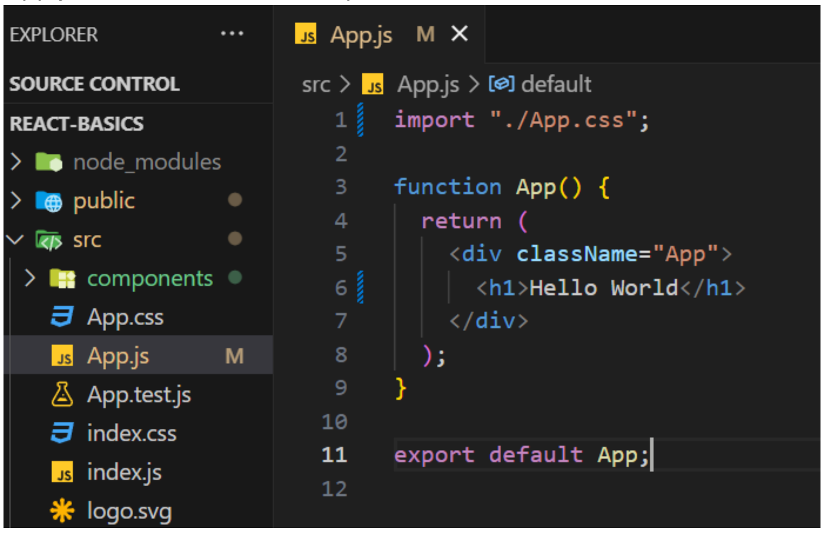
<u>REACT</u>

- ReactJS uses 3rd party libraries for routing and making http requests in react application. Axios http library used to make a rest API call
- Using create-react-app to setup react application -

```
npx create-react-app my-app
cd my-app
npm start
```

- App.js contains the root component that will be rendered to the DOM



```
import "./App.css";

function App() {
  return (
    <div className="App">
      <h1>Hello World</h1>
    </div>
  );
}

export default App;
```

O/P – Hello world on http://localhost:3000/

- DOM stands for Document Object Model it represents a tree structure of ur UI application. Whenever u load ur application in broswer, then browser will create a DOM for ur UI application.

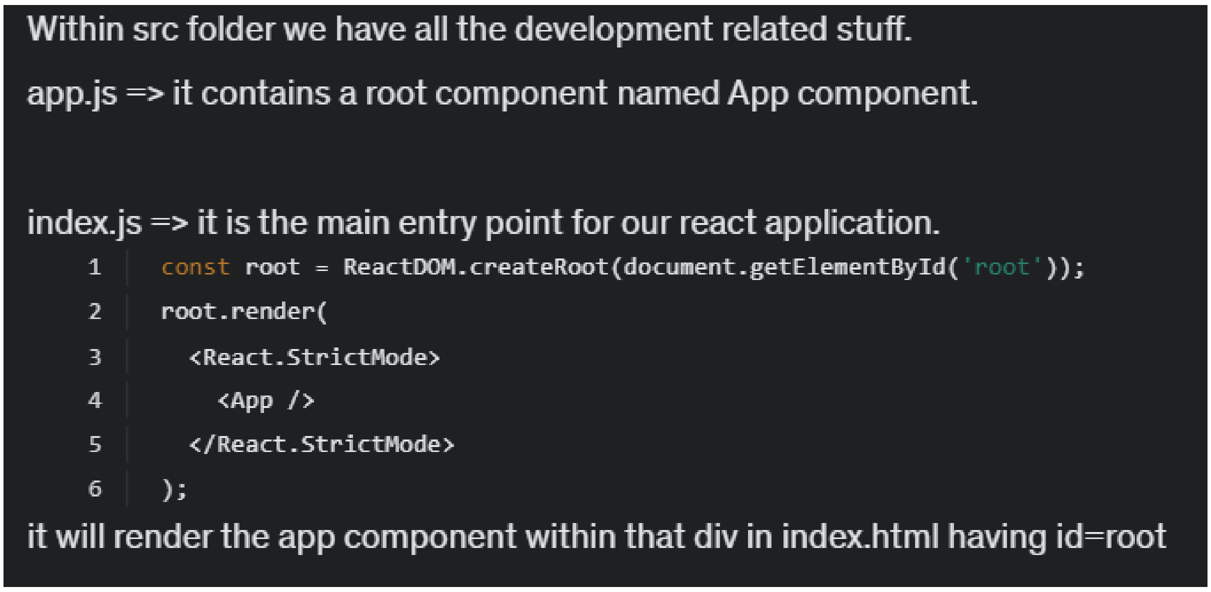
- The React Virtual DOM can be thought of as a lightweight copy of the actual DOM. Whenever the state or props change, before making updates to the real DOM, React first updates the Virtual DOM, calculating the differences between the previous and current states—a process called diffing.

  After identifying the differences, React only applies the necessary updates to the real DOM. This makes the rendering process more efficient since, instead of re-rendering the entire page, only the parts of the DOM that require changes are updated.

- **package.json** file contains all the dependencies and scripts required for out react application. Whenever we run npm install command, then npm will refer the package.json's dependency section and install these the dependencies in node_modules folder.

- In single page application there only 1 html file.
  Inside **public folder** we have **index.html** whose body should contain a single div and this div is responsible to render the react component. `<div id="root"></div>`

We may change the head section of html file to include css and script files but we don't change body section of index.html


Within src folder we have all the development related stuff.

app.js => it contains a root component named App component.

index.js => it is the main entry point for our react application.

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

it will render the app component within that div in index.html having id=root

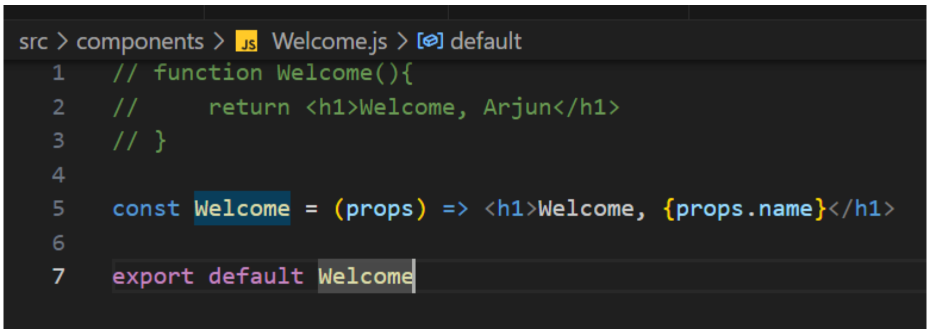
- App.css and index.css contain styling related stuffs
- **Flow of code -** Whenever we access the react application from the browser then index.html will get served in the browser. Next, the index.js file will load into the browser which will render the App component.

- App component contains JSX code which is returning some html. JSX is a syntax extension for JavaScript that allows users to write HTML-like markup in JavaScript files.

Functional Component

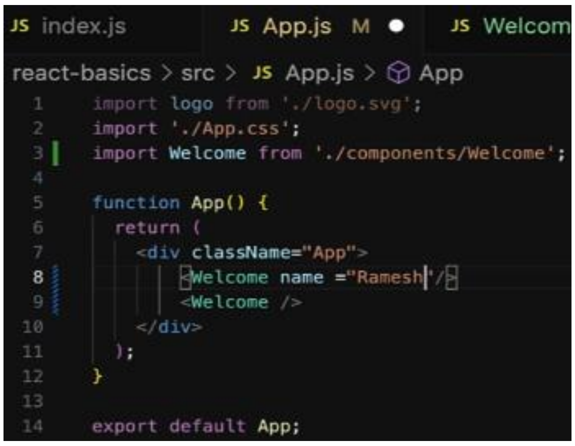
If u want to pass the data dynamically from one component to another component, then we can use props. Component file can have .js / jsx as extension.

In order to use one functional component in other components, we have to export it.



```
// function Welcome(){
//      return <h1>Welcome, Arjun</h1>
// }

const Welcome = (props) => <h1>Welcome, {props.name}</h1>

export default Welcome
```

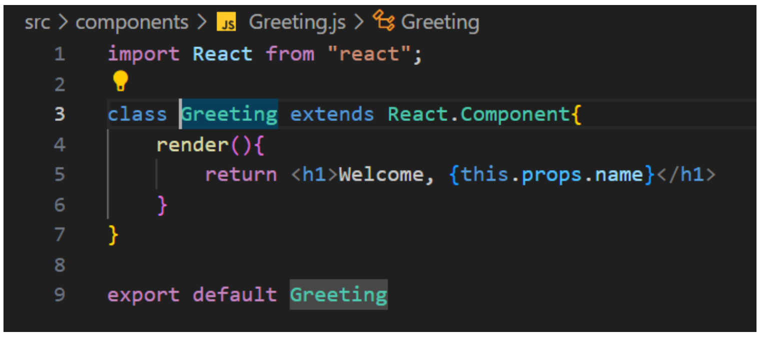
We have App component as parent and Welcome component as child component. Lets say we want to transfer the data from App component to Welcome component, then we can use **props** (properties) and we can pass the properties from parent component to child component.


JS index.js   JS App.js M ●   JS Welcom

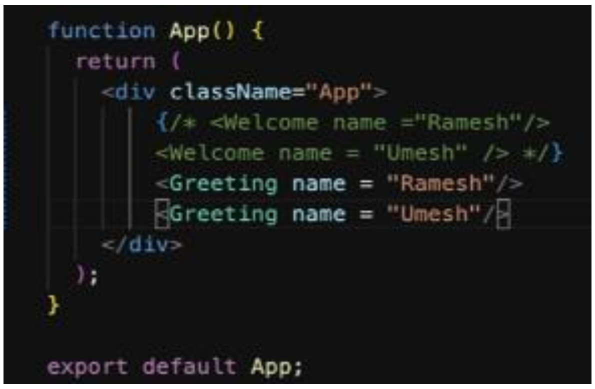
react-basics > src > JS App.js > App

```
import logo from './logo.svg';
import './App.css';
import Welcome from './components/Welcome';

function App() {
  return (
    <div className="App">
      <Welcome name ="Ramesh"/>
      <Welcome />
    </div>
  );
}

export default App;
```

<u>Class component</u> - props are automatically available in class component, we don't have to pass it to render method.

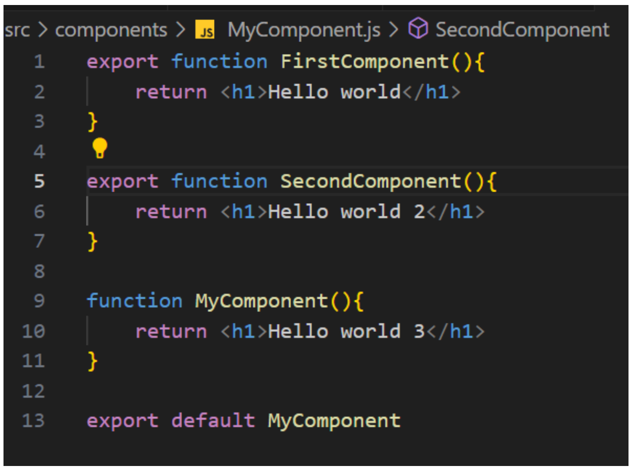

src > components > JS Greeting.js > Greeting

```
import React from "react";

class Greeting extends React.Component{
    render(){
        return <h1>Welcome, {this.props.name}</h1>
    }
}

export default Greeting
```



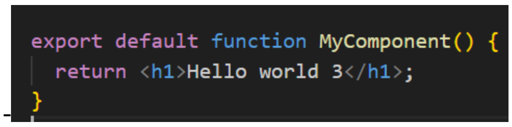
```
function App() {
  return (
    <div className="App">
      {/* <Welcome name ="Ramesh"/>
      <Welcome name = "Umesh" /> */}
      <Greeting name = "Ramesh"/>
      <Greeting name = "Umesh"/>
    </div>
  );
}

export default App;
```

Importing and Exporting Components

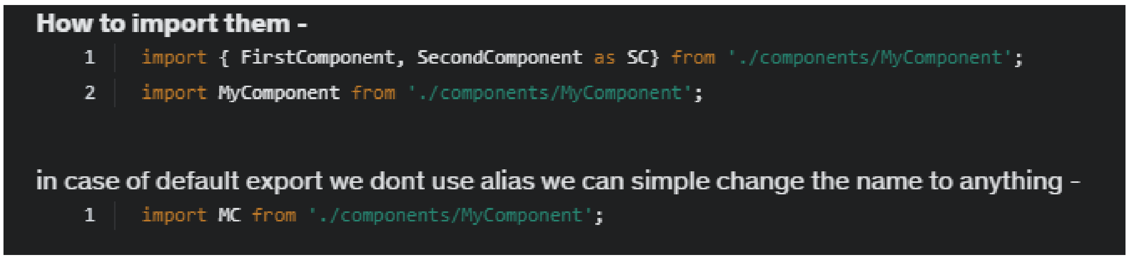

src > components > MyComponent.js > SecondComponent

```
export function FirstComponent(){
    return <h1>Hello world</h1>
}

export function SecondComponent(){
    return <h1>Hello world 2</h1>
}

function MyComponent(){
    return <h1>Hello world 3</h1>
}

export default MyComponent
```

or like this -



```
export default function MyComponent() {
  return <h1>Hello world 3</h1>;
}
```

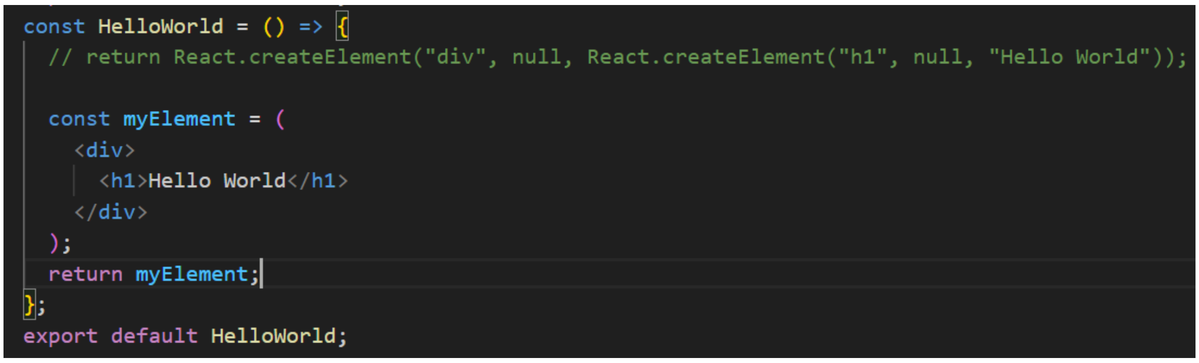

How to import them -

```
import { FirstComponent, SecondComponent as SC} from './components/MyComponent';
import MyComponent from './components/MyComponent';
```

in case of default export we dont use alias we can simple change the name to anything -

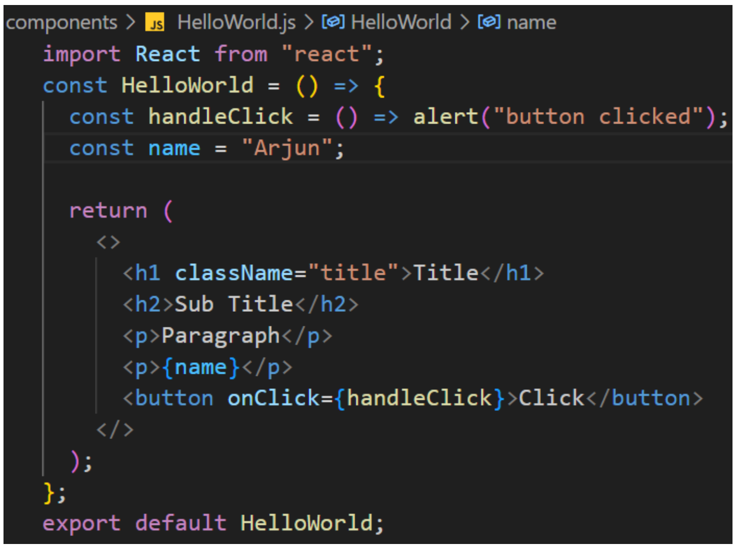
```
import MC from './components/MyComponent';
```

JSX – much easier to create element using it rather than using React.createElement() method



```
const HelloWorld = () => {
  // return React.createElement("div", null, React.createElement("h1", null, "Hello World"));

  const myElement = (
    <div>
      <h1>Hello World</h1>
    </div>
  );
  return myElement;
};
export default HelloWorld;
```

- Whenever we have multiple html elements in jsx, we have to wrap them inside an parent element like <div></div> or empty tag <> </> called react fragment.
- Use () to write html in multiple lines.

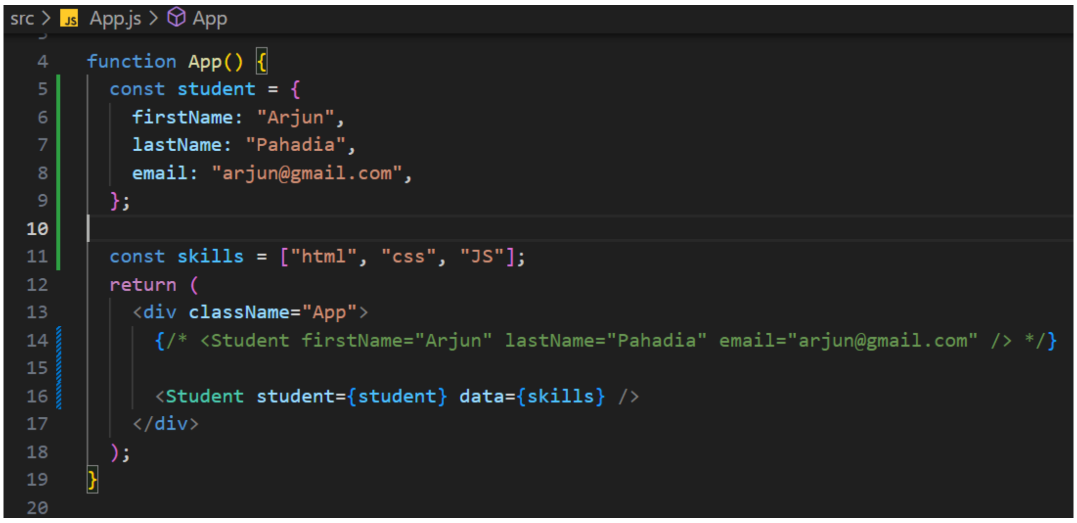


```
components > HelloWorld.js > HelloWorld > name
import React from "react";
const HelloWorld = () => {
  const handleClick = () => alert("button clicked");
  const name = "Arjun";

  return (
    <>
      <h1 className="title">Title</h1>
      <h2>Sub Title</h2>
      <p>Paragraph</p>
      <p>{name}</p>
      <button onClick={handleClick}>Click</button>
    </>
  );
};
export default HelloWorld;
```

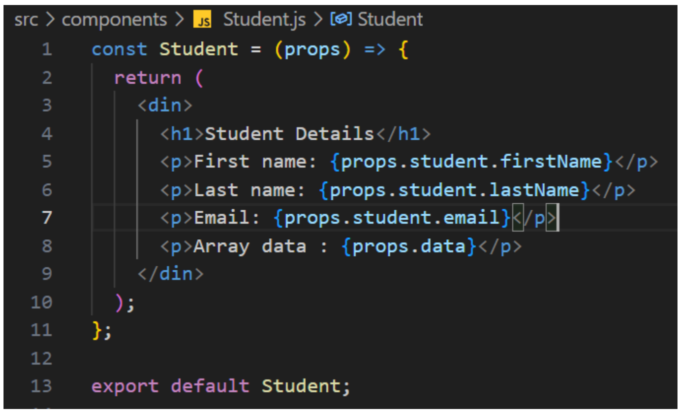
JSX Rules in React

1. Component should return single React element (div / section / article / fragment)
2. To write HTML on multiple lines, put the HTML inside parentheses
3. Use camelCase for HTML Attribute
   - To specify CSS classes for an element, use the **className** attribute instead of the **class** attribute. This is because **class** is a reserved keyword in JavaScript.
   - Event handlers in JSX are specified using camelCase syntax. For example, **onClick** instead of **onclick** or **onChange** instead of **onchange**.
4. **Expressions within Curly Braces:** You can embed JavaScript expressions within JSX elements by wrapping them in curly braces {}.
5. JSX follows XML rules, and therefore HTML elements must be properly closed. If an element doesn't have any children, you can use a self-closing tag.

Using props –

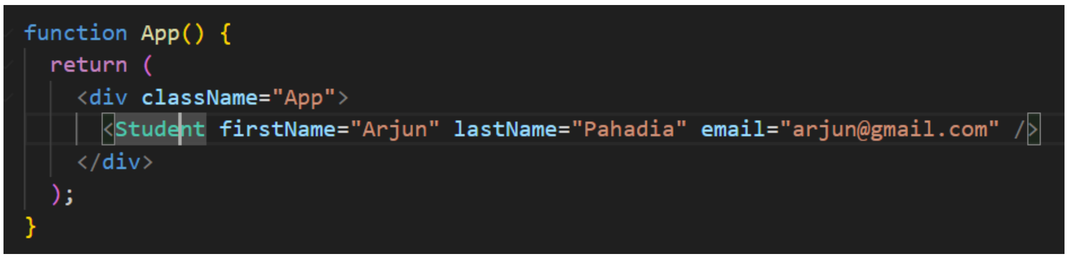


```
src > App.js > App
function App() {
  const student = {
    firstName: "Arjun",
    lastName: "Pahadia",
    email: "arjun@gmail.com",
  };

  const skills = ["html", "css", "JS"];
  return (
    <div className="App">
      {/* <Student firstName="Arjun" lastName="Pahadia" email="arjun@gmail.com" /> */}

      <Student student={student} data={skills} />
    </div>
  );
}
```

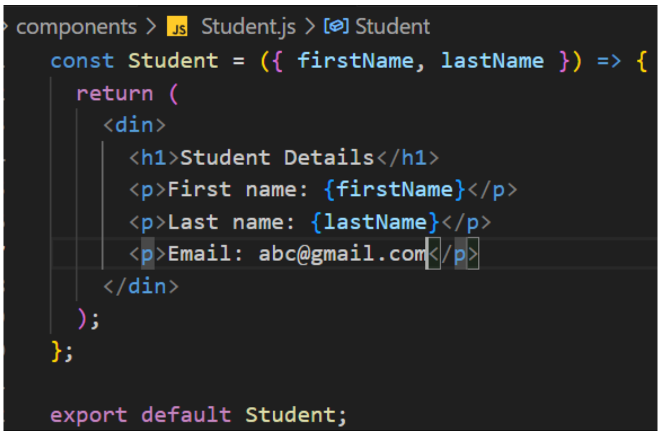


```
src > components > Student.js > Student
const Student = (props) => {
  return (
    <din>
      <h1>Student Details</h1>
      <p>First name: {props.student.firstName}</p>
      <p>Last name: {props.student.lastName}</p>
      <p>Email: {props.student.email}</p>
      <p>Array data : {props.data}</p>
    </din>
  );
};

export default Student;
```

Destructuring Props – extracting data from array or object and assign them to their own variable



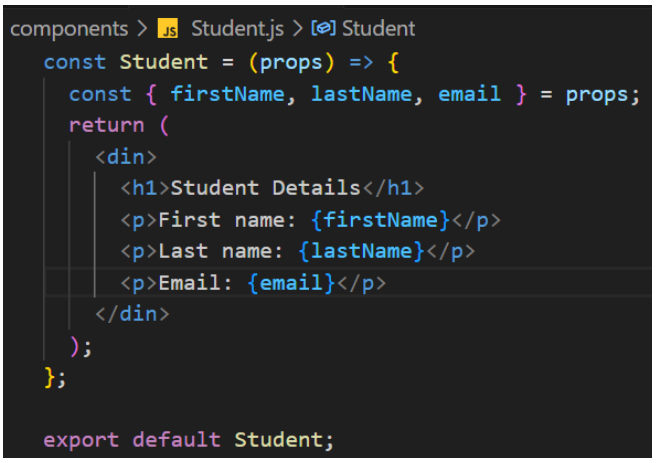
```
function App() {
  return (
    <div className="App">
      <Student firstName="Arjun" lastName="Pahadia" email="arjun@gmail.com" />
    </div>
  );
}
```



```
components > Student.js > Student
const Student = ({ firstName, lastName }) => {
  return (
    <din>
      <h1>Student Details</h1>
      <p>First name: {firstName}</p>
      <p>Last name: {lastName}</p>
      <p>Email: abc@gmail.com</p>
    </din>
  );
};

export default Student;
```

Or

Destructuring in function body -

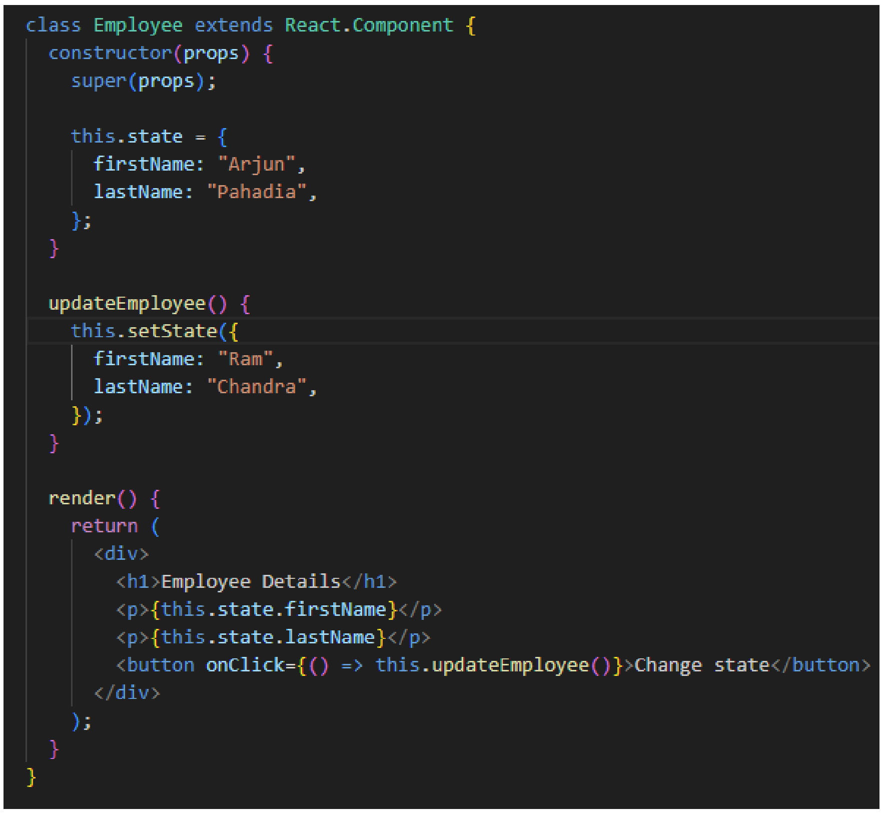


```
components > Student.js > Student
const Student = (props) => {
  const { firstName, lastName, email } = props;
  return (
    <din>
      <h1>Student Details</h1>
      <p>First name: {firstName}</p>
      <p>Last name: {lastName}</p>
      <p>Email: {email}</p>
    </din>
  );
};

export default Student;
```

<u>State and setState in Class Components</u>

In class components, use arrow function to call event handler

In constructor, we specify the state object. Within state object we can define the properties and values -

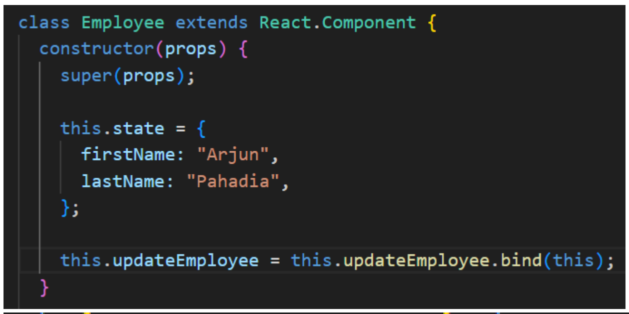


```
class Employee extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      firstName: "Arjun",
      lastName: "Pahadia",
    };
  }

  updateEmployee() {
    this.setState({
      firstName: "Ram",
      lastName: "Chandra",
    });
  }

  render() {
    return (
      <div>
        <h1>Employee Details</h1>
        <p>{this.state.firstName}</p>
        <p>{this.state.lastName}</p>
        <button onClick={() => this.updateEmployee()}>Change state</button>
      </div>
    );
  }
}
```

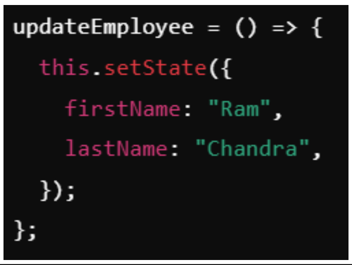
the arrow function () => this.updateEmployee() will **bind the correct context** (this) to updateEmployee, ensuring that the this inside updateEmployee refers to the Employee class instance. This is because arrow functions **lexically bind** this, meaning they inherit the this value from their surrounding context (the class instance).

In JS, the this context inside a normal function depends on how the function is called. When updateEmployee is passed as a callback (onClick={this.updateEmployee}), it is no longer called as a method of the Employee class instance. Instead, it is called in a different context, and the this inside updateEmployee will be undefined (in strict mode) or refer to the global object (in non-strict mode).

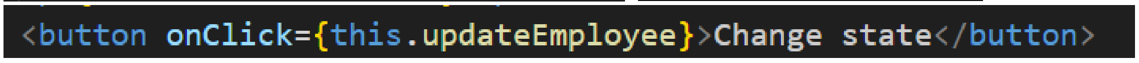
If you want pass updateEmployee directly, you need to bind it to the correct this context in the constructor or u write updateEmployee function as arrow function



```
class Employee extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      firstName: "Arjun",
      lastName: "Pahadia",
    };

    this.updateEmployee = this.updateEmployee.bind(this);
  }
```



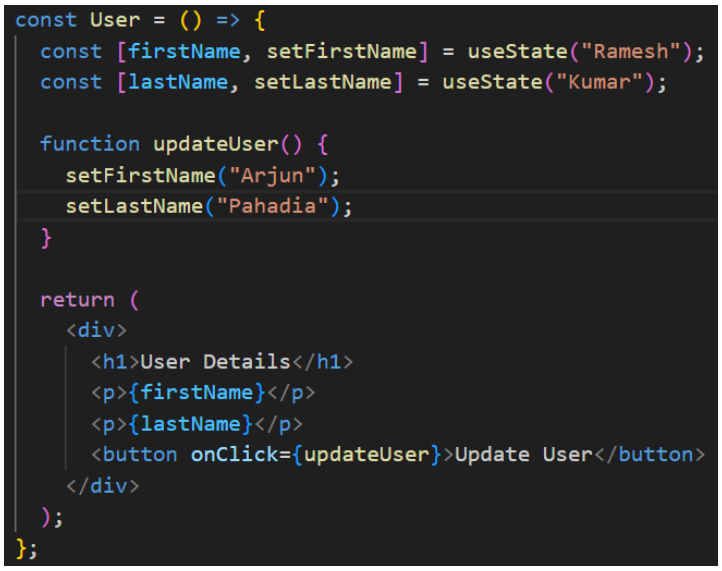
```
updateEmployee = () => {
  this.setState({
    firstName: "Ram",
    lastName: "Chandra",
  });
};
```



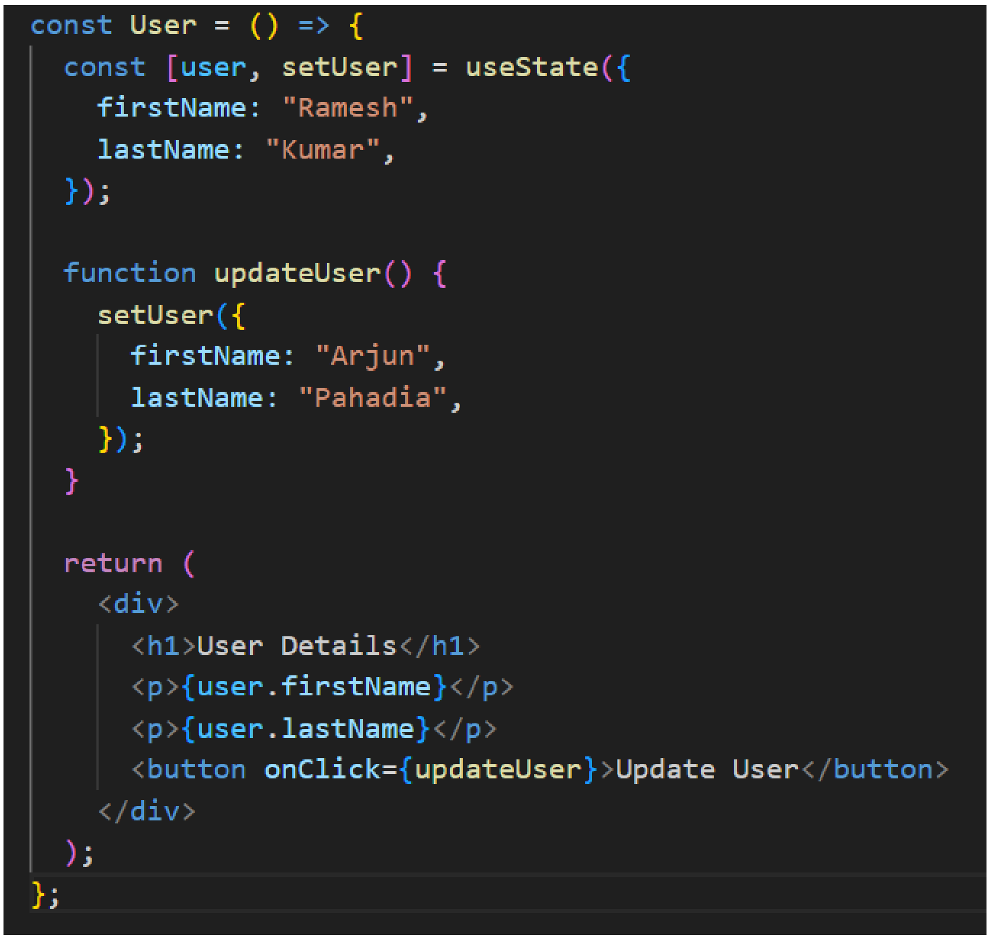
```
<button onClick={this.updateEmployee}>Change state</button>
```

useState Hook

It allows us to have state variables in funcitonal components. useState hook is itself a functional component in “react” library imported using named import. It takes one argument as initial state and returns an array which contain 2 values - current state and function that lets you update it.

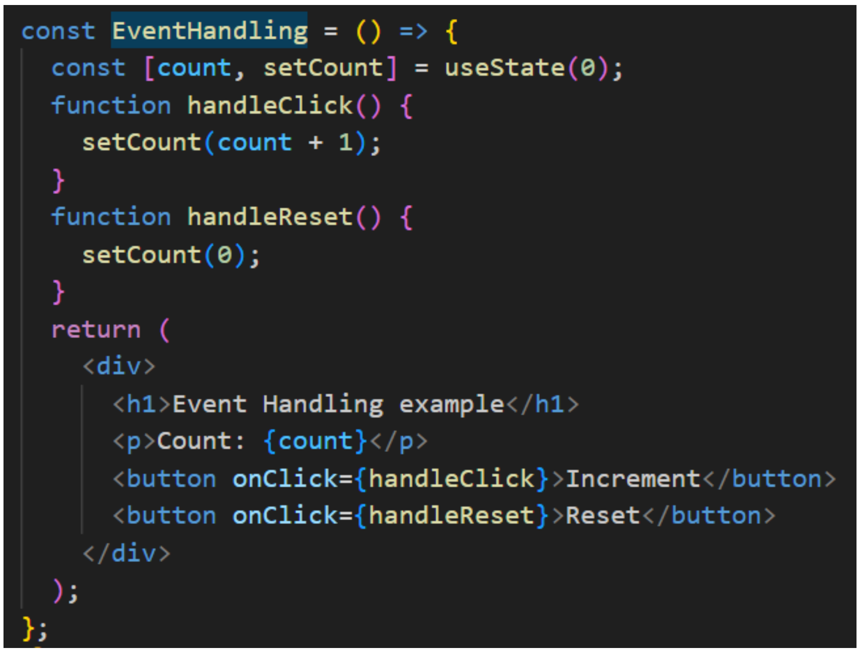


```
const User = () => {
  const [firstName, setFirstName] = useState("Ramesh");
  const [lastName, setLastName] = useState("Kumar");

  function updateUser() {
    setFirstName("Arjun");
    setLastName("Pahadia");
  }

  return (
    <div>
      <h1>User Details</h1>
      <p>{firstName}</p>
      <p>{lastName}</p>
      <button onClick={updateUser}>Update User</button>
    </div>
  );
};
```



```
const User = () => {
  const [user, setUser] = useState({
    firstName: "Ramesh",
    lastName: "Kumar",
  });

  function updateUser() {
    setUser({
      firstName: "Arjun",
      lastName: "Pahadia",
    });
  }

  return (
    <div>
      <h1>User Details</h1>
      <p>{user.firstName}</p>
      <p>{user.lastName}</p>
      <button onClick={updateUser}>Update User</button>
    </div>
  );
};
```

Event Handling



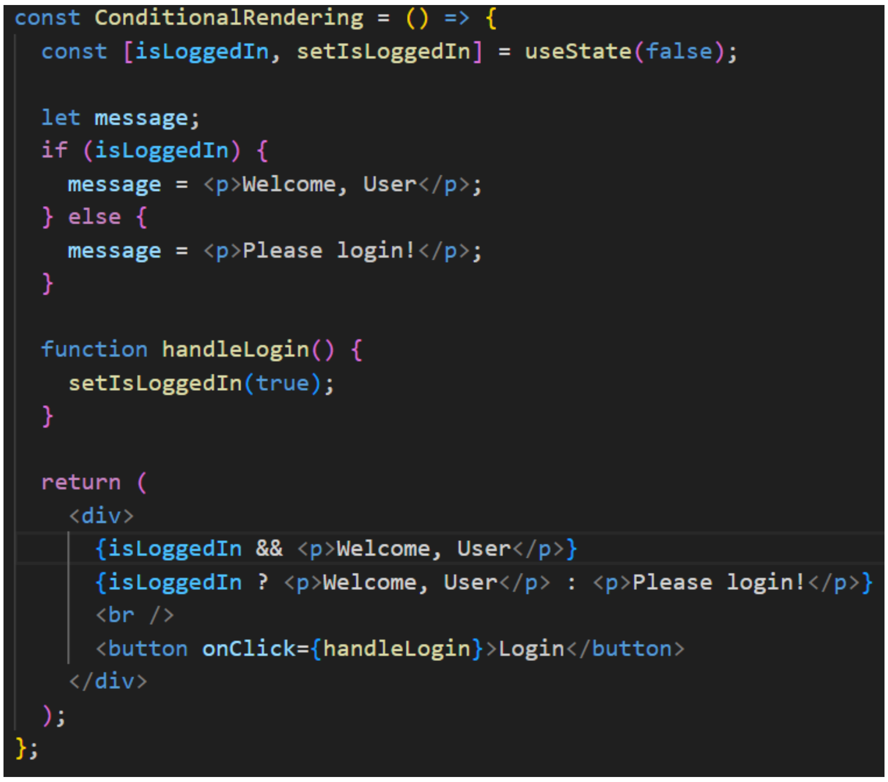
```
const EventHandling = () => {
  const [count, setCount] = useState(0);
  function handleClick() {
    setCount(count + 1);
  }
  function handleReset() {
    setCount(0);
  }
  return (
    <div>
      <h1>Event Handling example</h1>
      <p>Count: {count}</p>
      <button onClick={handleClick}>Increment</button>
      <button onClick={handleReset}>Reset</button>
    </div>
  );
};
```

Conditional Rendering

Using Short circuit operator

{ condition && <Component /> }

- If condition is **truthy**, <Component /> will be rendered.
- If condition is **falsy**, **nothing will be rendered** (the second part after && is skipped).



```
const ConditionalRendering = () => {
  const [isLoggedIn, setIsLoggedIn] = useState(false);

  let message;
  if (isLoggedIn) {
    message = <p>Welcome, User</p>;
  } else {
    message = <p>Please login!</p>;
  }

  function handleLogin() {
    setIsLoggedIn(true);
  }

  return (
    <div>
      {isLoggedIn && <p>Welcome, User</p>}
      {isLoggedIn ? <p>Welcome, User</p> : <p>Please login!</p>}
      <br />
      <button onClick={handleLogin}>Login</button>
    </div>
  );
};
```