここに書く！ }
```

▼ここに書くとエラーになる！

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

const [countA, setCountA] = useState(0);    ← エラー

const Example = () => {
  const [countA, setCountA] = useState(0);
  const [countB, setCountB] = useState(10);
```

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

const [countA, setCountA] = useState(0);

const Example = () => {
  const [countA, setCountA] = useState(0);
  const [countB, setCountB] = useState(10);
```

---

▼コンポーネント( Example )の中に

if , for , whileなど、ブロック({ })の中に useState を書くとエラーになる！

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

const Example = () => {
  if (true) {
    const [countA, setCountA] = useState(0);
  }
  const [countB, setCountB] = useState(10);
```

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

const Example = () => {
  if (true) {
    const [countA, setCountA] = useState(0);
  }
  const [countB, setCountB] = useState(10);
```

## 49. 【更新は即時ではない】ステート使用上の注意

カウントアップ & カウントダウン

```react-guide-material/05_state_and_event/src/050_prev_state/start/Example.jsx
import { useState } from "react";

const Example = () => {
  const [count, setCount] = useState(0);
  const countUp = () => {
    setCount(count + 1);
  };
  const countDown = () => {
    setCount(count - 1);
  };
  return (
    <>
      <p>現在のカウント数: {count}</p>
      <button
        onClick={countUp}
      >+</button>
      <button
        onClick={countDown}
      >-</button>
    </>
  );
};

export default Example;
```

現在のカウント数: 0

+ -

react-guide-material
05_state_and_event
src
050_prev_state
start
Example.jsx
055_practice_state
060_state_object
064_state_array
068_practice_obj_...
070_state_and_co...
080_state_and_pr...
090_practice_stat...
p.css
p.jsx
seErrorBounda...
index.css
lectures.js
main.jsx
.eslintrc.cjs
.gitignore
index.html
package-lock.json
package.json

↓続く ----------

▼ステートの特性

ボタンクリック、コンソールに「１」が表示されると思うのだが、すぐに反映されない！

```react-guide-material/05_state_and_event/src/050_prev_state/start/Example.jsx
const Example = () => {
  const [count, setCount] = useState(0);
  const countUp = () => {
    setCount(count + 1);
    console.log(count);
  };
```

ボタンクリックをクリック

「１」にならない！

でも↓は実行されている！
`setCount(count + 1);`

▼`setCount();`
stateの更新は予約される（すぐには更新されない。）
非同期で処理されている！

▼`setCount(count + 1);` から `[count, setCount] = useState(0);` の `count` に設定されるのも非同期処理

```
const Example = () => {
  const [count, setCount] = useState(0);
  const countUp = () => {
    setCount(count + 1);
    console.log(count);
  };
```

非同期処理

▼★重要★
関数コンポーネント(`Example = () => {` ～) が再レンダリング(再実行)されるときに
[count, setCount] = useState(0); の count の値が、console.log(count); に渡される！

↓続く ----------

★重要★
上記はこのように書いた方が良い！

```
setCount(count + 1);  →  setCount(prevstate => {
                           return prevstate + 1;
                         });
```

現在のstateの値

次のstateの値

```
setCount(prevstate => {
  return prevstate + 1;
});
```

▼本件のまとめ

```python
import { useState } from "react";

const Example = () => {
  const [count, setCount] = useState(0);
  const countUp = () => {
    setCount(count + 1);
    setCount(prevstate => {
      return prevstate + 1;
    });
    console.log(count);
  };
  const countDown = () => {
    setCount(count - 1);
  };
  return (
    <>
      <p>現在のカウント数: {count}</p>
      <button
        onClick={countUp}
      >+</button>
      <button
        onClick={countDown}
      >-</button>
    </>
  );
};

export default Example;
```

```python
import { useState } from "react";

const Example = () => {
  const [count, setCount] = useState(0);
  const countUp = () => {
    setCount(count + 1);
    setCount(prevstate => {
      return prevstate + 1;
    });
    console.log(count);
  };
  const countDown = () => {
    setCount(count - 1);
  };
  return (
    <>
      <p>現在のカウント数: {count}</p>
      <button
        onClick={countUp}
      >+</button>
      <button
        onClick={countDown}
      >-</button>
    </>
  );
};

export default Example;
```

51. オブジェクト型のステートを使う際の注意点！

<テキストボックス(input)に入力して、他の場所に表示させる> →

オブジェクト型のstateを変更する場合には必ず新しいオブジェクトを作成する！

※プリミティブ型：1, "str", bool, 10n, Symbol(), null, undefined
※オブジェクト型：{}, []などのプリミティブ型以外

▼以下だとブラウザのテキストボックスに名前(Tom/初期値)が表示されていない！

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

const Example = () => {
  const personObj = { name: "Tom", age: 18 };
  const [person, setPerson] = useState(personObj);

  const changeName = (e) => {
    setPerson({ name: e.target.value, age: person.age })
  }

  return (
    <>
      <h3>Name:{person.name}</h3>
      <h3>Age:{person.age}</h3>
      <input type="text" onChange={changeName} />
    </>
  );
};

export default Example;
```

import { useState } from "react";

const Example = () => {
  const personObj = { name: "Tom", age: 18 };
  const [person, setPerson] = useState(personObj);

  const changeName = (e) => {
    setPerson({ name: e.target.value, age: person.age })
  }

  return (
    <>
      <h3>Name:{person.name}</h3>
      <h3>Age:{person.age}</h3>
      <input type="text" onChange={changeName} />
    </>
  );
};

export default Example;

▼value を設定すれば表示される！

```
<input type="text" value={person.name} onChange={changeName} />
```

↓続く ----------

★完了 / 重要★
・テキストボックス(input)に入力して、他の場所にも表示させる
・type="number" を使用し、値変更したら、他の場所にも表示させる

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

const Example = () => {
  const personObj = { name: "Tom", age: 18 };
  const [person, setPerson] = useState(personObj);

  const changeName = (e) => {
    setPerson({ name: e.target.value, age: person.age })
  }
  const changeAge = (e) => {
    setPerson({ name: person.name, age: e.target.value })
  }
  const reset = () => {
    setPerson({ name: "", age: "" })
  }

  return (
    <>
      <h3>Name:{person.name}</h3>
      <h3>Age:{person.age}</h3>
      <input type="text" value={person.name} onChange={changeName} />
      <input type="number" value={person.age} onChange={changeAge} />
      <div>
        <button onClick={reset}>リセット</button>
      </div>
    </>
  );
};

export default Example;
```

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

const Example = () => {
  const personObj = { name: "Tom", age: 18 };
  const [person, setPerson] = useState(personObj);

  const changeName = (e) => {
    setPerson({ name: e.target.value, age: person.age })
  }
  const changeAge = (e) => {
    setPerson({ name: person.name, age: e.target.value })
```

```
  }
  const reset = () => {
    setPerson({ name: "", age: "" })
  }

  return (
    <>
      <h3>Name:{person.name}</h3>
      <h3>Age:{person.age}</h3>
      <input type="text" value={person.name} onChange={changeName} />
      <input type="number" value={person.age} onChange={changeAge} />
      <div>
        <button onClick={reset}>リセット</button>
      </div>
    </>
  );
};

export default Example;
```

52. 【重要】オブジェクトのステートは新しいオブジェクトを設定する！
【重要】オブジェクトを更新する際は、新しいオブジェクトを作成する

以下の赤枠を右図のように書き換えると、動かない…、誤り…

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

const Example = () => {
  const personObj = { name: "Tom", age: 18 };
  const [person, setPerson] = useState(personObj);

  const changeName = (e) => {
    setPerson({ name: e.target.value, age: person.age })
  }
  const changeAge = (e) => {
```

以下の書き方だと、input赤枠のTom欄が入力出来ない…

```
const changeName = (e) => {
  person.name = e.target.value;
  setPerson(person)
}
```

```
const changeName = (e) => {
  person.name = e.target.value;
  setPerson(person)
}
```

↓ ----------

以下ピンク文字箇所、新しいオブジェクトが生成される！という意味になる。正しい！

```
setPerson({ name: e.target.value, age: person.age })
```

```
setPerson({ name: e.target.value, age: person.age })
```

↓ ----------

以下のようにスプレッド演算子を使うことも可能！
...personでpersonが展開されて、新しいオブジェクトを作成！
setPerson({ ...person })
↓
さらに以下で上書きする場合
setPerson({ ...person, name: e.target.value })

正常に動く！

ちなみにconsoleで確認すると違いが分かる！

```
const changeName = (e) => {
  console.log({ ...person } === person);
  console.log({ ...person });
  setPerson({ ...person, name: e.target.value })
}
```

```
false
▶ {name: 'Tom', age: 18}
```

53. 【練習】オブジェクトのステートを更新

54. 配列のステートを使う際の注意点！
【重要】配列を更新する際は、新しい配列を作成する（52. 【重要】オブジェクト…と同じ）

スプレッド演算子(...)で配列を展開して、
...nums
↓
新しい配列を作成します。配列をコピーしている。
[ ...nums ]

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

// POINT 配列のstateの扱い方
const Example = () => {
  const numArray = [1, 2, 3, 4, 5];
  const [nums, setNums] = useState(numArray);

  const shuffle = () => {
    const newNums = [ ...nums ];
    const lastVal = newNums.pop();
    newNums.unshift(lastVal);
    setNums(newNums);
  }
  return (
```

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

// POINT 配列のstateの扱い方
const Example = () => {
  const numArray = [1, 2, 3, 4, 5];
  const [nums, setNums] = useState(numArray);

  const shuffle = () => {
    const newNums = [ ...nums ];
    const lastVal = newNums.pop();
    newNums.unshift(lastVal);
    setNums(newNums);
  }
  return (
    <>
```

```
    <>
      <h1>{nums}</h1>
      <button onClick={shuffle}>shuffle</button>
    </>
  );
};

export default Example;
```

```
      <h1>{nums}</h1>
      <button onClick={shuffle}>shuffle</button>
    </>
  );
};

export default Example;
```

55. ステートとコンポーネントの関係

key="" を設定すると、別のコンポーネントとみなされる！

```
<Count key="A" title="A"/>
<Count key="B" title="B"/>
```

```
{toggle ? <Count key="A" title="A"/> : <Count key="B" title="B"/>}
```

以下のコードは toggleボタンで切り替えると、count がリセットされてしまう…。解決方法は次回の56で！

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

// POINT stateとコンポーネントの関係
const Example = () => {
  const [ toggle, setToggle ] = useState(true);
  const toggleComponent = () => {
    setToggle(prev => !prev);
  }
  return (
    <>
    {/* POINT コンポーネントの位置によってstateが識別される */}
    <button onClick={toggleComponent}>toggle</button>
    {toggle ? <Count key="A" title="A"/> : <Count key="B" title="B"/>}
    {/* <Count title="A"/>
    {toggle && <Count title="B"/>} */}
    </>
  )
}
const Count = ({ title }) => {
  const [count, setCount] = useState(0);
  const countUp = () => {
    setCount((prevstate) => prevstate + 1);
  };
  const countDown = () => {
```

```
    setCount(count - 1);
  };
  return (
    <>
      <h3>{title}: {count}</h3>
      <button onClick={countUp}>+</button>
      <button onClick={countDown}>-</button>
    </>
  );
};

export default Example;
```

56. ステートを複数のコンポーネントで管理しよう！

toggleボタンで切り替えても、count がリセットされない方法！

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

const Example = () => {
  const [toggle, setToggle] = useState(true);
  const [countA, setCountA] = useState(0);
  const [countB, setCountB] = useState(0);
  const toggleComponent = () => {
    setToggle((prev) => !prev);
  };
  return (
    <>
      <button onClick={toggleComponent}>toggle</button>
      {toggle ? <Count key="A" title="A" count={countA} setCount={setCountA} />
      : <Count key="B" title="B" count={countB} setCount={setCountB} />}
    </>
  );
};
const Count = ({ title, count, setCount }) => {
  const countUp = () => {
    setCount((prevstate) => prevstate + 1);
  };
  const countDown = () => {
    setCount(count - 1);
  };
```

```
  return (
    <>
      <h3>
        {title}: {count}
      </h3>
      <button onClick={countUp}>+</button>
      <button onClick={countDown}>-</button>
    </>
  );
};

export default Example;
```

57. 【練習】ステートの受け渡し

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

const Example = () => {
    const [ count, setCount ] = useState(0);

    return (
      <>
        <CountResult title="カウント" count={count} />
        <CountUpdate setCount={setCount} />
      </>
    );
  };
  const CountResult = ({ title, count }) => <h3>{title}: {count}</h3>

  const CountUpdate = ({ setCount }) => {
    const countUp = () => {
        setCount(prev => prev + 1);
    };
    const countDown = () => {
        setCount(prev => prev - 1);
    };
    return (
      <>
        <button onClick={countUp}>+</button>
        <button onClick={countDown}>-</button>
```

```
    </>
  );
};

export default Example;
```

58. セクションまとめ

セクション6: 制御構文とフォームの制御

60. 配列をリスト表示

【重要】

for文はJSX内に記述出来ない、エラーになる！

for文はあまり使わない！

```
return (
  <>
    <ul>
      for (const animal of animals) {
        animalList.push(<li>{animal}</li>);
      }
    </ul>
  </>
);
```

map関数はJSX内に記述可能！ 配列のメソッドを良く使う！

```
return (
  <>
    <ul>
      {animals.map((animal) => <li>Hello, {animal}</li>)}
    </ul>
  </>
);
```

以下のコード、エラー発生… 次回61.で説明

▶Warning: Each child in a list should have a unique "key" prop.

```
const animals = ["Dog", "Cat", "Rat"];

const Example = () => {
  // POINT for文でJSXの配列を作成
  const animalList = [];
  for (const animal of animals) {
    animalList.push(<li>{animal}</li>);
  }

  // POINT map関数でJSXの配列を作成
  const helloAnimals = animals.map((animal) => {
    return <li>Hello {animal}</li>;
```

```
const animals = ["Dog", "Cat", "Rat"];

const Example = () => {
  // POINT for文でJSXの配列を作成
  const animalList = [];
  for (const animal of animals) {
    animalList.push(<li>{animal}</li>);
  }

  // POINT map関数でJSXの配列を作成
  const helloAnimals = animals.map((animal) => {
    return <li>Hello {animal}</li>;
  });
```

```
  });

  return (
    <>
      <h3>配列の操作</h3>
      <ul>
        {/* <li>{animals[0]}</li>
        <li>{animals[1]}</li>
        <li>{animals[2]}</li> */}
        {/* {animalList}
        {helloAnimals} */}
        {/* POINT map関数はJSX内に記述可能 */}
        {animals.map((animal) => <li>Hello, {animal}</li>)}
      </ul>
    </>
  );
};

export default Example;
```

**配列の操作**

- Hello, Dog
- Hello, Cat
- Hello, Rat

```
  return (
    <>
      <h3>配列の操作</h3>
      <ul>
        {/* <li>{animals[0]}</li>
        <li>{animals[1]}</li>
        <li>{animals[2]}</li> */}
        {/* {animalList}
        {helloAnimals} */}
        {/* POINT map関数はJSX内に記述可能 */}
        {animals.map((animal) => <li>Hello, {animal}</li>)}
      </ul>
    </>
  );
};

export default Example;
```

61. 【重要】リストには必ずキーを設定

【重要】配列をJSX内で記述する場合はキー(key)を設定する

```
animals.map((animal) => (
  <li key={animal}>Hello, {animal}</li>
))
```

```
return (
  <>
    <ul>
      {
        animals.map((animal) => (
          <li key={animal}>Hello, {animal}</li>
        ))
      }
    </ul>
  </>
);
```

<前提知識>

ReactはReact要素ツリー(厳密にはFiberツリー)の差分検出処理をしてDOMを更新している

key有 ← key無

<keyを付ける際の注意点>
・キーには必ず一意の値を設定する。
・キーに設定した値は変更しない。
・配列のインデックスはなるべく使わない。
↓

▼配列にインデックスを使用しない場合(ﾕﾆｰｸｷｰ)、使用した場合

↓1,2,3と入力後に[先頭に追加]ボタンクリック

```
import "./Example.css";
import { useState } from "react";

const Example = () => {
  const inputFact = () => ({
    key: Math.floor(Math.random() * 1e3),
    value: <input />,
  });

  const [inputs, setInputs] = useState([inputFact(), inputFact(), inputFact()]);

  const unshiftInput = () => {
    setInputs((prev) => [inputFact(), ...prev]);
  };
  return (
```

```
    <>
      <button onClick={unshiftInput}>先頭に追加</button>
      <div className="flex">
        <div>
          <strong>{`key={ユニークキー}`}</strong>
          <ul>
            {inputs.map((input) => (
              <li key={input.key}>
                {input.key}: {input.value}
              </li>
            ))}
          </ul>
        </div>
        <div>
          <strong>{`key={index}`}</strong>
          <ul>
            {inputs.map((input, index) => (
              <li key={index}>
                {input.key}: {input.value}
              </li>
            ))}
          </ul>
        </div>
      </div>
    </>
  );
};

export default Example;
```

62. 【練習】リストにキーを設定してみよう

010_list_compone...
020_list_and_key
024_why_key_uni...
030_practice_list
end
components
Profile.jsx
Example.jsx
start
040_list_and_filter
050_practice_filter
060_conditional_r...
070_refactor_com...
080_input_textarea
090_radio
100_single_check...
110_multi_checkb...
120_select
130_reminder
App.css
App.jsx
BaseErrorBounda...
index.css
lectures.js
main.jsx
.eslintrc.cjs
.gitignore
index.html
package-lock.json
package.json
README.md
vite.config.js
6 age: 18,
7 hobbies: ["sports", "music"],
8 },
9 {
10 name: "Tom",
11 age: 25,
12 hobbies: ["movie", "music"],
13 },
14 {
15 name: "Lisa",
16 age: 21,
17 hobbies: ["sports", "travel", "game"],
18 },
19 ];
20
21 const Example = () => {
22 return (
23 <>
24 <ul>
25 {/* mapで各要素に特定の処理を行ったものを新しい配列とする */}
26 {persons.map((person) => (
27 /* リストにはkeyを設定することを忘れないように！ */
28 <li key={person.name}>
29 <Profile {...person} />
30 </li>
31 ))}
32 </ul>
33 </>
34 );
35 };
36
37 export default Example;
Hobby:
sports
music
Name: Tom
Age: 25
Hobby:
movie
music
Name: Lisa
Age: 21
Hobby:
sports
travel
game

Example.jsx
Profile.jsx
react-guide-material > 06_control_and_form > src > 030_practice_list > end > components > Profile
react-guide-material
06_control_and_form
node_modules
public
src
1 const Profile = ({ name, age, hobbies }) => {
2 return (
3 
4 <hr />
5 Name: {name}
6 Age: {age}

```jsx
import Profile from "./components/Profile";

const persons = [
  {
    name: "Geo",
    age: 18,
    hobbies: ["sports", "music"],
  },
  {
    name: "Tom",
    age: 25,
    hobbies: ["movie", "music"],
  },
  {
    name: "Lisa",
    age: 21,
    hobbies: ["sports", "travel", "game"],
  },
];

const Example = () => {
  return (
    <>
      <ul>
        {/* mapで各要素に特定の処理を行ったものを新しい配列とする */}
        {persons.map((person) => (
```

```jsx
const Profile = ({ name, age, hobbies }) => {
  return (
    <div>
      <hr />
      <div>Name: {name}</div>
      <div>Age: {age}</div>
      <div>
        <div>Hobby:</div>
        <ul>
          {hobbies.map((hobby) => (
            /* リストにはkeyを設定することを忘れないように！ */
            <li key={hobby}>{hobby}</li>
          ))}
        </ul>
      </div>
    </div>
  );
};

export default Profile;
```

```
            /* リストにはkeyを設定することを忘れないように！ */
            <li key={person.name}>
            <Profile {...person} />
          </li>
        ))}
      </ul>
    </>
  );
};

export default Example;
```

63. 配列のフィルターメソッドの使い方　★わりと良く使う機能！

filterの書き方：1

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

const animals = ["Dog", "Cat", "Rat"];
// POINT filterメソッドの使い方
const Example = () => {
  const [filterVal, setFilterVal] = useState("");
  return (
    <>
      <h3>配列のフィルター</h3>
      <input type="text" value={filterVal} onChange={(e) => setFilterVal(e.target.value)} />
      <ul>
        {animals
          .filter(animal => animal.indexOf(filterVal) !== -1)
          .map((animal) => (
            <li key={animal}>{animal}</li>
          ))
        }
      </ul>
    </>
  );
};

export default Example;
```

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

```
const animals = ["Dog", "Cat", "Rat"];
// POINT filterメソッドの使い方
const Example = () => {
  const [filterVal, setFilterVal] = useState("");
  return (
    <>
      <h3>配列のフィルター</h3>
      <input type="text" value={filterVal} onChange={(e) => setFilterVal(e.target.value)} />
      <ul>
        {animals
          .filter(animal => animal.indexOf(filterVal) !== -1)  ⇥ tab
          .map((animal) => (
            <li key={animal}>{animal}</li>
          ))
        }
      </ul>
    </>
  );
};

export default Example;
```

filterの書き方：2

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

const animals = ["Dog", "Cat", "Rat"];
// POINT filterメソッドの使い方
const Example = () => {
  const [filterVal, setFilterVal] = useState("");
  return (
    <>
      <h3>配列のフィルター</h3>
      <input type="text" value={filterVal} onChange={(e) => setFilterVal(e.target.value)} />
      <ul>
        {animals
          .filter(animal => {
            const isMatch = animal.indexOf(filterVal) !== -1;
            console.log(animal.indexOf(filterVal))
            return isMatch
          })
```

```
        .map((animal) => (
        <li key={animal}>{animal}</li>
      ))}
    </ul>
  </>
 );
};

export default Example;
```

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

const animals = ["Dog", "Cat", "Rat"];
// POINT filterメソッドの使い方
const Example = () => {
  const [filterVal, setFilterVal] = useState("");
  return (
    <>
      <h3>配列のフィルター</h3>
      <input type="text" value={filterVal} onChange={(e) => setFilterVal(e.target.value)} />
      <ul>
        {animals
          .filter(animal => {
            const isMatch = animal.indexOf(filterVal) !== -1;
            console.log(animal.indexOf(filterVal))
            return isMatch
          })
          .map((animal) => (
          <li key={animal}>{animal}</li>
        ))}
      </ul>
    </>
  );
};

export default Example;
```

64. 【練習】フィルターメソッドの使い方を練習してみよう

```jsx
import Profile from "./components/Profile";
import { useState } from "react";

const persons = [
  {
    name: "Geo",
    age: 18,
    hobbies: ["sports", "music"],
  },
  {
    name: "Tom",
    age: 25,
    hobbies: ["movie", "music"],
  },
  {
    name: "Lisa",
    age: 21,
    hobbies: ["sports", "travel", "game"],
  },
];

const Example = () => {
  const [filterVal, setFilterVal] = useState("");
  return (
    <>
      <input type="text" value={filterVal} onChange={(e) => setFilterVal(e.target.value)} />
      <ul>
        {persons
        // filterを追加
        .filter(person => {
          const isMatch = person.name.indexOf(filterVal) !== -1;
          return isMatch
        })
        .map((person) => (
          <li key={person.name}>
            <Profile {...person} />
          </li>
        ))}
      </ul>
    </>
  );
};
```

- components
  - Profile.jsx
  - Example.jsx

```jsx
const Profile = ({ name, age, hobbies }) => {
  return (
    <div>
      <hr />
      <div>Name: {name}</div>
      <div>Age: {age}</div>
      <div>
        <div>Hobby:</div>
        <ul>
          {hobbies.map((hobby) => (
            <li key={hobby}>{hobby}</li>
          ))}
        </ul>
      </div>
    </div>
  );
};

export default Profile;
```

```
export default Example;
```

- travel
- game

65. 条件分岐を設ける方法まとめ

3項演算子で書いた方がよい！
・コードが重複しない、修正箇所が１つになる
・メンテナンスしやすい

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

/*
  条件分岐 if文、&&、3項演算子、

  ?? (Null合体演算子)
  A ?? B (Aがnull or undefinedの時、Bを使う)
*/
const Example = () => {
  const animals = ["Dog", "Cat", null, "Rat"];

  const [filterVal, setFilterVal] = useState("");

  return (
    <>
      <input
        type="text"
        value={filterVal}
        onChange={(e) => setFilterVal(e.target.value)}
      />
      <ul>
        {animals
          .filter((animal) => {
            const animalStr = animal ?? "";
            const isMatch = animalStr.indexOf(filterVal) !== -1;

            return isMatch;
          })
          .map((animal) => {
            return (
              <li key={animal}>
                {
```

- Dog★
- Cat
- null,undefinedでした
- Rat

t

- Cat
- Rat

```
                // < if文 >
                // if(animal === "Dog") {
                //   return <li key={animal}>{animal}★</li>
                // } else {
                //   return <li key={animal}>{animal}</li>
                // }

                // < 3項演算子 その1 >
                // animal === "Dog"
                //  ? animal + "★"
                //  : animal)

                // < 3項演算子 その2 ★こちらの方がメンテナンスしやすい★ >
                // animal + (animal === "Dog"
                //  ? "★"
                //  : "")

                // < ??（null合体演算子） >
                animal ?? "null,undefinedでした"
              }{
                /*
                < &&演算子を使う場合、{}ブロックを追加して記述 >
                trueの場合は"★"が表示される
                falseの場合は何も表示されない
                */
                animal === "Dog" && "★"
              }
            </li>
          );
        })}
      </ul>
    </>
  );
};

export default Example;
```

66. コンポーネントのリファクタリング

リファクタリング前の状態

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

const Example = () => {
  const animals = ["Dog", "Cat", "Rat"];

  const [filterVal, setFilterVal] = useState("");

  return (
    <>
      <input
        type="text"
        value={filterVal}
        onChange={(e) => setFilterVal(e.target.value)}
      />
      <ul>
        {animals
          .filter((animal) => {
            const isMatch = animal.indexOf(filterVal) !== -1;
            return isMatch;
          })
          .map((animal) => {
            return (
              <li key={animal}>
                {animal}
                {animal === "Dog" && "★"}
              </li>
            );
          })}
      </ul>
    </>
  );
};

export default Example;
```

リファクタリング後

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

import AnimalList from "./components/AnimalList"
import AnimalFilter from "./components/AnimalFilter"

const Example = () => {
  const animals = ["Dog", "Cat", "Rat"];

  const [filterVal, setFilterVal] = useState("");

  const filterdAnimals = animals.filter((animal) => {
    const isMatch = animal.indexOf(filterVal) !== -1;
    return isMatch;
  });

  return (
    <>
      <AnimalFilter filterState={[filterVal, setFilterVal]}/>
      <AnimalList animals={filterdAnimals} />
    </>
  );
};
```

```
export default Example;
```

```
const AnimalFilter = ({ filterState }) => {
  const [filterVal, setFilterVal] = filterState;

  return (
    <input
      type="text"
      value={filterVal}
      onChange={(e) => setFilterVal(e.target.value)}
    />
  );
};
export default AnimalFilter;
```

```
const AnimalItem = ({ animal }) => {
  return (
    <li>
      {animal}
      {animal === "Dog" && "★"}
    </li>
  );
};

export default AnimalItem;
```

```
080_input_textarea
090_radio
100_single_checkbox
110_multi_checkbox
120_select
```

```
12        </ul>
13      );
14    };
15
16    export default AnimalList;
```

```
import AnimalItem from "./AnimalItem";
const AnimalList = ({ animals }) => {
  if (animals.length === 0) {
    return <h3>アニマルが見つかりません。</h3>;
  }

  return (
    <ul>
      {animals.map((animal) => {
        return <AnimalItem animal={animal} key={animal} />;
      })}
    </ul>
  );
};

export default AnimalList;
```

67. 【Form】inputとtextareaの作成方法

【重要】
htmlでは、for だが、<label for="456">
JSXでは、htmlFor にする。<label htmlFor="456">。for文はjavascriptで良く使うので。
---
htmlでは、終了タグを記述するのだが、<textarea>aaa</textarea>
JSXでは、終了タグ無し。入力したい値は val に記述。<textarea value={val} />

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

// POINT input要素、textarea要素の使い方
const Example = () => {

  const [val, setVal] = useState("");
  const clearVal = () => setVal("");
```

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

// POINT input要素、textarea要素の使い方
const Example = () => {

  const [val, setVal] = useState("");
  const clearVal = () => setVal("");

  return (
```

```
  return (
    <div>
      <label htmlFor="456">ラベル</label>
      <div>
        <input
          id="123"
          placeholder="こんにちは"
          value={val}
          onChange={(e) => setVal(e.target.value)}
        />
        <textarea
          id="456"
          placeholder="こんにちは"
          value={val}
          onChange={(e) => setVal(e.target.value)}
        />
      </div>
      <h3>{val}</h3>
      <button onClick={clearVal}>クリア</button>
    </div>
  );
};

export default Example;
```

```
    <div>
      <label htmlFor="456">ラベル</label>
      <div>
        <input
          id="123"
          placeholder="こんにちは"
          value={val}
          onChange={(e) => setVal(e.target.value)}
        />
        <textarea
          id="456"
          placeholder="こんにちは"
          value={val}
          onChange={(e) => setVal(e.target.value)}
        />
      </div>
      <h3>{val}</h3>
      <button onClick={clearVal}>クリア</button>
    </div>
  );
};

export default Example;
```

68. 【Form】ラジオボタンの作成方法

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

// POINT ラジオボタンの実装
const Example = () => {
  const [fruit, setFruit] = useState("Apple");
  const onChange = (e) => setFruit(e.target.value);

  const RADIO_COLLECTION = ["Apple", "Banana", "Cherry"];

  return (
    <>
```

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

// POINT ラジオボタンの実装
const Example = () => {
  const [fruit, setFruit] = useState("Apple");
  const onChange = (e) => setFruit(e.target.value);

  const RADIO_COLLECTION = ["Apple", "Banana", "Cherry"];

  return (
    <>
      {RADIO_COLLECTION.map((value) => {
```

```
{RADIO_COLLECTION.map((value) => {
  return (
    <label key={value}>
      <input
        type="radio"
        value={value}
        checked={fruit === value}
        onChange={onChange}
      />
      {value}
    </label>
  );
})}
{/* <label>  ⇥ tab
  <input
    type="radio"
    value="Banana"
    checked={fruit === "Banana"}
    onChange={onChange}
  />
  Banana
</label> */}
<h3>私は{fruit}がたべたい</h3>
</>
);
};

export default Example;
```

Apple Banana Cherry

**私はAppleがたべたい**

```
        return (
          <label key={value}>
            <input
              type="radio"
              value={value}
              checked={fruit === value}
              onChange={onChange}
            />
            {value}
          </label>
        );
      })}
      {/* <label>
        <input
          type="radio"
          value="Banana"
          checked={fruit === "Banana"}
          onChange={onChange}
        />
        Banana
      </label>
      <label>
        <input
          type="radio"
          value="Cherry"
          checked={fruit === "Cherry"}
          onChange={onChange}
        />
        Cherry
      </label> */}
      <h3>私は{fruit}がたべたい</h3>
    </>
  );
};

export default Example;
```

69. 【Form】チェックボックスの作成方法

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

// POINT チェックボックスの実装
```

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

// POINT チェックボックスの実装
```

```react-checkbox.jsx
const Example = () => {
  const [isChecked, setIsChecked] = useState(true);

  // const toggleChecked = (e) => {
  //   setIsChecked(prevState => !prevState);
  // };

  return (
    <div>
      <label htmlFor="my-check">
        チェック：
      </label>
      <input
        type="checkbox"
        id="my-check"
        checked={isChecked}
        onChange={() => setIsChecked(prevState => !prevState)}
        // onChange={(e) => toggleChecked(e)}
      />
      <div>{isChecked ? "ON!" : "OFF!"}</div>
    </div>
  );
};

export default Example;
```

チェック：☑
ON!

```react-checkbox.jsx
const Example = () => {
  const [isChecked, setIsChecked] = useState(true);

  // const toggleChecked = (e) => {
  //   setIsChecked(prevState => !prevState);
  // };

  return (
    <div>
      <label htmlFor="my-check">
        チェック：
      </label>
      <input
        type="checkbox"
        id="my-check"
        checked={isChecked}
        onChange={() => setIsChecked(prevState => !prevState)}
        // onChange={(e) => toggleChecked(e)}
      />
      <div>{isChecked ? "ON!" : "OFF!"}</div>
    </div>
  );
};

export default Example;
```

70. 【Form】複数選択チェックボックスの作成方法

```react-checkbox.jsx
import { useState } from "react";

// 複数チェックボックスの実装
const Example = () => {
  const [fruits, setFruits] = useState([
    { label: "Apple", value: 100, checked: false },
    { label: "Banana", value: 200, checked: false },
    { label: "Cherry", value: 300, checked: false },
  ]);
```

☐ Apple:100

```react-checkbox.jsx
import { useState } from "react";

// 複数チェックボックスの実装
const Example = () => {
  const [fruits, setFruits] = useState([
    { label: "Apple", value: 100, checked: false },
    { label: "Banana", value: 200, checked: false },
    { label: "Cherry", value: 300, checked: false },
  ]);
```

```
const [sum, setSum] = useState(0);

const handleChange = (e) => {
  const newFruits = fruits.map((fruit) => {
    const newFruit = { ...fruit };
    if (newFruit.label === e.target.value) {
      newFruit.checked = !fruit.checked;
    }

    return newFruit;
  });

  setFruits(newFruits);
  // forEachバージョン
  let sumVal = 0;
  newFruits.forEach(fruit => {
    if(fruit.checked) {
      sumVal = sumVal + fruit.value;
    }
  });

  // filter + forEachバージョン
  // let sumVal = 0;
  // newFruits
  //   .filter((fruit) => fruit.checked)
  //   .forEach((fruit) => (sumVal = sumVal + fruit.value));

  // filter + reduceバージョン 上級者向け
  // let sumVal = newFruits
  //   .filter((fruit) => fruit.checked)
  //   .reduce((sumVal, fruit) => sumVal + fruit.value, 0);

  setSum(sumVal);
};
return (
  <div>
    {fruits.map((fruit) => {
      return (
        <div key={fruit.label}>
          <input
            id={fruit.label}
            type="checkbox"
            value={fruit.label}
```

Banana:200
Cherry:300
合計：500

```
11  const [sum, setSum] = useState(0);
12
13  const handleChange = (e) => {
14    const newFruits = fruits.map((fruit) => {
15      const newFruit = { ...fruit };
16      if (newFruit.label === e.target.value) {
17        newFruit.checked = !fruit.checked;
18      }
19
20      return newFruit;
21    });
22
23    setFruits(newFruits);
24    // forEachバージョン
25    let sumVal = 0;
26    newFruits.forEach(fruit => {
27      if(fruit.checked) {
28        sumVal = sumVal + fruit.value;
29      }
30    });
31
32    // filter + forEachバージョン
33    // let sumVal = 0;
34    // newFruits
35    //   .filter((fruit) => fruit.checked)
36    //   .forEach((fruit) => (sumVal = sumVal + fruit.value));
37
38    // filter + reduceバージョン 上級者向け
39    // let sumVal = newFruits
40    //   .filter((fruit) => fruit.checked)
41    //   .reduce((sumVal, fruit) => sumVal + fruit.value, 0);
42
43    setSum(sumVal);
44  };
45  return (
46    <div>
47      {fruits.map((fruit) => {
48        return (
49          <div key={fruit.label}>
50            <input
51              id={fruit.label}
```

```
            checked={fruit.checked}
            onChange={handleChange}
          />
          <label htmlFor={fruit.label}>
            {fruit.label}:{fruit.value}
          </label>
        </div>
      );
    })}
    <div>合計：{sum}</div>
  </div>
);
};

export default Example;
```

```
            id={fruit.label}
            type="checkbox"
            value={fruit.label}
            checked={fruit.checked}
            onChange={handleChange}
          />
          <label htmlFor={fruit.label}>
            {fruit.label}:{fruit.value}
          </label>
        </div>
      );
    })}
    <div>合計：{sum}</div>
  </div>
);
};

export default Example;
```

71. 【Form】プルダウンの作成方法

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

// POINT プルダウンの実装
const Example = () => {
  const [selected, setSelected] = useState("Banana");

  const OPTIONS = ["Apple", "Banana", "Cherry"];

  return (
    <>
      <select
        value={selected}
        onChange={(e) => setSelected(e.target.value)}
      >
        {OPTIONS.map(opt => <option key={opt} value={opt}>{opt}</option>)}
        {
        /* <option value="Apple">Apple</option>
        <option value="Banana">Banana</option>
        <option value="Cherry">Cherry</option> */
```

Banana
選択された果物：Banana

```
        }
      </select>
      <div>選択された果物：{selected}</div>
    </>
  );
};

export default Example;
```

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

// POINT プルダウンの実装
const Example = () => {
  const [selected, setSelected] = useState("Banana");

  const OPTIONS = ["Apple", "Banana", "Cherry"];

  return (
    <>
      <select
        value={selected}
        onChange={(e) => setSelected(e.target.value)}
      >
        {OPTIONS.map(opt => <option key={opt} value={opt}>{opt}</option>)}
        {
        /* <option value="Apple">Apple</option>
        <option value="Banana">Banana</option>
        <option value="Cherry">Cherry</option> */
        }
      </select>
      <div>選択された果物：{selected}</div>
    </>
  );
};

export default Example;
```

72. Todoアプリを作ってみよう

```
react-guide-material
06_control_and_form
src
130_reminder
end
components
Form.jsx
List.jsx
Todo.jsx
Example.jsx
start
App.css
App.jsx
```

```
// POINT Reminder(Todoアプリ)の作り方
import Todo from "./components/Todo"

const Example = () => {
  return (
    <>
      <h2>Reminder</h2>
      <Todo />
    </>
  );
};

export default Example;
```


```
// POINT Reminder(Todoアプリ)の作り方
import Todo from "./components/Todo"

const Example = () => {
  return (
    <>
      <h2>Reminder</h2>
      <Todo />
    </>
  );
};

export default Example;
```


```
react-guide-material
06_control_and_form
src
130_reminder
end
components
Form.jsx
List.jsx
Todo.jsx
Example.jsx
start
App.css
App.jsx
BaseErrorBounda...
index.css
```

```
import { useState } from "react";
import List from "./List";
import Form from "./Form";

const Todo = () => {
  const todosList = [
    {
      id: 1,
      content: "店予約する",
    },
    {
      id: 2,
      content: "卵買う",
    },
    {
      id: 3,
```

クリックしてブレークポイントを追加します


```
import { useState } from "react";
import List from "./List";
import Form from "./Form";

const Todo = () => {
  const todosList = [
    {
      id: 1,
      content: "店予約する",
    },
    {
      id: 2,
      content: "卵買う",
    },
    {
      id: 3,
      content: "郵便出す",
```

```
lectures.js
main.jsx
.eslintrc.cjs
.gitignore
index.html
package-lock.json
package.json
README.md
vite.config.js
07_styling_compone...
08_other_function
09_debugging
10_functional_progr...
11_hooks_p1
12_hooks_p2
13_redux
14_class_component
15_performance
16_rest_api
17_nextjs_p1
18_nextjs_p2
19_test
20_typescript
EPADS
ライン
ライン
```

```
    content: "郵便出す",
  },
];

const [todos, setTodos] = useState(todosList);

const deleteTodo = (id) => {
  const newTodos = todos.filter((todo) => {
    return todo.id !== id;
  });

  setTodos(newTodos);
};

const createTodo = (todo) => {
  setTodos([...todos, todo]);
};

return (
  <>
    <List todos={todos} deleteTodo={deleteTodo} />
    <Form createTodo={createTodo} />
  </>
);
};
export default Todo;
```

```
},
];

const [todos, setTodos] = useState(todosList);

const deleteTodo = (id) => {
  const newTodos = todos.filter((todo) => {
    return todo.id !== id;
  });

  setTodos(newTodos);
};

const createTodo = (todo) => {
  setTodos([...todos, todo]);
};

return (
  <>
    <List todos={todos} deleteTodo={deleteTodo} />
    <Form createTodo={createTodo} />
  </>
);
};
export default Todo;
```

```
eact-guide-material
06_control_and_form
src
130_reminder
end
components
Form.jsx
List.jsx
Todo.jsx
Example.jsx
start
App.css
App.jsx
```

```
const List = ({todos, deleteTodo}) => {
    const complete = (id) => {
        deleteTodo(id)
    }
    return (
        <div>
            {todos.map(todo => {
                return (
                    <div key={todo.id}>
                        <button onClick={() => complete(todo.id)}>完了</button>
                        <span>{todo.content}</span>
                    </div>
                )
```

```
BaseErrorBounda...
index.css
lectures.js
main.jsx
.eslintrc.cjs
.gitignore
```

```
14            })}
15          </div>
16        );
17      }
18
19      export default List;
```

```
const List = ({todos, deleteTodo}) => {
    const complete = (id) => {
        deleteTodo(id)
    }
    return (
        <div>
            {todos.map(todo => {
                return (
                    <div key={todo.id}>
                        <button onClick={() => complete(todo.id)}>完了</button>
                        <span>{todo.content}</span>
                    </div>
                )
            })}
        </div>
    );
}

export default List;
```

```
ct-guide-material
_control_and_form
rc
130_reminder
end
components
Form.jsx
List.jsx
Todo.jsx
Example.jsx
start
App.css
App.jsx
BaseErrorBounda...
index.css
```

```
1   import { useState } from "react";
2   const Form = ({ createTodo }) => {
3     const [enteredTodo, setEnteredTodo] = useState("");
4
5     const addTodo = (e) => {
6       e.preventDefault();
7
8       const newTodo = {
9         id: Math.floor(Math.random() * 1e5),
10        content: enteredTodo,
11      };
12
13      createTodo(newTodo);
14
15      setEnteredTodo("");
```

```
import { useState } from "react";
const Form = ({ createTodo }) => {
  const [enteredTodo, setEnteredTodo] = useState("");

  const addTodo = (e) => {
    e.preventDefault();

    const newTodo = {
      id: Math.floor(Math.random() * 1e5),
      content: enteredTodo,
    };

    createTodo(newTodo);

    setEnteredTodo("");
```

```
lectures.js
main.jsx
eslintrc.cjs
gitignore
ndex.html
package-lock.json
package.json
README.md
ite.config.js
_styling_compone...
_other_function
_debugging
_functional_progr...
_hooks_p1
_hooks_p2
_redux
```

```
16    };
17    return (
18      <div>
19        <form onSubmit={addTodo}>
20          <input
21            type="text"
22            value={enteredTodo}
23            onChange={(e) => setEnteredTodo(e.target.value)}
24          />
25          <button>追加</button>
26        </form>
27      </div>
28    );
29  };
30
31  export default Form;
```

```
  };
  return (
    <div>
      <form onSubmit={addTodo}>
        <input
          type="text"
          value={enteredTodo}
          onChange={(e) => setEnteredTodo(e.target.value)}
        />
        <button>追加</button>
      </form>
    </div>
  );
};

export default Form;
```

<補足>
乱数を作成する場合、便利なライブラリ(パッケージ)有
npm docs nanoid
https://github.com/ai/nanoid#readme

```
import { nanoid } from 'nanoid'
model.id = nanoid() //=> "V1StGXR8_Z5jdHi6B-myT"
```

import { nanoid } from 'nanoid'
model.id nanoid() //=> "V1StGXR8_Z5jdHi6B-myT"

セクション7: スタイリング

73. セクション紹介 ※内容不要

74. インラインスタイルの使い方！

★インラインスタイルは基本的に使わない！★

< POINT >

・再利用性が低い
・疑似要素やメディアクエリが使用できない
・レンダリングの度に計算されるのでパフォーマンスが劣る
・動的で頻繁に計算されるスタイルの適用

< インラインスタイルのメリットとデメリット >

・メリット
直感的に記述することができる。

・デメリット
再レンダリングの度に値が計算されるのでパフォーマンス的に優れていない
要素に直接記述しているので詳細度が一番高くなり、cssのスタイルが何も効かない

疑似セレクタやメディアクエリにも対応していないため、実装しようとするとわかりづらいコードになってしまう

```
ide-material > 07_styling_component > src > 010_inline_style > end > Example.jsx > ...
import { useState } from "react";

const Example = () => {
  const [isSelected, setIsSelected] = useState(false);

  const clickHandler = () => setIsSelected(prev => !prev);

  /* POINT style属性に適応させるスタイルをオブジェクトで記述します */
  const style = {
    margin: "auto",
    "border-radius": "9999px",
    border: "none",
    display: "block",

    /* POINT 単位を書かない場合
    単位を書かずに文字列ではなく数字を与えてあげると、reactが自動で解釈し値にpxを付けてくれます。 */
    width: 120,
    height: 60,
    fontWeight: "bold",
    cursor: "pointer",
    /* POINT 三項演算子を使用して isSelected が true の場合は 'pink' false の場合は空文字( '' ) を与えています。
    valueに空文字を与えた場合プロパティは適応されません。 */
    backgroundColor: isSelected ? "pink" : "",
    /* POINT 直接記述することによって可読性が大きく低下するので、可読性が向上する方法を考えて実装してみよう */
  };

  return (
    <>
      <button style={style} onClick={clickHandler}>
        ボタン
      </button>
      <div style={{ textAlign: "center" }}>{isSelected && "クリックされました。"}</div>  ⇥ tab
    </>
  );
};

export default Example;
```

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

```
const Example = () => {
  const [isSelected, setIsSelected] = useState(false);

  const clickHandler = () => setIsSelected(prev => !prev);

  /* POINT style属性に適応させるスタイルをオブジェクトで記述します */
  const style = {
    margin: "auto",
    "border-radius": "9999px",
    border: "none",
    display: "block",

    /* POINT 単位を書かない場合
    単位を書かずに文字列ではなく数字を与えてあげると、reactが自動で解釈し値にpxを付けてくれます。 */
    width: 120,
    height: 60,
    fontWeight: "bold",
    cursor: "pointer",
    /* POINT 三項演算子を使用して isSelected が true の場合は 'pink' false の場合は空文字( '' ) を与えています。
    valueに空文字を与えた場合プロパティは適応されません。 */
    backgroundColor: isSelected ? "pink" : "",
    /* POINT 直接記述することによって可読性が大きく低下するので、可読性が向上する方法を考えて実装してみよう */
  };

  return (
    <>
      <button style={style} onClick={clickHandler}>
        ボタン
      </button>
      <div style={{ textAlign: "center" }}>{isSelected && "クリックされました。"}</div>
    </>
  );
};

export default Example;
```

75. インラインスタイルの注意点！

上記(前回)に記載されている。

プラス以下、使用不可
::before, ::after, :hover, :active
@media (min-width: 600px){-}

インラインスタイルは基本的に使わない！

76. 外部CSSのimportを使ったスタイリング　　※76の書き方はあまり良くない…

cssファイルを以下のように同じディレクトリに置くのはあまり良くない…

【注意】
＜動的なスタイルの適用＞
クラスの付け外しに論理積 (&&) は使用してはいけません！
isSelected が false の場合、 className='btn false' になってしまいます。
例：`className={`btn ${isSelected && 'selected'}`}`

react-guide-material > 07_styling_component > src > 020_import_css > end > Example.jsx > Exam

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

import SubButton from "./components/SubButton";
import "./Example.css";

const Example = () => {
  const [isSelected, setIsSelected] = useState(false);

  const clickHandler = () => setIsSelected((prev) => !prev);

  return (
    <>
      <button
        className={`btn ${isSelected ? "selected" : ""}`}
        onClick={clickHandler}
      >
        ボタン
      </button>
      <SubButton />
```

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

import SubButton from "./components/SubButton";
import "./Example.css";

const Example = () => {
  const [isSelected, setIsSelected] = useState(false);

  const clickHandler = () => setIsSelected((prev) => !prev);

  return (
    <>
      <button
        className={`btn ${isSelected ? "selected" : ""}`}
        onClick={clickHandler}
      >
        ボタン
      </button>
      <SubButton />
      <div style={{ textAlign: "center" }}>
        {isSelected && "クリックされました。"}
      </div>
    </>
  );
};

export default Example;
```

```
.btn {
    margin: auto;
    border-radius: 9999px;
    border: none;
```

```
    display: block;
    width: 120px;
    height: 60px;
    font-weight: bold;
    cursor: pointer;
}

.selected {
    background-color: pink;
}
```

```
import "./SubButton.css";

const SubButton = () => {
    return <button className="btn">サブボタン</button>

}
export default SubButton;
```

```
.btn {
    margin: auto;
    border-radius: 9999px;
    border: none;
    display: block;
    width: 120px;
    height: 60px;
    font-weight: bold;
    cursor: pointer;
    background: orange;
}

.selected {
    background-color: pink;
}
```

77. CSS Modulesを使ったスタイリング

<補足>
CSS Modules 近い将来、廃止される可能性がある！
★次回で説明する CSS-in-JS を使用したほうがよい！★

<CSS Moduleのメリットとデメリット>
・メリット
class名を気にすることなくcssを記述することができる。
cssと変わらないので学習コストがない
標準の機能なので、導入コストがない
・デメリット
将来日推奨になる可能性がある。
https://github.com/webpack-contrib/css-loader/issues/1050#:~:text=In%20the%20near%20future%20we%20want%20to%20deprecate%20CSS%20modules

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

import SubButton from "./components/SubButton";
import styles from "./Example.module.css";

// console.log(styles);

const Example = () => {
  const [isSelected, setIsSelected] = useState(false);

  const clickHandler = () => setIsSelected((prev) => !prev);

  return (
    <>
      <button className={`${styles.btn} ${isSelected ? styles.selected : ""}`} onClick={clickHandler}>
        ボタン
      </button>
      <SubButton />
      <div style={{ textAlign: "center" }}>
        {isSelected && "クリックされました。"}
      </div>
    </>
  );
};

export default Example;
```

react-guide-material > 07_styling_component > src > 030_css_module > end > Example.jsx > ...

```
eact-guide-material
03_js_basic
04_react_basic
05_state_and_event
06_control_and_form
07_styling_component
node_modules
public
src
> 010_inline_style
> 020_import_css
v 030_css_module
v end
v components
SubButton.jsx
# SubButton.module.css
Example.jsx
# Example.module.css
> start
> 040_css_in_js
> 045_practice_css_in_js
> 050_chakra_ui
# App.css
App.jsx
BaseErrorBoundary.jsx
# index.css
lectures.js
```

```jsx
import { useState } from "react";

import SubButton from "./components/SubButton";
import styles from "./Example.module.css";

// console.log(styles);

const Example = () => {
  const [isSelected, setIsSelected] = useState(false);

  const clickHandler = () => setIsSelected((prev) => !prev);

  return (
    <>
      <button className={`${styles.btn} ${isSelected ? styles.selected : ""}`} onClick={clickHandler}>
        ボタン
      </button>
      <SubButton />
      <div style={{ textAlign: "center" }}>
        {isSelected && "クリックされました。"}
      </div>
    </>
  );
};

export default Example;
```

Example.jsx:16

```
▼ {btn: '_btn_16hsy_1', selected: '_selected_16hsy_12'}
  btn: "_btn_16hsy_1"
  selected: "_selected_16hsy_12"
  ▶ [[Prototype]]: Object
```

```
<button class="_btn_16hsy_1 _selected_16hsy_12">ボタン</button>
<button class="_btn_12uhm_1">サブボタン</button>
```

react-guide-material > 07_styling_component > src > 030

```
eact-guide-material
07_styling_component
src
v 030_css_module
v end
v components
SubButton.jsx
# SubButton.module.css
Example.jsx
# Example.module.css
> start
> 040_css_in_js
```

```css
.btn {
    margin: auto;
    border-radius: 9999px;
    border: none;
    display: block;
    width: 120px;
    height: 60px;
    font-weight: bold;
    cursor: pointer;
}

.selected {
```

```
.btn {
    margin: auto;
    border-radius: 9999px;
    border: none;
    display: block;
    width: 120px;
    height: 60px;
    font-weight: bold;
    cursor: pointer;
}

.selected {
    background-color: pink;
```

045_practice_css_in_js
050_chakra_ui
App.css

```
    background-color: pink;
}
```

```
}
```

react-guide-material > 07_styling_component > src > 030_css_module > end > components > SubB

- react-guide-material
- 07_styling_component
- src
  - 030_css_module
    - end
      - components
        - SubButton.jsx
        - SubButton.module.css
      - Example.jsx
      - Example.module.css

```
import styles from "./SubButton.module.css";

const SubButton = () => {
    return <button className={styles.btn}>サブボタン</button>
}

export default SubButton;
```

```
import styles from "./SubButton.module.css";

const SubButton = () => {
    return <button className={styles.btn}>サブボタン</button>
}

export default SubButton;
```

react-guide-material > 07_styling_component > src > 030_cs

- react-guide-material
- 07_styling_component
- src
  - 030_css_module
    - end
      - components
        - SubButton.jsx
        - SubButton.module.css
      - Example.jsx
      - Example.module.css
    - start
  - 040_css_in_js
  - 045_practice_css_in_js
  - 050_chakra_ui
  - App.css
  - App.jsx
  - BaseErrorBoundary.jsx
  - index.css
  - lectures.js

```
.btn {
    margin: auto;
    border-radius: 9999px;
    border: none;
    display: block;
    width: 120px;
    height: 60px;
    font-weight: bold;
    cursor: pointer;
    background: orange;
}

.btn:hover {
    background-color: red;
}

.btn:active {
    background-color: purple;
}
```

```
.btn {
    margin: auto;
    border-radius: 9999px;
    border: none;
    display: block;
    width: 120px;
    height: 60px;
    font-weight: bold;
    cursor: pointer;
    background: orange;
}

.btn:hover {
    background-color: red;
}

.btn:active {
    background-color: purple;
}
```

```
.selected {
    background-color: pink;
}

@media (max-width: 600px) {
    .btn {
        border-radius: 0;
    }
}
```

78. 【styled-components】CSS-in-JSを使ったスタイリング

▼以下のコマンドでインストール可能
npm i styled-components
https://github.com/styled-components/styled-components

▼以下の styled.button は関数！ タグ付きテンプレート という！
`const StyledButton = styled.button``
https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Template_literals
※この拡張機能を使えばcssが見やすくなる！
vscode-styled-components

▼2024年 現在 CSS in JS がトレンドになっている

**CSS in JS**

```
const StyledButton = styled.div``
```

**“CSSをJSファイル内に記載して、CSSを適用したコンポーネントを作成する”**

**特徴**
- **クラス名の衝突が起きない**
- **ライブラリを導入する必要がある**
- **CSSとJSが１つのファイルにまとまる**
- **propsを参照して動的にスタイルできる**
- **疑似要素やメディアクエリが使用できる**

79. 【styled-components】【発展】CSS-in-JSを使ったスタイリング

< css-in-jsのメリットとデメリット >

■メリット
- スタイルをコンポーネントで定義するので、外部のcssに依存することなくコンポーネント単体で動作する
- JavaScriptで記述するため、JSの文法が使用出来たり、propsとして値を渡すこともできる
- ユニークなクラス名が自動生成され他のコンポーネントに影響を与えないことが保証される → →
- cssの設計が必要なくなる
- コンポーネントで完結しているため、他のプロジェクトで再利用がしやすい

★ランダムなクラス名になる。 他のスタイルに影響されない。

```
<button class="sc-FEMpB gVreue">ボタン</button>
<button class="sc-FEMpB sc-gjcoXW gVreue fxZowW">サブボタン</button>
<button class="sc-FEMpB sc-dVBluf gVreue bdqnPo">…</button>
```

■デメリット
- 自動生成されるユニークなクラス名が読めない
- cssに比べパフォーマンスに劣る
  ※ 些細な差なのでデメリットというほどでも無い
  ※ どうしても気になる方は、Nextjsを使用することでパフォーマンスの面は気にしなくてよくなります。

【重要】

```
import styled from "styled-components";
```

↓

console.dir(styled); で確認すると複数のプロパティ有、htmlのタグと一致する。

```
import styled from "styled-components";
console.dir(styled);
```

↓

<button>タグを使いたい場合は、styled.button と書く！

↓

styled.button は関数。タグ付きテンプレート という。

```
const StyledButton = styled.button``
```

```
<StyledButton $isS
  ボタン
</StyledButton>
```

```
<button class="sc-FEMpB gVreue">ボタン</button>
```

```
▼ ƒ pt(e)
  ▶ a: ƒ (t)
  ▶ abbr: ƒ (t)
  ▶ address: ƒ (t)
  ▶ area: ƒ (t)
  ▶ article: ƒ (t)
  ▶ aside: ƒ (t)
  ▶ audio: ƒ (t)
  ▶ b: ƒ (t)
  ▶ base: ƒ (t)
  ▶ bdi: ƒ (t)
  ▶ bdo: ƒ (t)
  ▶ big: ƒ (t)
  ▶ blockquote: ƒ (t)
  ▶ body: ƒ (t)
  ▶ br: ƒ (t)
  ▶ button: ƒ (t)
  ▶ canvas: ƒ (t)
  ▶ caption: ƒ (t)
  ▶ circle: ƒ (t)
  ▶ cite: ƒ (t)
  ▶ clipPath: ƒ (t)
  ▶ code: ƒ (t)
  ▶ col: ƒ (t)
  ▶ colgroup: ƒ (t)
  ▶ data: ƒ (t)
  ▶ datalist: ƒ (t)
  ▶ dd: ƒ (t)
  ▶ defs: ƒ (t)
  ▶ del: ƒ (t)
  ▶ details: ƒ (t)
  ▶ dfn: ƒ (t)
  ▶ dialog: ƒ (t)
  ▶ div: ƒ (t)
  ▶ dl: ƒ (t)
  ▶ dt: ƒ (t)
  ▶ ellipse: ƒ (t)
  ▶ em: ƒ (t)
  ▶ embed: ƒ (t)
  ▶ fieldset: ƒ (t)
  ▶ figcaption: ƒ (t)
  ▶ figure: ƒ (t)
  ▶ footer: ƒ (t)
  ▶ foreignObject: ƒ
  ▶ form: ƒ (t)
  ▶ g: ƒ (t)
  ▶ h1: ƒ (t)
  ▶ h2: ƒ (t)
  ▶ h3: ƒ (t)
  ▶ h4: ƒ (t)
```

```
1  import { useState } from "react";
2  import styled from "styled-components";
3
4  /* POINT 生成する要素を指定し、スタイルをテンプレートリテラルで記述します */
5  /* React要素扱いなので変数名は大文字で記述！ */
6  const StyledButton = styled.button`
7    margin-inline: auto;
8    border-radius: 9999px;
9    border: none;
10   display: block;
11   width: 120px;
12   height: 60px;
13   margin: 10px auto;
14   font-weight: bold;
15   cursor: pointer;
16   text-align: center;
17   line-height: 60px;
18   /*
19   [POINT] valueを関数にすることで、引数にpropsを受け取ることができる。
20   { $isSelected } の部分を変更することで受け取る名前を変更することができる。
21   2024/04 Transient props を使用するように修正 isSelected → $isSelected
22   参考: https://styled-components.com/docs/api#transient-props
23   */
24   background-color: ${( $isSelected ) => ($isSelected ? "pink" : "darkcyan")};
25   transition: all 0.3s ease-out;
26
27   /* 疑似クラスの追加 */
28   /* 2924/04 ホバー時のスタイルが正しく適用していないの修正*/
```

```
console.dir(styled);
```

```
29    &:hover,
30    :active {
31      opacity: 0.7;
32      transform: scale(1.1);
33    }
34    span {
35      color: purple;
36    }
37    /* メディアクエリ */
38    @media (max-width: 600px) {
39      border-radius: 0;
40    }
41
42    :global {
43      background-color: black;
44    }
45  `;
46
47  // [POINT] 上記のスタイルの継承。styled()でラップする
48  // 2024/04 Transient props を使用するように修正
49  const StyledSubButton = styled(StyledButton)`
50    background-color: ${({ $isSelectedSub }) =>
51      $isSelectedSub ? "crimson" : "coral"};
52  `;
53
54  const StyledOliveButton = styled(StyledButton)`
55    background-color: olive;
56  `;
57
58  const Example = () => {
59    const [isSelected, setIsSelected] = useState(false);
60    const [isSelectedSub, setIsSelectedSub] = useState(false);
61
62    const onClickHandler = () => setIsSelected(!isSelected);
63    const onClickSubHandler = () => setIsSelectedSub(!isSelectedSub);
64
65    return (
66      // 属性にある isSelected は 上記で定義されています。
67      // background-color: ${({ isSelected }) => ~~~ };
68      <>
69        <StyledButton $isSelected={isSelected} onClick={onClickHandler}>
```

```
70          ボタン
71        </StyledButton>
72
73        <StyledSubButton
74          $isSelectedSub={isSelectedSub}
75          onClick={onClickSubHandler}
76        >
77          サブボタン
78        </StyledSubButton>
79
80        <StyledOliveButton>
81          <span>オリーブ</span>
82        </StyledOliveButton>
83      </>
84    );
85  };
86
87  export default Example;
88
```

```
import { useState } from "react";
import styled from "styled-components";

/* POINT 生成する要素を指定し、スタイルをテンプレートリテラルで記述します */
/* React要素扱いなので変数名は大文字で記述！ */
const StyledButton = styled.button`
  margin-inline: auto;
  border-radius: 9999px;
  border: none;
  display: block;
  width: 120px;
  height: 60px;
  margin: 10px auto;
  font-weight: bold;
  cursor: pointer;
  text-align: center;
  line-height: 60px;
  /*
  [POINT] valueを関数にすることで、引数にpropsを受け取ることができる。
  { $isSelected } の部分を変更することで受け取る名前を変更することができる。
  2024/04 Transient props を使用するように修正 isSelected → $isSelected
  参考: https://styled-components.com/docs/api#transient-props
  */
```

```
  background-color: ${( $isSelected ) => ($isSelected ? "pink" : "darkcyan")};
  transition: all 0.3s ease-out;

  /* 疑似クラスの追加 */
  /* 2924/04 ホバー時のスタイルが正しく適用していないの修正*/
  &:hover,
  :active {
    opacity: 0.7;
    transform: scale(1.1);
  }
  span {
    color: purple;
  }
  /* メディアクエリ */
  @media (max-width: 600px) {
    border-radius: 0;
  }

  :global {
    background-color: black;
  }
`;

// [POINT] 上記のスタイルの継承。styled()でラップする
// 2024/04 Transient props を使用するように修正
const StyledSubButton = styled(StyledButton)`
  background-color: ${({ $isSelectedSub }) =>
    $isSelectedSub ? "crimson" : "coral"};
`;

const StyledOliveButton = styled(StyledButton)`
  background-color: olive;
`;

const Example = () => {
  const [isSelected, setIsSelected] = useState(false);
  const [isSelectedSub, setIsSelectedSub] = useState(false);

  const onClickHandler = () => setIsSelected(!isSelected);
  const onClickSubHandler = () => setIsSelectedSub(!isSelectedSub);

  return (
    // 属性にある isSelected は 上記で定義されています。
```

```
    // background-color: ${({ isSelected }) => ~~~ };
    <>
      <StyledButton $isSelected={isSelected} onClick={onClickHandler}>
        ボタン
      </StyledButton>

      <StyledSubButton
        $isSelectedSub={isSelectedSub}
        onClick={onClickSubHandler}
      >
        サブボタン
      </StyledSubButton>

      <StyledOliveButton>
        <span>オリーブ</span>
      </StyledOliveButton>
    </>
  );
};

export default Example;
```

80. 【練習&解答】styled-components

```
import styled from "styled-components";

const FirstButton = styled.button`
  display: inline-block;
  border-radius: 3px;
  padding: 0.5rem 0;
  margin: 0.5rem 1rem;
  width: 11rem;
  background: pink;
  border: none;
`;

const SecondButton = styled(FirstButton)`
  background: red;
  color: white;
`;
```

ThirdButton / ボタン3 / dark

```
const ThirdButton = styled(SecondButton)`
  background: ${(props) => (props.dark ? "black" : "green")};
`;

const Example = () => {
  return (
    <>
      <FirstButton>FirstButton / ボタン1</FirstButton>

      <p>下のボタン2<br />FirstButtonを継承したSecondButtonを作成し、
      backgroudをredに、colorをwhiteへ</p>
      <SecondButton>SecondButton / ボタン2</SecondButton>

      <p>下のボタン3<br />SecondButtonを継承したThirdButtonを作成し、
      props.darkがある場合のみbackgroudがblackに、無い場合はgreenへ</p>
      <ThirdButton>ThirdButton / ボタン3</ThirdButton>
      <ThirdButton dark>ThirdButton / ボタン3 / dark</ThirdButton>
    </>
  );
};

export default Example;
```

```
import styled from "styled-components";

const FirstButton = styled.button`
  display: inline-block;
  border-radius: 3px;
  padding: 0.5rem 0;
  margin: 0.5rem 1rem;
  width: 11rem;
  background: pink;
  border: none;
`;

const SecondButton = styled(FirstButton)`
  background: red;
  color: white;
`;

const ThirdButton = styled(SecondButton)`
```

```
  background: ${(props) => (props.dark ? "black" : "green")};
`;

const Example = () => {
  return (
    <>
      <FirstButton>FirstButton / ボタン1</FirstButton>

      <p>下のボタン2<br />FirstButtonを継承したSecondButtonを作成し、
      backgroudをredに、colorをwhiteへ</p>
      <SecondButton>SecondButton / ボタン2</SecondButton>

      <p>下のボタン3<br />SecondButtonを継承したThirdButtonを作成し、
      props.darkがある場合のみbackgroudがblackに、無い場合はgreenへ</p>
      <ThirdButton>ThirdButton / ボタン3</ThirdButton>
      <ThirdButton dark>ThirdButton / ボタン3 / dark</ThirdButton>
    </>
  );
};

export default Example;
```

81. 【まとめ】Reactでのスタイルの適用方法

<スタイリング方法の比較>

- CSSファイルの読み込み
- インラインスタイル
- CSS Modules
- CSS in JS

**CSSファイルの読み込み** `import "./Example.css";`

“CSSファイルにclassを定義して、JSXのclassNameに適用する”

特徴

・グローバルスコープとなるためクラス名の衝突が起きやすい

・ルートファイル(App.js等)でグローバルなスタイルを当てたいときに使用する

★インラインスタイルはあまり使わない…★

## インラインスタイル

```
style={{ color: 'red' }}
```

“JSXのstyle属性にオブジェクトを渡す”

特徴

・再利用性が低い
・疑似要素やメディアクエリが使用できない
・レンダリングの度に計算されるのでパフォーマンスが劣る
・動的で頻繁に計算されるスタイルの適用

## CSS Modules

```
import styles from
"./Example.module.css";
```

“CSSファイルをモジュールとしてJSファイルに読み込んで、コンポーネントごとにローカルスコープを作ってスタイルを適用する”

特徴

・クラス名の衝突が起きない
・create-react-appで設定済みのため、すぐ使える
・将来、非推奨になる可能性がある
・CSSとJSが２つのファイルに分かれる

▼2024年 現在 CSS in JS がトレンドになっている

## CSS in JS

```
const StyledButton = styled.div``
```

“CSSをJSファイル内に記載して、CSSを適用したコンポーネ

ントを作成する”

**特徴**

- クラス名の衝突が起きない
- ライブラリを導入する必要がある
- CSSとJSが１つのファイルにまとまる
- propsを参照して動的にスタイルできる
- 疑似要素やメディアクエリが使用できる

## ポータル

Portal

ポータルの子要素を、直接の親要素ではなく別のDOM要素にマウントすることができる。

セクション8: ReactでDOM操作を行う方法

86. 【createPortal】モーダルの作り方

＜ createPortalはどんなときに使うか？ ＞

子要素は親要素のスタイルによって表示に制限を受ける場合があります。

（overflow: hidden 、 z-index 、 width　など・・・）

それらの制限なく、子要素が親要素を「飛び出して」表示する必要があるときにcreatePortalを使うのが有効です。

モーダル、ポップアップ、トーストは使用の代表例です。

＜ createPortal()の使い方 ＞

第一引数: React の子要素としてレンダー可能なもの （要素、文字列、フラグメント、コンポーネントなど）

第二引数: レンダー先のDOM要素

react-guide-material > 08_other_function > src > 010_portals > end > Example.jsx > ...

```
import { useState } from "react";
import { createPortal } from "react-dom";
import Modal from "./components/Modal";

const ModalPortal = ({ children }) => {
  const target = document.querySelector(".container.end");
  return createPortal(children, target);
};

const Example = () => {
  const [modalOpen, setModalOpen] = useState(false);
  return (
    <div>
      <div className="container end"></div>

      <button
```

モーダルを表示する

モーダル

閉じる

```
import { useState } from "react";
import { createPortal } from "react-dom";
import Modal from "./components/Modal";

const ModalPortal = ({ children }) => {
  const target = document.querySelector(".container.end");
  return createPortal(children, target);
};

const Example = () => {
  const [modalOpen, setModalOpen] = useState(false);
  return (
    <div>
      <div className="container end"></div>

      <button
        type="button"
        onClick={() => setModalOpen(true)}
        disabled={modalOpen}
      >
        モーダルを表示する
      </button>

      {modalOpen && (
```

```
        <ModalPortal>
          <Modal handleCloseClick={() => setModalOpen(false)} />
        </ModalPortal>
      )}
    </div>
  );
};

export default Example;
```

react-guide-material > 08_other_function > src > 010_portals > end > components > Modal.jsx

```
import "./Modal.css";

const Modal = ({ handleCloseClick }) => {
  return (
    <div className="modal">
      <div className="modal__content">
        <p>モーダル</p>
        <button type="button" onClick={handleCloseClick}>
          閉じる
        </button>
      </div>
    </div>
  );
};

export default Modal;
```

react-guide-material > 08_other_function > src > 010_portals > end > comp

```
.App-end .container .modal {
  position: absolute;
  top: 0;
  left: 0;
  bottom: 0;
  right: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.3);
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 100;
}
```

```
> start
> 020_practice_port...
> 030_useRef
> 040_forwardRef
> 050_useImperativ...
> 060_practice_ref
# App.css
```

```
  z-index: 100;
}

.App-end .container .modal__content {
  background-color: #fff;
  padding: 20px 50px;
}
```

```
.App-end .container .modal__content {
  background-color: #fff;
  padding: 20px 50px;
}
```

87. 【Bubbling】Portalを使う際の注意点！

<注意点>
Bubbling(バブリング)：イベントが子要素から親要素へ伝播していくこと

モーダルの[閉じる]ボタンを押すと、
以下の console.log('container発火しない…') が発火しそうだが、発火しない…
console.log('空のdiv')}が発火する！

```
<div onClick={() => console.log('空のdiv')}>
   <div className="container end" onClick={() => console.log('container発火しない！')}></div>
```

理由：React要素のツリーの場合は、React要素のツリーの構成にしたがってバブリングする！上の図が分かりやすいかも。

```
react-guide-material
08_other_function
> node_modules
> public
```

react-guide-material > 08_other_function > src > 010_portals > end > Example.jsx > Example

```
import { createPortal } from "react-dom";
import Modal from "./components/Modal";

const ModalPortal = ({ children }) => {
```

```jsx
import { useState } from "react";
import { createPortal } from "react-dom";
import Modal from "./components/Modal";

const ModalPortal = ({ children }) => {
  const target = document.querySelector(".container.end");
  return createPortal(children, target);
};

const Example = () => {
  const [modalOpen, setModalOpen] = useState(false);
  return (
    <div onClick={() => console.log('空のdiv')}>
```

```react
      <div className="container end" onClick={() => console.log('container発火しない…')}></div>

      <button
        type="button"
        onClick={() => setModalOpen(true)}
        disabled={modalOpen}
      >
        モーダルを表示する
      </button>

      {modalOpen && (
        <ModalPortal>
          <Modal handleCloseClick={() => setModalOpen(false)} />
        </ModalPortal>
      )}
    </div>
  );
};

export default Example;
```

react-guide-material > 08_other_function > src > 010_portals > end > components > Modal.jsx

```react
import "./Modal.css";

const Modal = ({ handleCloseClick }) => {
  return (
    <div className="modal">
      <div className="modal__content">
        <p>モーダル</p>
        <button type="button" onClick={handleCloseClick}>
          閉じる
        </button>
      </div>
    </div>
  );
};

export default Modal;
```

react-guide-material > 08_other_function > src > 010_portals > end > comp

```css
.App-end .container .modal {
  position: absolute;
  top: 0;
```

```react
import "./Modal.css";

const Modal = ({ handleCloseClick }) => {
  return (
    <div className="modal">
      <div className="modal__content">
        <p>モーダル</p>
        <button type="button" onClick={handleCloseClick}>
          閉じる
        </button>
      </div>
    </div>
  );
};

export default Modal;
```

```css
.App-end .container .modal {
  position: absolute;
  top: 0;
  left: 0;
```

```
public
src
010_portals
end
components
Modal.css
Modal.jsx
_Example_コピー...
Example.jsx
start
020_practice_port...
030_useRef
040_forwardRef
050_useImperativ...
060_practice_ref
App.css
```

```
  left: 0;
  bottom: 0;
  right: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.3);
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 100;
}

.App-end .container .modal__content {
  background-color: #fff;
  padding: 20px 50px;
}
```

```
  bottom: 0;
  right: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.3);
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 100;
}

.App-end .container .modal__content {
  background-color: #fff;
  padding: 20px 50px;
}
```

88. 【練習＆解答】createPortalでトーストを作成してみよう

```
react-guide-material
08_other_function
node_modules
public
src
010_portals
020_practice_port...
end
components
Toast.css
Toast.jsx
Example.jsx
start
030_useRef
040_forwardRef
050_useImperativ...
060_practice_ref
App.css
App.jsx
```

```
import { useState } from "react";
import { createPortal } from "react-dom";
import Toast from "./components/Toast";

const ToastPortal = ({ children }) => {
  const target = document.querySelector(".container.end");
  return createPortal(children, target);
};

const Example = () => {
  const [toastOpen, setToastOpen] = useState(false);
  return (
    <div>
      <div className="container end"></div>

      <button
        type="button"
        onClick={() => setToastOpen(true)}
        disabled={toastOpen}
```

BaseErrorBounda...
index.css
lectures.js
main.jsx
.eslintrc.cjs
.gitignore
index.html
package-lock.json
package.json
README.md
vite.config.js
09_debugging
10_functional_progr...
11_hooks_p1
12_hooks_p2
13_redux
14_class_component

```
      >
        トーストを表示する
      </button>

      {toastOpen && (
        <ToastPortal>
          <Toast
            visible={toastOpen}
            handleCloseClick={() => setToastOpen(false)}
          />
        </ToastPortal>
      )}
    </div>
  );
};

export default Example;
```

```
import { useState } from "react";
import { createPortal } from "react-dom";
import Toast from "./components/Toast";

const ToastPortal = ({ children }) => {
  const target = document.querySelector(".container.end");
  return createPortal(children, target);
};

const Example = () => {
  const [toastOpen, setToastOpen] = useState(false);
  return (
    <div>
      <div className="container end"></div>

      <button
        type="button"
        onClick={() => setToastOpen(true)}
        disabled={toastOpen}
      >
        トーストを表示する
      </button>

      {toastOpen && (
        <ToastPortal>
```

```
          <Toast
            visible={toastOpen}
            handleCloseClick={() => setToastOpen(false)}
          />
        </ToastPortal>
      )}
    </div>
  );
};

export default Example;
```

react-guide-material > 08_other_function > src > 020_practice_portals > end > components > Toast.jsx

react-guide-material
- 08_other_function
  - node_modules
  - public
  - src
    - 010_portals
    - 020_practice_port...
      - end
        - components
          - Toast.css
          - Toast.jsx
        - Example.jsx
        - start
    - 030_useRef
    - 040_forwardRef
    - 050_useImperativ...
    - 060_practice_ref
    - App.css
    - App.jsx
    - BaseErrorBounda...
    - index.css

```
import "./Toast.css";

const Toast = ({ visible, handleCloseClick }) => {
  const toastClassName = visible ? "toast is-visible" : "toast";
  return (
    <div className={toastClassName}>
      <div className="toast__content">
        <p>トースト</p>
        <button
          type="button"
          className="toast__button"
          onClick={handleCloseClick}
        >
          ×
        </button>
      </div>
    </div>
  );
};

export default Toast;
```

```
import "./Toast.css";

const Toast = ({ visible, handleCloseClick }) => {
  const toastClassName = visible ? "toast is-visible" : "toast";
  return (
    <div className={toastClassName}>
      <div className="toast__content">
```

```
        <p>トースト</p>
        <button
          type="button"
          className="toast__button"
          onClick={handleCloseClick}
        >
          ×
        </button>
      </div>
    </div>
  );
};

export default Toast;
```

react-guide-material > 08_other_function > src > 020_practice_portals > end > comp

- react-guide-material
  - 08_other_function
    - node_modules
    - public
    - src
      - 010_portals
      - 020_practice_port...
        - end
          - components
            - Toast.css
            - Toast.jsx
          - Example.jsx
          - start
      - 030_useRef
      - 040_forwardRef
      - 050_useImperativ...
      - 060_practice_ref
      - App.css
      - App.jsx
      - BaseErrorBounda...
      - index.css
      - lectures.js
      - main.jsx
    - .eslintrc.cjs
    - .gitignore

```
.App-end .container .toast {
  position: absolute;
  bottom: 0;
  left: 0;
  width: 100%;
  padding: 10px;
  transform: translateY(100%);
  z-index: 100;
  display: none;
}
.App-end .container .toast.is-visible {
  transform: translateY(0);
  transition: transform 0.3s ease-in;
  display: block;
}

.App-end .container .toast__content {
  position: relative;
  background-color: rgba(255, 255, 255, 0.3);
  box-shadow: 0 0 8px gray;
  padding: 12px 50px;
}

.App-end .container .toast__button {
  background-color: transparent;
  border: none;
  cursor: pointer;
```

```
  outline: none;
  padding: 0;
  appearance: none;
  position: absolute;
  top: 10px;
  right: 10px;
  width: 30px;
  height: 30px;
  font-size: 30px;
}

.App-end .container .toast__button:active {
  opacity: 0.3;
}
```

89. 【useRef】refでDOMを直接操作してみよう

セクション21: 【付録】TypeScript

250. プリミティブな値の型定義について学ぼう

```
let str: string = 'Hello';
```

↑この string は型

```
// POINT プリミティブとは
// プリミティブとは、TypeScriptが扱うことの出来る基本的な値のこと
// 具体的な例としては、文字列、数値、巨大な数値、真偽値、null、undefinedなどがある

const Example = () => {
  let str: string = 'Hello';
  str = 'Bye';
  console.log(str);

  let num: number = 102;
```


```
// POINT プリミティブとは
// プリミティブとは、TypeScriptが扱うことの
   出来る基本的な値のこと
// 具体的な例としては、文字列、数値、巨大な数値、
   真偽値、null、undefinedなどがある
const Example = () => {
  let str: string = 'Hello';
  str = 'Bye';
  console.log(str);

  let num: number = 102;
```

```
  console.log(num);

  let bignum: bigint = 103n;
  console.log(bignum);

  let bool: boolean = true;
  console.log(bool);

  let nullish: null = null;
  let undefinedValue: undefined = undefined;

  // リテラル型
  let trueVal: true = true;
  let num123: 123 = 123;
  let strHello: 'Hello' = 'Hello';

  // any型(何でも格納可能)
  let anyValue: any = [];
};

export default Example;
```

```
  console.log(num);

  let bignum: bigint = 103n;
  console.log(bignum);

  let bool: boolean = true;
  console.log(bool);

  let nullish: null = null;
  let undefinedValue: undefined = undefined;

  // リテラル型
  let trueVal: true = true;
  let num123: 123 = 123;
  let strHello: 'Hello' = 'Hello';

  // any型(何でも格納可能)
  let anyValue: any = [];
};

export default Example;
```

251. ユニオン型の定義方法について学ぼう

ユニオン型(union type)は、複数の型を組み合わせて「型Tまたは型U」のような、「または」の意味を表すことができる

```
let strOrNum: string | number = 'Hello';
```

←ユニオン型

↑ |「または」

```
guide-material > 20_typescript > src > 040_union_types > end > Example.tsx > ...
// POINT ユニオン型(union type)は、複数の型を組み合わせて「型Tまたは型U」
のような、「または」の意味を表すことができる
// POINT T | U というように、| を用いてユニオン型を表す

const Example = () => {
  let strOrNum: string | number = 'Hello';
  strOrNum = 123;
  console.log(strOrNum);

  let strOrNumOrBool: string | number | boolean = 'Hello';
```

```
// POINT ユニオン型(union type)は、複数の型を組み合わせて
  「型Tまたは型U」のような、「または」の意味を表すことができる
// POINT T | U というように、| を用いてユニオン型を表す

const Example = () => {
  let strOrNum: string | number = 'Hello';
  strOrNum = 123;
  console.log(strOrNum);

  let strOrNumOrBool: string | number | boolean = 'Hello';
  strOrNumOrBool = true;
```

```typescript
  strOrNumOrBool = true;
  console.log(strOrNumOrBool);

  let helloOrNumOrBool: 'Hello' | number | boolean = false;

  type HelloOrNum = 'Hello' | number;
  const hello: HelloOrNum = 'Hello';

  type DayOfWeek =
    | 'Monday'
    | 'Tuesday'

  const day: DayOfWeek = 'Monday';
};

export default Example;
```

```typescript
  console.log(strOrNumOrBool);

  let helloOrNumOrBool: 'Hello' | number | boolean = false;

  type HelloOrNum = 'Hello' | number;
  const hello: HelloOrNum = 'Hello';

  type DayOfWeek =
    | 'Monday'
    | 'Tuesday'

  const day: DayOfWeek = 'Monday';
};

export default Example;
```

252. オブジェクトの型定義について学ぼう

配列やオブジェクトの型定義

```typescript
type Person = { name: string, age?: number }
```

↑ ? は、ageプロパティ あっても無くても良いという意味

guide-material > 20_typescript > src > 050_objects > end > Example.tsx > ...

```typescript
// POINT TypeScriptにおけるオブジェクトの型定義ついて学ぼう
const Example = () => {
  const arry1: number[] = [1,2,3];
  const arry2: string[] = ['hello','bye'];
  const arry3: Array<number> = [1,2,3];
  const arry4: (string | number)[] = [1,'bye'];
  const arry5: Array<string | number> = [1,'bye'];

  type Person = { name: string, age?: number }
  const obj1: Person = { name: 'Taro' };

  const users: { name: string, age?: number }[] = [
    { name: 'Taro' },
    { name: 'Hanako', age: 30 },
    { name: 'Jiro', age: 30 }
  ]
```

```typescript
// POINT TypeScriptにおけるオブジェクトの型定義ついて学ぼう
const Example = () => {
  const arry1: number[] = [1,2,3];
  const arry2: string[] = ['hello','bye'];
  const arry3: Array<number> = [1,2,3];
  const arry4: (string | number)[] = [1,'bye'];
  const arry5: Array<string | number> = [1,'bye'];

  type Person = { name: string, age?: number }
  const obj1: Person = { name: 'Taro' };

  const users: { name: string, age?: number }[] = [
    { name: 'Taro' },
    { name: 'Hanako', age: 30 },
    { name: 'Jiro', age: 30 }
  ]
};
```

```
};

export default Example;
```

```
export default Example;
```

253. 型推論とは？暗黙的な型定義について学ぼう

▼str にホバーすると以下のように string型 が設定してある。型推論。

```
const
  let str: string
  let str = 'hello';
```

▼ obj にホバー、型が定義されている。型推論。

```
// POINT TypeScriptでは型推論(type infere
// 基本
let obj: {
    name: string;
    age: number;
}
const
  let
  let obj = { name: 'Taro', age: 10 };
```

// POINT TypeScriptでは型推論(type inference)によって型をある程度推定してくれる機能がある
// 基本的には、明らかに型が分かるような場合の型定義は型推論に任せるようにする

```
// POINT TypeScriptでは型推論(type inference)に
よって型をある程度推定してくれる機能がある
// 基本的には、明らかに型が分かるような場合の型定義
は型推論に任せるようにする

const Example = () => {
  let str = 'hello';
  let obj = { name: 'Taro', age: 10 };

  const bye = 'bye';
  const num = 123;
};

export default Example;
```

```
const Example = () => {
  let str = 'hello';
  let obj = { name: 'Taro', age: 10 };

  const bye = 'bye';
  const num = 123;
};

export default Example;
```

## 254. 型エイリアスについて学ぼう

type を使って、型エイリアスを作って
type User = {

TypeScriptでは、type文を用いて型に別名を付けることができる。これを 型エイリアス(type alias) と呼ぶ

```
const Example = () => {
  type User = {
    name: string,
    age: number
  }
  const user: User = { name: 'Taro', age: 10 }

  type UserName = string;
  type UserAge = number;
  type UserGender = "man" | "woman" | "others";

  type UserProfile = {
    name: UserName,
    age: UserAge,
    gender: UserGender,
  }

  const userProfile: UserProfile = {
    name: 'Hanako',
    age: 21,
    gender: "woman"
  }
};

export default Example;
```

```
const Example = () => {
  type User = {
    name: string,
    age: number
  }
  const user: User = { name: 'Taro', age: 10 }

  type UserName = string;
  type UserAge = number;
  type UserGender = "man" | "woman" | "others";

  type UserProfile = {
    name: UserName,
    age: UserAge,
    gender: UserGender,
  }

  const userProfile: UserProfile = {
    name: 'Hanako',
    age: 21,
    gender: "woman"
  }
};

export default Example;
```

## 255. 関数の型定義の方法について学ぼう（引数）

```
const Example = () => {
  function sum1(x: number, y: number = 1) {
    console.log(y);
```

```
const Example = () => {
  function sum1(x: number, y: number = 1) {
    console.log(y);
    return x + y;
```

```
    return x + y;
  }
  const result1 = sum1(1);
  console.log(result1)

  const sum2 = (x: number, y: number) => x + y;
  const result2 = sum2(10, 20);
  console.log(result2);
};

export default Example;
```

```
1
2
30
```

```
  }
  const result1 = sum1(1);
  console.log(result1)

  const sum2 = (x: number, y: number) => x + y;
  const result2 = sum2(10, 20);
  console.log(result2);
};

export default Example;
```

256. 関数の型定義の方法について学ぼう（戻り値）

```
// POINT 関数の型定義を学ぼう

const Example = () => {
  function sum1(x: number, y: number): void {
    console.log(y);
    // return x + y;
  }
  const result1 = sum1(1, 2);
  console.log(result1);

  const sum2 = (x: number, y: number): string => "hello";
  const result2 = sum2(10, 20);
  console.log(result2);

  type Sum = (x: number, y:number) => number;
  const sum3: Sum = (x, y) => x + y;
  const result3 = sum3(100, 500);
  console.log(result3);
};

export default Example;
```

```
2
undefined
hello
600
```

```
function sum1(x: number, y: number): void {
  console.log(y);
  return x + y;
}
```

↑関数の戻り値が存在しない場合。
void（型）：関数の実行結果として、何も返らない。
なので return は不要。記述するとエラーになる。

```
const sum2 = (x: number, y: number): string => "hello";
```

↑戻り値は string の型ですよ、といっている。

```
type Sum = (x: number, y:number) => number;
```

↑このSumは型情報を定義。関数ではない。

```
const Example = () => {
  function sum1(x: number, y: number): void {
    console.log(y);
```

```
    // return x + y;
  }
  const result1 = sum1(1, 2);
  console.log(result1);

  const sum2 = (x: number, y: number): string => "hello";
  const result2 = sum2(10, 20);
  console.log(result2);

  type Sum = (x: number, y:number) => number;
  const sum3: Sum = (x, y) => x + y;
  const result3 = sum3(100, 500);
  console.log(result3);
};

export default Example;
```

257. ジェネリクスとは？型引数の定義について学ぼう

以下の関数2つ、処理はほとんど同じ、string と number が違うだけ。

```
const Example = () => {
  const repeatStr = (value: string, times: number): string[] => {
    return new Array(times).fill(value);
  }
  const strArry = repeatStr("hello", 3);
  console.log(strArry);

  const repeatNum = (value: number, times: number): number[] => {
    return new Array(times).fill(value);
  }
  const numArry = repeatNum(10, 3);
  console.log(numArry);
```

```
▸ (3) ['hello', 'hello', 'hello']
▸ (3) [10, 10, 10]
```

上記だとコードが増えてしまうので、以下のようにまとめられる。

↓続く ----------

↓<T>がジェネリクス。これを使って上記の関数2個をまとめられる！

```
const repeat = <T>(value: T, times: number): T[] => {
  return new Array(times).fill(value);
}
const numArry = repeat<number>(10, 3);
```

↓T が number になっている。という意味。

```
const repeat = <number>(value: number, times: number): number[] => {}
```

★ジェネリクスとは、型引数(type parameters)を受け取る関数を作る機能のことを指す

```
ide-material > 20_typescript > src > 090_generics > start > Example.ts > ...
// ジェネリクスとは、型引数(type parameters)を受け取る関数を作る機能のことを指す

const Example = () => {
  const repeatStr = (value: string, times: number): string[] => {
    return new Array(times).fill(value);
  }
  const strArry1 = repeatStr("hello", 3);

  const repeatNum = (value: number, times: number): number[] => {
    return new Array(times).fill(value);
  }
  const numArry1 = repeatNum(10, 3);

  ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓

  // 上記の関数2つをジェネリクスを使って書き換える
  const repeat = <T>(value: T, times: number): T[] => {
    return new Array(times).fill(value);
  }
  const numArry = repeat<number>(10, 3);
  const numArry2 = repeat(10, 3); // 型推論できるので型<number>を省略できる。
                                  // ※基本、型<number>は省略したほうが良い？
  console.log(numArry2);
  const strArry = repeat<string>("hello", 3);
  console.log(strArry);
  const boolArry = repeat<boolean>(true, 3);
  console.log(boolArry);
  const byeArry = repeat<"bye">("bye", 3);
  console.log(byeArry);
};

export default Example;
```

```
▸ (3) [10, 10, 10]
▸ (3) ['hello', 'hello', 'hello']
▸ (3) [true, true, true]
▸ (3) ['bye', 'bye', 'bye']
```

```
const Example = () => {
  const repeatStr = (value: string, times: number): string[] => {
    return new Array(times).fill(value);
```

```
}
const strArry1 = repeatStr("hello", 3);

const repeatNum = (value: number, times: number): number[] => {
  return new Array(times).fill(value);
}
const numArry1 = repeatNum(10, 3);
```

↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓

▼上記の関数2つをジェネリクスを使って書き換える

```
const repeat = <T>(value: T, times: number): T[] => {
  return new Array(times).fill(value);
}
const numArry = repeat<number>(10, 3);
const numArry2 = repeat(10, 3); ← 型推論できるので型<number>を省略できる。
                                  ※基本、型<number>は省略したほうが良い？
console.log(numArry2);
const strArry = repeat<string>("hello", 3);
console.log(strArry);
const boolArry = repeat<boolean>(true, 3);
console.log(boolArry);
const byeArry = repeat<"bye">("bye", 3);
console.log(byeArry);
};

export default Example;
```

258. 関数コンポーネントに型を導入してみよう

TypeScriptでReactコンポーネントを作成しよう

React.FC = React.FunctionComponent

```
import Hello from "./Hello";

const Example: React.FC = () => {
  return <Hello />
};

export default Example;
```

```
Hello.tsx
export default Example;
```

```
const Hello: React.FC = () => {
    return <h1>Hello TypeScript</h1>
}
export default Hello;
```

259. propsに型定義を導入してみよう

props の場合、type HelloProps = {} が必須。無いとエラーになる。

```
type HelloProps = {
    text: string
}

const Hello: React.FC<HelloProps> = (props) => {
    return <h1>Hello {props.text}</h1>
}
```

型を定義 <HelloProps>

interface：入出力の部分だけ定義する、キーワード。
中の実態は実装しない。どういった入力から、どういった出力が出てくるか定義する。

```
interface HelloProps {
    text: string
}

const Hello: React.FC<HelloProps> = (props) => {
    return <h1>Hello {props.text}</h1>
}

export default Hello;
```

```
import Hello from "./Hello";

const Example: React.FC = () => {
```

src
120_props
end
Example.tsx
Hello.tsx
start

```
3  const Example: React.FC = () => {
4    return <Hello text="TypeScriptだよー"/>
5  };
6
7  export default Example;
8
```

```
  return <Hello text="TypeScriptだよー"/>
};

export default Example;
```

**Hello TypeScriptだよー**

▼children を含めるパターン

react-guide-material > 20_typescript > src > 120_props > start > Example.tsx > ...

react-guide-material
20_typescript
src
120_props
start
Example.tsx
Hello.tsx

```
1  import Hello from "./Hello";
2
3  const Example: React.FC = () => {
4    return <Hello text="TypeScriptスクリプト">Childrenなのだ</Hello>
5  };
6
7  export default Example;
8
```

react-guide-material > 20_typescript > src > 120_props > start > Hello.tsx > default

react-guide-material
20_typescript
src
120_props
start
Example.tsx
Hello.tsx
130_class
140_form_event
150_usestate

```
1  type HelloProps = {
2      text: string,
3      children: React.ReactNode
4  }
5
6  const Hello: React.FC<HelloProps> = (props) => {
7      return <h2>Hello {props.text}!!! {props.children}</h2>
8  }
9
10 export default Hello;
```

**Hello TypeScriptスクリプト!!! Childrenなのだ**

```
import Hello from "./Hello";

const Example: React.FC = () => {
  return <Hello text="TypeScriptスクリプト">Childrenなのだ</Hello>
};

export default Example;
```

```
type HelloProps = {
    text: string,
    children: React.ReactNode
}
```

```
const Hello: React.FC<HelloProps> = (props) => {
    return <h2>Hello {props.text}!!! {props.children}</h2>
}

export default Hello;
```

▼関数 パターン

react-guide-material > 20_typescript > src > 120_props > end > Example.tsx > ...

```
import Hello, { Btn } from "./Hello";

const Example: React.FC = () => {
  return (
    <>
      <Btn fn={(text) => console.log(`Hello ${text}`)} />
      <Hello text="タイプスクリプト">Childrenなのだ</Hello>
    </>
  );
};

export default Example;
```

ボタン

**Hello タイプスクリプト!!! Childrenなのだ**

ボタン クリック ↓

Hello TypeScriptですよ

react-guide-material > 20_typescript > src > 120_props > end > Hello.tsx > ...

```
type HelloProps = {
    text: string,
    children: React.ReactNode
}

const Hello: React.FC<HelloProps> = (props) => {
    return <h2>Hello {props.text}!!! {props.children}</h2>
}

type FnProps = {
    fn: (text: string) => void
}

export const Btn: React.FC<FnProps> = (props) => {
    return <button onClick={() => props.fn('TypeScriptですよ')}>ボタン</button>
}
```

良くわからない…
↓
関数( fn('TypeScriptですよ') )の
戻り値は無くて良いので、
void にしている。

```
BaseErrorBounda...
# index.css
18  export default Hello;
```

```
import Hello, { Btn } from "./Hello";

const Example: React.FC = () => {
  return (
    <>
      <Btn fn={(text) => console.log(`Hello ${text}`)} />
      <Hello text="タイプスクリプト">Childrenなのだ</Hello>
    </>
  );
};

export default Example;
```

```
type HelloProps = {
    text: string,
    children: React.ReactNode
}

const Hello: React.FC<HelloProps> = (props) => {
    return <h2>Hello {props.text}!!! {props.children}</h2>
}

type FnProps = {
    fn: (text: string) => void
}

export const Btn: React.FC<FnProps> = (props) => {
    return <button onClick={() => props.fn('TypeScriptですよ')}>ボタン</button>
}

export default Hello;
```

260. 【おまけ】classの型定義方法について学ぼう