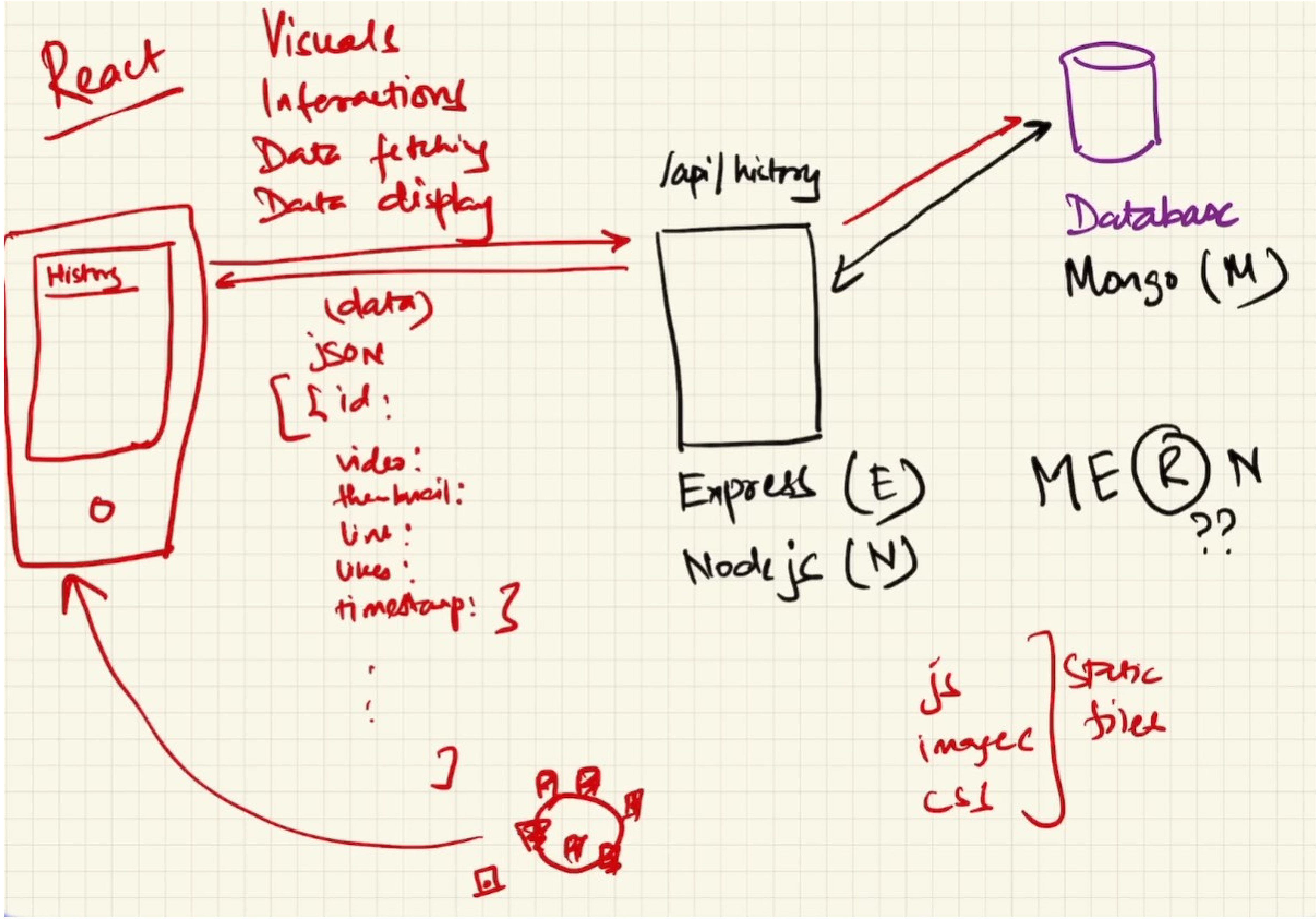
MERN STACK


React
Visuals
Interactions
Data fetching
Data display
History
(data)
JSON
[{ id :
video :
thumbnail :
link :
likes :
timestamp : }
:
:
]
/api/history
Express (E)
Node js (N)
Database
Mongo (M)
ME Ⓡ N
??
js
images
css
Static files

React Tutorial

- React is a JavaScript library for building user interfaces.

- React is used to build single-page applications.

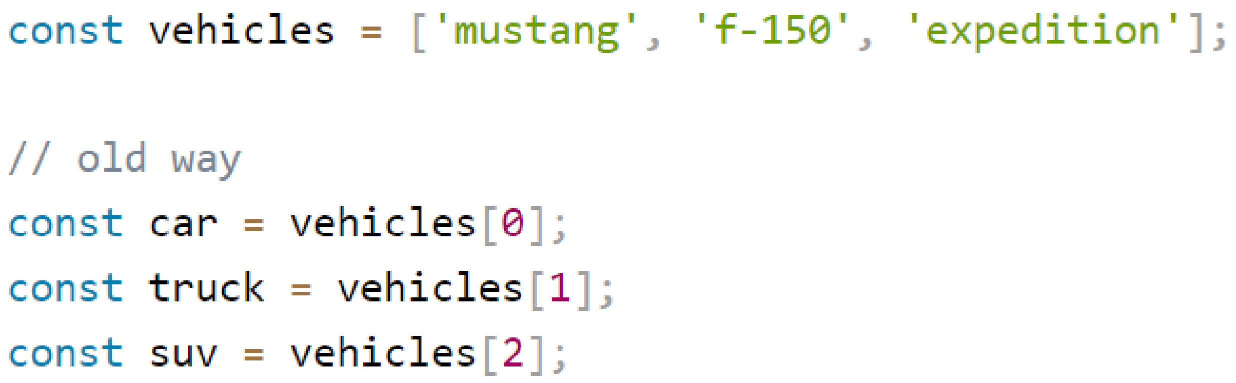
- React allows us to create reusable UI components.
- React, sometimes referred to as a frontend JavaScript framework, is a JavaScript library created by Facebook.
- React only changes what needs to be changed!

React Installation

- First check node version through cmd. Node -v
- Install node.js if not installed
- Npm create vite@latest
- Give project name
- Then go to project folder and install third party deficiencies npm i or(install)
- Npm run dev.

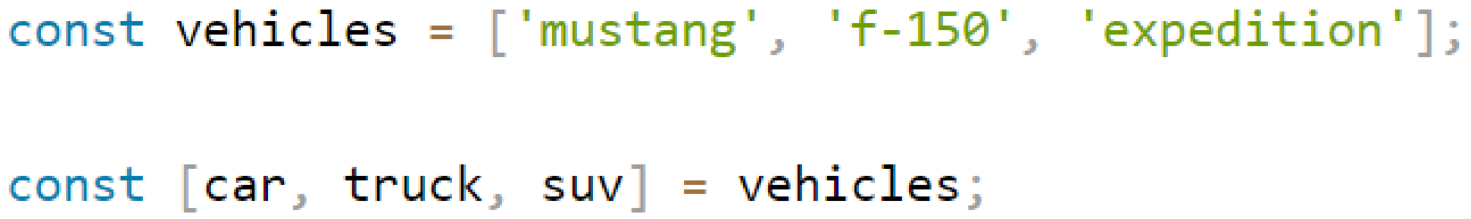
Destructuring:
Destructuring makes it easy to extract only what is needed.

Old way



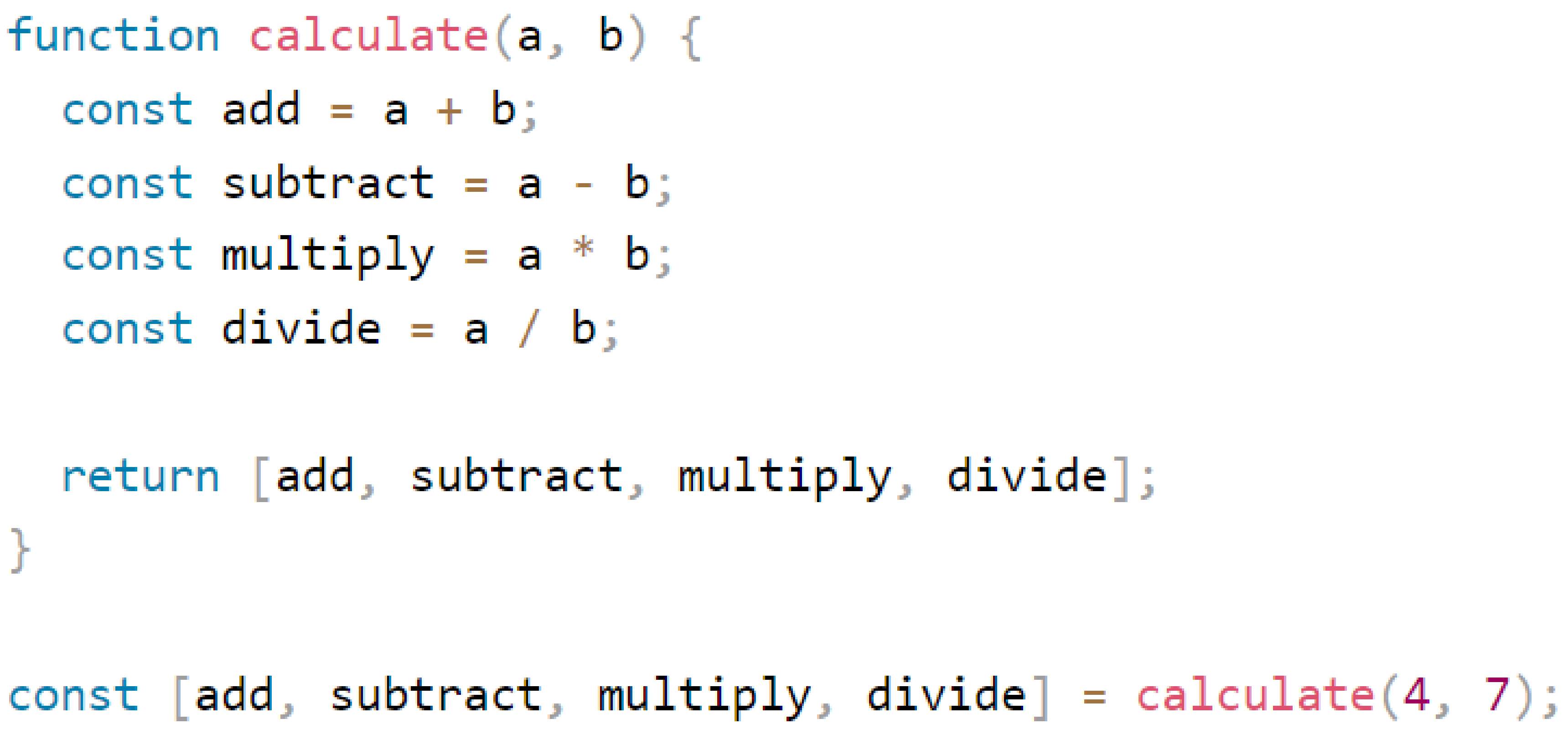
```
const vehicles = ['mustang', 'f-150', 'expedition'];

// old way
const car = vehicles[0];
const truck = vehicles[1];
const suv = vehicles[2];
```

New way



```
const vehicles = ['mustang', 'f-150', 'expedition'];

const [car, truck, suv] = vehicles;
```

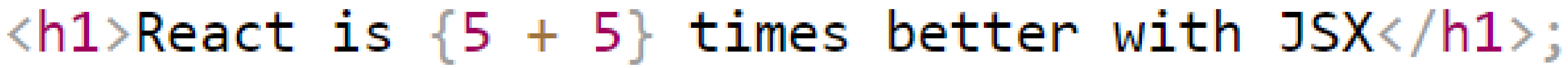
Example



```
function calculate(a, b) {
  const add = a + b;
  const subtract = a - b;
  const multiply = a * b;
  const divide = a / b;

  return [add, subtract, multiply, divide];
}

const [add, subtract, multiply, divide] = calculate(4, 7);
```

Expressions in JSX

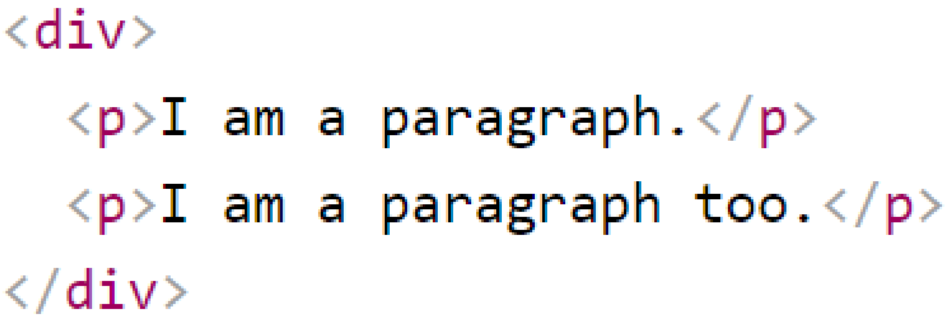
- With JSX you can write expressions inside curly braces { }.
- The expression can be a React variable, or property, or any other valid JavaScript expression.



```
<h1>React is {5 + 5} times better with JSX</h1>;
```

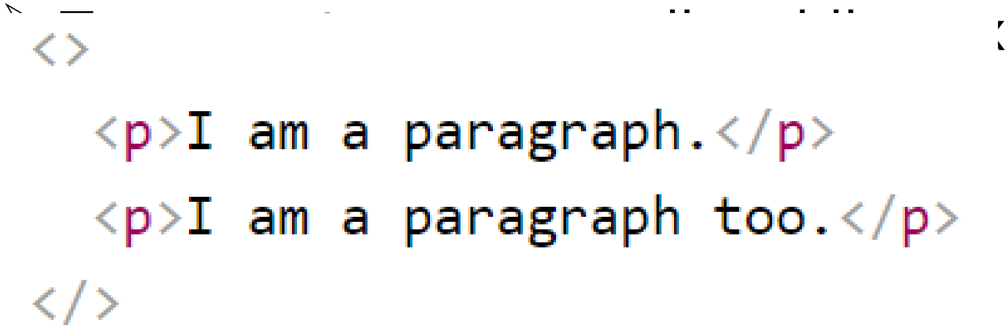
One Top Level Element

- The HTML code must be wrapped in ONE top level element.
- For Example:



```
<div>
  <p>I am a paragraph.</p>
  <p>I am a paragraph too.</p>
</div>
```

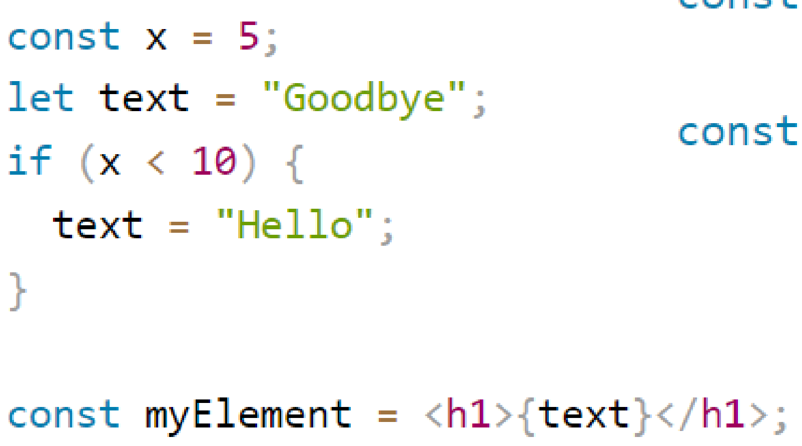
- This approach is not a good practice because we adding one **extra tag div tag** to the **DOM**.
- xtra nodes to the **DOM. Use fragment.**



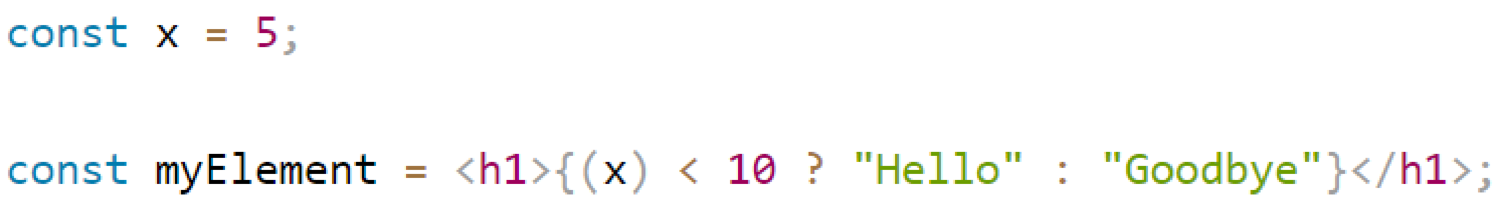
```
<>
  <p>I am a paragraph.</p>
  <p>I am a paragraph too.</p>
</>
```

Attribute class = className and Conditions - if statements

- The **class** keyword is a reserved word in JavaScript, Therefore not allowed to use it in JSX.
- Use attribute **className.**
- **React** supports **if** statements, but not inside JSX.
- To be able to use conditional statements in JSX, put the **if** statements outside of the JSX, or use a **ternary expression** instead.



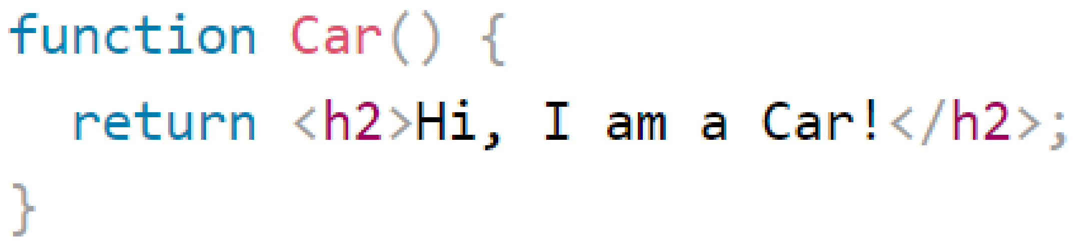
```
const x = 5;
let text = "Goodbye";
if (x < 10) {
  text = "Hello";
}

const myElement = <h1>{text}</h1>;
```



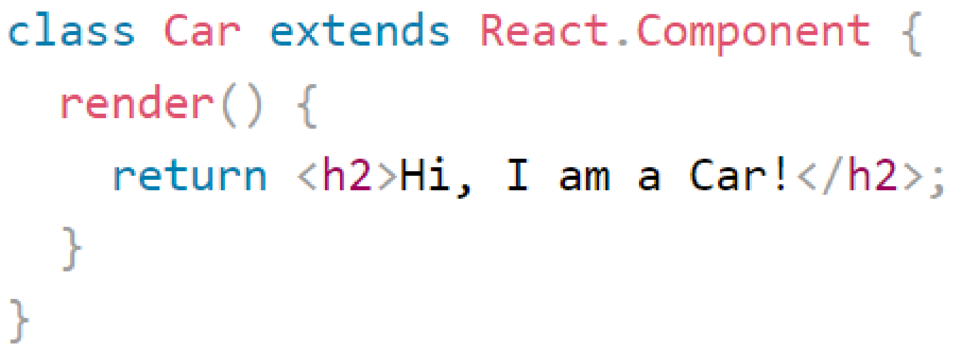
```
const x = 5;

const myElement = <h1>{(x) < 10 ? "Hello" : "Goodbye"}</h1>;
```

React Components

- Components are like functions that return HTML elements.
- Components are independent and reusable.
- Components come in two types.
  - Class components
  - Function components



```
function Car() {
  return <h2>Hi, I am a Car!</h2>;
}
```



```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}
```

A class component must include the extends React.Component statement.
The component also requires a render() method, this method returns HTML.

Install Bootstrap

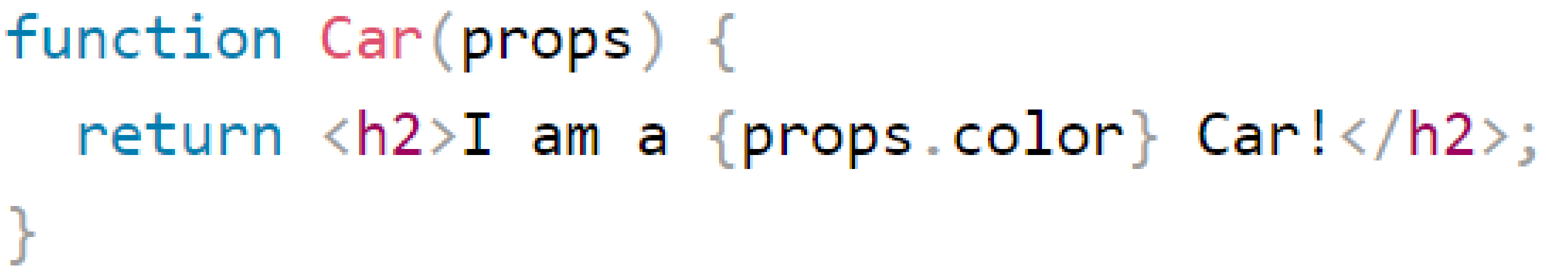
- Npm i bootstrap@latest

Now how to import

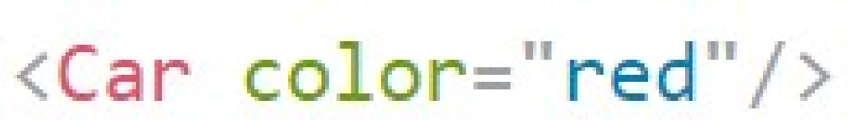
```
import "bootstrap/dist/css/bootstrap.css";
```

React Props

- Components can be passed as props, which stands for properties.
- Props are like function arguments, and we send them into the component as attributes

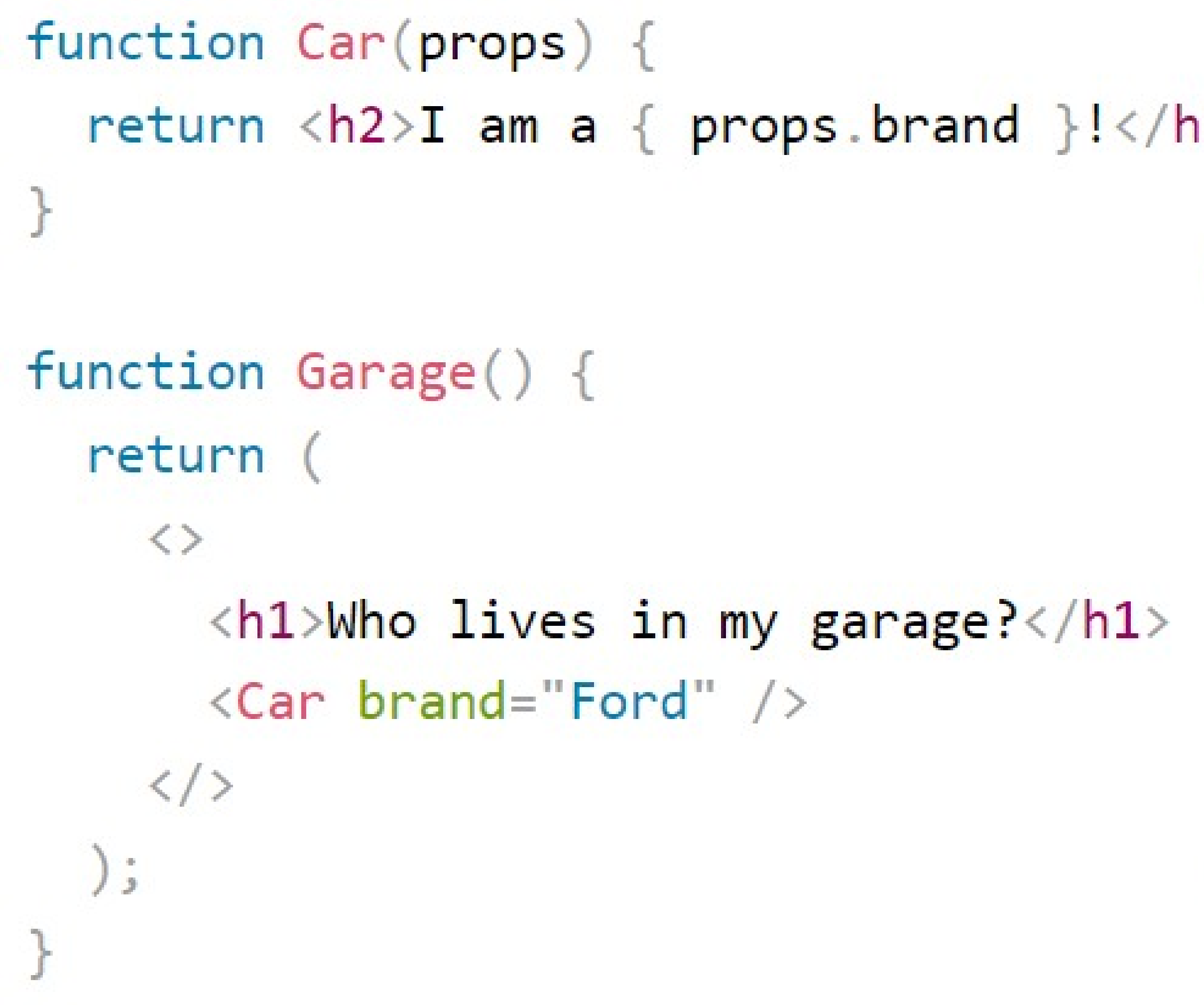


```
function Car(props) {
  return <h2>I am a {props.color} Car!</h2>;
}
```

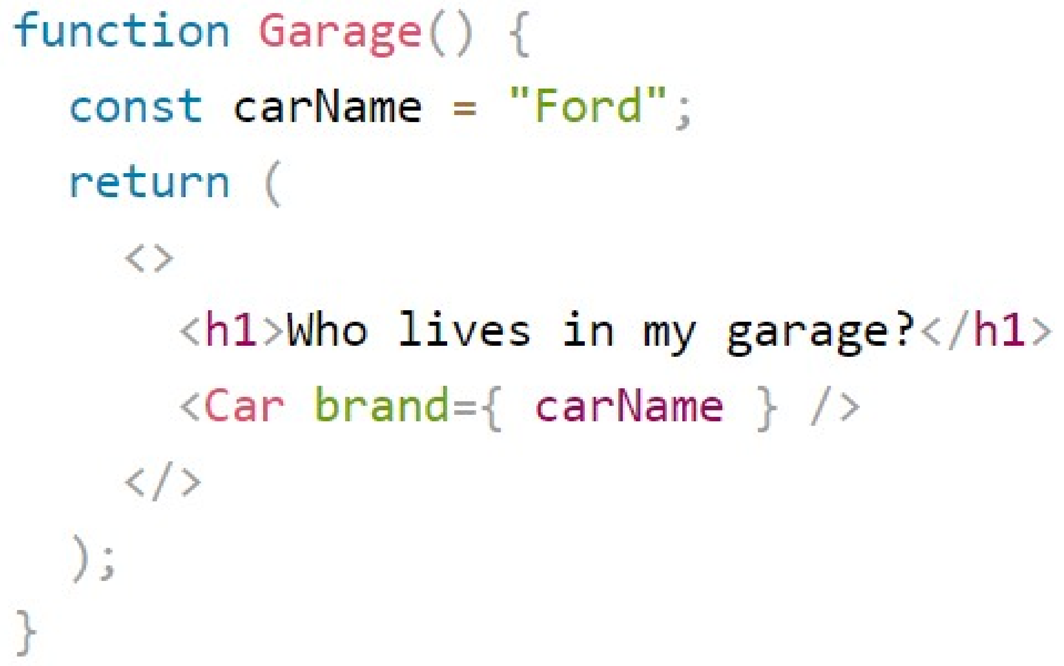


```
<Car color="red"/>
```

React Props



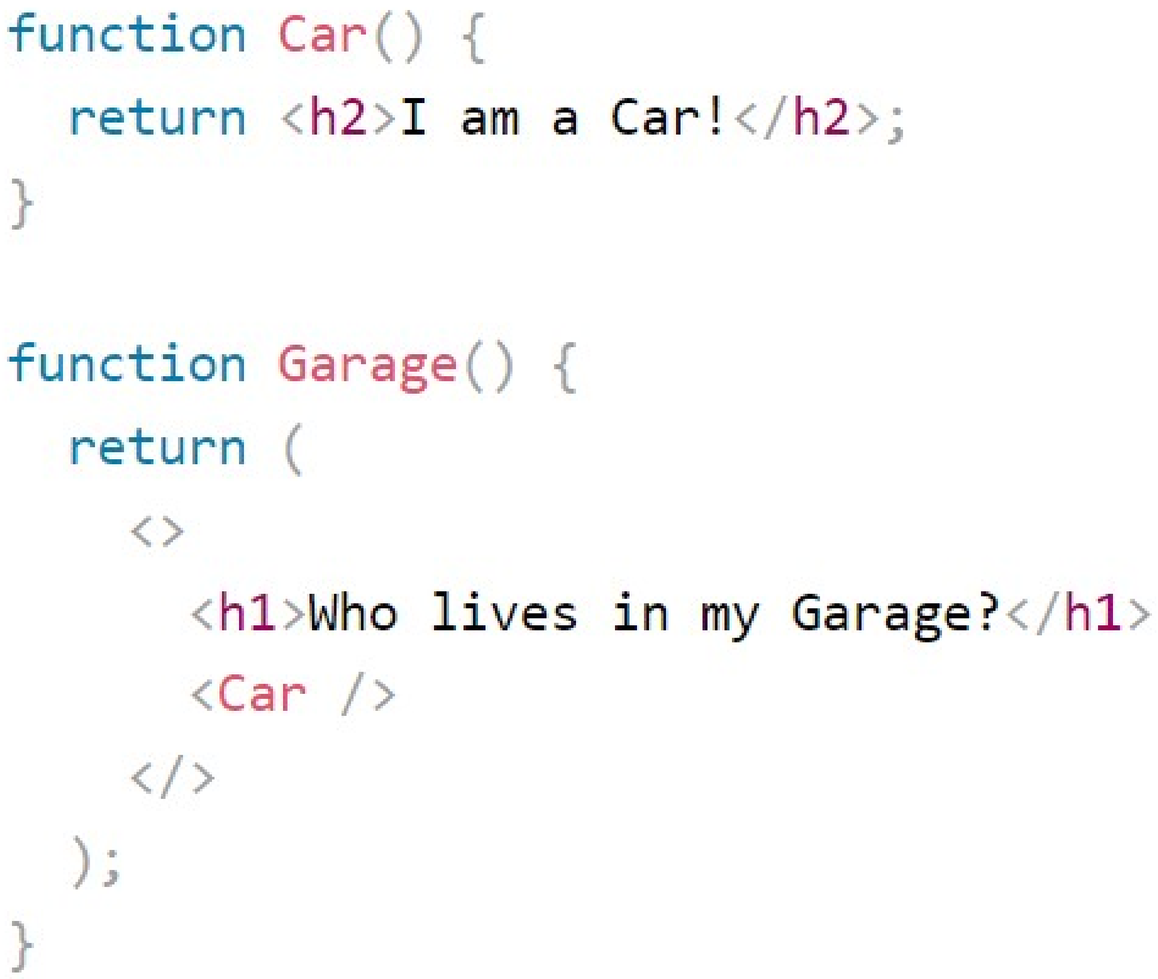
```
function Car(props) {
  return <h2>I am a { props.brand }!</h2>;
}

function Garage() {
  return (
    <>
      <h1>Who lives in my garage?</h1>
      <Car brand="Ford" />
    </>
  );
}
```



```
function Garage() {
  const carName = "Ford";
  return (
    <>
      <h1>Who lives in my garage?</h1>
      <Car brand={ carName } />
    </>
  );
}
```

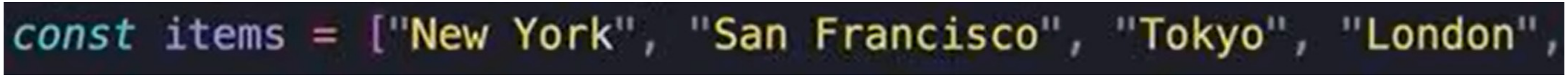
Components in Components

- We can refer to components inside other components.
- React is all about re-using code.

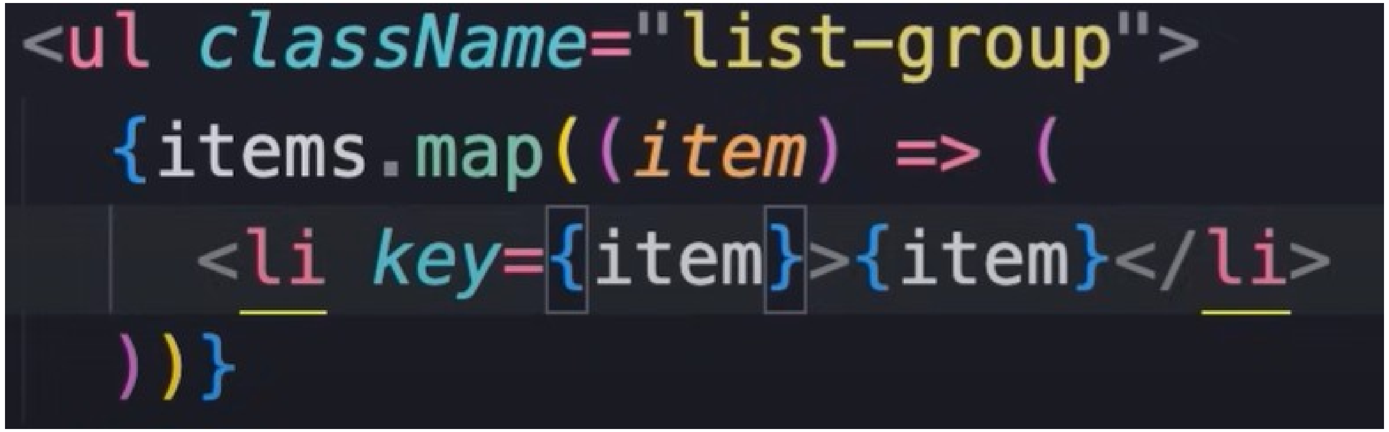


```
function Car() {
  return <h2>I am a Car!</h2>;
}

function Garage() {
  return (
    <>
      <h1>Who lives in my Garage?</h1>
      <Car />
    </>
  );
}
```

Map function

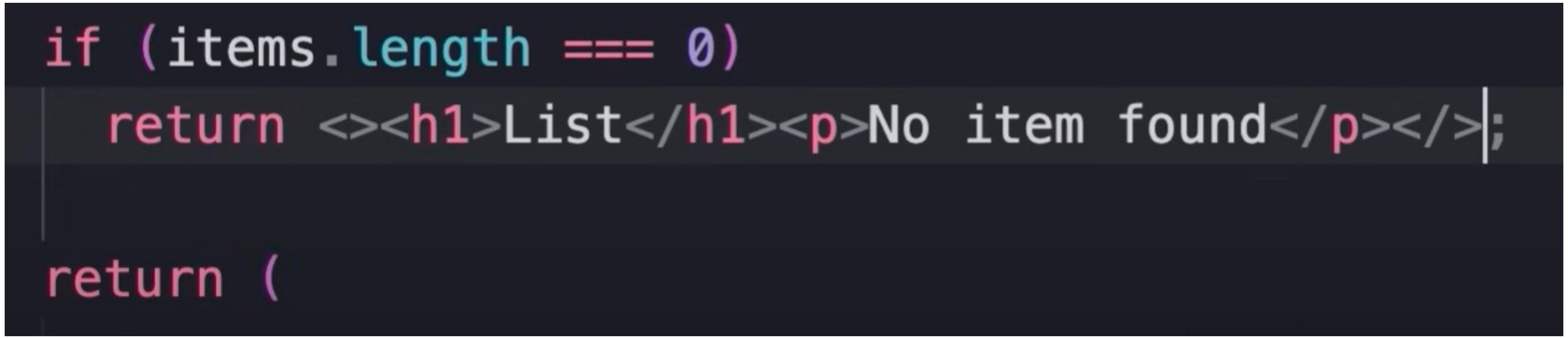


```
const items = ["New York", "San Francisco", "Tokyo", "London",
```

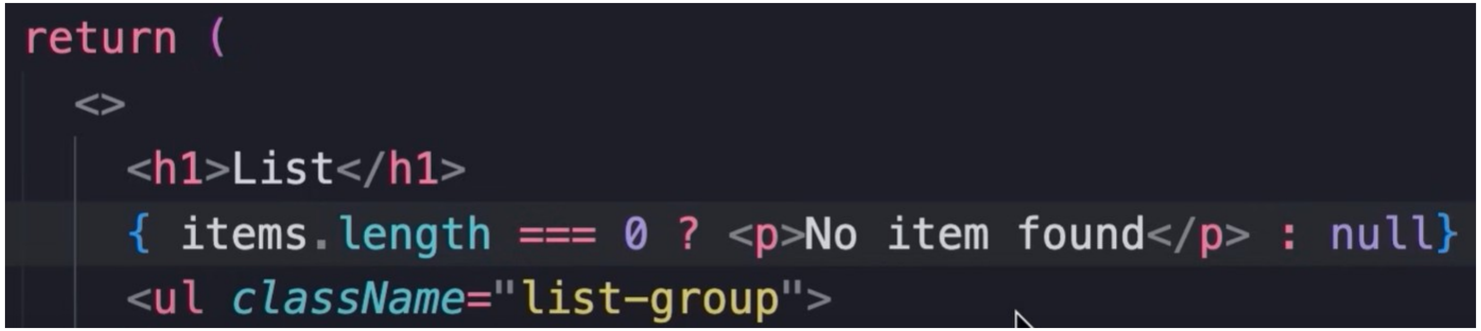


```
<ul className="list-group">
  {items.map((item) => (
    <li key={item}>{item}</li>
  ))}
```

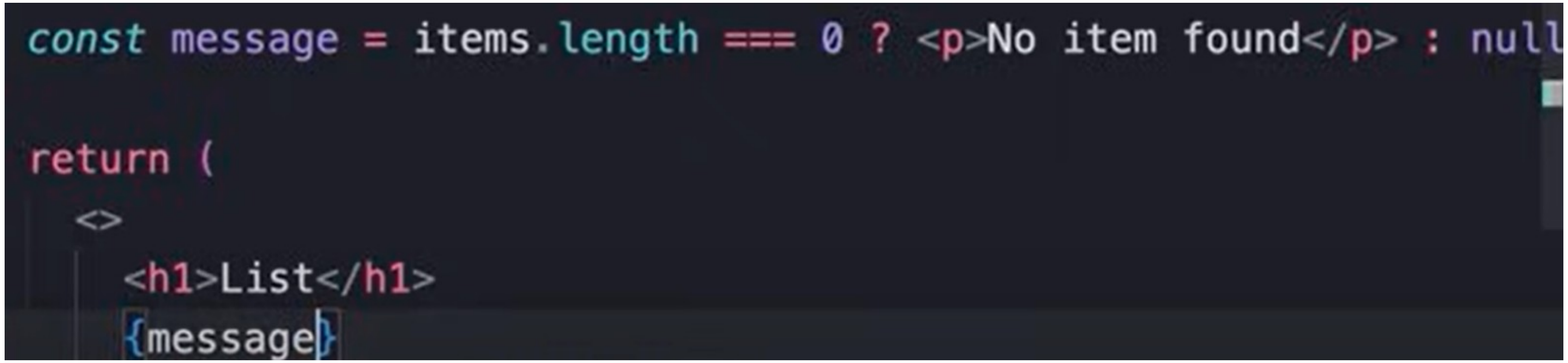
Conditional rendering



```
if (items.length === 0)
  return <><h1>List</h1><p>No item found</p></>;

return (
```

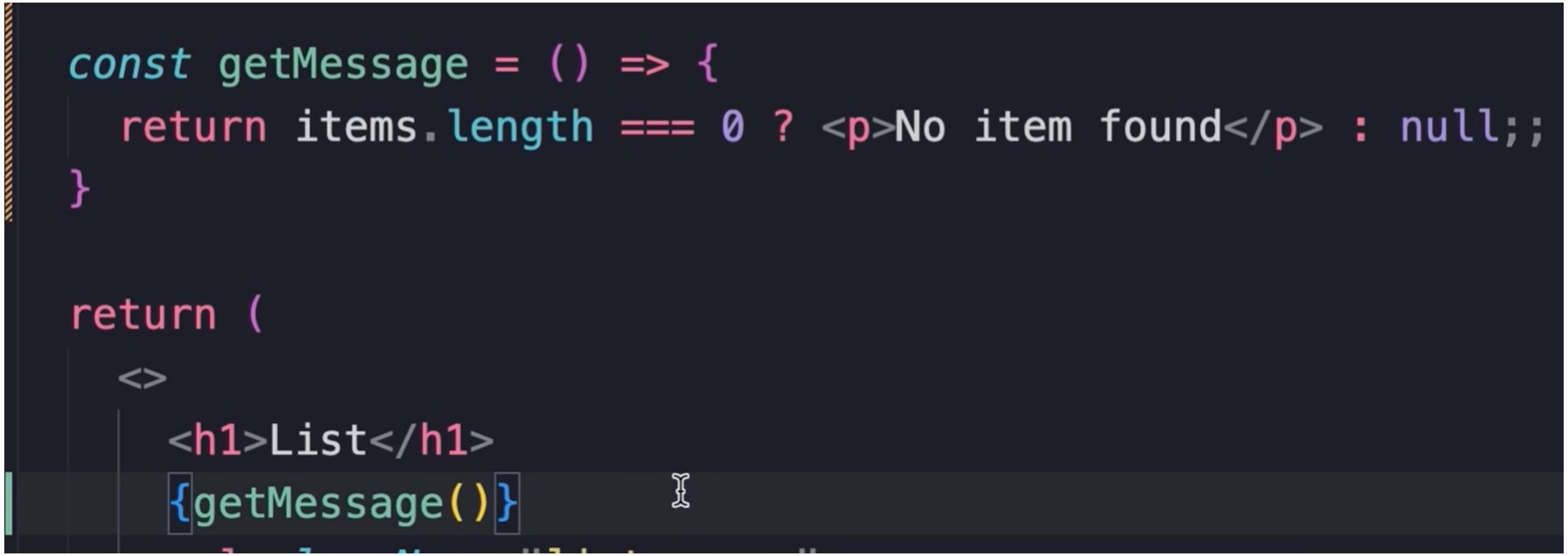


```
return (
  <>
    <h1>List</h1>
    { items.length === 0 ? <p>No item found</p> : null}
    <ul className="list-group">
```



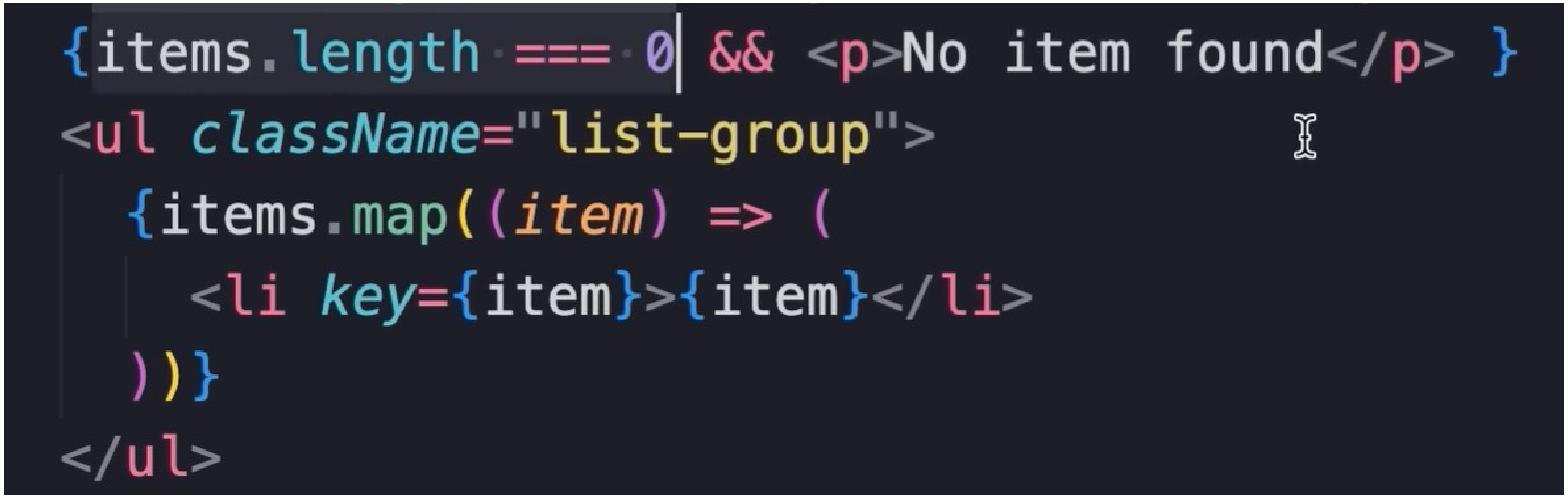
```
const message = items.length === 0 ? <p>No item found</p> : null

return (
  <>
    <h1>List</h1>
    {message}
```

Conditional rendering

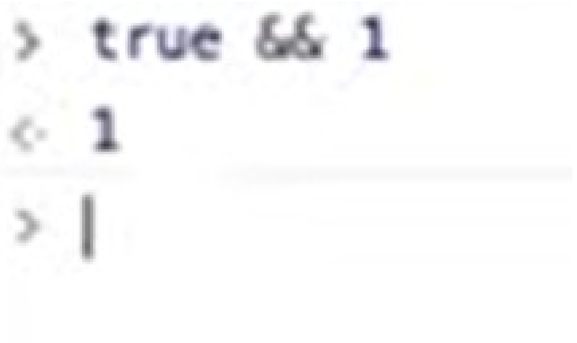


```
const getMessage = () => {
  return items.length === 0 ? <p>No item found</p> : null;;
}

return (
  <>
    <h1>List</h1>
    {getMessage()}
```

More concise and better way of Conditional rendering

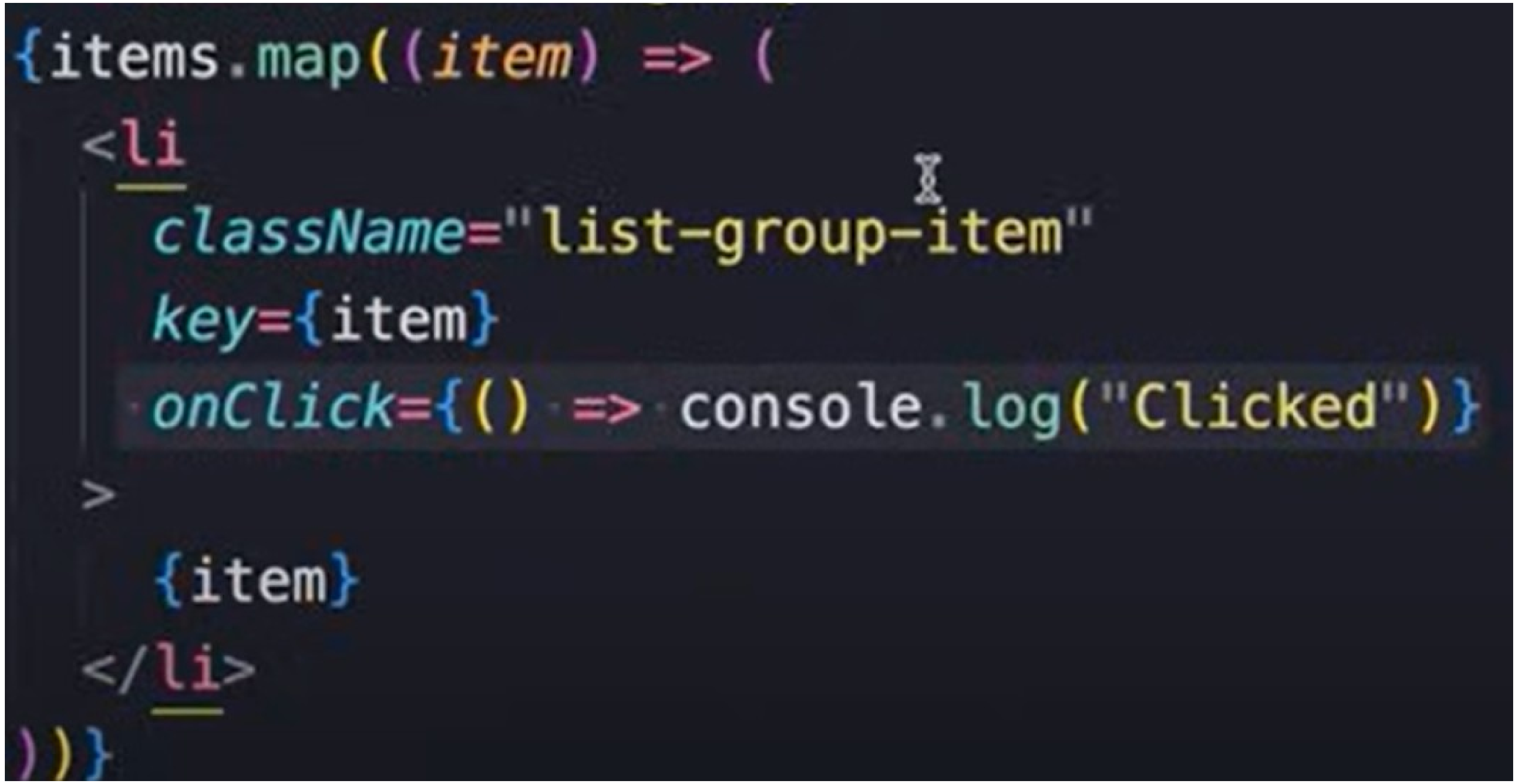


```
{items.length === 0 && <p>No item found</p> }
<ul className="list-group">
  {items.map((item) => (
    <li key={item}>{item}</li>
  ))}
</ul>
```

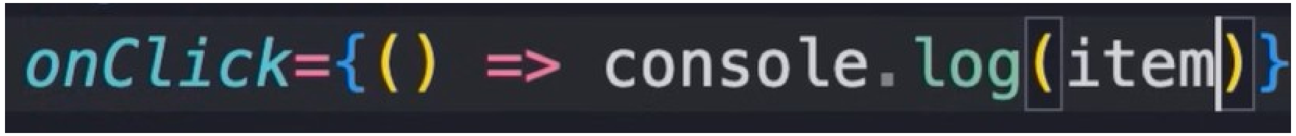


```
> true && 1
<· 1
> |
```

Handling events

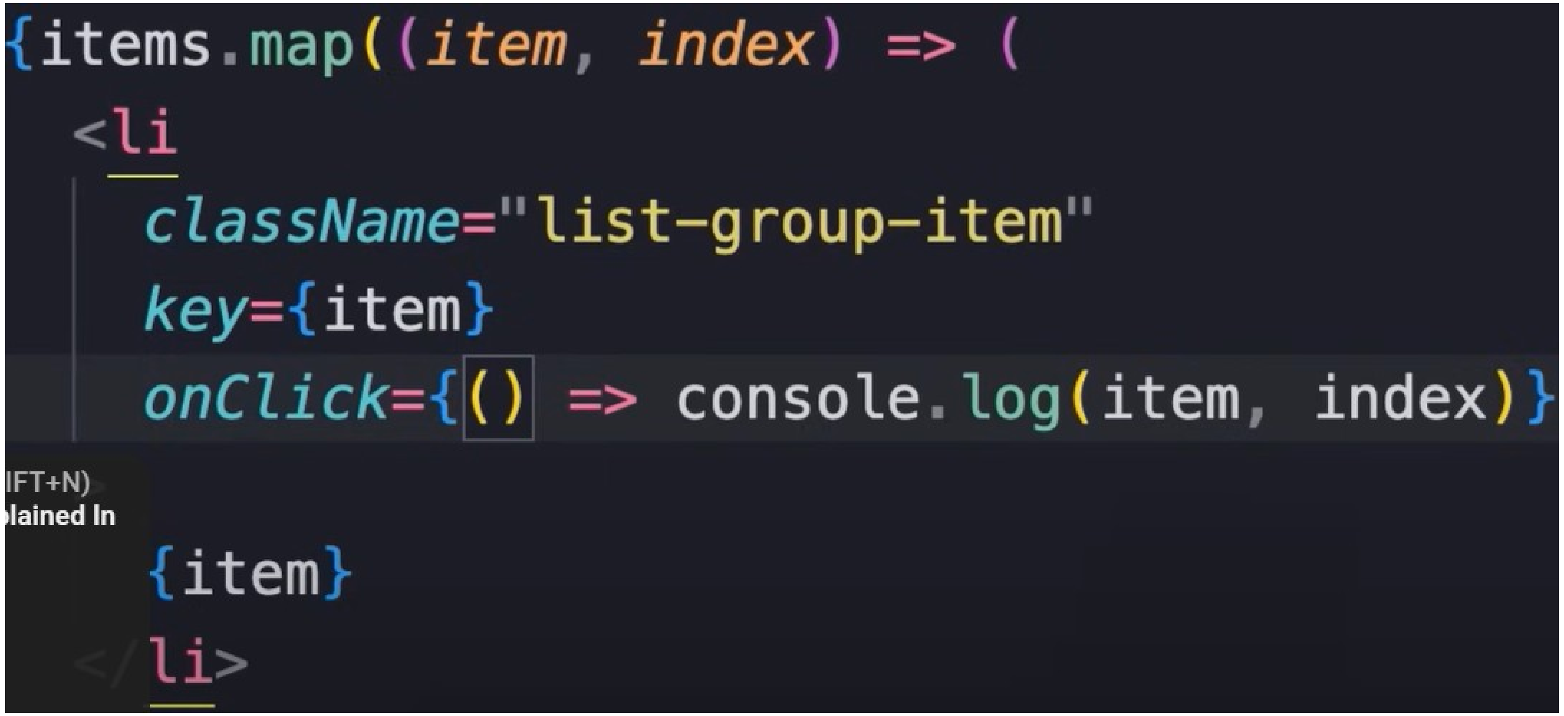


```
{items.map((item) => (
  <li
    className="list-group-item"
    key={item}
    onClick={() => console.log("Clicked")}
  >
    {item}
  </li>
))}
```

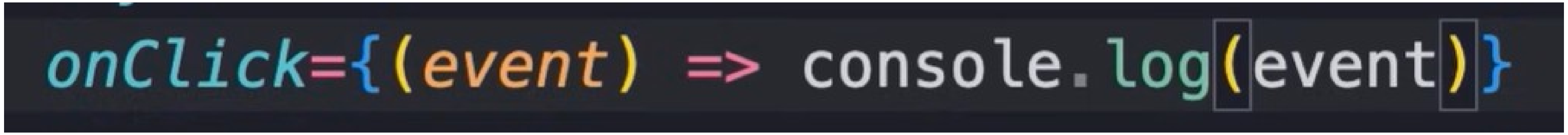


```
onClick={() => console.log(item)}
```

Handling events



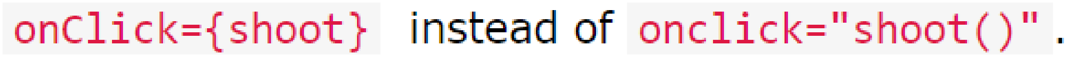
```
{items.map((item, index) => (
  <li
    className="list-group-item"
    key={item}
    onClick={() => console.log(item, index)}

    {item}
  </li>
```



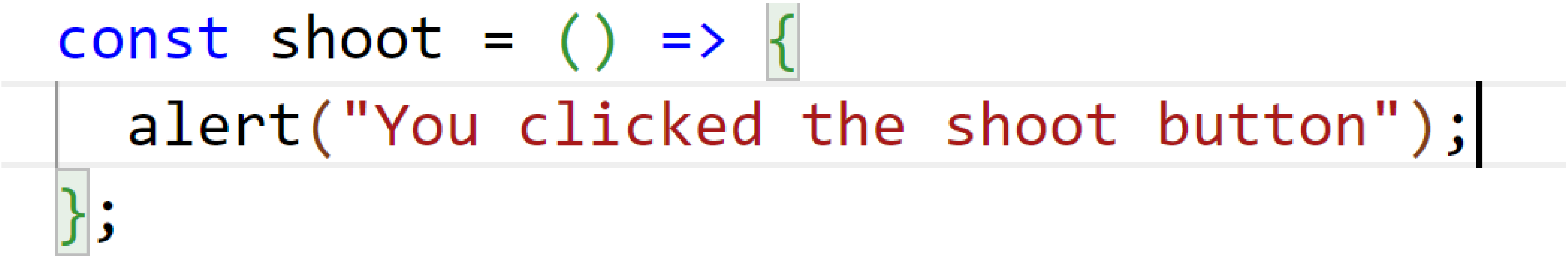
```
onClick={(event) => console.log(event)}
```

React Events

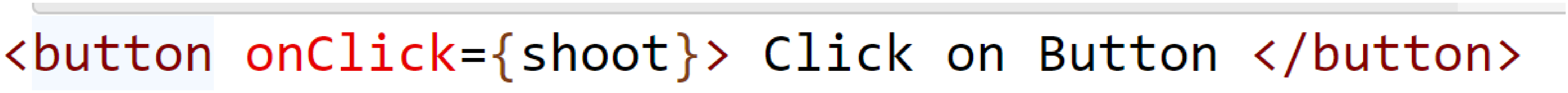
- Just like HTML DOM events, React can perform actions based on user events.
- React events are written in camelCase syntax:
  - onClick instead of onclick.
  - **React event** handlers are written **inside curly braces**:



`onClick={shoot}` instead of `onclick="shoot()"` .

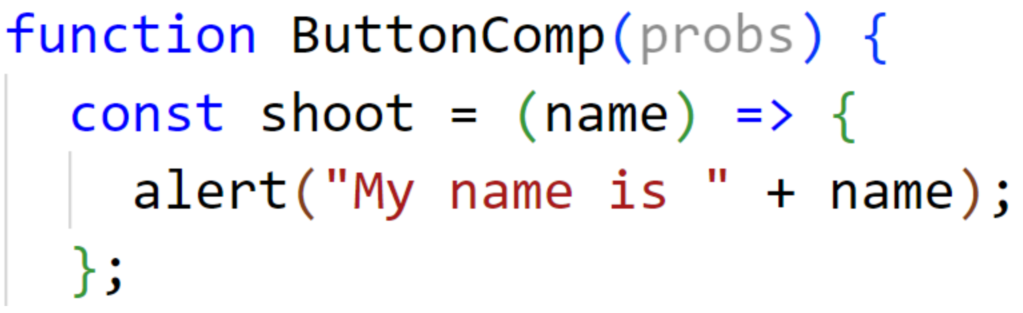


```
const shoot = () => {
  alert("You clicked the shoot button");
};
```

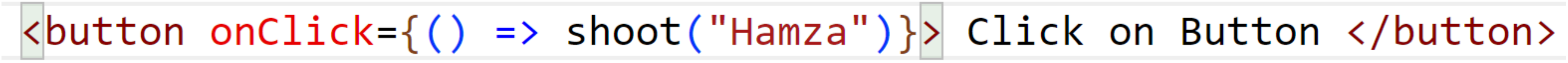


```
<button onClick={shoot}> Click on Button </button>
```

Passing Arguments to React Events

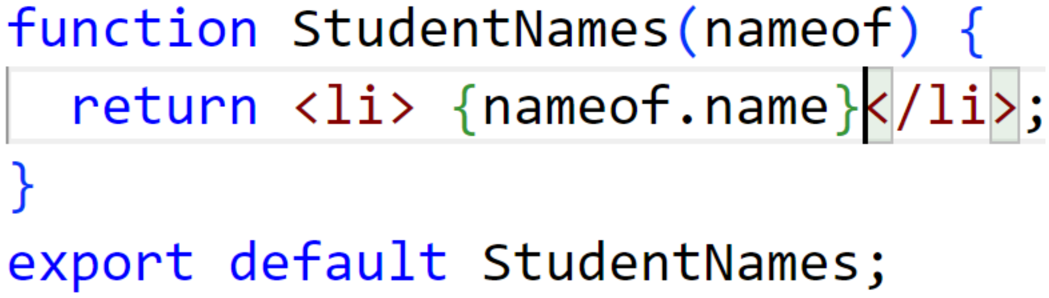


```
function ButtonComp(probs) {
  const shoot = (name) => {
    alert("My name is " + name);
  };
```

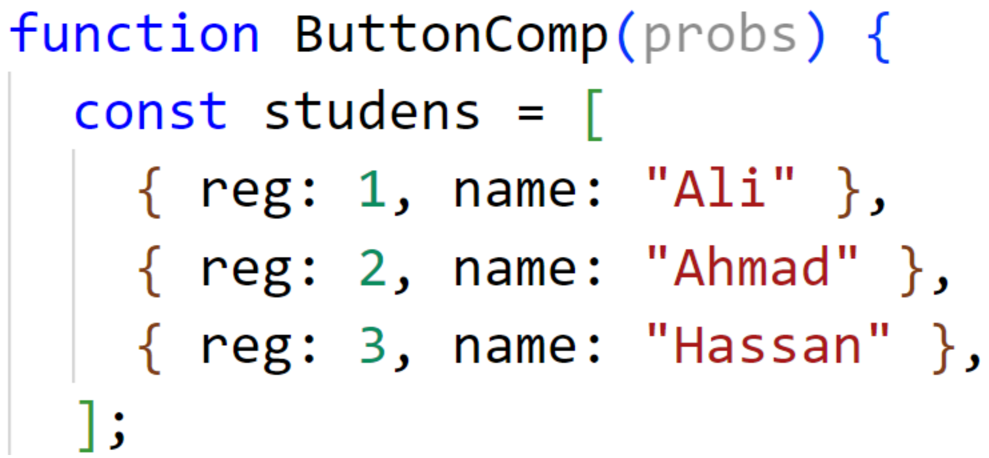


```
<button onClick={() => shoot("Hamza")}> Click on Button </button>
```

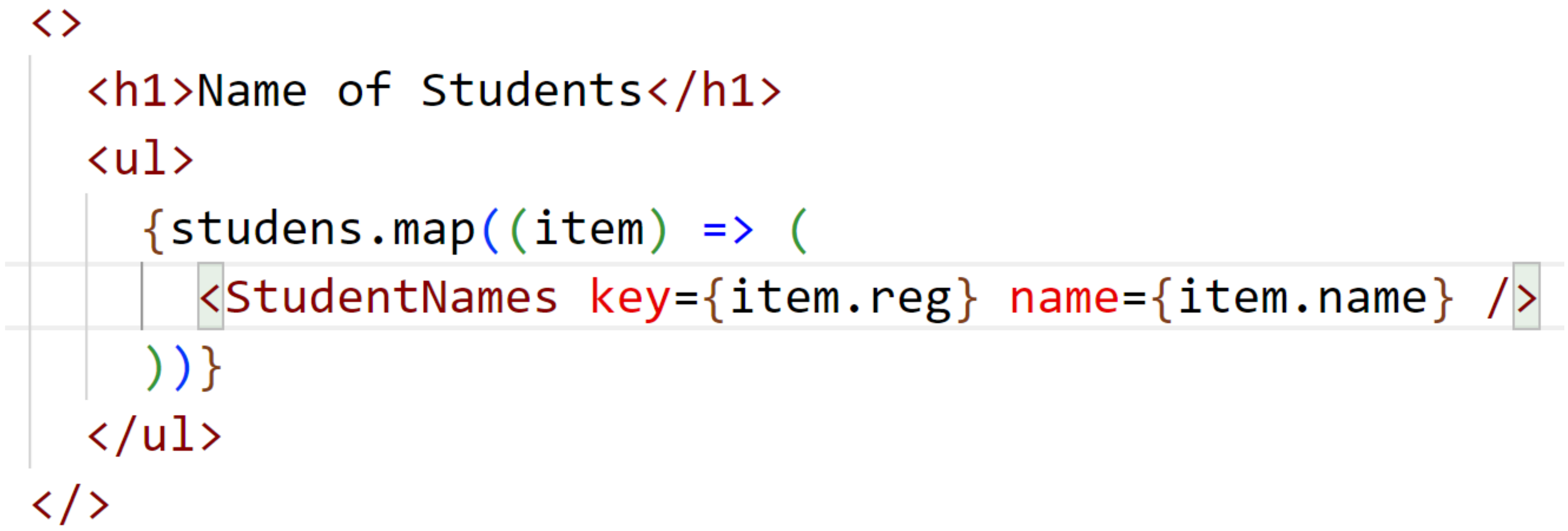
React List



```
function StudentNames(nameof) {
  return <li> {nameof.name}</li>;
}
export default StudentNames;
```

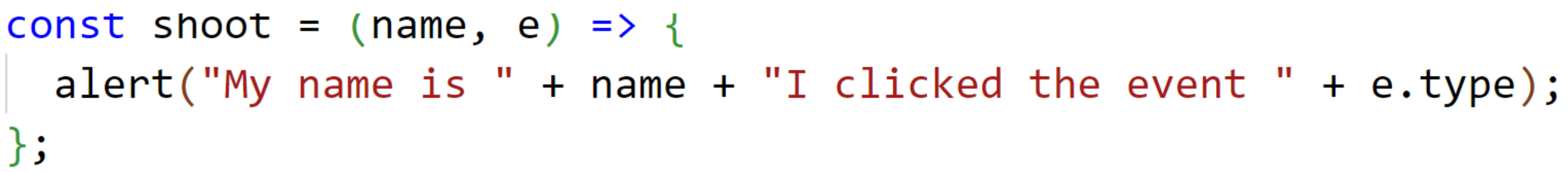


```
function ButtonComp(probs) {
  const studens = [
    { reg: 1, name: "Ali" },
    { reg: 2, name: "Ahmad" },
    { reg: 3, name: "Hassan" },
  ];
```

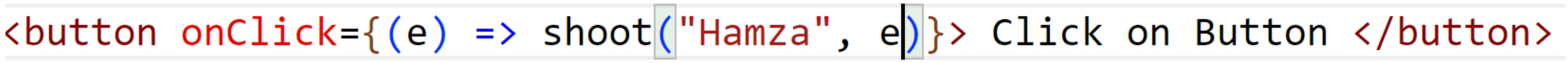


```
<>
  <h1>Name of Students</h1>
  <ul>
    {studens.map((item) => (
      <StudentNames key={item.reg} name={item.name} />
    ))}
  </ul>
</>
```

React Event Object

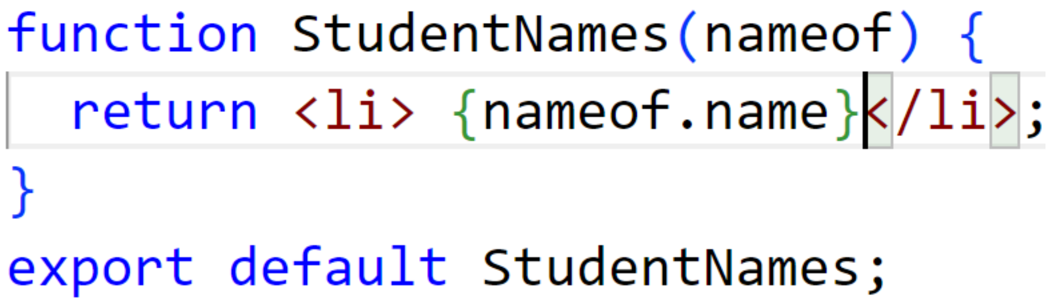


```
const shoot = (name, e) => {
  alert("My name is " + name + "I clicked the event " + e.type);
};
```

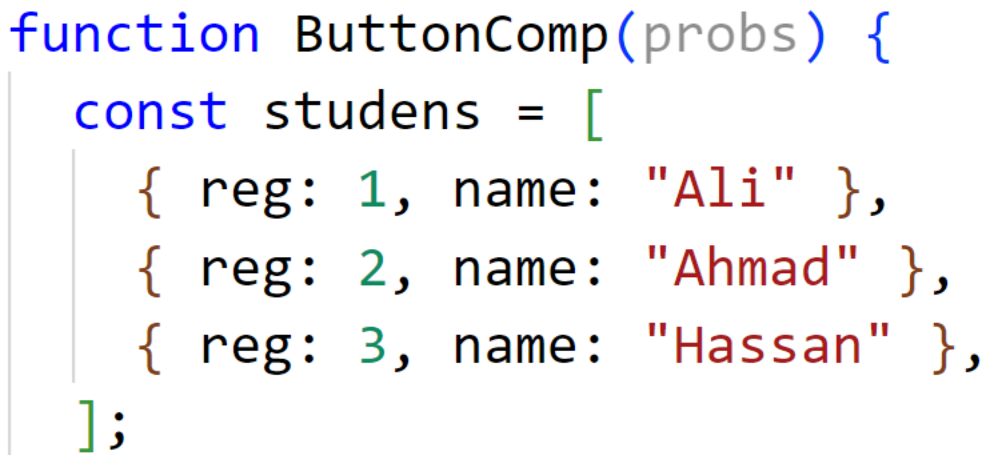


```
<button onClick={(e) => shoot("Hamza", e)}> Click on Button </button>
```

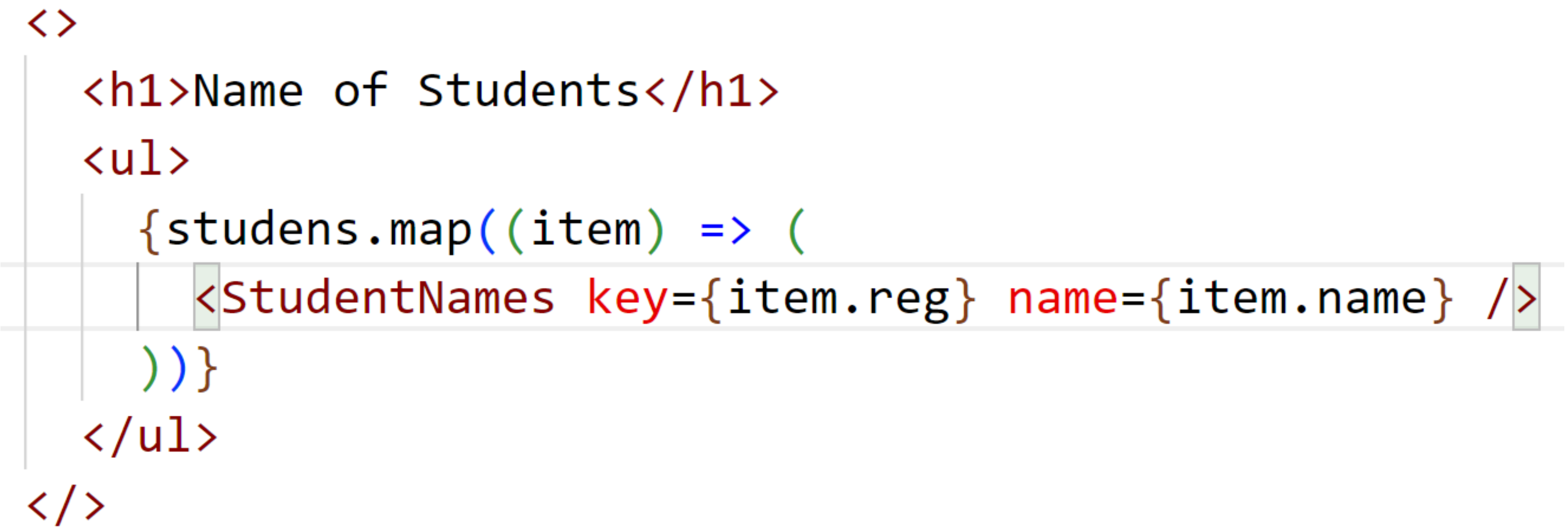
React List



```
function StudentNames(nameof) {
  return <li> {nameof.name}</li>;
}
export default StudentNames;
```



```
function ButtonComp(probs) {
  const studens = [
    { reg: 1, name: "Ali" },
    { reg: 2, name: "Ahmad" },
    { reg: 3, name: "Hassan" },
  ];
```



```
<>
  <h1>Name of Students</h1>
  <ul>
    {studens.map((item) => (
      <StudentNames key={item.reg} name={item.name} />
    ))}
  </ul>
</>
```

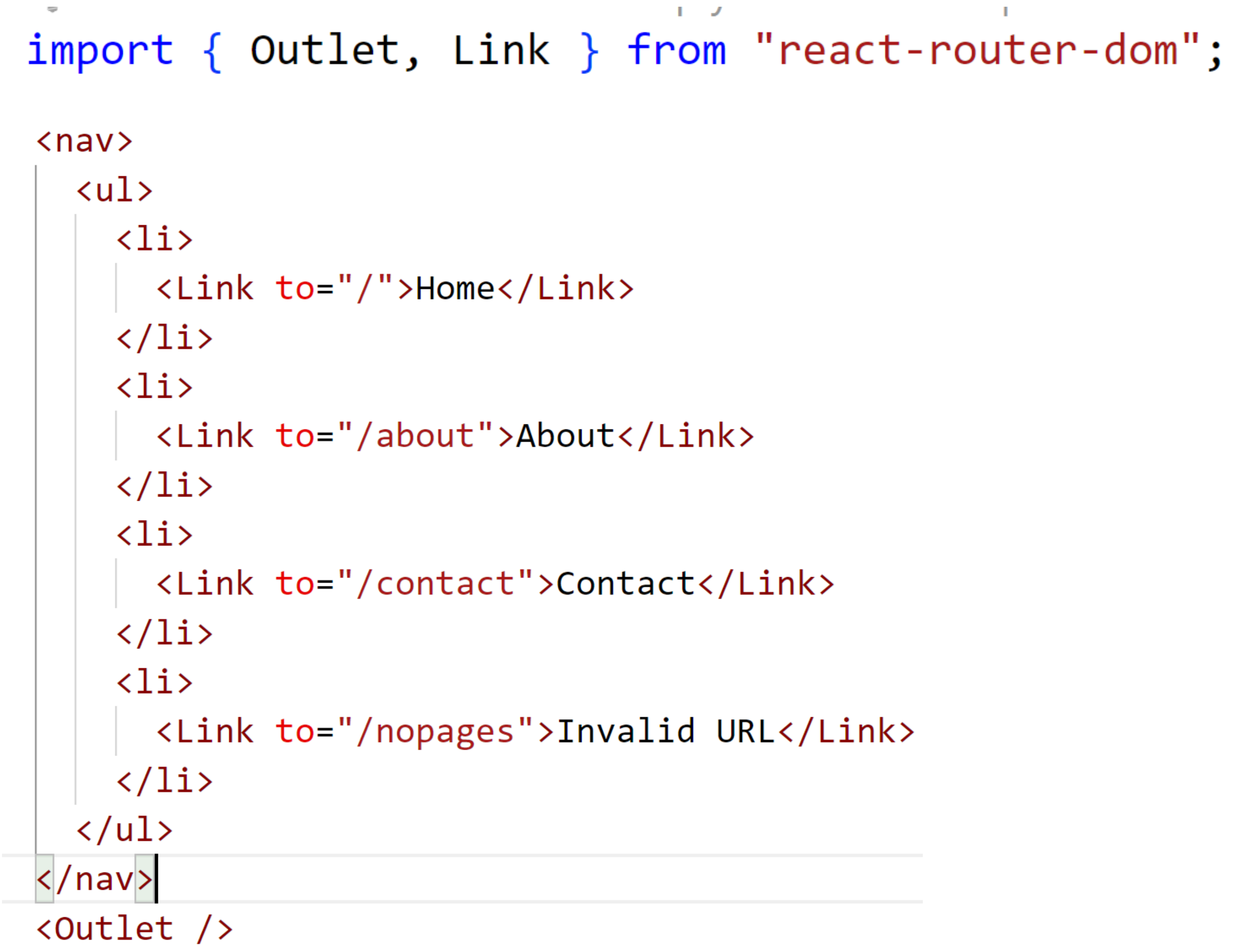
Adding Forms in React

- In HTML, form data is usually handled by the DOM.
- In React, form data is usually handled by the components.
- When the data is handled by the components, all the data is stored in the component state.
- We can control changes by adding event handlers in the onChange attribute.
- We can use the useState Hook to keep track of each inputs value.

React Router

- React App doesn't include page routing.
- React Router is the most popular solution.
- **npm i react-router-dom or npm i react-router-dom@latest**
- **Folder Structure**
  - Within the **src** folder, we'll create a folder named **pages** with several files:

React Router



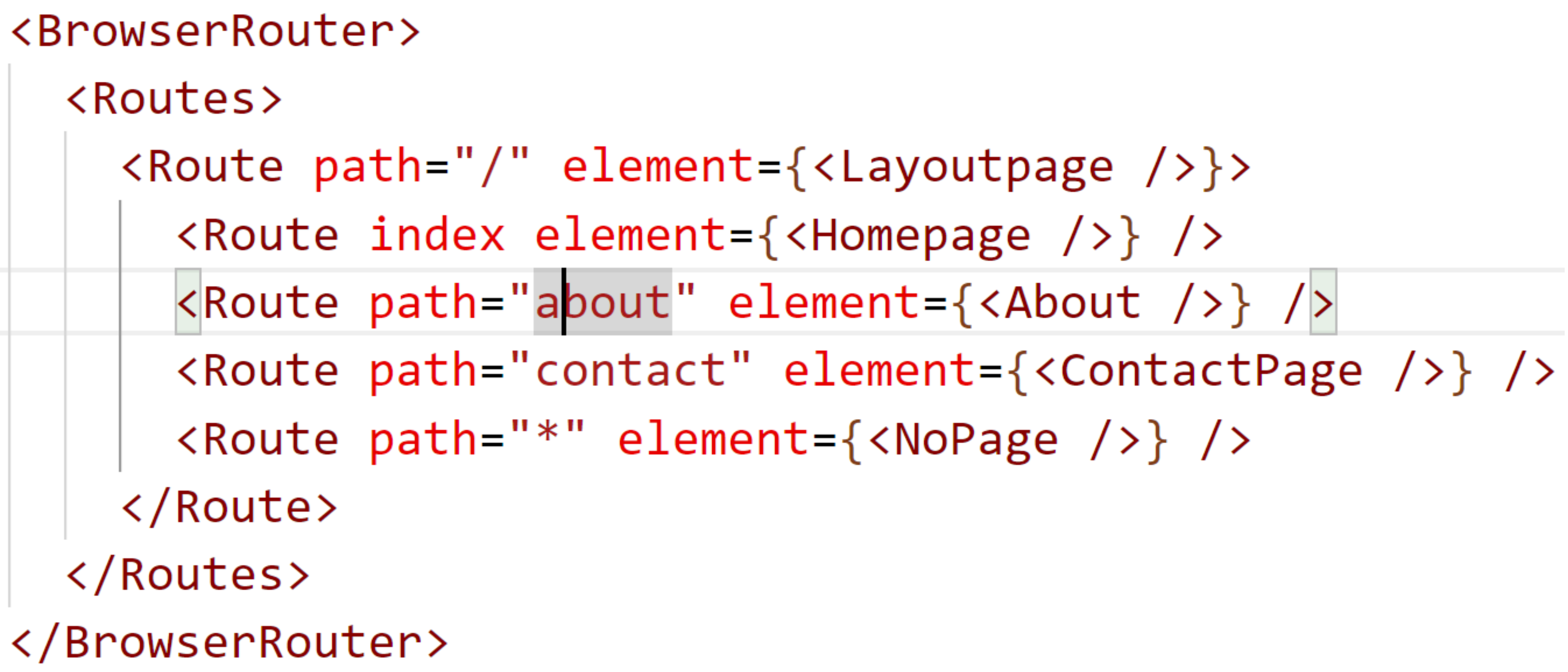
```
import { Outlet, Link } from "react-router-dom";

<nav>
  <ul>
    <li>
      <Link to="/">Home</Link>
    </li>
    <li>
      <Link to="/about">About</Link>
    </li>
    <li>
      <Link to="/contact">Contact</Link>
    </li>
    <li>
      <Link to="/nopages">Invalid URL</Link>
    </li>
  </ul>
</nav>
<Outlet />
```

LayoutPage .jsx

React Router

```
import { BrowserRouter, Route, Routes } from "react-router-dom";
```

App.jsx



```
<BrowserRouter>
  <Routes>
    <Route path="/" element={<Layoutpage />}>
      <Route index element={<Homepage />} />
      <Route path="about" element={<About />} />
      <Route path="contact" element={<ContactPage />} />
      <Route path="*" element={<NoPage />} />
    </Route>
  </Routes>
</BrowserRouter>
```

React Router

- We wrap our content first with **<BrowserRouter>.**
- Then we define our **<Routes>.**
- An application can have multiple **<Routes>.**
- **<Route>s** can be nested. The first **<Route>** has a path of **/** and renders the **Layout component**.
- The nested <Route>s inherit and add to the **parent route**. So the **About** path is combined with the parent and becomes **/ About**.
- The **Homepage** component route does not have a path but has an index attribute. That specifies this route as the default route for the parent route, which is **/**.
- Setting the path **to *** will act as a **catch-all** for any **undefined URLs**. This is great for a **404 error** page.

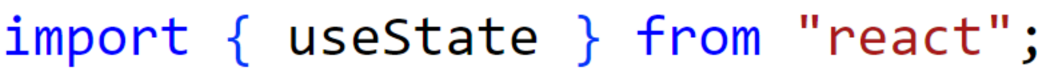
React Hooks

- Hooks generally replace class components, there are no plans to remove classes from React.
- Hooks allow function components to have access to state and other React features.
- We must import Hooks from react.
- **Hook Rules**
  - Hooks can only be called inside React function components.
  - Hooks can only be called at the top level of a component.
  - Hooks cannot be conditional.

React useState Hooks

- ➢ React useState Hook allows us to track state in a function components.
- ➢ State generally refers to data or properties that need to be tracking in an application.

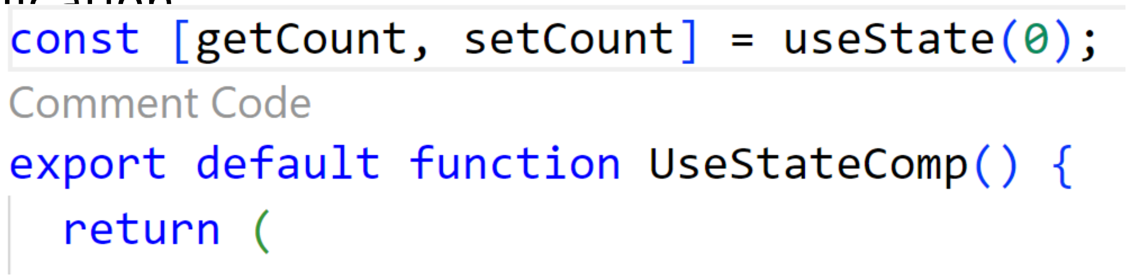
Import `useState`



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

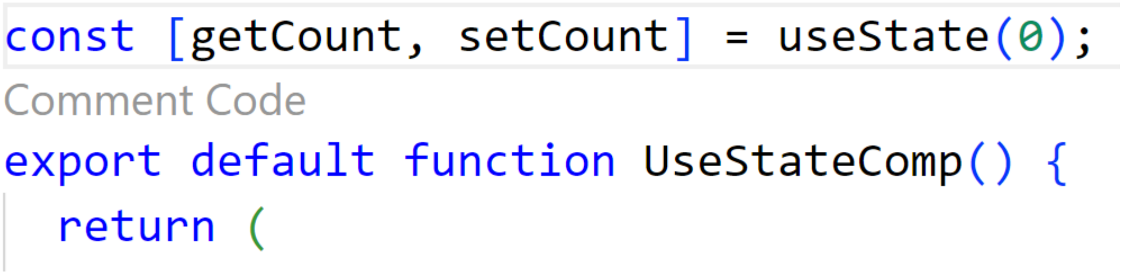
Initialize `useState`

- ➢ useState accepts an initial state and returns two values:
- ➢ The current state.
  - ➢ A function that updates the state. a function components.
  - ➢ A variable that generally refers to data or properties that need to be tracking in an application.



```
const [getCount, setCount] = useState(0);
Comment Code
export default function UseStateComp() {
  return (
```

React useState Hooks

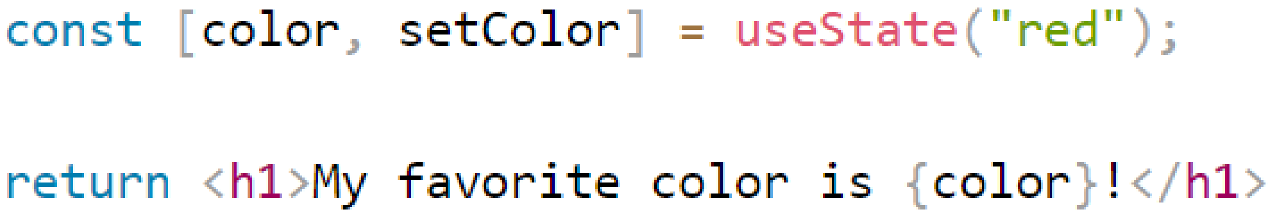


```
const [getCount, setCount] = useState(0);
```

Comment Code

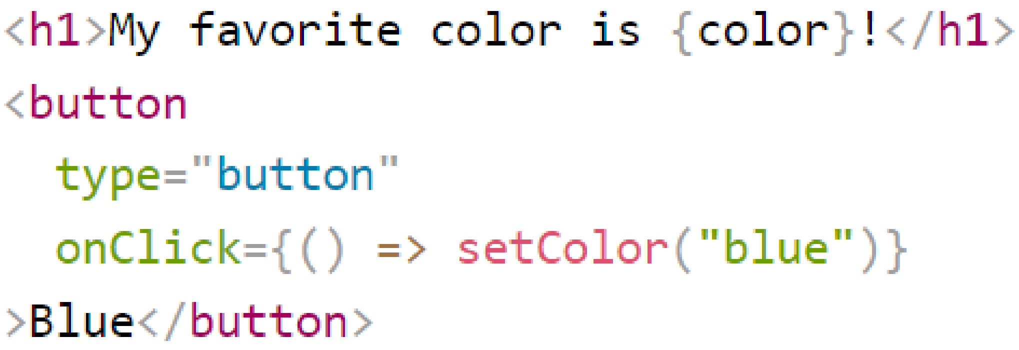
```
export default function UseStateComp() {
  return (
```

- ➢ we are destructuring the returned values from useState.
- ➢ The first value, **getcount**, is our current state.
- ➢ The second value, **setCount**, is the function that is used to update our state.
- ➢ We can now include our state anywhere in our component.

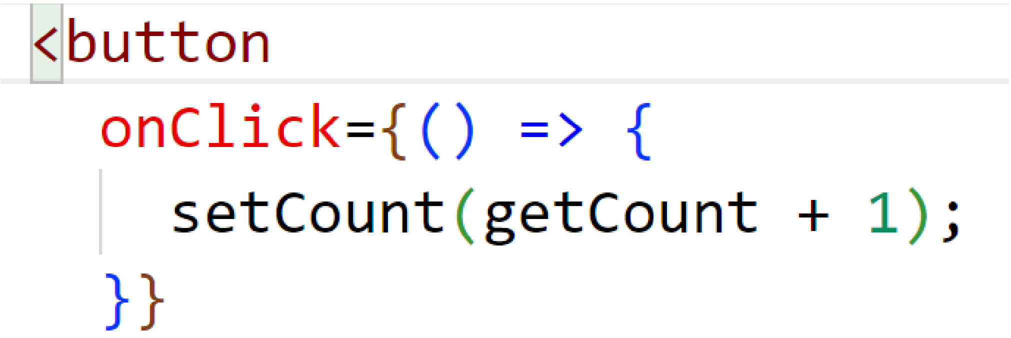


```
const [color, setColor] = useState("red");

return <h1>My favorite color is {color}!</h1>
```

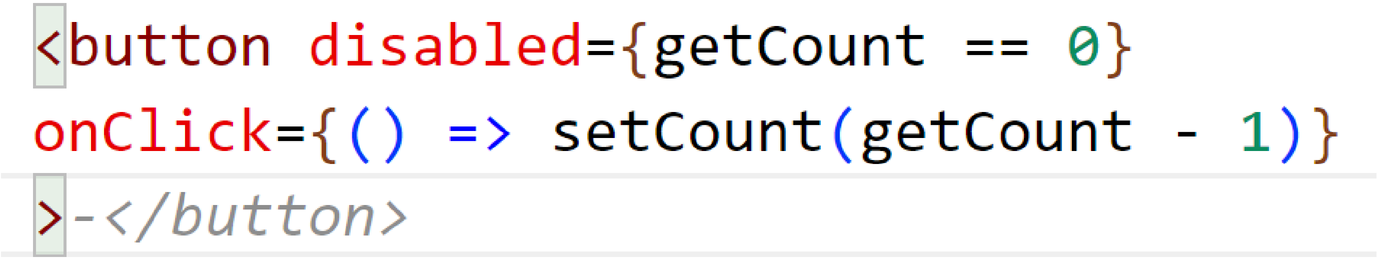
Update State



```
<h1>My favorite color is {color}!</h1>
<button
  type="button"
  onClick={() => setColor("blue")}
>Blue</button>
```



```
<button
  onClick={() => {
    setCount(getCount + 1);
  }}
```



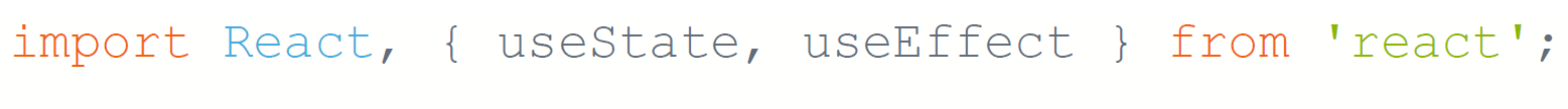
```
<button disabled={getCount == 0}
onClick={() => setCount(getCount - 1)}
>-</button>
```

React useEffect Hooks

- The **useEffect** Hook allows you to perform side effects in your components.
- For instance: **fetching data**, **directly updating the DOM**, and **timers**.
- **useEffect** accepts **two arguments**. The second argument is **optional**.
- useEffect(<function>, <dependency>).
- Syntax:
  - **useEffect**(() => {

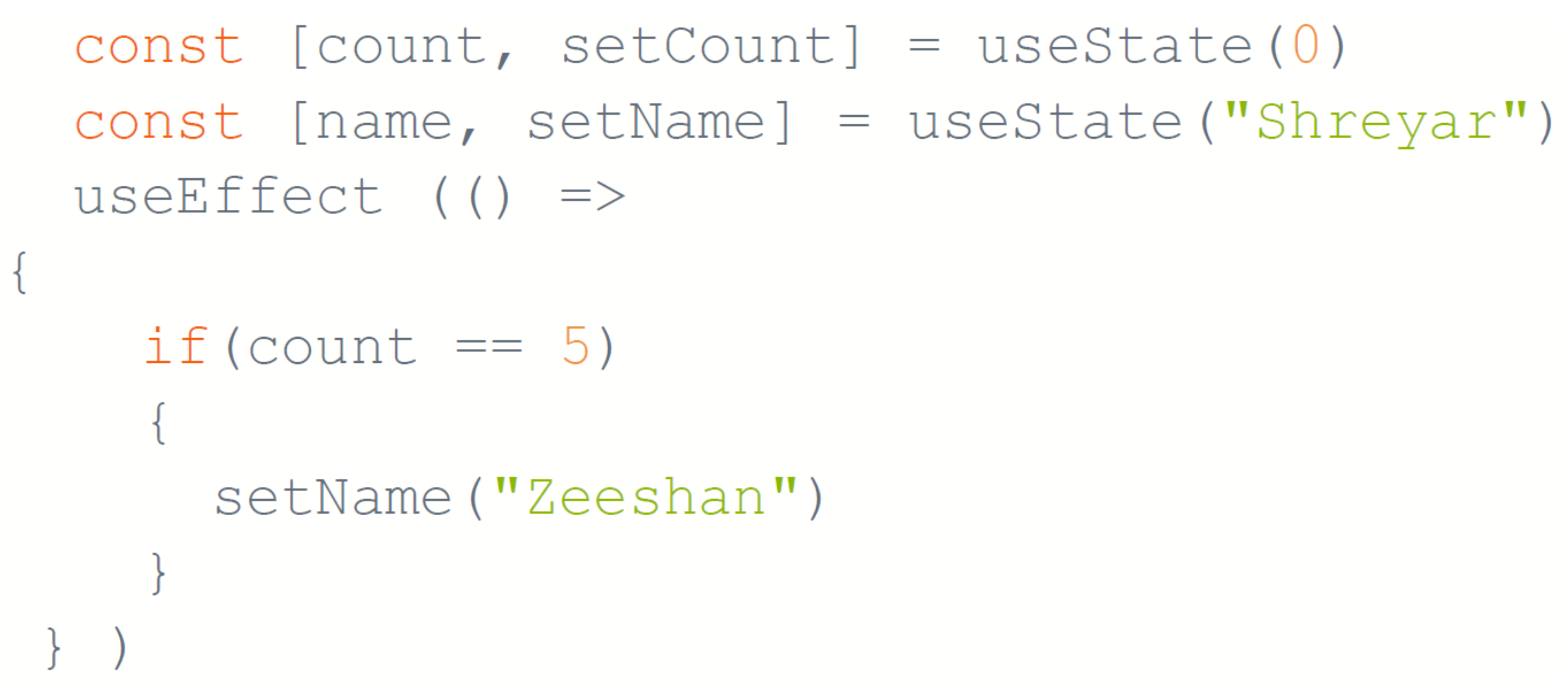
    //statements

    }, [**dependency**])
  - The **dependency** argument is **optional**.



```
import React, { useState, useEffect } from 'react';
```

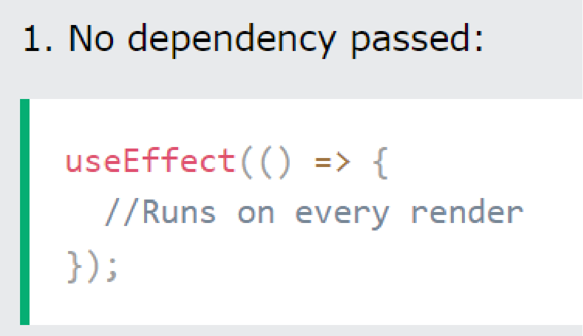
React useEffect Hooks



```
  const [count, setCount] = useState(0)
  const [name, setName] = useState("Shreyar")
  useEffect (() =>
{
    if(count == 5)
    {
      setName("Zeeshan")
    }
 } )
```

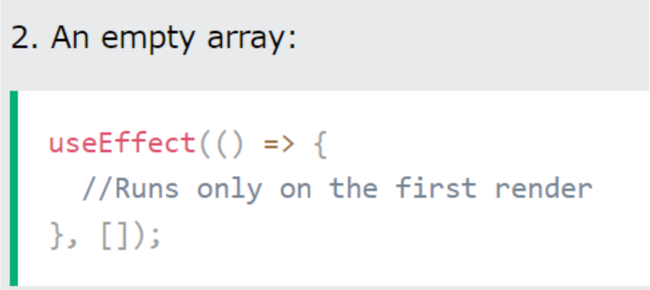
```
<h1> {name} </h1>
<button title="Add" onPress={() => setCount(count + 1)}>
</button>
```

React useEffect Hooks



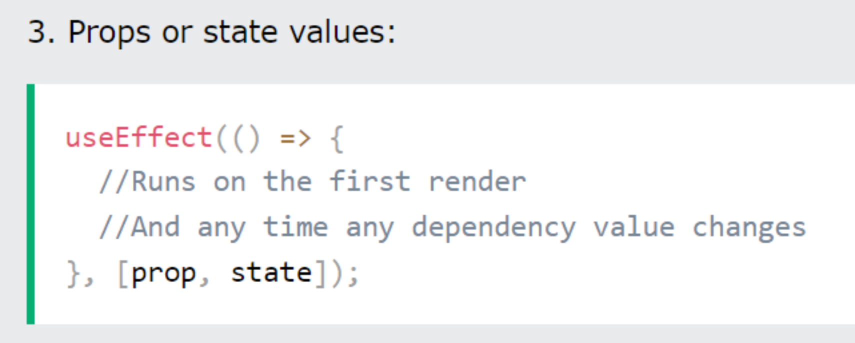
1. No dependency passed:

```
useEffect(() => {
  //Runs on every render
});
```



2. An empty array:

```
useEffect(() => {
  //Runs only on the first render
}, []);
```



3. Props or state values:

```
useEffect(() => {
  //Runs on the first render
  //And any time any dependency value changes
}, [prop, state]);
```

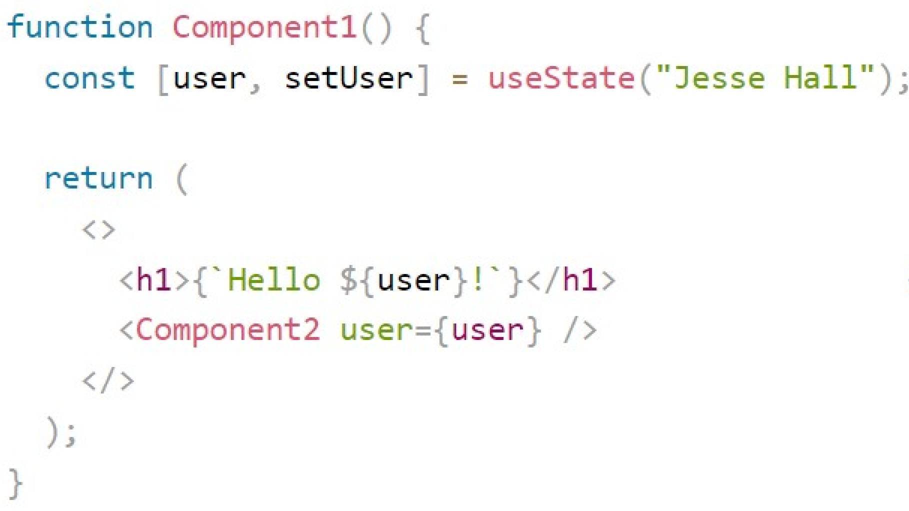
React useContext Hooks

- React Context is a way to manage state globally.
- It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone。

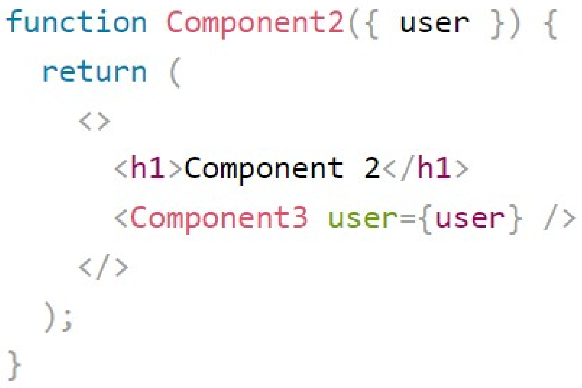
Problem

  - State should be held by the highest parent component in the stack that requires access to the state. To illustrate, we have many nested components. The component at the top and bottom of the stack need access to the state.
  - To do this without Context, we will need to pass the state as "props" through each nested component. This is called "**prop drilling**".

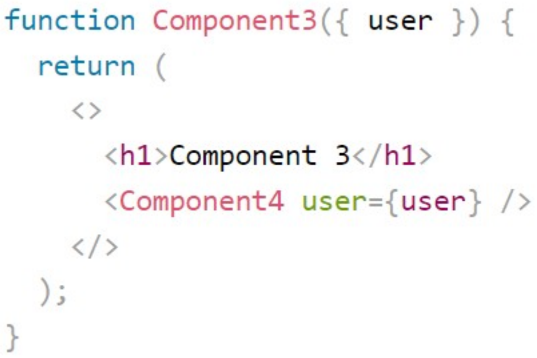
Problem （ prop drilling ）



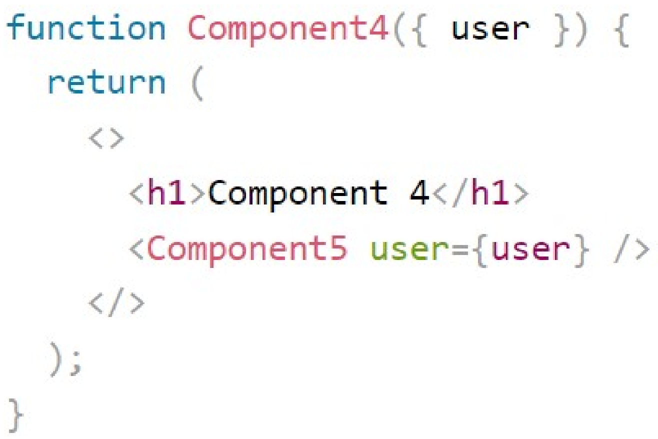
```
function Component1() {
  const [user, setUser] = useState("Jesse Hall");

  return (
    <>
      <h1>{`Hello ${user}!`}</h1>
      <Component2 user={user} />
    </>
  );
}
```



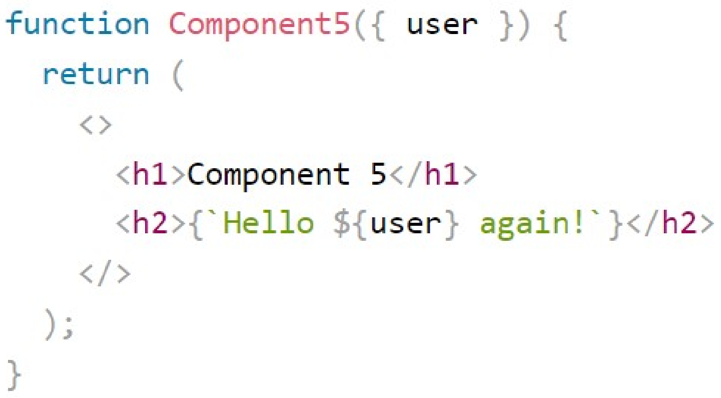
```
function Component2({ user }) {
  return (
    <>
      <h1>Component 2</h1>
      <Component3 user={user} />
    </>
  );
}
```



```
function Component3({ user }) {
  return (
    <>
      <h1>Component 3</h1>
      <Component4 user={user} />
    </>
  );
}
```



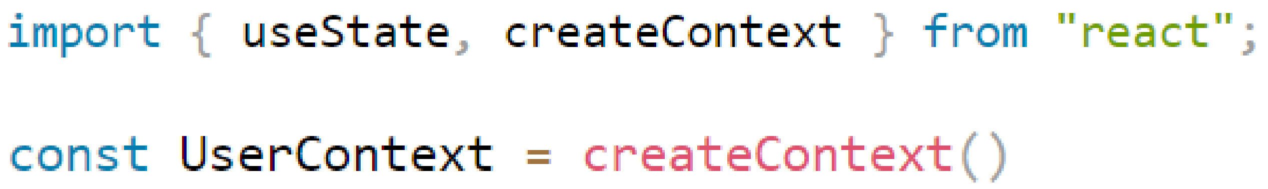
```
function Component4({ user }) {
  return (
    <>
      <h1>Component 4</h1>
      <Component5 user={user} />
    </>
  );
}
```



```
function Component5({ user }) {
  return (
    <>
      <h1>Component 5</h1>
      <h2>{`Hello ${user} again!`}</h2>
    </>
  );
}
```

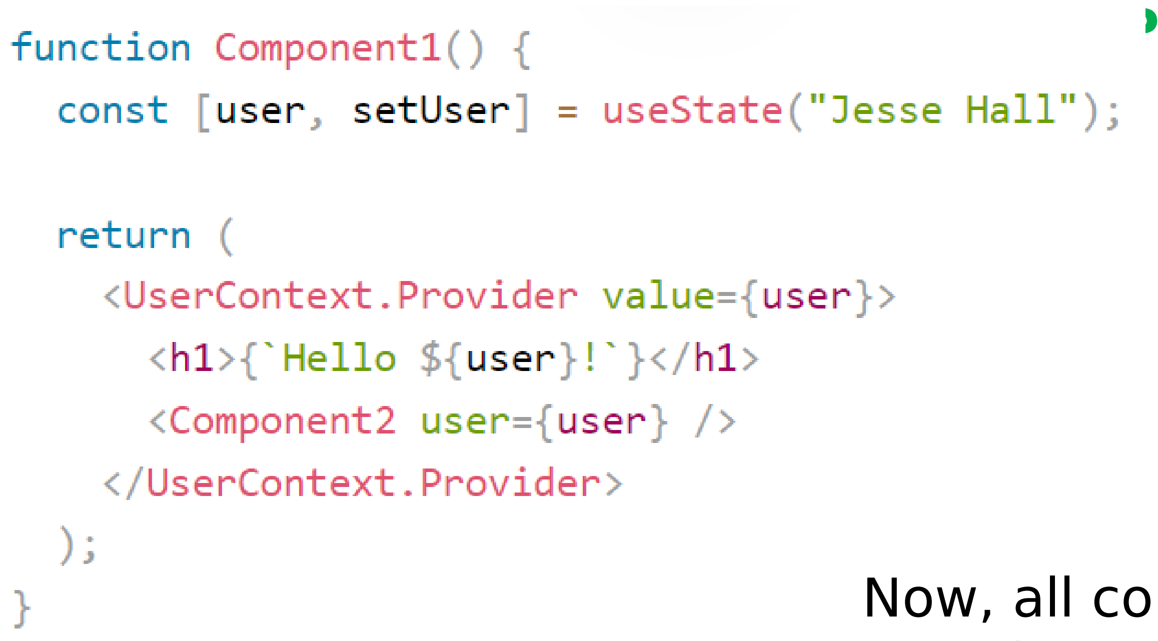
Solution (React useContext)

- To create context, We must import **createContext** and initialize it:



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

const UserContext = createContext()
```

- Next use the **Context Provider** to wrap the tree of components that need the state Context.



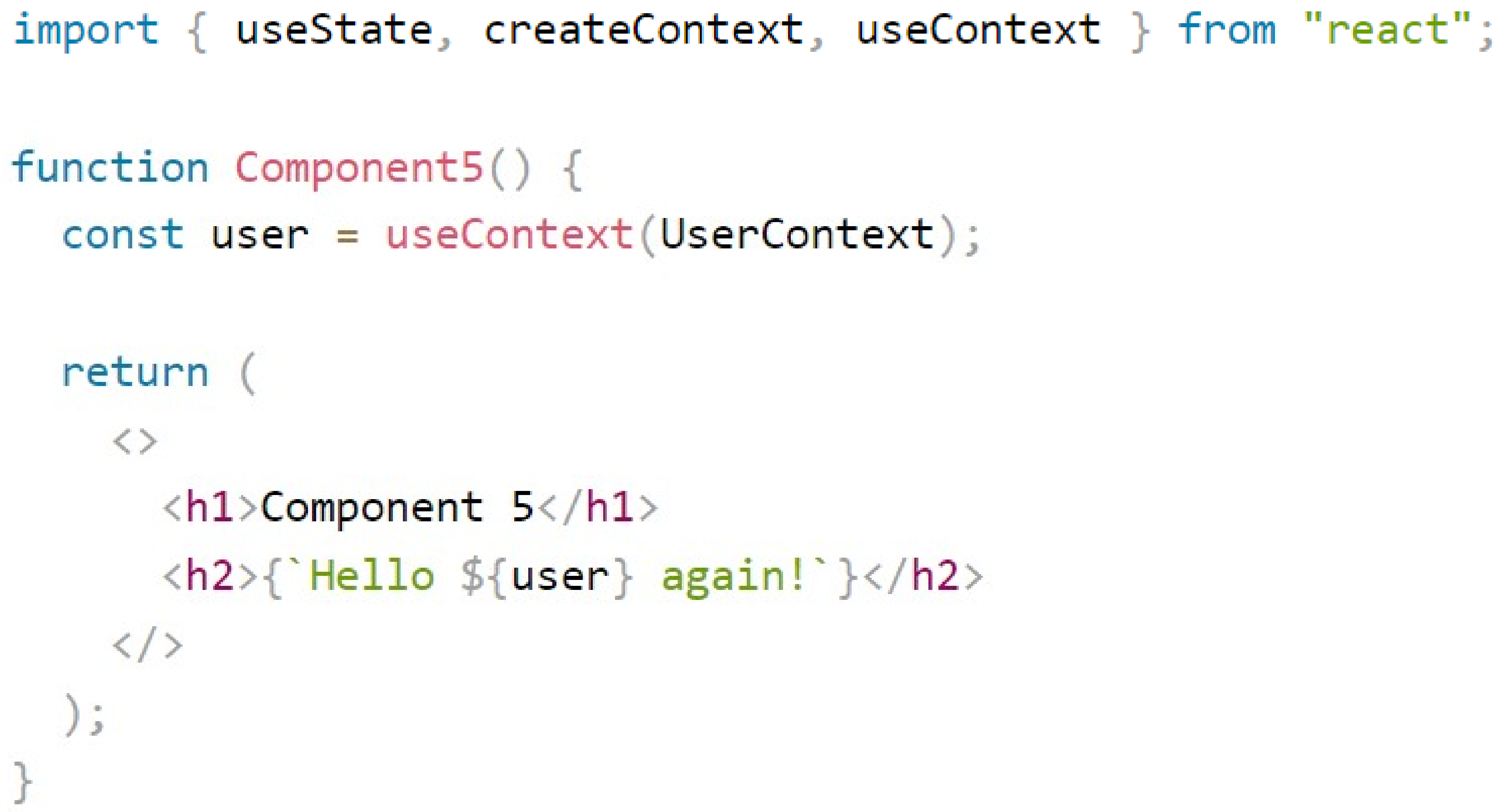
```
function Component1() {
  const [user, setUser] = useState("Jesse Hall");

  return (
    <UserContext.Provider value={user}>
      <h1>{`Hello ${user}!`}</h1>
      <Component2 user={user} />
    </UserContext.Provider>
  );
}
```

Provider and supply the **state** value.

Now, all components in this tree will have access to the user Context.

Solution (React useContext)

- ➢ Use the **useContext** Hook:
- ➢ To use the Context in a **child component**, we need to access it using the

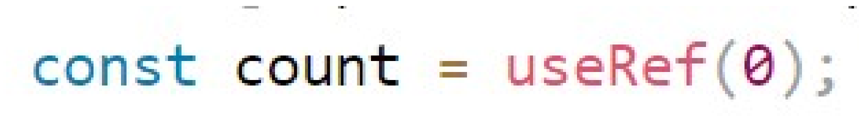


```
import { useState, createContext, useContext } from "react";

function Component5() {
  const user = useContext(UserContext);

  return (
    <>
      <h1>Component 5</h1>
      <h2>{`Hello ${user} again!`}</h2>
    </>
  );
}
```

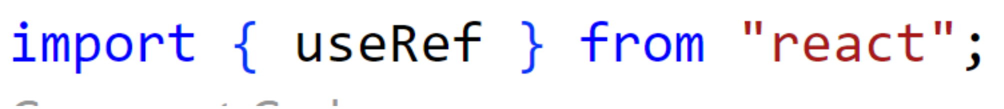
React **useRef** Hooks

- The **useState**, which creates state variables that cause re-renders when they are updated.
- **For instance:** If we tried to count how many times our application renders using the **useState Hook.**
  - we would be caught in an infinite loop since this Hook itself causes a re-render.
- The **useRef hook** creates a mutable object whose value can be changed without causing re-renders
- This makes **useRef useful** for storing values that we don't want to trigger re-renders.
- **Examples**
  - Access **DOM elements directly** within functional components and **focusing an input field**, measuring the dimensions of an element, or triggering

React useRef Hooks

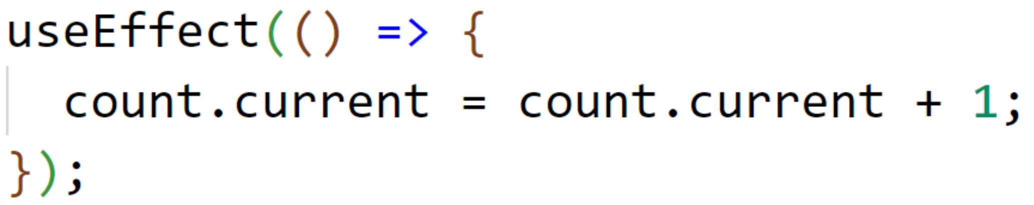


```
const count = useRef(0);
```

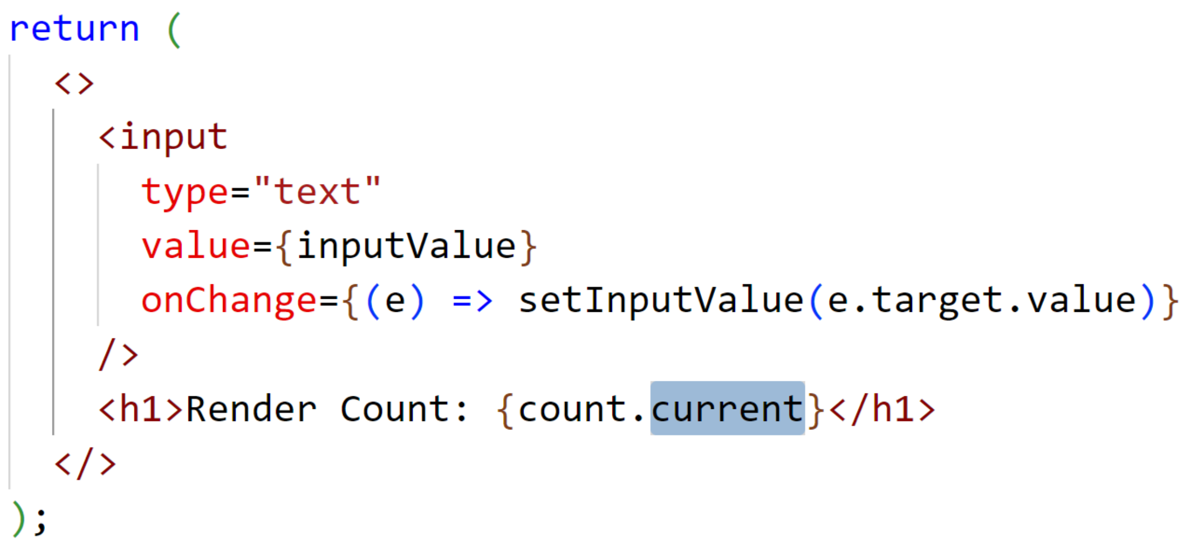


```
import { useRef } from "react";
```

- The **useRef()** only **returns one item**. It returns an **Object called current**.
- We set the initial value: **useRef(0).**
- It's like doing this: **const** count = {current: 0}. We can access the count by using count.current

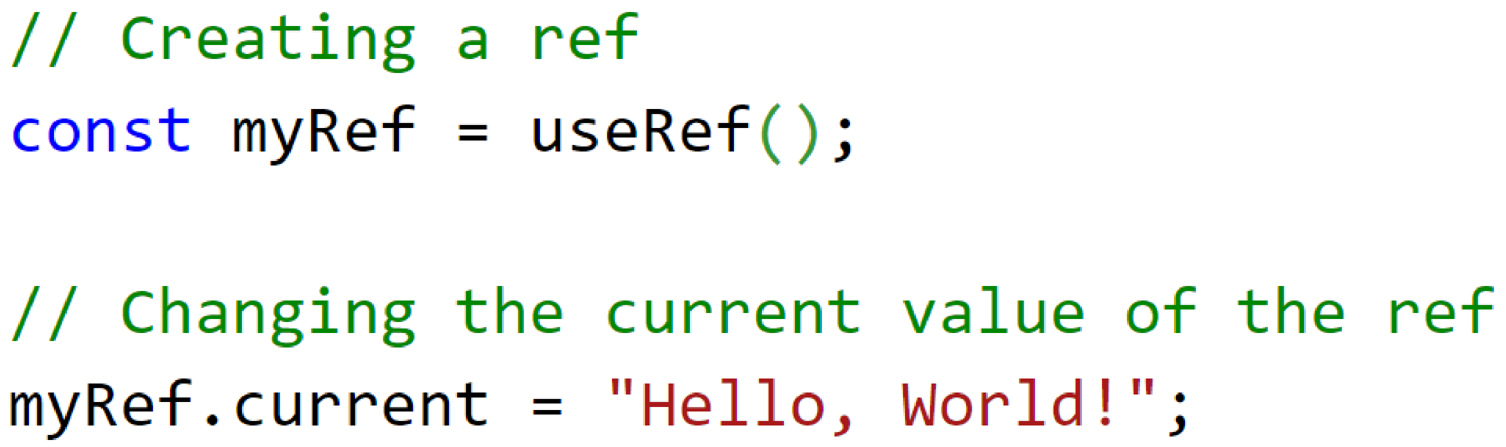


```
useEffect(() => {
  count.current = count.current + 1;
});
```



```
return (
  <>
    <input
      type="text"
      value={inputValue}
      onChange={(e) => setInputValue(e.target.value)}
    />
    <h1>Render Count: {count.current}</h1>
  </>
);
```

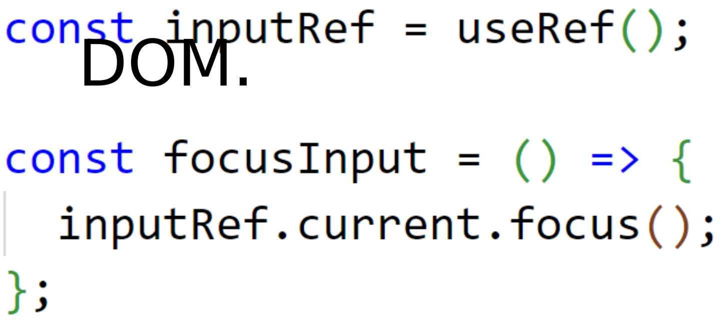
React useRef Hooks



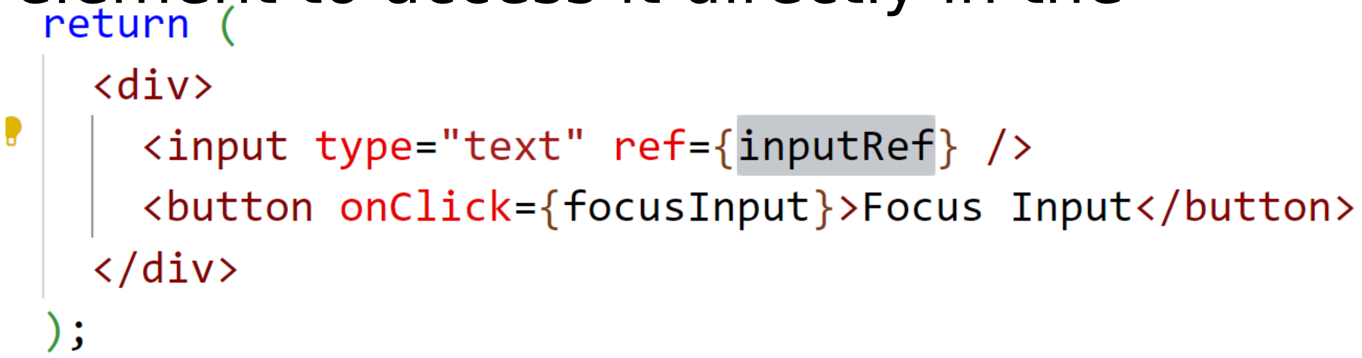
```
// Creating a ref
const myRef = useRef();

// Changing the current value of the ref
myRef.current = "Hello, World!";
```

Accessing DOM Elements in React useRef Hooks

- **useRef** can be used to access DOM elements without causing issues.
- Add a **ref** attribute to an element to access it directly in the DOM.



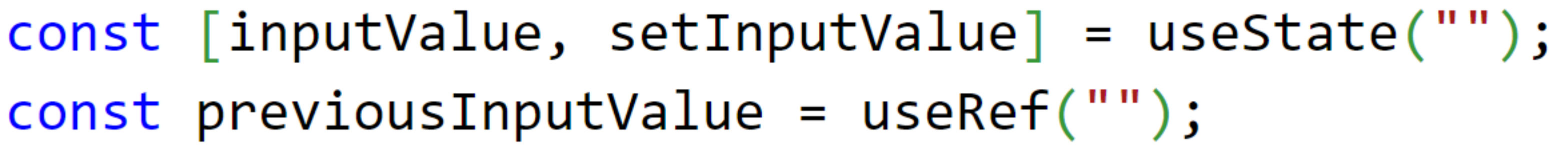
```
const inputRef = useRef();

const focusInput = () => {
  inputRef.current.focus();
};
```



```
return (
  <div>
    <input type="text" ref={inputRef} />
    <button onClick={focusInput}>Focus Input</button>
  </div>
);
```

Tracking State Changes in React useRef Hooks

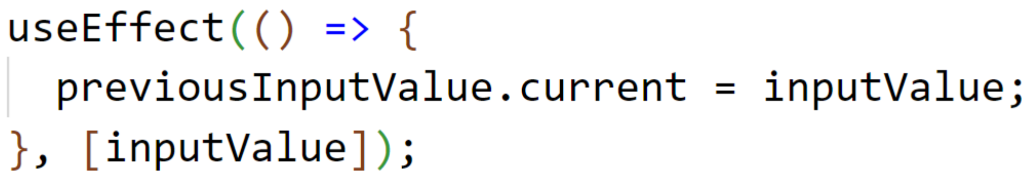
- **useRef** Hook can also be used to keep track of previous state values.



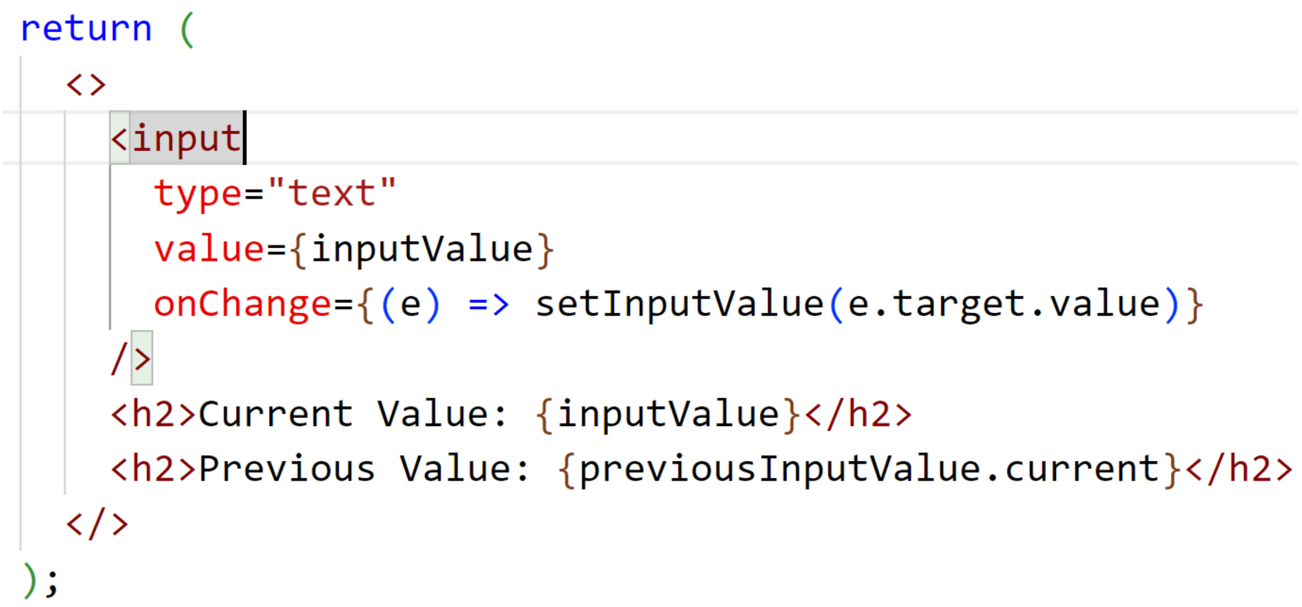
```
const [inputValue, setInputValue] = useState("");
const previousInputValue = useRef("");
```

values between

renders



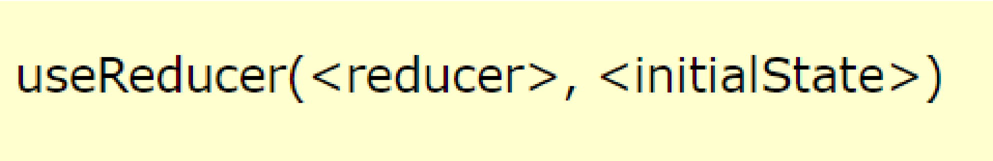
```
useEffect(() => {
  previousInputValue.current = inputValue;
}, [inputValue]);
```



```
return (
  <>
    <input
      type="text"
      value={inputValue}
      onChange={(e) => setInputValue(e.target.value)}
    />
    <h2>Current Value: {inputValue}</h2>
    <h2>Previous Value: {previousInputValue.current}</h2>
  </>
);
```

React **useReducer** Hooks

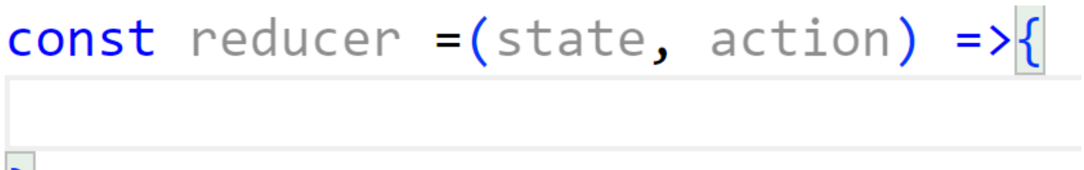
- The **useReducer** Hook is similar to the **useState** Hook.
- It's an alternative to **useState** hook, especially when the **state logic** become **complex** or involves multiple sub-values or when the next state depends on the previous one.
- **The useReducer** is a hook allows you to manage complex state logic in a more organized way.
- Syntax
  - The useReducer Hook accepts two arguments



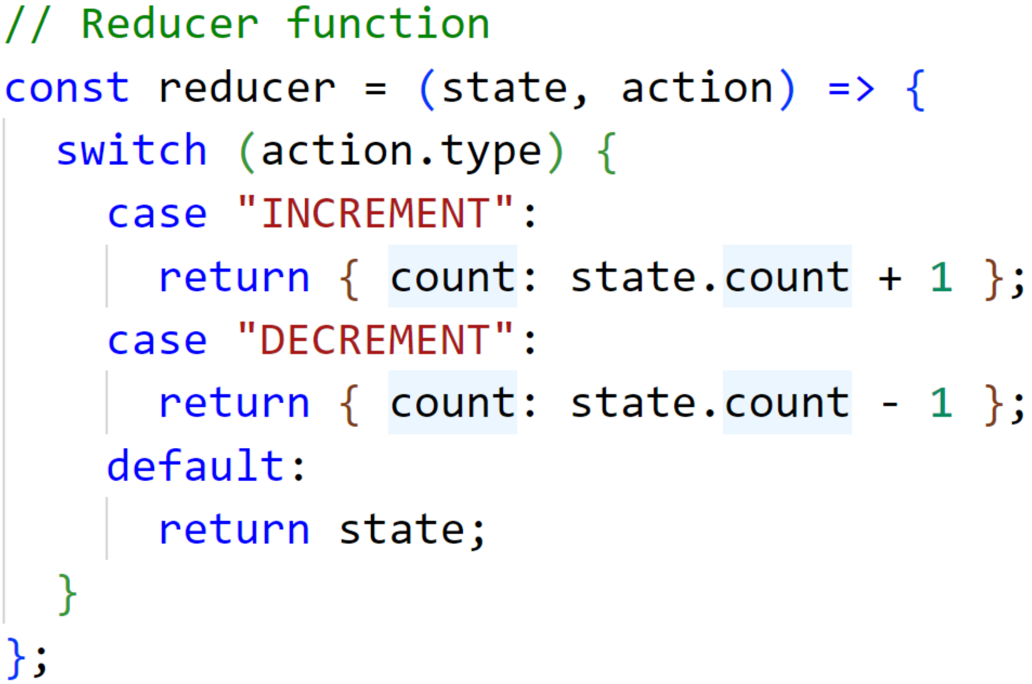
```
useReducer(<reducer>, <initialState>)
```

React **useReducer** Hooks

- With **useReducer**, We define a **reducer function** that takes the current state and an action as arguments, and returns the **new** s



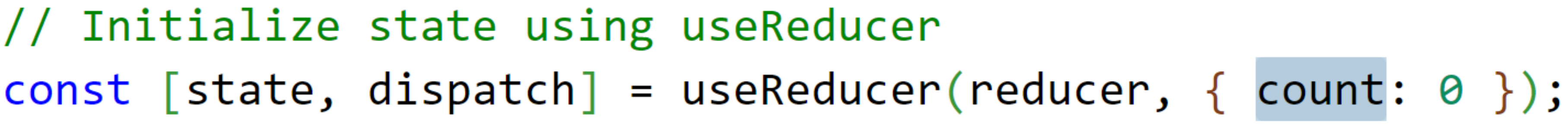
```
const reducer =(state, action) =>{

};
```



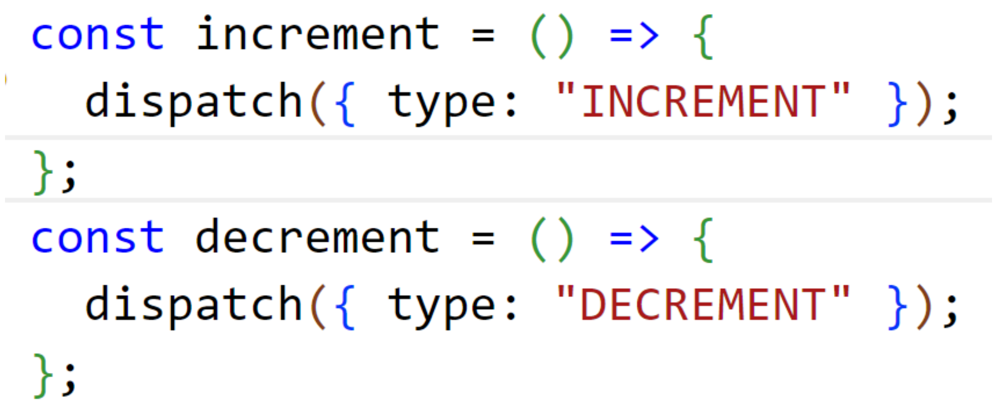
```
// Reducer function
const reducer = (state, action) => {
  switch (action.type) {
    case "INCREMENT":
      return { count: state.count + 1 };
    case "DECREMENT":
      return { count: state.count - 1 };
    default:
      return state;
  }
};
```

React useReducer Hooks

- We then use the useReducer hook by passing the **reducer function** and an initial state value.
- React will manage the state for us, and whenever we want to update the state, we dispatch an action to the reducer function.
- **The useReducer Hook returns the current state and a dispatch**



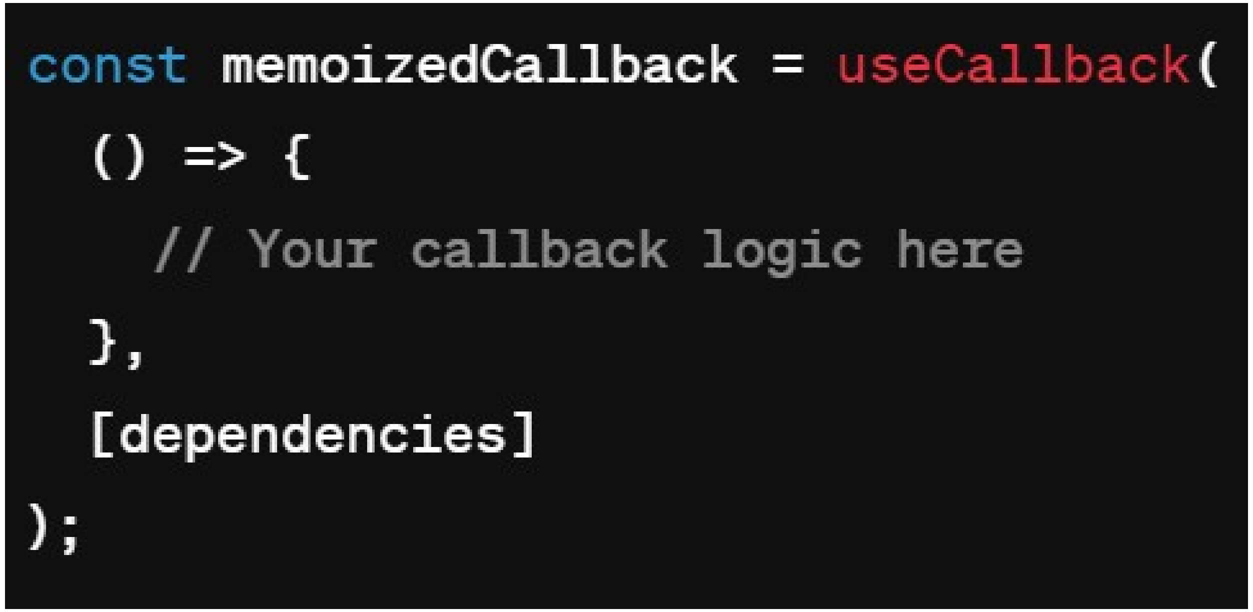
```
// Initialize state using useReducer
const [state, dispatch] = useReducer(reducer, { count: 0 });
```



```
const increment = () => {
  dispatch({ type: "INCREMENT" });
};
const decrement = () => {
  dispatch({ type: "DECREMENT" });
};
```

React useCallback Hooks

- **useCallback** Hook return a memoized function.



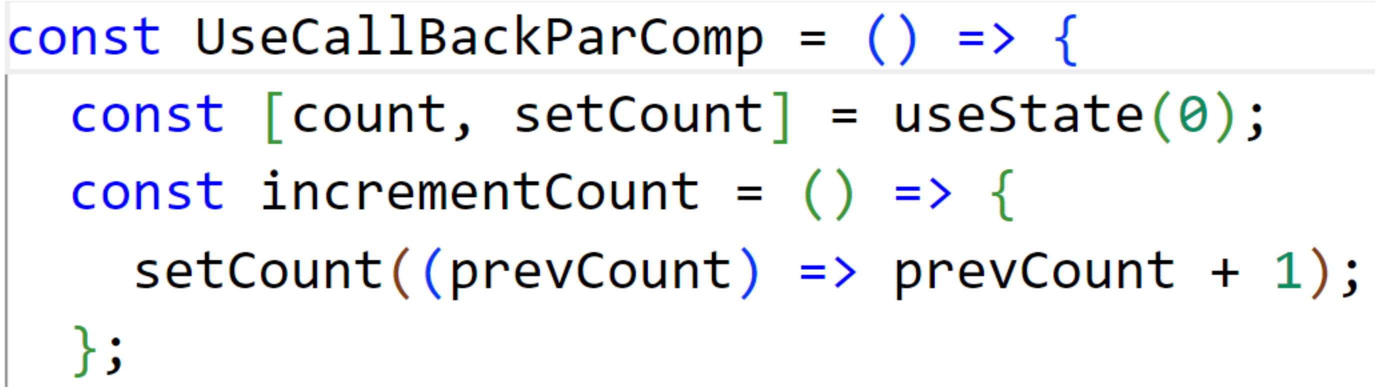
```
const memoizedCallback = useCallback(
  () => {
    // Your callback logic here
  },
  [dependencies]
);
```

- It’s preventing unnecessary re-renders of components that receive the callback as a prop.

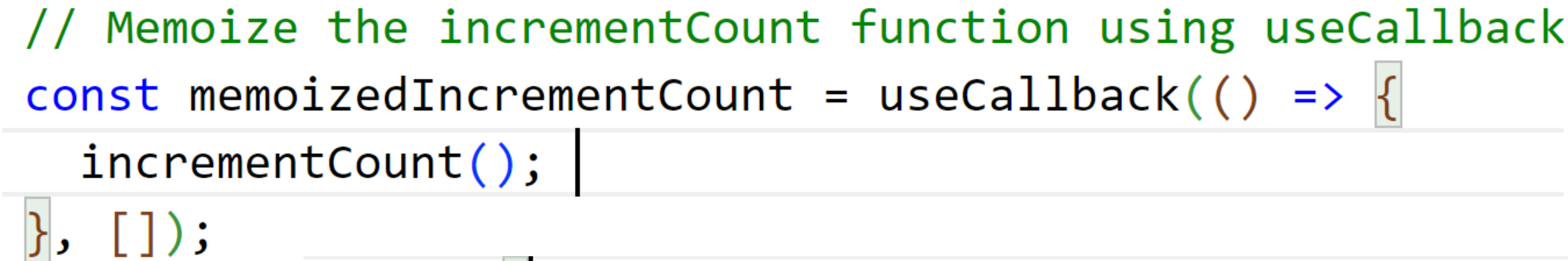
React useCallback Hooks

- **It** allows us to memoize (or cache) a function. When we use useCallback, React will return a memoized version of our function.
- This memoized function will only change if one of the dependencies passed to **useCallback** changes.
- Think of memoization as caching a value so that it does not need to be recalculated.
- Memoization is a technique used to optimize performance by storing the result of expensive function calls and returning the cached result when the same inputs occur again.
- When a function is memoized, its return value is stored in memory, and subsequent calls with the same inputs can be served from the

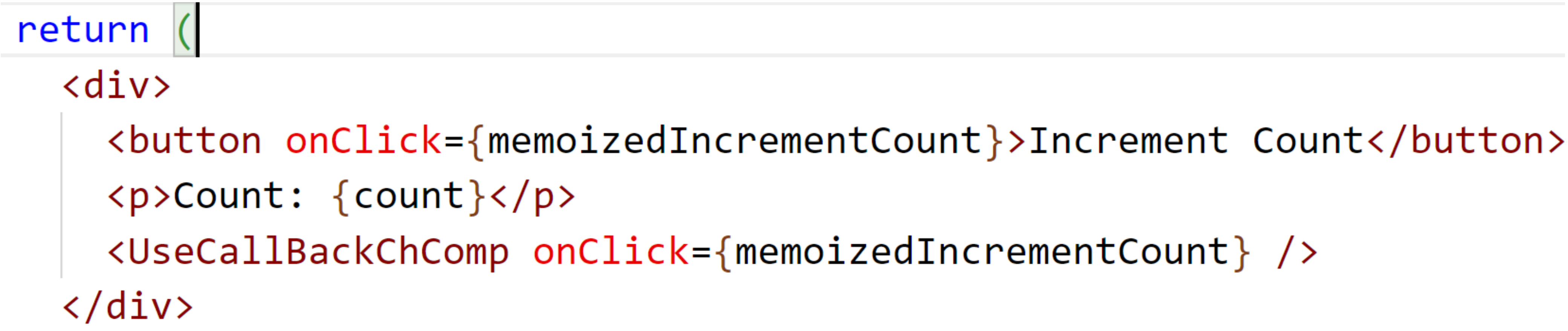
useCallback Hooks Example (Parent Component)



```
const UseCallBackParComp = () => {
  const [count, setCount] = useState(0);
  const incrementCount = () => {
    setCount((prevCount) => prevCount + 1);
  };
```

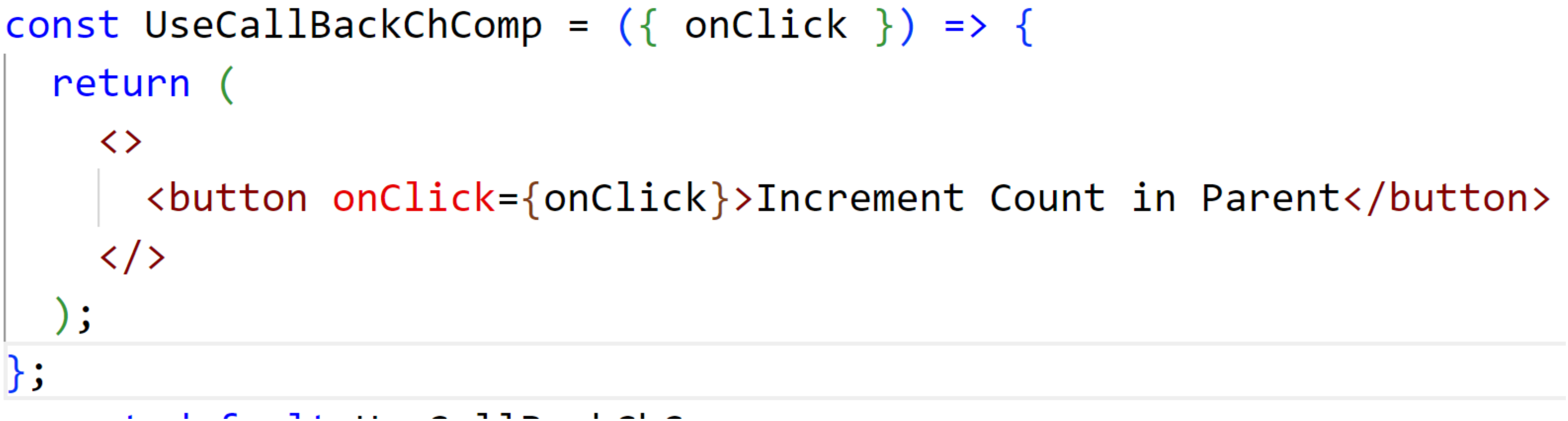


```
// Memoize the incrementCount function using useCallback
const memoizedIncrementCount = useCallback(() => {
  incrementCount();
}, []);
```



```
return (
  <div>
    <button onClick={memoizedIncrementCount}>Increment Count</button>
    <p>Count: {count}</p>
    <UseCallBackChComp onClick={memoizedIncrementCount} />
  </div>
```

useCallback Hooks Example (Child Component)



```
const UseCallBackChComp = ({ onClick }) => {
  return (
    <>
      <button onClick={onClick}>Increment Count in Parent</button>
    </>
  );
};
```

React useCallback Hooks

- **useCallback** will not automatically run on every render.
- The useCallback Hook only runs when one of its dependencies update.
- This can improve performance.
- The **useCallback** and **useMemo** Hooks are similar.
- The main difference is that **useMemo** returns a **memoized value** and **useCallback** returns a **memoized function.**

React useMemo Hooks

- **useMemo** Hook return a memoized value.
- It is used to memorize expensive computations so that they are only re-executed when their dependencies change.
- Syntax:
- Const memoizedValue = useMemo(

```
                  ()=>
              {
                  computeValue();
              },[dependencies]
  );
```

- Any Question