

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JavaScriptの関数として定義する。

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

<Welcome />

★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
```

↓
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"  
},
```

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

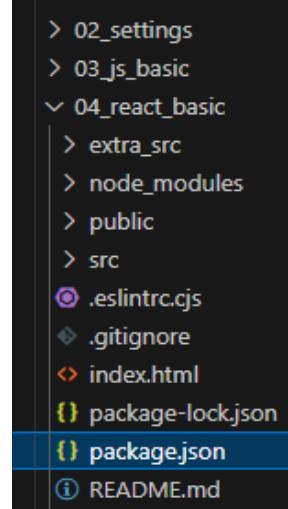
↓

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;  
}
```

```
> 050_project_sample  
< 060_styling (selected)  
< end  
  # Example.css  
  Example.jsx (highlighted in blue)  
< start  
  Example.jsx  
> 070_component_nest  
> 073_practice_compo...  
> 075_fragment  
> 080_expr_in_jsx  
> 085_over_and_out
```

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

Hello Component

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;
```

```
04_react_basic
  src
    070_component_nest
      > end
      > start
        components
          List.jsx
          # Example.css
          Example.jsx
          > 073_practice_compo...
          > 075_fragment
          > 080_expr_in_jsx
          > 085_expr_and_state
          > 087_practice_expr
```

```
1  import "./Example.css";
2  import { List } from "./components/List";
3
4  const Example = () => {
5    return (
6      <div className="component">
7        <h3>Hello Component</h3>
8        <List />
9      </div>
10 );
11 }
12
13 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>

```
04_react_basic
  src
    070_component_nest
      > end
      > start
        > return()
```

```
start
components
  List.jsx
# Example.css
Example.jsx
> 073_practice_compo...
> 075_fragment
> 080_expr_in_jsx
> 085_expr_and_state
> 087_practice_exor
5   return (
6     <div className="component">
7       <h3>Hello Component</h3>
8       <List />
9     </div>
10    );
11  };
12
13  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
1  const List = () => {
2   return (
3     <ul>
4       <li>item-1</li>
5       <li>item-2</li>
6       <li>item-3</li>
7       <li>item-4</li>
8       <li>item-5</li>
9     </ul>
10   );
11 }
12
13 // POINT List コンポーネントを 名前付き export
14 export { List };
```

```
070_component_nest
  end
    components
      # Child.css
      Child.jsx
      List.jsx
      Example.jsx
4 .App-end .component {
5   padding: 1rem;
6   color: blue;
7   border: 5px solid blue;
8
9 }
10
```

```
react-guide-material
  04_react_basic
    src
      070_component_nest
        end
          components
            # Child.css
            Child.jsx
            List.jsx
            Example.jsx
          start
1 // POINT List コンポーネントを名前付き import
2 import { List } from "./List";
3 import "./Child.css"; ←
4
5 const Child = () => {
6   return (
7     <div className="component">
8       <h3>Hello Component</h3>
9       <List />
10      </div>
11    );
12 }
```

```
> 073_practice_compon... 12 };
> 075_fragment 13
> 080_expr_in_jsx 14 // POINT コンポーネントを default export
> 085_expr_and_state 15 export default Child;
> 087
```

★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
      > .
```

4 import Child from "./components/Child";
5
6 const Example = () => <Child />;
7 // const Example = () => {
8 // return <Child />
9 // };
10
11 export default Example;
12

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！`<div>`で一つにまとめる！

だが、不要の`<div>`は書きたくない！無駄な`<div>`。

```
const Child = () => {
  return (
    <div>
      <div className="component">
        <h3>Hello Component</h3>
      </div>
      <h3>Hello</h3>
      <p>テストテスト</p>
    </div>
  ),
}
```

```
};
```

```
</div>
```

```
);
```

↓
上記右の場合だと、不要の<div>(ルート要素)がある。無駄な<div>。

★重要★一般的な書き方

<div>(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

```
react-guide-material  
  04_react_basic  
    src  
      080_expr_in_jsx  
        start  
          components  
            # Child.css  
            # Child.jsx  
            # Expression.css  
            # Expression.jsx  
            Example.jsx  
        085_expr_and_state
```

```
1 import Child from "./components/Child";  
2 import Expression from "./components/Expression";  
3  
4 const Example = () => (  
5   <>  
6     <Child />  
7     <Expression />  
8   </>  
9 );  
10  
11 export default Example;
```

Hello Component

Hello Expression

メソッドも実行できる！

```
import "./Expression.css";  
  
const Expression = () => {  
  const title = "Expression";  
  return (  
    <h3>Hello {title}</h3>  
  );  
};  
  
Expression.prototype.handleClick = function() {  
  console.log("Hello Clicked");  
};  
  
export default Expression;
```

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> ← ここに配列が埋め込まれます
    </div>
  );
};

```

配列の中身が展開されて表示される！

```

Hello Expression
item1item2item3

```

```

▼ <div class="expression">
  ▶ <h3>...</h3>
  ▼ <h3> == $0
    ...
    "item1"
    "item2"
    "item3"
  </h3>
</div>

```

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`; ← こちらは省略しないver
  // };

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

```

```

Hello Function

```

```

▼ <div>
  <h3>Hello Function</h3>
</div>

```

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>
);

```

JSXです！

▼<div>
 <h3>JSXです！</h3>
</div>

※ javascript のコードとみなされる。

変数に代入したJSXも埋め込みます。

```
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" />;
<090_props>
  <start>
    <components>
      # Child.css
      Child.jsx
    <Example.jsx>
  > 100_practice_props
```

▼props 記述

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

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

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

★補足★
props → p でもOK

```
# App.css          15
App.jsx           16  export default Child;
BaseErrorBoundary.jsx 17
```

↓ -----

color="red" 設定

```
090_props          4  import Child from "./components/Child";
start             5
components        6  const Example = () => <Child color="red" />;
# Child.css
Child.jsx
Example.jsx      8  export default Example;
100_practice_props 9
```

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

```
src               3  import "./Child.css";
090_props         4
start             5  const Child = (props) => {
components        6    return (
# Child.css
Child.jsx        7      <div className={`component ${props.color}`}>
8        <h3>Hello Component</h3>
9      </div>
);
10
11  };
12
13  export default Child;
14
```

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

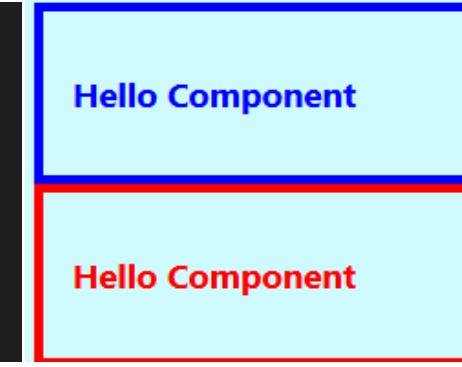
Hello Component

↓ -----

▼2個記述することで、以下のような結果になる！

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

```
react-guide-material
  04_react_basic
    src
      090_props
        start
          components
            # Child.css
            Child.jsx
            Example.jsx
          100_practice_props
            1 import Child from "./components/Child";
            2
            3 const Example = () => {
            4   return (
            5     <>
            6       <Child color="" />
            7       <Child color="red" />
            8     </>
            9   )
            10 }
```



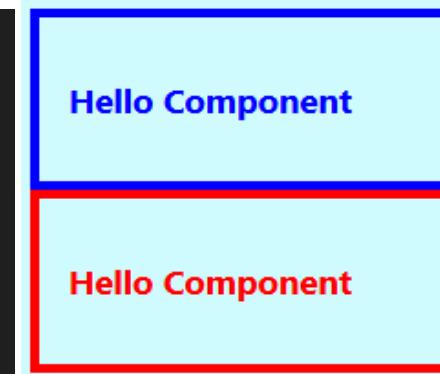
```
> 100_practice_props    10  };
> 110_props_children    11
> 120_props_rules       12  export default Example;
> 130_whats_jsx         13
```

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

```
<div className={`component ${color}`}>
  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;
 15
```

★結果は上記と同じ



※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' }) => {
  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;
 15
```

```
react-guide-material
  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
  1 import Child from "./components/Child";
  2
  3 const Example = () => {
  4   return (
  5     <>
  6       <Child />
  7       <Child color="red" />
  8     </>
  9   );
 10 }
 11
 12 export default Example;
 13

```

```

120_props_rules
130_whats_jsx
140_react_eleme...
# App.css
13
14
15

```

※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}`}>
      ...
    </div>
  );
}

```

```

src
  3 import "./Child.css";
  4
  5 const Child = ({ color: c = 'green' }) => {
  6
    return (
      <div className={`component ${c}`}>
        <h3>Hello Component</h3>
      </div>
    );
  12 }
  14 export default Child;

```

▼結果は上記と同じ



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

```

```

        2
        3 const Example = () => {
        4   const hello = (arg) => `Hello ${arg}`;
        5
        6   return (
        7     <>
        8       <Child
        9         num={123}
       10        fn={hello}
       11        bool
       12        obj={{ name: 'Tom', age: 18 }}
       13      />
       14    </>
       15  )
       16 };
       17
       18 export default Example;
       19

```

★ propsには全てのタイプの値を渡すことができます。

```

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

```

import Child from "./components/Child";

```

react-guide-material

04_react_basic

src

090_props

- > end
- start
- components
 - # Child.css
 - Child.jsx

Example.jsx

```

1 import Child from './components/Child';
2
3 const Example = () => {
4   const o = {
5     color: "red", ←
6     num: 123
7   }
8
9   return (
10    <>
11    <Child
12      {...o} ← このように書ける！ {...o} ※ ... はスプレッド演算子
13    // 以下でもOKだが面倒くさい
14    // color={o.color}
15    // num={o.num}
16    />
17  </>
18 )
19
20 };
21
22 export default Example;
23 
```

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

▼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

```

1 import Profile from "./components/Profile";
2 import Container from "./components/Container";
3 
```

▼1.Profile(子)コンポーネント

```

110_props_children
  4 const Example = () => {
  5   return (
  6     <div>
  7       <Container title="Childrenとは?">
  8         <Profile />
  9       </Container>
 10     </div>
 11   );
 12 };
 14 export default Example;
 15

```

```

react-guide-material
  1 import "./Profile.css";
  2
  3 const Profile = ({ name, age, country, color }) => {
  4   return (
  5     <div className={`profile ${color}`}>
  6       <h3>Name: {name}</h3>
  7       <p>Age: {age}</p>
  8       <p>From: {country}</p>
  9     </div>
 10   );
 11 };
 13 export default Profile;

```

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

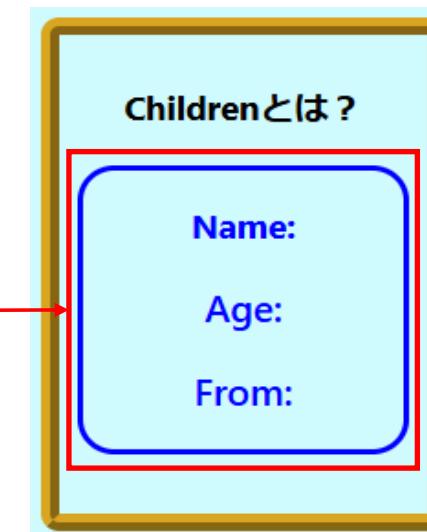
★POINT★ `props.children` として子要素を受け取る

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

```

react-guide-material
  1 import "./Container.css";
  2
  3 const Container = ({ title, children }) => {
  4   console.log(children);
  5   return (
  6     <div className="container">
  7       <h3>{title}</h3>
  8       {children}
  9     </div>
 10   );
 11 };
 13 export default Container;
 14

```



↓ 続く -----

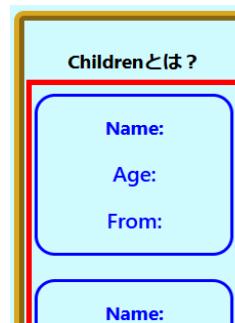
★補足★

<Profile/> 追加すれば2個表示される(赤枠)

```

# Profile.css
# Profile.jsx
  9 const Example = () => {
 10   return (
 11     <div>
 12       <Container title="Childrenとは?">
 13         <Profile/>
 14         <Profile/>
 15       </Container>
 16     </div>
 17   );
 18 };
 20 export default Example;

```



```
# App.css  
App.jsx  
BaseErrorBoundary.jsx  
# index.css  
lecture15
```

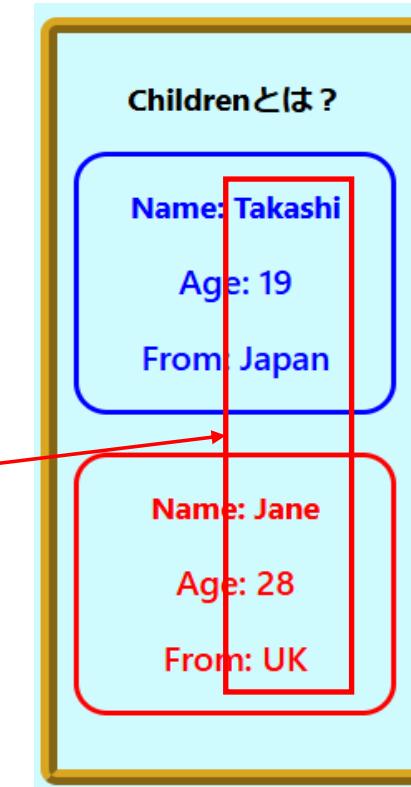
↓ 続く -----

★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  
  4 react_basic  
    src  
      085_expr_and_state  
      > 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...
```

```
1 import Profile from "./components/Profile";  
2 import Container from "./components/Container";  
3  
4 const profile = [  
5   { name: "Takashi", age: 19, country: "Japan" },  
6   { name: "Jane", age: 28, country: "UK", color: "red" },  
7 ];  
8  
9 const Example = () => {  
10   return (  
11     <div>  
12       <Container title="Childrenとは?">  
13         <Profile {...profile[0]} />  
14         <Profile {...profile[1]} />  
15       </Container>  
16     </div>  
17   );  
18 };  
19  
20 export default Example;
```



```
react-guide-material  
  4 react_basic  
    src  
      110_props_children  
        start  
          components  
            Container.css  
            Container.jsx  
            Profile.css  
            Profile.jsx  
          Example.jsx  
        111_jsx_compo...
```

```

> 120_props_rules      12
> 130_whats.jsx       13  export default Container;
> 140_react_element   14

react-guide-material    1  import "./Profile.css";
                        2
                        3  const Profile = ({ name, age, country, color }) => {
                        4      return (
                        5          <div className={`profile ${color}`}>
                        6              <h3>Name: {name}</h3>
                        7              <p>Age: {age} </p>
                        8              <p>From: {country}</p>
                        9          </div>
                       10     );
                       11 };
                       12
                       13  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。

```

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>
  );
};

```

✖ Warning: Each child in a list should have a unique "key" prop.
Check the render method of `Example`. See <https://reactjs.org/link/warning-keys> for more information.

▼`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_expl_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

 BaseErrorBoundary...

 index.css

 lectures.js

 main.jsx

 .eslintrc.cjs

 .gitignore

 index.html

 package-lock.json

 package.json

```
1 import Profile from "./components/Profile";
2 import Container from "./components/Container";
3
4 const profile = [
5   { name: "Takashi", age: 19, country: "Japan" },
6   { name: "Jane", age: 28, country: "UK", color: "red" },
7 ];
8
9 const Example = () => {
10   return (
11     <div>
12       <Container title="Childrenとは?" children={[
13         [
14           <Profile key={profile[0].name} {...profile[0]} />,
15           <Profile key={profile[1].name} {...profile[1]} />
16         ]
17       } />
18
19       <Container
20         title="個別に渡せる"
21         first={<Profile key={profile[0].name} {...profile[0]} />}
22         second={<Profile key={profile[1].name} {...profile[1]} />}
23       />
24     </div>
25   );
26 }
27
28 export default Example;
```



react-guide-material

 04_react_basic

 src

 085_expl_and_state

 087_practice_expr

 090_props

 100_practice_props

 110_props_children

 end

 start

 components

```
1 import "./Container.css";
2
3 const Container = ({ title, children, first, second }) => {
4   console.log(children);
5   return (
6     <div className="container">
7       <h3>{title}</h3>
8       {children}
9       {first}
10      {second}
11    </div>
12  );
13}
```

```

# Container.css
Container.jsx
# Profile.css
Profile.jsx
Example.jsx

```

```

11   </div>
12 );
13 };
14
15 export default Container;
16

```



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

```

4 import Bye from "./components/Bye"
5 import Hello from "./components/Hello"
6
7 const Example = () => {
8
9   // POINT propsの流れは一方通行
10  const name = 'Tom';
11
12  return (
13    <>
14      <Hello name={name}>
15        <Bye name={name} />
16    </>
17  );
18};
19
20 export default Example;
21

```

Hello Tom
Bye Tom

react-guide-material

```

1 const Hello = (props) => {
2   return (
3     <div>
4       <h3>Hello {props.name}</h3>
5     </div>
6   );
7 };
8
9 export default Hello;

```

react-guide-material

```

1 const Bye = (props) => {
2   return (
3     <div>
4       <h3>Bye {props.name}</h3>
5     </div>
6   );
7 };
8
9 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！

▼エラーになる！

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

react-guide-material

```

1 const Hello = (props) => {
2
3   props.name = 'Bob';
4   console.log(props.name);
5

```

```
at Hello (http://localhost:3001/src/Hello.jsx)
at Example (http://localhost:3001/src/Example.jsx)
```

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

※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');
```

```
const Hello = (props) => {
  // POINT propsは読み取り専用
  // props.name = 'Bob';
  // console.log(props.name);

  const desc = Reflect.getOwnPropertyDescriptor(props, 'name');
  console.log(desc)

  return (
    <div>
      <h3>Hello {props.name}</h3>
    </div>
  );
}

export default Hello;
```

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

```
<div id="app"></div> 
```

 マウント（利用可能な状態にする）

```
const element = <h1>Hello!</h1>
const root = ReactDOM.createRoot(
  document.getElementById('app')
);
root.render(element)
```

<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>;
}
```

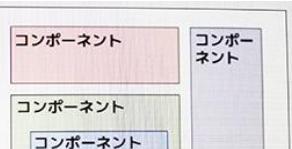
→ <Welcome name="Tom" />

```
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。

関数コンポーネント
Function Component

関数コンポーネントは



propsを受け取り、JSXを返す。



- 再利用性の向上（コードが使いまわせる）
- 可読性の向上（コードが整理される）
- 疎結合になる（バグを減らせる）



コンポーネントの親子関係

コンポーネントは出力するJSXの中に他のコンポーネントを含めることができる。

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



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

セクション5: イベントリスナーと状態管理 (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★

```
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;
```

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
onBlur検知
onFocus検知
⑧ onChange検知

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

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

/* POINT イベントリスナーの登録方法 */
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>
  )
}
```

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

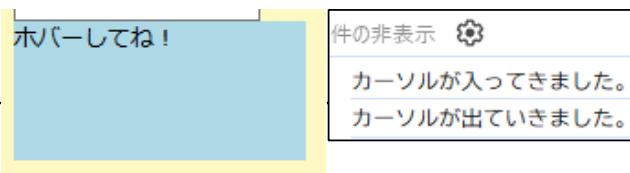
★e.target.valueで入力値を取得
入力値を取得：
abcdef

牛の非表示
a
ab
abc
abcd
abcde
abcdef

react-guide-material
05_state_and_event
` src
` 015_other_events
> end
Example.css

```
# Example.css  
Example.jsx  
> start  
> 020_useState  
> 030_useState_ren...  
> 040_multiple_state  
> 050_prev_state  
> 055_practice_state  
> 060_state_object  
6 <div>  
7   <h3>コンソールを確認してください。</h3>  
8   /* POINT e.target.valueで入力値を取得 */  
9   <div>  
10    <label>  
11      入力値を取得：  
12      <input type="text" onChange={(e) => console.log(e.target.value)} />  
13    </label>  
14  </div>
```

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



```
<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…

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

```
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

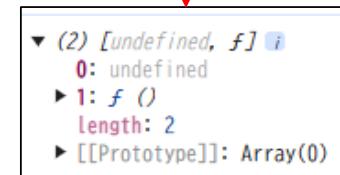
```

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]} →

```

配列の0番目に参照用の値
が渡ってきます。
配列の0番目：参照用の値
配列の1番目：更新用の関数



=
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

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

★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
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

46. 【重要】ステートとは？

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

変数の値を変えても画面の
表示は変わらないのは、
なぜ？？

```
<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; Exampleが実行される度に値  
    が空になってしまいます。  
  return (  
    <>  
      <input  
        type="text"  
        onChange={(e) =>  
          | displayVal = e.target.value  
        } /> = {displayVal}  
    </>  
  );
```

};

値が空で表示される。

```
export default Example;
```

画面が変更するために必要な処理

- ・Reactにコンポーネントの再実行（再レンダリング）を依頼し、新しいReact要素を作成してもらう必要がある。
- ・変更した値をどこかに保持しておく必要がある。（stateに保存）



これらを可能にする仕組みを提供するのがuseStateという関数

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

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

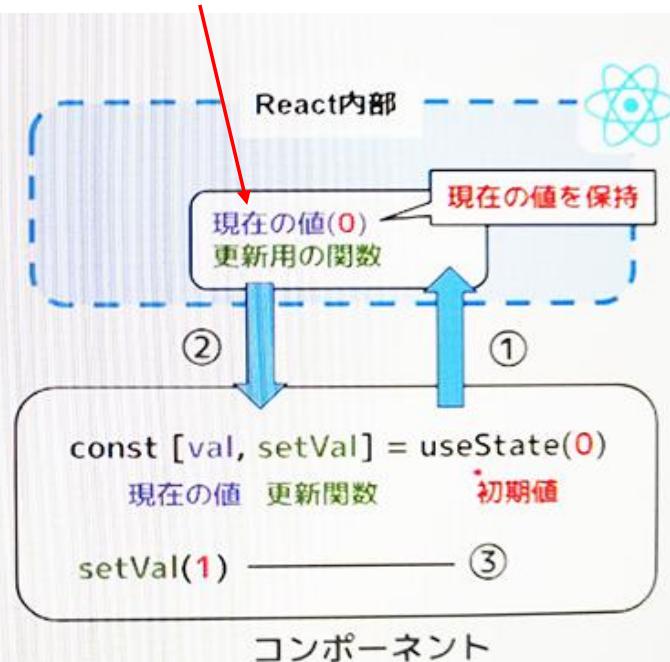
useStateの役割と使い方

①接続 (Hook into)

React内部と接続。状態が管理されるようになる。

②”現在の値”と”更新関数”を返却

③更新関数で新しい値をReactに渡す。また、Reactに自身のコンポーネントを再実行するように依頼する。



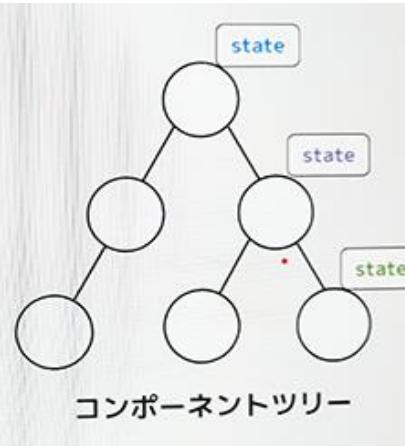
useState は React hooks と呼ばれる

React hooks

state（状態）とは？

コンポーネント毎に保持・管理される値

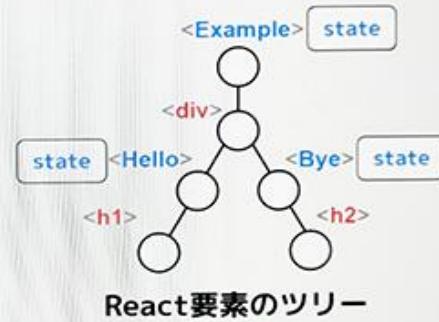
※コンポーネント内に定義した普通の変数はレンダリングのたびに初期化され保持されない



state(状態)とは？

stateはコンポーネント毎に保持

```
const Bye = () => <h2>GoodBye!</h2>
const Hello = () => <h1>Hello</h1>
const Example = () => {
  return (
    <div>
      <Hello/>
      <Bye/>
    </div>
  );
};
```



React要素のツリー

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！

The screenshot shows a code editor with a file named `Example.jsx` and a browser window. The browser displays a text input field containing "12345" and a list below it. The list contains the numbers 1, 2, 123, 1234, and 12345, each on a new line. A red arrow points from the explanatory text above to the browser's list, indicating that the list was rendered again even though no state update occurred.

```
const Example = () => {
  let displayVal;
  console.log('再レンダリングされました');
  return (
    <>
      <input type="text" value={displayVal}>
  
```

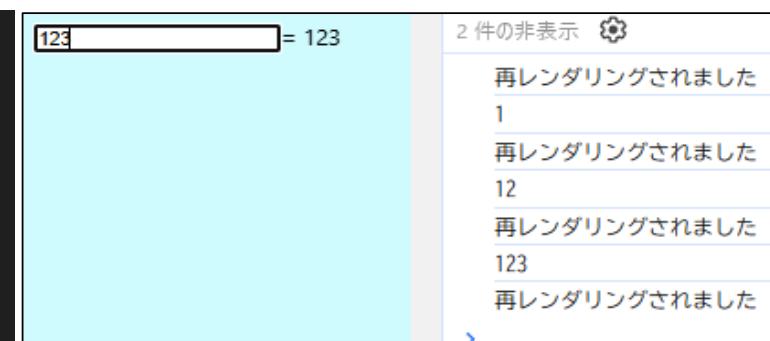
```
> 055_practice_state    9
> 060_state_object     10      type="text"
> 064_state_array      11      onChange={(e) => {
> 068_practice_obj_st... 12          console.log(e.target.value);
> 070_state_and_com... 13          displayVal = e.target.value;
> 080_state_and_props 14      }
> 090_practice_state_... 15      />
# App.css               16      = {displayVal}
# App.jsx                17  );
# BaseErrorBoundary... 18  };
# index.css              19
JS lectures.js           20  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
```



▼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";
```

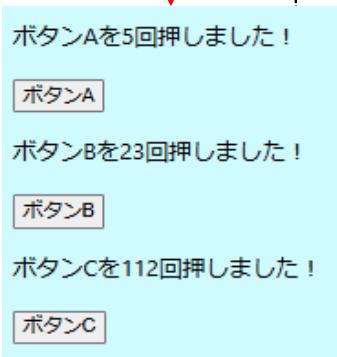
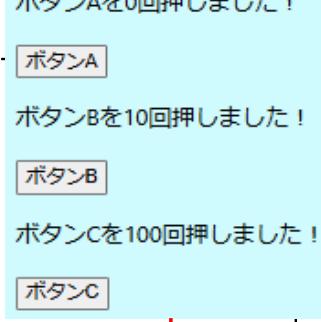
```
const Example = () => {
  const [countA, setCountA] = useState(0);
  const [countB, setCountB] = useState(10);
  const [countC, setCountC] = useState(100);
```

```
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;
```



```
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...
        > 090_state_and_t...
```

```
1 import { useState } from "react";
2
3 const Example = () => {
4   const [countA, setCountA] = useState(0);
5   const [countB, setCountB] = useState(10);
6   const [countC, setCountC] = useState(100);

7
8   return (
9     <>
10       <p>ボタンAを{countA}回押しました！</p>
11       <button onClick={() => {
12         setCountA(countA + 1);
13       }}>ボタンA</button>
14     </>
15   );
16}
```

```
> 090_practice_state... 14
# App.css 15    <p>ボタンBを{countB}回押しました！</p>
# App.jsx 16    <button onClick={() => {
# BaseErrorBoundar... 17      setCountB(countB + 1);
# index.css 18    }>ボタンB</button>
JS lectures.js 19
# main.jsx 20    <p>ボタンCを{countC}回押しました！</p>
# .eslintrc.cjs 21    <button onClick={() => setCountC(countC + 1)}>ボタンC</button>
# .gitignore 22  </>
# index.html 23  };
# package-lock.json 24  };
# package.json 25
# README.md 26  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);
```

カウントアップ & カウントダウン

```
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;
```

```
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...
          op.css
          op.jsx
          useErrorBounda...
          # index.css
          JS lectures.js
          main.jsx
          .eslintrc.cjs
          .gitignore
          index.html
          package-lock.json
          package.json
          README.md
          1   import { useState } from "react";
          2
          3   const Example = () => {
          4     const [count, setCount] = useState(0);
          5     const countUp = () => {
          6       setCount(count + 1);
          7     };
          8     const countDown = () => {
          9       setCount(count - 1);
         10   };
         11   return (
         12     <>
         13       <p>現在のカウント数: {count}</p>
         14       <button
         15         onClick={countUp}
         16       >+</button>
         17       <button
         18         onClick={countDown}
         19       >-</button>
         20     </>
         21   );
         22
         23
         24   export default Example;
         25
```

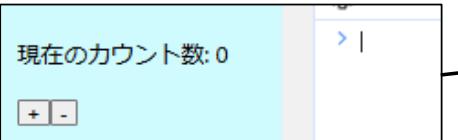
↓ 続く -----

▼ステートの特性

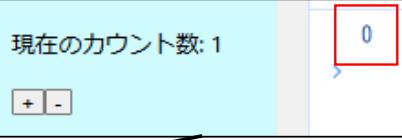
ボタンクリック、コンソールに「1」が表示されると思うのだが、**すぐに反映されない！**

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

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;
});

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

現在のstateの値

次のstateの値

▼本件のまとめ

state更新用関数とレンダリング

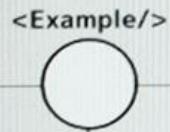
React内部
に保持される値

state
(count) 0

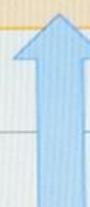
0

2

2

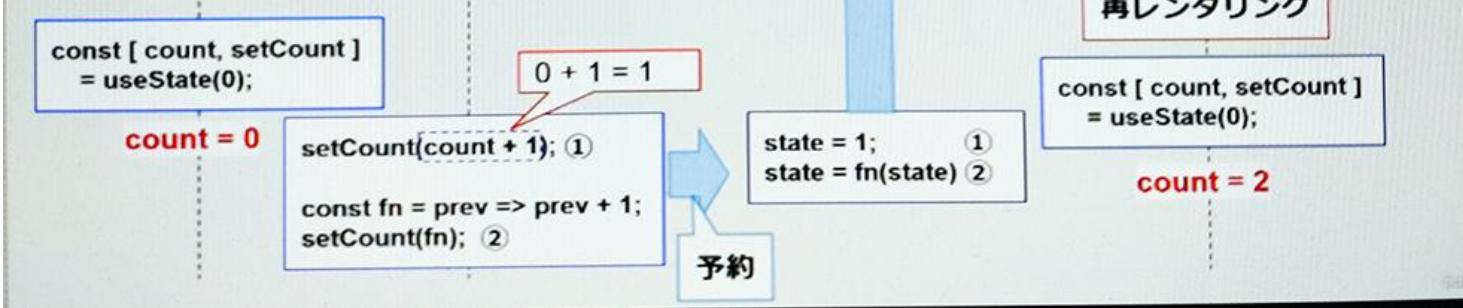


onClick



<Example/>

→



```
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;
```

```
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;
```

Name:Tom

Age:18

Tom
18
リセット

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} />
```

Tom

↓ 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;

```

Name:Tom ABC
Age:26

The screenshot shows a light blue header with the text 'Name:Tom ABC' and 'Age:26'. Below it is a white input field containing 'Tom ABC' and a number input field containing '26'. A red rectangular button labeled 'リセット' (Reset) is positioned below the inputs.

```

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…

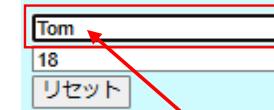
```

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！

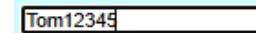
```
setPerson({ ...person })
```

↓

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); false
  console.log({ ...person });
  setPerson({ ...person, name: e.target.value })
}
```



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

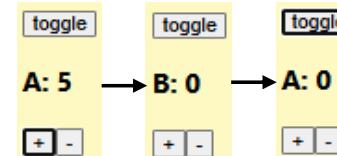
```

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"/>} */}
    </>
  )
}

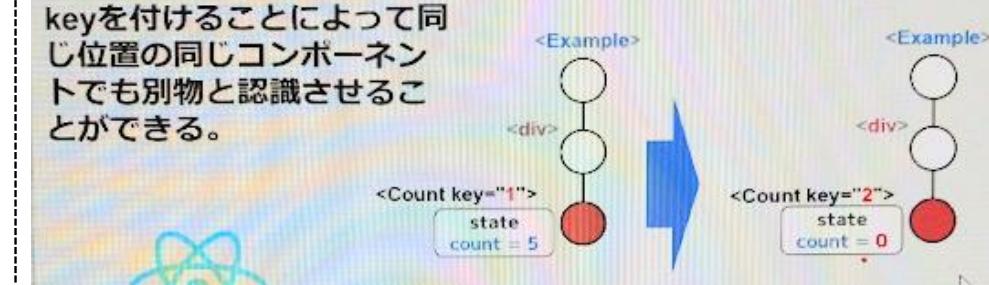
const Count = ({ title }) => {
  const [ count, setCount ] = useState(0);
  const countUp = () => {
    setCount((prevstate) => prevstate + 1);
  };
  const countDown = () => {

```



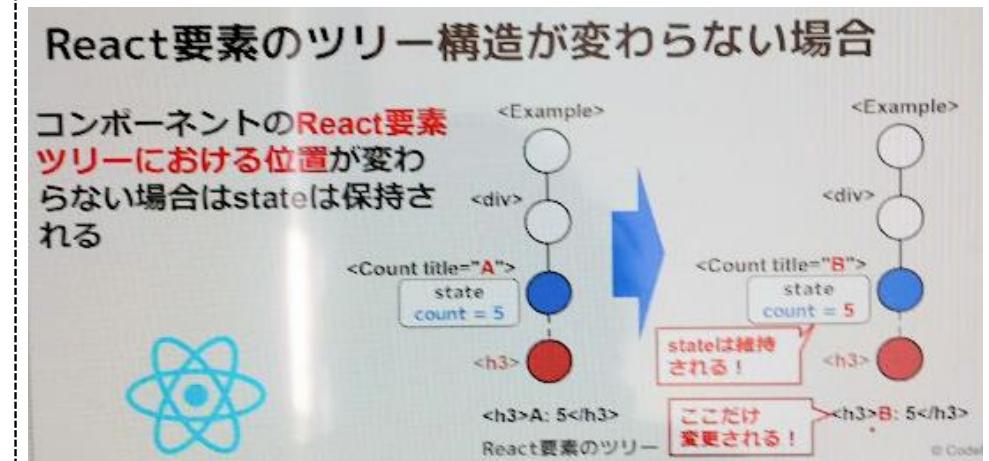
同じ位置に独立してコンポーネントを表示

keyを付けることによって同じ位置の同じコンポーネントでも別物と認識させることができる。



React要素のツリー構造が変わらない場合

コンポーネントのReact要素ツリーにおける位置が変わらない場合はstateは保持される



```

    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、count がリセットされない方法！

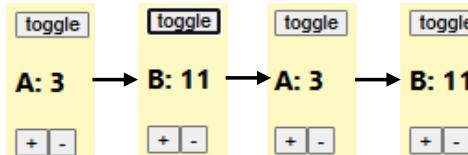
```

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} />}
    </>
  );
};

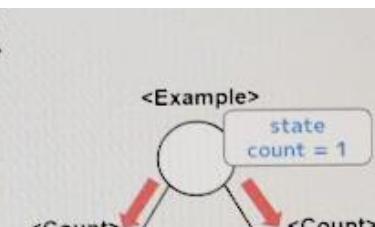
const Count = ({ title, count, setCount }) => {
  const countUp = () => {
    setCount((prevState) => prevState + 1);
  };
  const countDown = () => {
    setCount(count - 1);
  };

```



stateをpropsで渡すケース

- ・コンポーネントが消滅する可能性がある時。



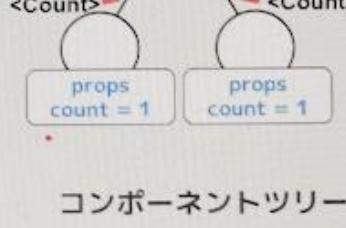
```

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

export default Example;

```

・特定のstateを複数の子コンポーネントで共有したいとき。



57. 【練習】ステートの受け渡し

```

import { useState } from "react";

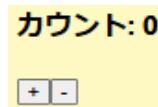
const Example = () => {
  const [ count, setCount ] = useState(0);

  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！



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>
    for (const animal of animals) { [→ tail]
      animalList.push(<li>{animal}</li>);
    }
  </ul>
);
```

```
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>
  </>
);

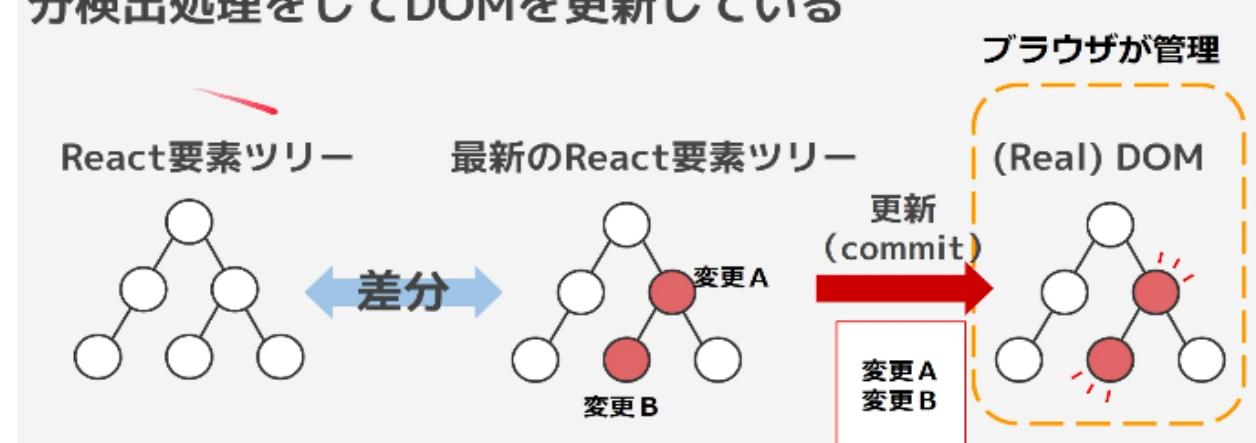
```

<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

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

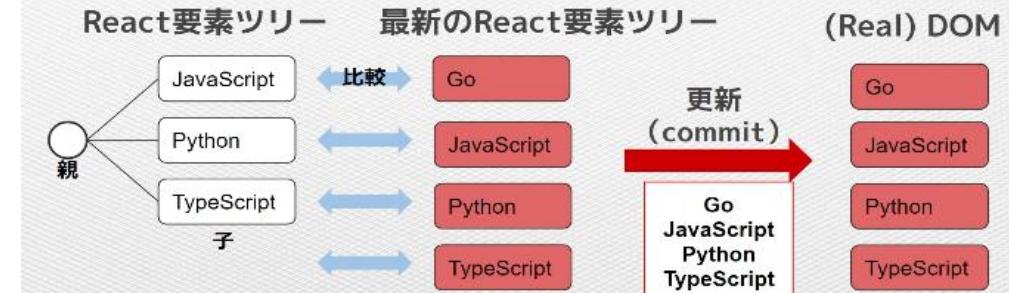
先頭に挿入した場合



子要素にkeyを持たせると、Reactはどの要素が変更、追加、削除されたかを識別できるようになるため、差分のみ更新することが可能になる。

key無

先頭に挿入した場合



Reactは全ての子のReact要素を変更してしまい、全てのReal DOMを洗い替える。

<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

```
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 (
    <div>
      <ul>
        {inputs.map((input, index) => (
          <li key={input.key}>
            {input.value}
          </li>
        ))}
      </ul>
      <button onClick={unshiftInput}>追加</button>
    </div>
  );
}
```

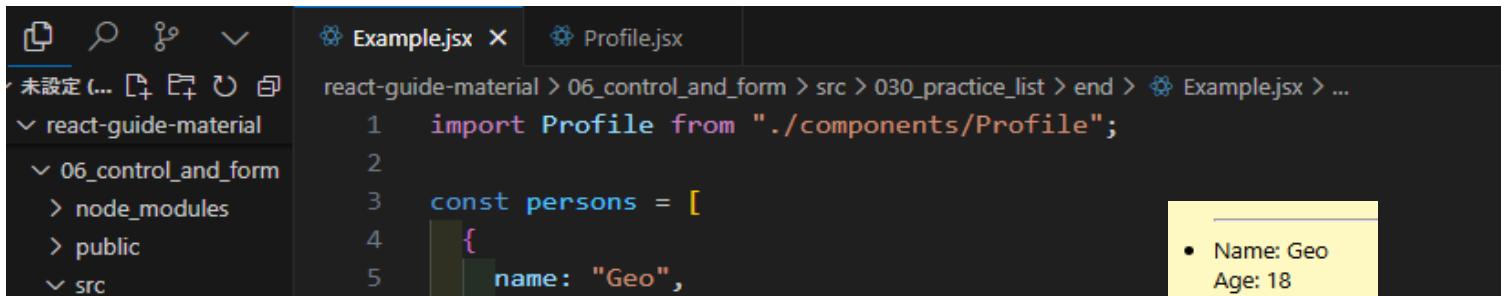
↓1,2,3と入力後に[先頭に追加]ボタンクリック

先頭に追加	
key={ユニークキー}	key={index}
188: <input type="text"/>	188: <input type="text" value="1"/>
329: <input type="text" value="1"/>	329: <input type="text" value="2"/>
530: <input type="text" value="2"/>	530: <input type="text" value="3"/>
558: <input type="text" value="3"/>	558: <input type="text"/>

```
<>
<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



The screenshot shows a code editor with a dark theme. The file is named `Example.jsx`. The code defines a variable `persons` as an array of objects. A tooltip is displayed over the first item in the array, showing its properties: `name: "Geo"` and `Age: 18`.

```
import Profile from "./components/Profile";
const persons = [
  {
    name: "Geo",
    Age: 18
  }
];
```

```
> 010_list_componen... 6    age: 18,
> 020_list_and_key 7    hobbies: ["sports", "music"],
> 024_why_key_uni... 8  },
> 030_practice_list 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！ */ → tab
28             <li key={person.name}>
29               <Profile {...person} />
30             </li>
31           ))
32         </ul>
33       </>
34     );
35   };
36
37 export default Example;
```

```
Hobby:
  o sports
  o music
  • Name: Tom
  Age: 25
  Hobby:
    o movie
    o music
  • Name: Lisa
  Age: 21
  Hobby:
    o sports
    o travel
    o game
```

未設定... [?] [?] [?] [?] [?] react-guide-material > 06_control_and_form > src > 030_practice_list > end > components > Profile.jsx

```
1  const Profile = ({ name, age, hobbies }) => {
2    return (
3      <div>
4        <hr />
5        <div>Name: {name}</div>
6        <div>Age: {age}</div>
```

```

> 010_list_compon...
> 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...

```

```

    <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;

```

```

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) => (

```

```

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

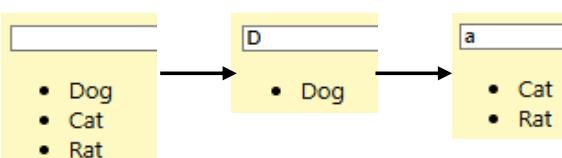
```

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

```
.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;
```

```

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

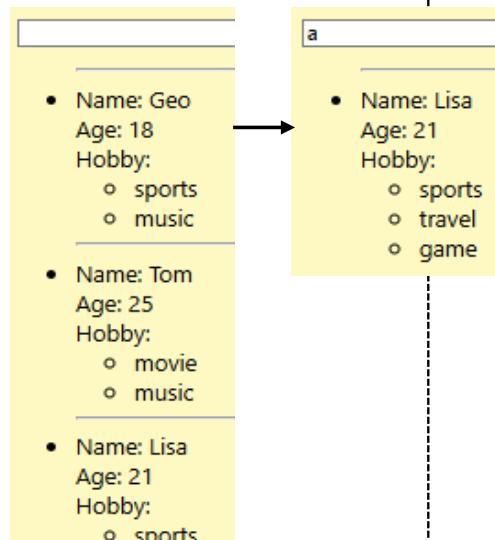
- Profile.jsx
- Example.jsx

```

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、

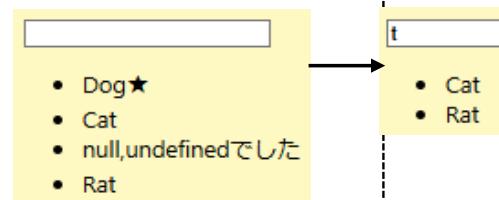
?? (Null合体演算子)
A ?? B (Aがnull or undefinedの時、Bを使う)
*/
const Example = () => {
  const animals = ["Dog", "Cat", null, "Rat"];

  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}>
                {

```



```
// < 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シスしやすい★ >
// 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";

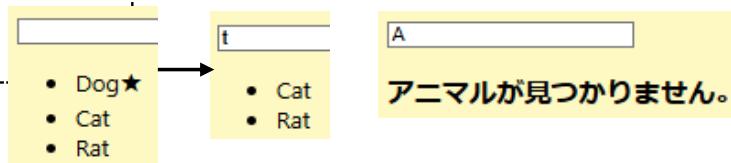
const Example = () => {
  const animals = ["Dog", "Cat", "Rat"];

  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

```

src
└── 070_refactor_comp...
  └── end
    └── components
      └── AnimalFilter.jsx
          1
          2
          3 import { useState } from "react";
          4
          5 import AnimalList from "./components/AnimalList"
          6 import AnimalFilter from "./components/AnimalFilter"
          7

```

```
AnimalItem.jsx
AnimalList.jsx
Example.jsx
start
Example.jsx
080_input_textarea
090_radio
100_single_checkbox
110_multi_checkbox
120_select
130_reminder
App.css
App.jsx
BaseErrorBoundary....
index.css
lectures.jsx
main.jsx
.eslintrc.cjs
.gitignore
8 const Example = () => {
9   const animals = ["Dog", "Cat", "Rat"];
10
11   const [filterVal, setFilterVal] = useState("");
12
13   const filteredAnimals = animals.filter((animal) => {
14     const isMatch = animal.indexOf(filterVal) !== -1;
15     return isMatch;
16   });
17
18   return (
19     <>
20       <AnimalFilter filterState={[filterVal, setFilterVal]} />
21       <AnimalList animals={filteredAnimals} />
22     </>
23   );
24 }
25
26 export default Example;
27
```

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

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

```
export default Example;
```

```
react-guide-material > 06_control_and_form > src > 070_refactor_components > end > components > AnimalFilter.jsx  
1 const AnimalFilter = ({ filterState }) => {  
2   const [filterVal, setFilterVal] = filterState;  
3  
4   return (  
5     <input  
6       type="text"  
7       value={filterVal}  
8       onChange={(e) => setFilterVal(e.target.value)}  
9     />  
10  );  
11};  
12 export default AnimalFilter;
```

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

```
react-guide-material > 06_control_and_form > src > 070_refactor_components > end > components > AnimalItem.jsx  
1 const AnimalItem = ({ animal }) => {  
2   return (  
3     <li>  
4       {animal}  
5       {animal === "Dog" && "★"}  
6     </li>  
7   );  
8 };  
10 export default AnimalItem;
```

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

```
react-guide-material > 06_control_and_form > src > 070_refactor_components > end > components > AnimalList.jsx  
1 import AnimalItem from "./AnimalItem";  
2 const AnimalList = ({ animals }) => {  
3   if (animals.length === 0) {  
4     return <h3>アニマルが見つかりません。</h3>;  
5   }  
6  
7   return (  
8     <ul>  
9       {animals.map((animal) => {  
10         return <AnimalItem animal={animal} key={animal} />;  
11       })}
```

```
> 080_input_textarea 12   </ul>
> 090_radio        13 );
> 100_single_checkbox 14 };
> 110_multi_checkbox 15
> 120_select       16   export default AnimalList;
> 17
```

```
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>
  );
};

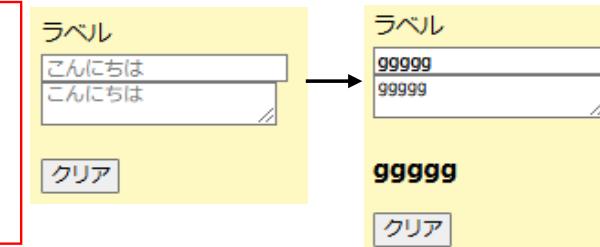
export default AnimalList;
```

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} />`



```
use_material / 00_controller_and_form / src / 000_input_textarea / end / 00_Example
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;

```

私はAppleがたべたい

● Apple ○ Banana ○ Cherry

```

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

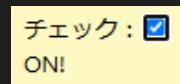
```

```
// const チェックボックスの実装
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;
| Ctrl+L to chat Ctrl+K to generate
```



```
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

```
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

```
1 import { useState } from "react";
2
3 // 複数チェックボックスの実装
4 const Example = () => {
5   const [fruits, setFruits] = useState([
6     { label: "Apple", value: 100, checked: false },
7     { label: "Banana", value: 200, checked: false },
8     { label: "Cherry", value: 300, checked: false },
9   ]);
10
```

```
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}

```

Apple:100
Banana:200
Cherry:300
合計 : 500

```
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}

```

```

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

export default Example;

```

```

51   id={fruit.label}
52   type="checkbox"
53   value={fruit.label}
54   checked={fruit.checked}
55   onChange={handleChange}
56   />
57   <label htmlFor={fruit.label}>
58     {fruit.label}:{fruit.value}
59   </label>
60   </div>
61 );
62 );
63 <div>合計 : {sum}</div>
64 </div>
65 );
66 };
67
68 export default Example;
69

```

71. 【Form】プルダウンの作成方法

```

import { useState } from "react";

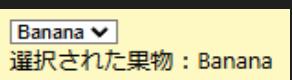
// POINT プルダウンの実装
const Example = () => {
  const [selected, setSelected] = useState("Banana");

  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> */
        }
      </select>
    </>
  );
};

export default Example;

```



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

export default Example;
```

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

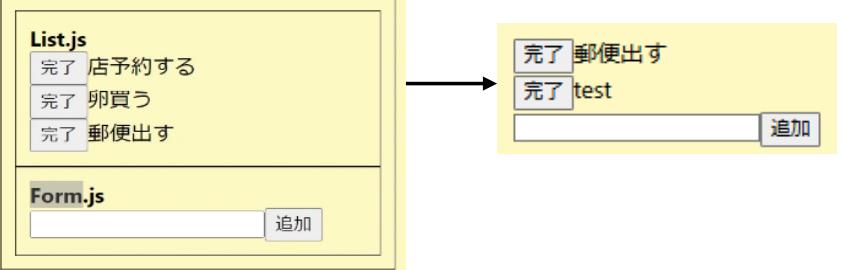
// POINT プルダウンの実装
const Example = () => {
  const [selected, setSelected] = useState("Banana");

  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;
```

Reminder



```
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
```

```
1 // POINT Reminder(Todoアプリ)の作り方
2 import Todo from "./components/Todo"
3
4 const Example = () => {
5     return (
6         <>
7             <h2>Reminder</h2>
8             <Todo />
9         </>
10    );
11 }
12
13 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;
```

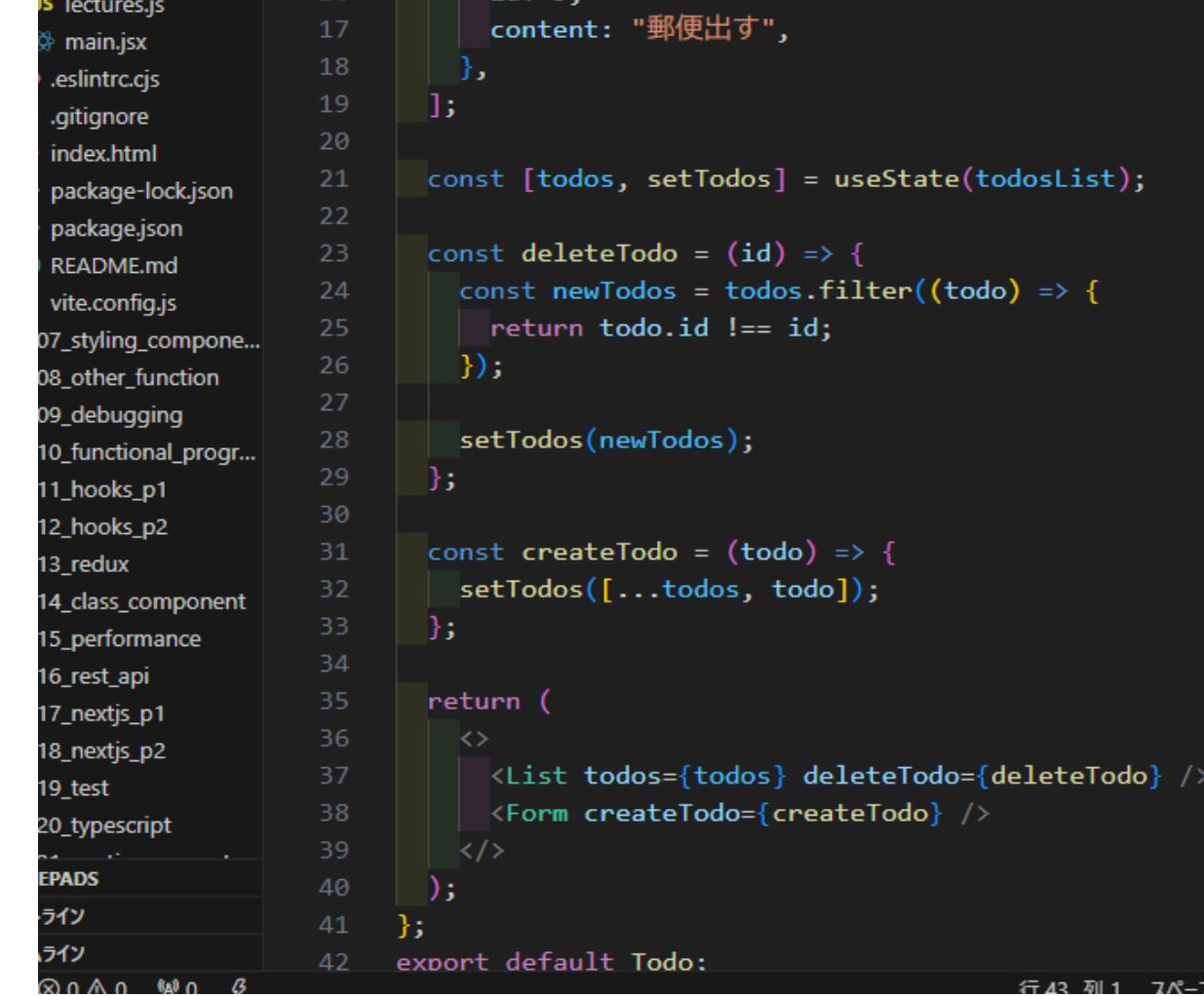
```
react-guide-material
06_control_and_form
└── src
    └── 130_reminder
        └── end
            └── components
                └── Form.jsx
                └── List.jsx
                └── Todo.jsx
                └── Example.jsx
            > start
        # App.css
        # App.jsx
        #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

```
1 import { useState } from "react";
2 import List from "./List";
3 import Form from "./Form";
4
5 const Todo = () => {
6     const todosList = [
7         {
8             content: "店予約する",
9             id: 1,
10            },
11            {
12                id: 2,
13                content: "卵買う",
14            },
15            {
16                id: 3,
```

```
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_compon...
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
行 43 列 1 78-79
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;
```

```
],
];

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;
```

```
react-guide-material > 06_control_and_form > src > 130_reminder > end > components > List.jsx > ...
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
行 43 列 1 78-79
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;
```



```
App.jsx
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;
```

react-guide-material > oo_control_and_form > src > ts0_reminder > end > components

```
L+ L+ ⌂ EP
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("");
```

```

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;
32

```

```

};

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tyling

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
- ・動的に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

```
node-material > 0/_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}>
        ポタン
      </button>
      <div style={{ textAlign: "center" }}>{isSelected && "クリックされました。"}</div>
    </>
  );
};

export default Example;

```

75. インライнстyles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を使ったスタイル ※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…

```
▽ components
  # SubButton.css
  ⚡ SubButton.jsx
  # Example.css
  ⚡ Example.jsx
```

【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！

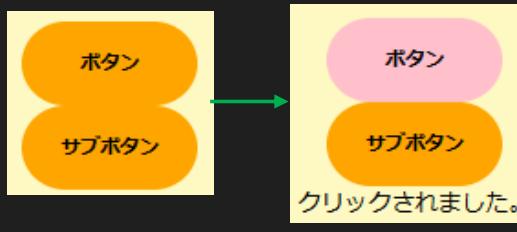
isSelectedがfalseの場合、className='btn false'になってしまいます。

例：`className={`btn ${isSelected && 'selected'}`}`

```
react-guide-material > 07_styling_component > src > 020_import_css > end > ⚡ Example.jsx > [Ex] Exam
dot
threeJs
react-guide-material
.vscode
02_settings
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
  end
    components
      # SubButton.css
      ⚡ SubButton.jsx
```

```
1 import { useState } from "react";
2
3 import SubButton from "./components/SubButton";
4 import "./Example.css";
5
6 const Example = () => {
7   const [isSelected, setIsSelected] = useState(false);
8
9   const clickHandler = () => setIsSelected((prev) => !prev);
10
11   return (
12     <>
13       <button
14         className={`${btn ${isSelected ? "selected" : ""}}`}
15         onClick={clickHandler}
16       >
17         ボタン
18       </button>
19       <SubButton />
20     </>
21   );
22 }
23
24 export default Example;
```

```
# Example.css  
例題  
# Example.jsx  
例題  
start  
# Example.css  
例題  
# Example.jsx  
例題  
> 030_css_module  
> 040_css_in_js  
> 045_practice_css_i...  
> 050_chakra_ui  
20   <div style={{ textAlign: "center" }}>  
21     {isSelected && "クリックされました。"}  
22   </div>  
23 </>  
24 );  
25 };  
26  
27 export default Example;  
28
```



```
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;
```

```
act-guide-material  
7_styling_component  
src  
1 .btn {  
2   margin: auto;  
3   border-radius: 9999px;  
4   border: none;
```

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

```
020_import_css
  4   border: none;
  5   display: block;
  6   width: 120px;
  7   height: 60px;
  8   font-weight: bold;
  9   cursor: pointer;
10 }
11
12 .selected {
13   background-color: #pink;
14 }
```

設定の... E F O 白
react-guide-material
07_styling_component
src
 020_import_css
 end
 components
 # SubButton.css
 SubButton.jsx
 # Example.css
 Example.jsx

```
1 import "./SubButton.css";
2
3 const SubButton = () => {
4   return <button className="btn">サブボタン</button>
5 }
6
7 export default SubButton;
```

E F L+ L+ O 白
react-guide-material
07_styling_component
src
 020_import_css
 end
 components
 # SubButton.css
 SubButton.jsx
 # Example.css
 Example.jsx
> start
> 030_css_module
> 040_css_in_js
> 045_practice_css_i...
11
12 .selected {
13 background-color: #pink;
14 }

```
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を使ったスタイル

<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>

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

```
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 > ...

```

1 import { useState } from "react";
2
3 import SubButton from "./components/SubButton";
4 import styles from "./Example.module.css";
5
6 // console.log(styles); ←
7
8 const Example = () => {
9   const [isSelected, setIsSelected] = useState(false);
10
11   const clickHandler = () => setIsSelected((prev) => !prev);
12
13   return (
14     <>
15       <button className={`${styles.btn} ${isSelected ? styles.selected : ""}`} onClick={clickHandler}>
16         ボタン
17       </button>
18       <SubButton />
19       <div style={{ textAlign: "center" }}>
20         {isSelected && "クリックされました。"}
21       </div>
22     </>
23   );
24 }
25
26 export default Example;
27

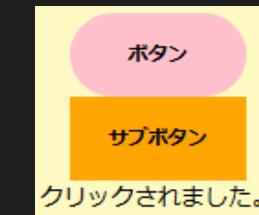
```

Example.jsx:16

```

▼ {btn: '_btn_16hsy_1', selected: '_selected_16hsy_12'} ⓘ
  btn: "_btn_16hsy_1"
  selected: "selected_16hsy_12"
  ► [[Prototype]]: Object

```



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

```

1 .btn {
2   margin: auto;
3   border-radius: 9999px;
4   border: none;
5   display: block;
6   width: 120px;
7   height: 60px;
8   font-weight: bold;
9   cursor: pointer;
10 }
11
12 .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    13 | background-color: pink;
> 050_chakra_ui             14 | }
```

```
react-guide-material > 07_styling_component > src > 030_css_module > end > components > SubButton.js
1 import styles from "./SubButton.module.css";
2
3 const SubButton = () => {
4     return <button className={styles.btn}>サブボタン</button>
5 }
6
7 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_css_module > end > components > SubButton.js
1 .btn {
2     margin: auto;
3     border-radius: 9999px;
4     border: none;
5     display: block;
6     width: 120px;
7     height: 60px;
8     font-weight: bold;
9     cursor: pointer;
10    background: orange;
11 }
12
13 .btn:hover {
14     background-color: red;
15 }
16
17 .btn:active {
18     background-color: purple;
19 }
```

```
.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;
}
```

```
19
20
21 .selected {
22     background-color: pink;
23 }
24
25 @media (max-width: 600px) {
26     .btn {
27         border-radius: 0;
28     }
29 }
```

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

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

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

▼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](#)

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

<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

■デメリット

- ・自動生成されるユニークなクラス名が読みない
- ・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。

▼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

★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>
```

```
import styled from "styled-components"; 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。
↓
<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→ <button class="sc-FEMpB gVreue">ボタン</button>
ボタン
</StyledButton>
```

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

```
1 import { useState } from "react";
2 import styled from "styled-components"; console.dir(styled); →
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  /* 2024/04 ホバー時のスタイルが正しく適用していないの修正*/
29
```

```
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)`  

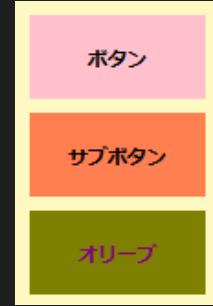
50   background-color: ${({ $isSelectedSub }) =>  

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>

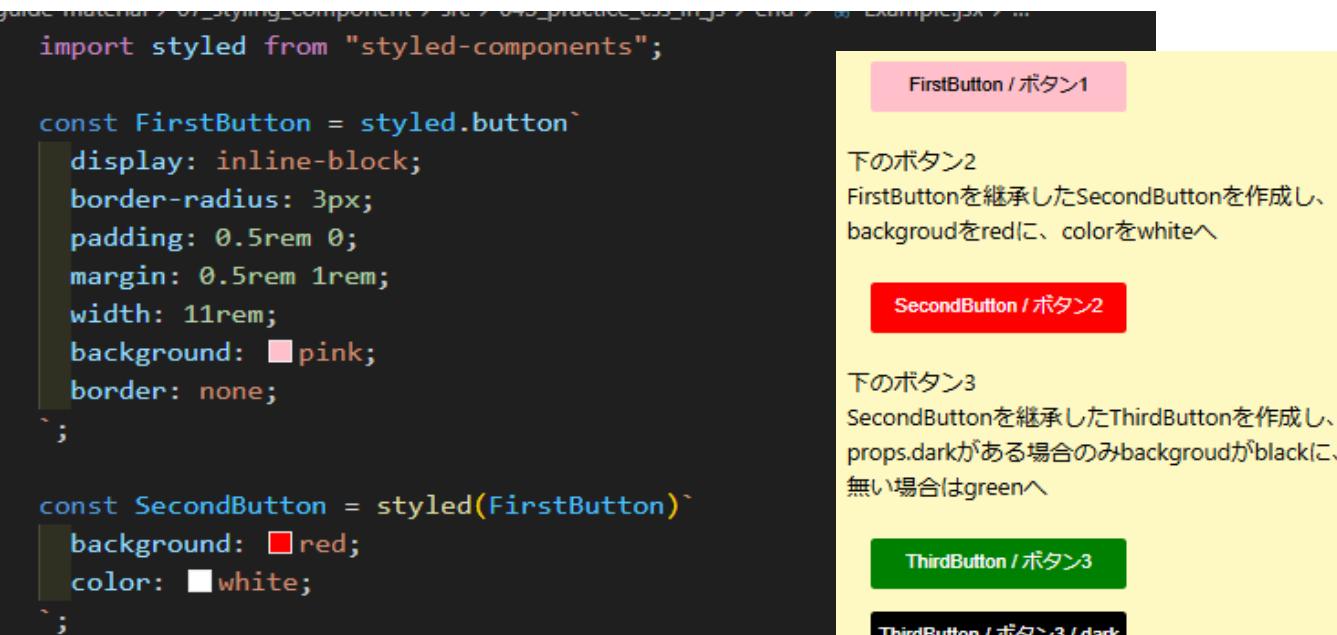
  <StyledSubButton
    $isSelectedSub={isSelectedSub}
    onClick={onClickSubHandler}>
    >
      サブボタン
    </StyledSubButton>

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

export default Example;

```

80. 【練習&解答】styled-components



The screenshot shows a code editor with a file named `Example.js`. The code uses styled-components to create three buttons: `FirstButton`, `SecondButton`, and `ThirdButton`.

```

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;
`;

```

FirstButton / ボタン1

下のボタン2
FirstButtonを継承したSecondButtonを作成し、backgroundをredに、colorをwhiteへ

SecondButton / ボタン2

下のボタン3
SecondButtonを継承したThirdButtonを作成し、props.darkがある場合のみbackgroundがblackに、無い場合はgreenへ

ThirdButton / ボタン3

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、  
background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background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

★INLINE STYLEはあまり使わない…★

INLINE STYLE

```
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
- ・動的に頻繁に計算されるスタイルの適用

```
import styles from
```

CSS Modules

```
"./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り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

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・・・)

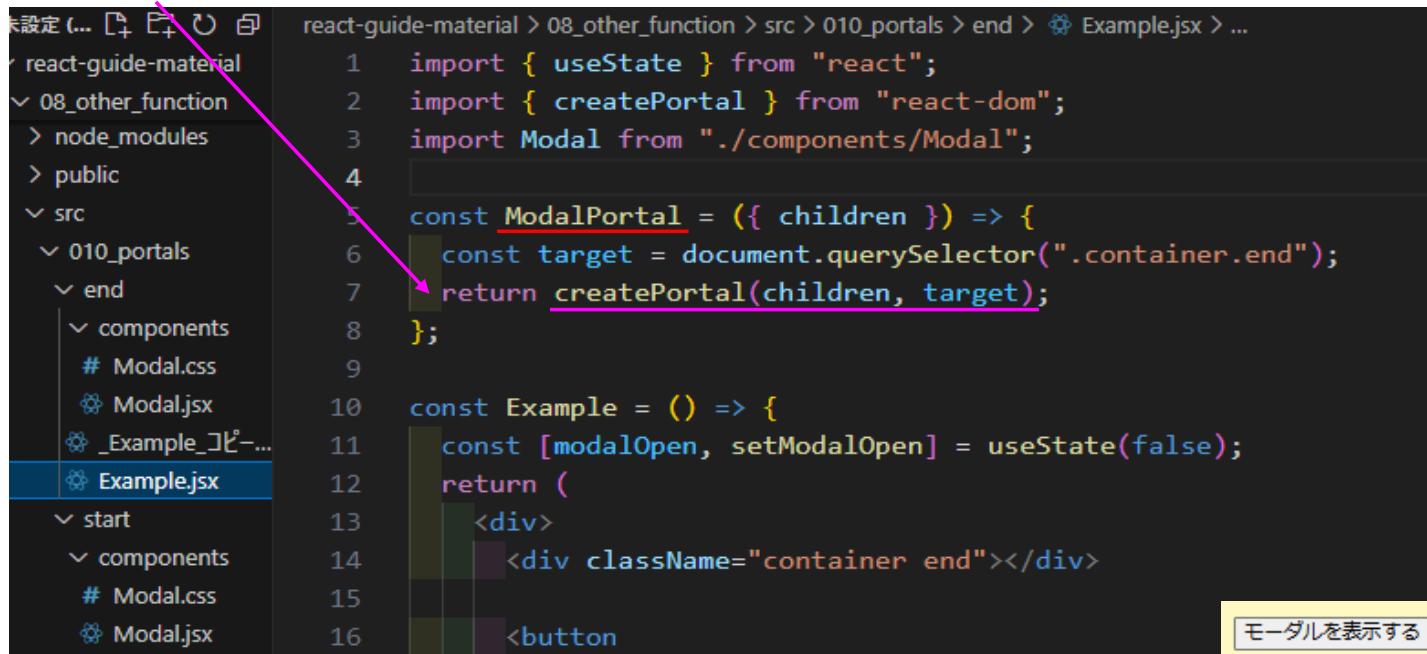
それらの制限なく、子要素が親要素を「飛び出して」表示する必要があるときにcreatePortalを使うのが有効です。

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ラー可能なもの (要素、文字列、フラグメント、コンポーネントなど)

第二引数: レンダラー先のDOM要素



```
react-guide-material > 08_other_function > src > 010_portals > end > Example.jsx > ...
1 import { useState } from "react";
2 import { createPortal } from "react-dom";
3 import Modal from "./components/Modal";
4
5 const ModalPortal = ({ children }) => {
6   const target = document.querySelector(".container.end");
7   return createPortal(children, target);
8 }
9
10 const Example = () => {
11   const [modalOpen, setModalOpen] = useState(false);
12   return (
13     <div>
14       <div className="container end"></div>
15       <button>
16         モーダルを表示する
17       </button>
18     </div>
19   );
20 }
```

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。



The screenshot shows a code editor on the left and a browser preview on the right. The code editor displays a file named 'Example.jsx' with the following content:

```
例 Example.jsx 17 type="button"
> 020_practice_port... 18 onClick={() => setModalOpen(true)}
> 030_useRef 19 disabled={modalOpen}
> 040_forwardRef 20 >
> 050_useImperativ... 21 <div>
> 060_practice_ref 22   モーダルを表示する
# App.css 23 </div>
@ App.jsx 24   {modalOpen && (
@ BaseErrorBounda... 25     <ModalPortal>
# index.css 26       <Modal handleCloseClick={() => setModalOpen(false)} />
JS lectures.js 27     </ModalPortal>
@ main.jsx 28   )
.eslintrc.cjs 29 </div>
.gitignore 30 );
index.html 31 };
package-lock.json 32
package.json 33 export default Example;
```

The browser preview on the right shows a dark-themed UI with a light gray modal overlay. A purple arrow points from the code editor's button to the modal's content area. The modal contains a single button labeled "モーダルを表示する".

```
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;
```

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

export default Example;
```

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

```
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 > components > Modal.css

```
.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;
}
```

```
.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
13 |   z-index: 100;
14 }
15
16 .App-end .container .modal__content {
17   background-color: #fff;
18   padding: 20px 50px;
19 }
20

```

```

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

```

87. 【Bubbling】Portalを使う際の注意点！

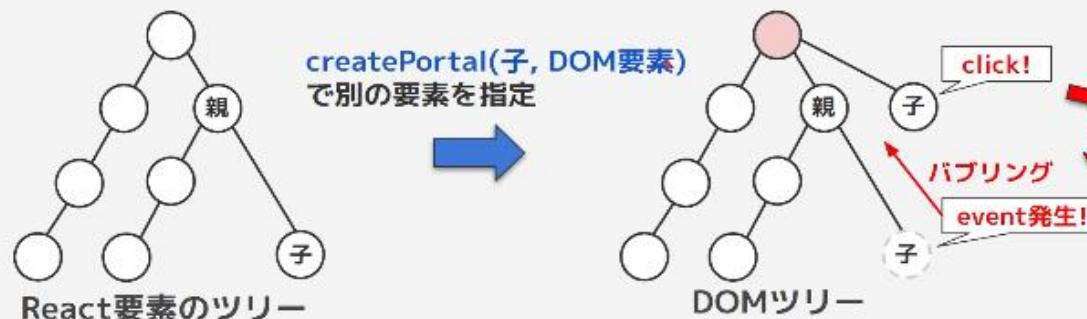
<注意点>

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

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。



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…

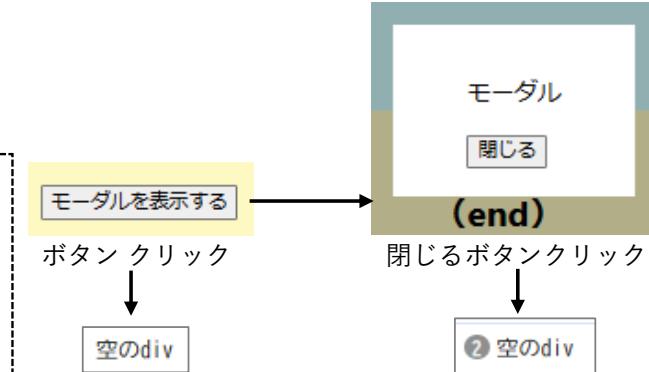
`console.log('空のdiv')` が発火する！

```

<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。



設定 (...) L+ E+ ⌂ ⌂ react-guide-material > 08_other_function > src > 010_portals > end > Example.jsx > [Example]

```

react-guide-material
  2 import { createPortal } from "react-dom";
  3 import Modal from "./components/Modal";
  4
  5 const ModalPortal = ({ children }) => {

```

```
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
  # App.jsx
  # BaseErrorBounda...
  # index.css
  lectures.js
  main.jsx
  .eslintrc.cjs
  .gitignore
  index.html
  package-lock.json
  package.json
  README.md
  vite.config.js
  09_debugging

  6   const target = document.querySelector(".container.end");
  7   return createPortal(children, target);
  8 }

  9
 10  const Example = () => {
 11    const [modalOpen, setModalOpen] = useState(false);
 12    return (
 13      <div onClick={() => console.log('空のdiv')}>
 14        <div className="container end" onClick={() => console.log('container発火しない...')}></div>
 15
 16        <button
 17          type="button"
 18          onClick={() => setModalOpen(true)}
 19          disabled={modalOpen}
 20        >
 21          モーダルを表示する
 22        </button>
 23
 24        {modalOpen && (
 25          <ModalPortal>
 26            <Modal handleCloseClick={() => setModalOpen(false)} />
 27          </ModalPortal>
 28        )}
 29      </div>
 30    );
 31  };
 32
 33  export default Example;
```

```
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')}>
```

```
<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
  1  import "./Modal.css";
  2
  3  const Modal = ({ handleCloseClick }) => {
  4    return (
  5      <div className="modal">
  6        <div className="modal__content">
  7          <p>モーダル</p>
  8          <button type="button" onClick={handleCloseClick}>
  9            閉じる
 10          </button>
 11        </div>
 12      </div>
 13    );
 14  };
 15
 16  export default Modal;
```

```
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;
```

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

```
> public
  4   left: 0;
  5   bottom: 0;
  6   right: 0;
  7   width: 100%;
  8   height: 100%;
  9   background-color: □rgba(0, 0, 0, 0.3);
10  display: flex;
11  justify-content: center;
12  align-items: center;
13  z-index: 100;
14 }

16 .App-end .container .modal__content {
17   background-color: ■#fff;
18   padding: 20px 50px;
19 }

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 > src > 020_practice_portals > end > Example.jsx > ...
react-guide-material
  1 import { useState } from "react";
  2 import { createPortal } from "react-dom";
  3 import Toast from "./components/Toast";
  4
  5 const ToastPortal = ({ children }) => {
  6   const target = document.querySelector(".container.end");
  7   return createPortal(children, target);
  8 };
  9
 10 const Example = () => {
 11   const [toastOpen, setToastOpen] = useState(false);
 12   return (
 13     <div>
 14       <div className="container end"></div>
 15       <button
 16         type="button"
 17         onClick={() => setToastOpen(true)}
 18         disabled={toastOpen}
 19       >
```

```
	BaseErrorBounda... 20
# index.css 21
JS lectures.js 22
main.jsx 23
.eslintrc.cjs 24
.gitignore 25
index.html 26
package-lock.json 27
package.json 28
README.md 29
vite.config.js 30
09_debugging 31
10_functional_progr... 32
11_hooks_p1 33
12_hooks_p2 34
13_redux 35
14_class_component 36
export default Example;
```

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

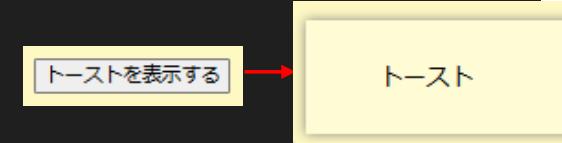
const ToastPortal = ({ children }) => {
  const target = document.querySelector(".container.end");
  return createPortal(children, target);
};

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;
```



```
<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

```
1 import "./Toast.css";
2
3 const Toast = ({ visible, handleCloseClick }) => {
4   const toastClassName = visible ? "toast is-visible" : "toast";
5   return (
6     <div className={toastClassName}>
7       <div className="toast__content">
8         <p>トースト</p>
9         <button
10           type="button"
11           className="toast__button"
12           onClick={handleCloseClick}
13         >
14           <span>X</span>
15         </button>
16       </div>
17     </div>
18   );
19 };
20
21 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;

```

未設定 (…)

- ✓ react-guide-material
- ✗ 08_other_function
 - > node_modules
 - > public
 - ✗ src
 - > 010_portals
 - ✗ 020_practice_portals
 - ✗ 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
- JS lectures.js
- main.jsx
- eslintrc.js
- .gitignore

```

.react-guide-material > 08_other_function > src > 020_practice_portals > end > components > # Toast.css
  1 .App-end .container .toast {
  2   position: absolute;
  3   bottom: 0;
  4   left: 0;
  5   width: 100%;
  6   padding: 10px;
  7   transform: translateY(100%);
  8   z-index: 100;
  9   display: none;
 10 }
 11 .App-end .container .toast.is-visible {
 12   transform: translateY(0);
 13   transition: transform 0.3s ease-in;
 14   display: block;
 15 }
 16
 17 .App-end .container .toast__content {
 18   position: relative;
 19   background-color: rgba(255, 255, 255, 0.3);
 20   box-shadow: 0 0 8px gray;
 21   padding: 12px 50px;
 22 }
 23
 24 .App-end .container .toast__button {
 25   background-color: transparent;

```

```

.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;
}

```

```
↳ index.html      26 border: none;
{} package-lock.json 27 cursor: pointer;
{} package.json     28 outline: none;
① README.md       29 padding: 0;
⚡ vite.config.js   30 appearance: none;
> 09_debugging    31 position: absolute;
> 10_functional_progr... 32 top: 10px;
> 11_hooks_p1      33 right: 10px;
> 12_hooks_p2      34 width: 30px;
> 13_redux          35 height: 30px;
> 14_class_component 36 font-size: 30px;
> 15_performance    37 }
> 16_rest_api        38
> 17 nextjs n1        39 .App-end .container .toast__button:active {
> NOTEPADS           40 | opacity: 0.3;
> アウトライン         41 }
> タイムライン         42 }
```

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;

```

```

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;
}

```

```

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

```

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

```

```

// POINT TypeScriptにおけるオブジェクトの型定義について学ぼう
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 }
]

```

```

// POINT TypeScriptにおけるオブジェクトの型定義について学ぼう
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。

```
2 // 基本的には、明らかに型定義
3
4 const let str: string
5 let str = 'hello';
6 let obj = { name: 'Taro'}
```

▼ 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 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
// 分かるような場合の型定義は型推論に任せようとする

const let obj: {
  name: string;
  age: number;
}

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ようとする
```

```
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 = {
```

```
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)

```
guide-material > 20_typescript > src > 080_function_type > start > Example
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);      ▶ (3) ['hello', 'hello', 'hello']
  console.log(strArry);                      ▶ (3) [10, 10, 10]

  const repeatNum = (value: number, times: number): number[] => {
    return new Array(times).fill(value);
  }
  const numArry = repeatNum(10, 3);
  console.log(numArry);
```

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

```
// 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);
```

▶ (3) [10, 10, 10]
▶ (3) ['hello', 'hello', 'hello']
▶ (3) [true, true, true]
▶ (3) ['bye', 'bye', 'bye']

```
export default Example;
```

```
const Example = () => {
  const repeatStr = (value: string, times: number): string[] => {
```

```
}

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