State:

It is second way to render data on browser by using state (1.Props 2.state)

State is local to the component and can only be initialized and updated within the component.

State is similar to props, but it is private and fully controlled by the component. We can create state only in class component. It is possible to update the state/Modify the state.

There are two way to initialize state in React Component :-

- Directly inside class
- Inside the constructor

<u>Directly inside class</u>

```
class Student extends Component {

  // States  - Here it is a class property

state = {

    name: “Jay”,

    prop1: this.props.prop1

  }

  render() {      }

}
```

Note –

The state property is referred as state.

This is a class instance property.

Inside the Constructor

```
class Student extends Component {
  constructor(props) {
    // It is required to call the parent class constructor
    super(props);
    // States
    this.state = {
      name: “Jay”,
      prop1: this.props.prop1
    }
  }  render() {  }
}
```

When the component class is created, the constructor is the first method called, so it’s the right place to add state.

The class instance has already been created in memory, so you can use *this* to set properties on it.

When we write a constructor, make sure to call the parent class’ constructor by super(props)

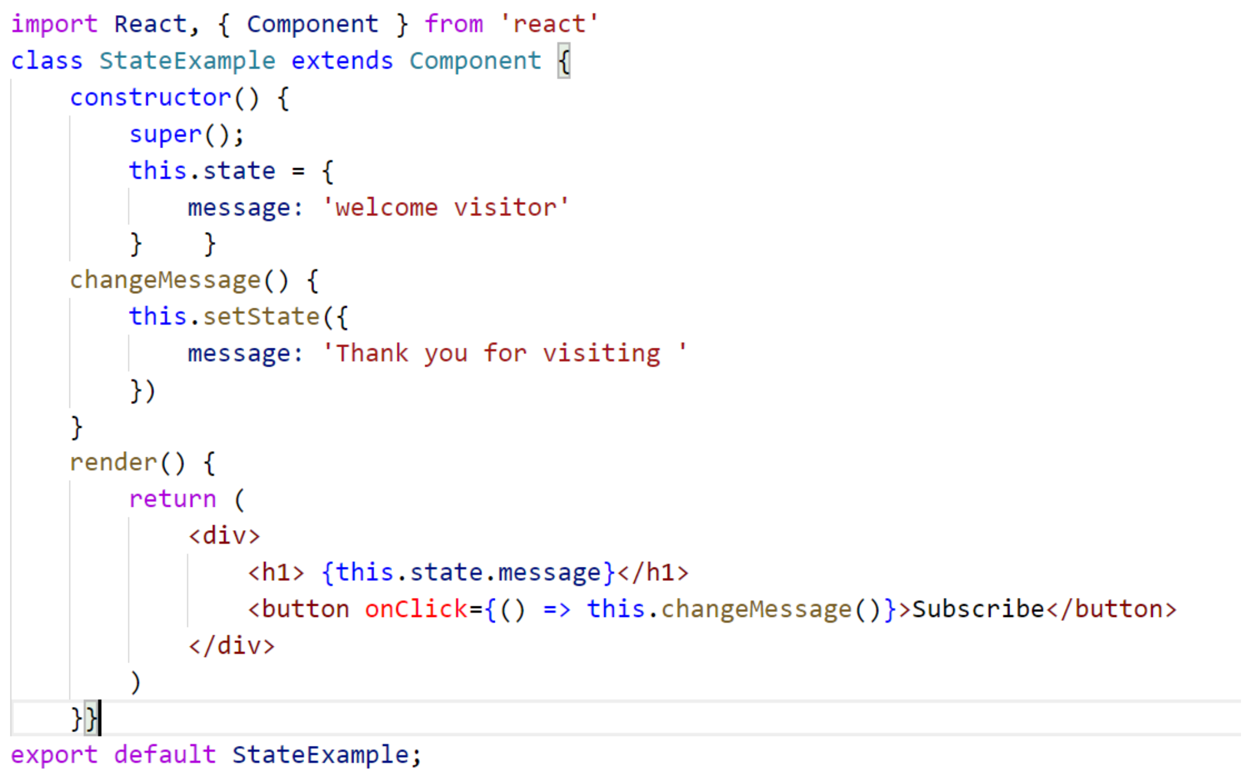
When you call super with props. React will make props available across the component through this.props

State with class component:

We can change message run time by using state not props

State is nothing but an object that is privately maintain inside a component. State can influence what is render in browser. State can change within component

stateExample.js

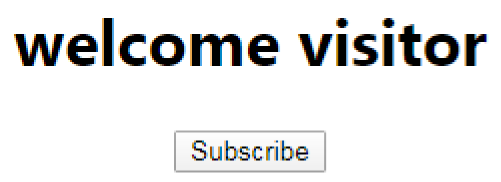


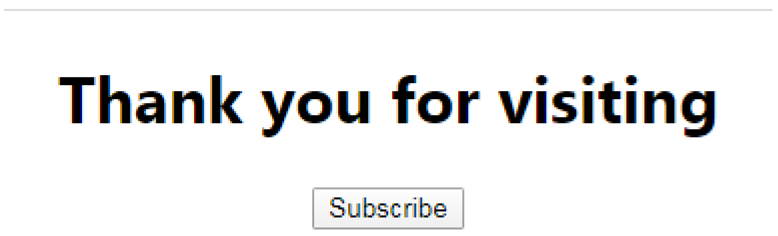
```
import React, { Component } from 'react'
class StateExample extends Component {
    constructor() {
        super();
        this.state = {
            message: 'welcome visitor'
        }    }
    changeMessage() {
        this.setState({
            message: 'Thank you for visiting '
        })
    }
    render() {
        return (
            <div>
                <h1> {this.state.message}</h1>
                <button onClick={() => this.changeMessage()}>Subscribe</button>
            </div>
        )
    }}
export default StateExample;
```

App.js

```
import React from 'react';
import logo from './logo.svg';
import './App.css';
import Greet from './components/Greet'
import Mycomponent from './components/Greet'
/* import { Greet } from './components/Greet' */
import Welcome from './components/Welcome'
import Hello from './components/Hello'
import StateExample from './components/StateExample';

class App extends React.Component {
  render() {
    return (
          <div className="App">
       <StateExample />

      </div>
    );
  }
}
export default App;
```

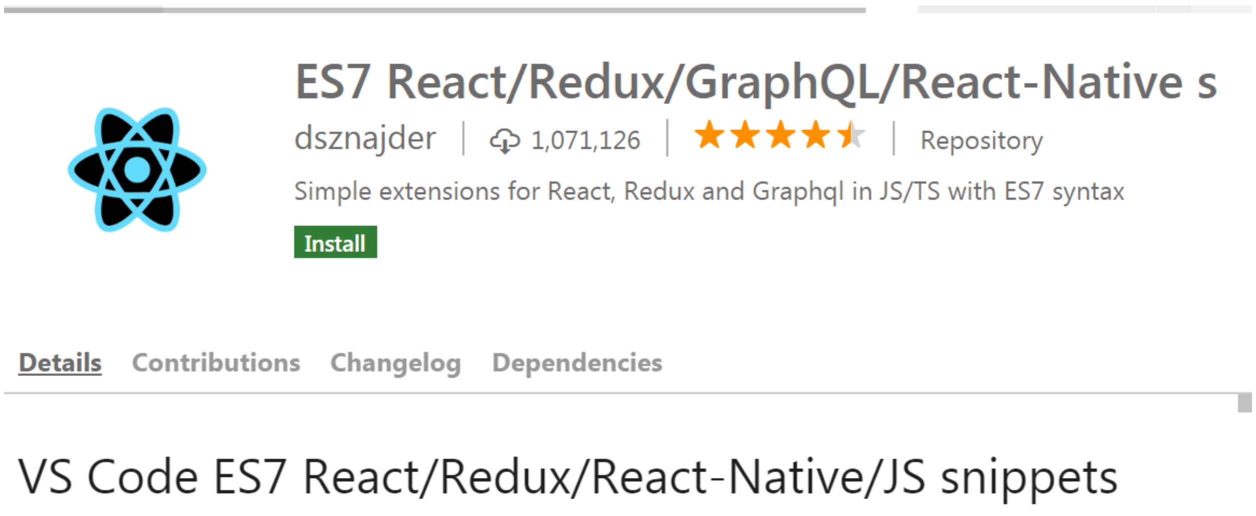
Output:


welcome visitor



Thank you for visiting

Add extension in vb



ES7 React/Redux/GraphQL/React-Native s

dsznajder | 1,071,126 | ★★★★½ | Repository

Simple extensions for React, Redux and Graphql in JS/TS with ES7 syntax

VS Code ES7 React/Redux/React-Native/JS snippets

Create file then type: - **rce** then enter- implement class definition automatically

rconst- implement constructor

SetState:-

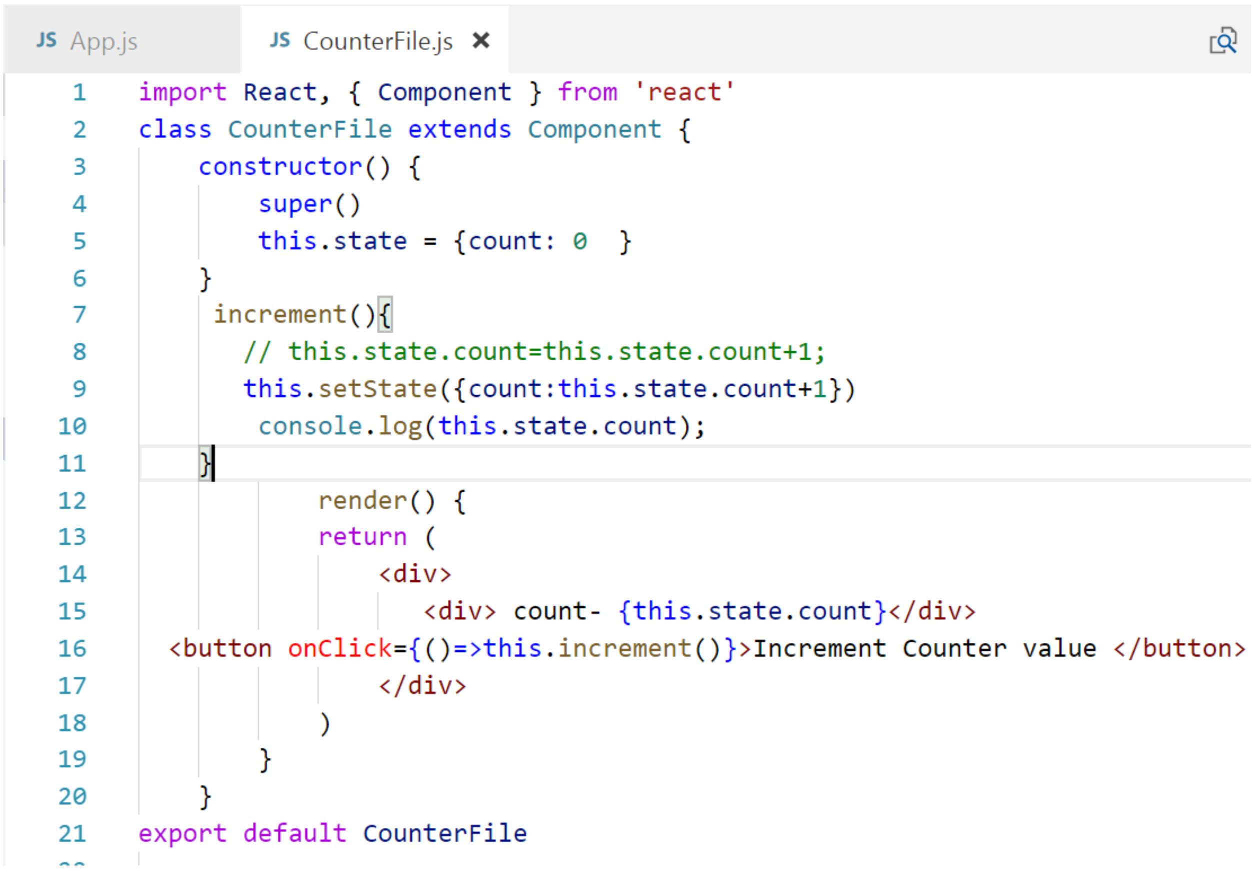
UI is not re- rendering whenever the state is changing this is main reason we should never modify state directly

We never change state directly except in constructor that is we can't. We can change state anywhere then use **setState**

State call asynchronous

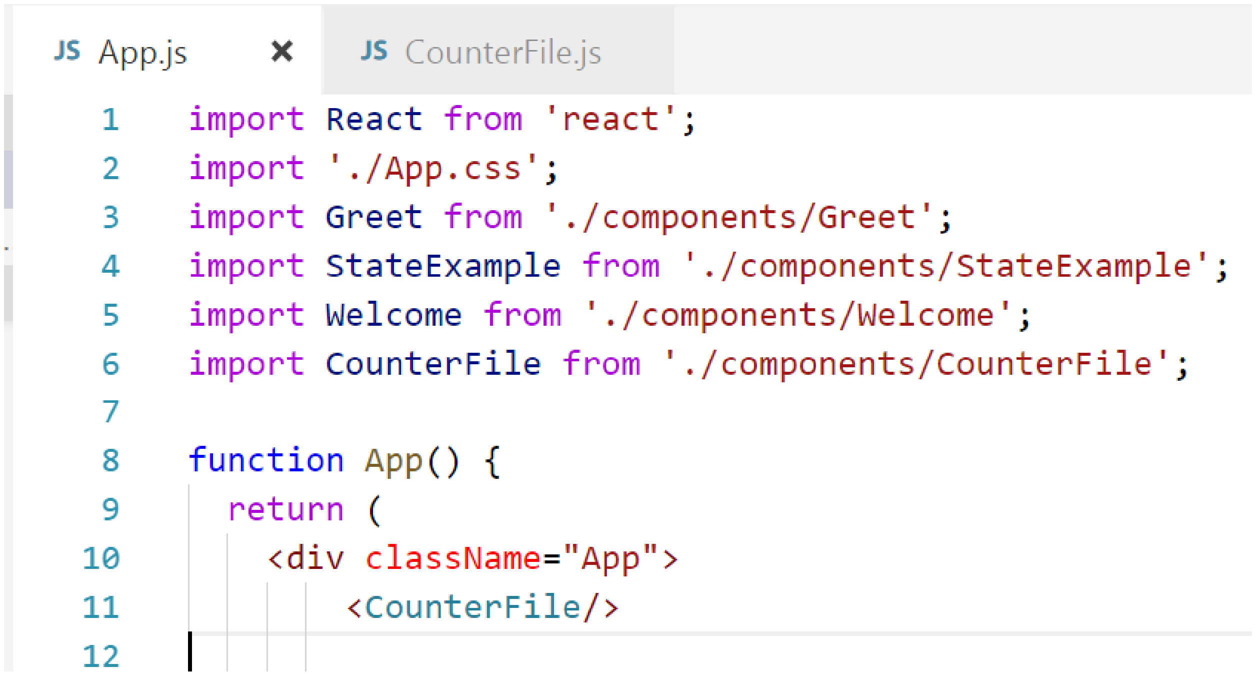
Example: - Increment counter value

CounterFile.js



```
import React, { Component } from 'react'
class CounterFile extends Component {
    constructor() {
        super()
        this.state = {count: 0  }
    }
     increment(){
       // this.state.count=this.state.count+1;
       this.setState({count:this.state.count+1})
        console.log(this.state.count);
    }
            render() {
            return (
                <div>
                   <div> count- {this.state.count}</div>
  <button onClick={()=>this.increment()}>Increment Counter value </button>
                </div>
            )
        }
    }
export default CounterFile
```

App.js



App.js CounterFile.js

```
import React from 'react';
import './App.css';
import Greet from './components/Greet';
import StateExample from './components/StateExample';
import Welcome from './components/Welcome';
import CounterFile from './components/CounterFile';

function App() {
  return (
    <div className="App">
      <CounterFile/>

```

Difference between State and Props

Props	State
Props are read-only.	State changes can be asynchronous.
Props are immutable	State is mutable.
Props allow you to pass data from one component to other components as an argument.	State holds information about the components.
Props can be accessed by the child component.	State cannot be accessed by child components.
Props are used to communicate between components.	States can be used for rendering dynamic changes with the component
Stateless component can have Props.	Stateless components cannot have State.
Props make components reusable.	State cannot make components reusable.
Props are external and controlled by whatever renders the component.	The State is internal and controlled by the React Component itself.

Destructuring props and state:-

Destructuring is ES6 Feature that is making possible to unpack values from arrays or properties from objects into distinct variables. In react Destructuring props and state improve code readability.

Destructuring was introduced in ES6. It's a JavaScript feature that allows us to extract multiple pieces of data from an array or object and assign them to their own variables.

Destructuring is a convenient way to extract multiple keys from an object or array simultaneously and assign the values to local variables.

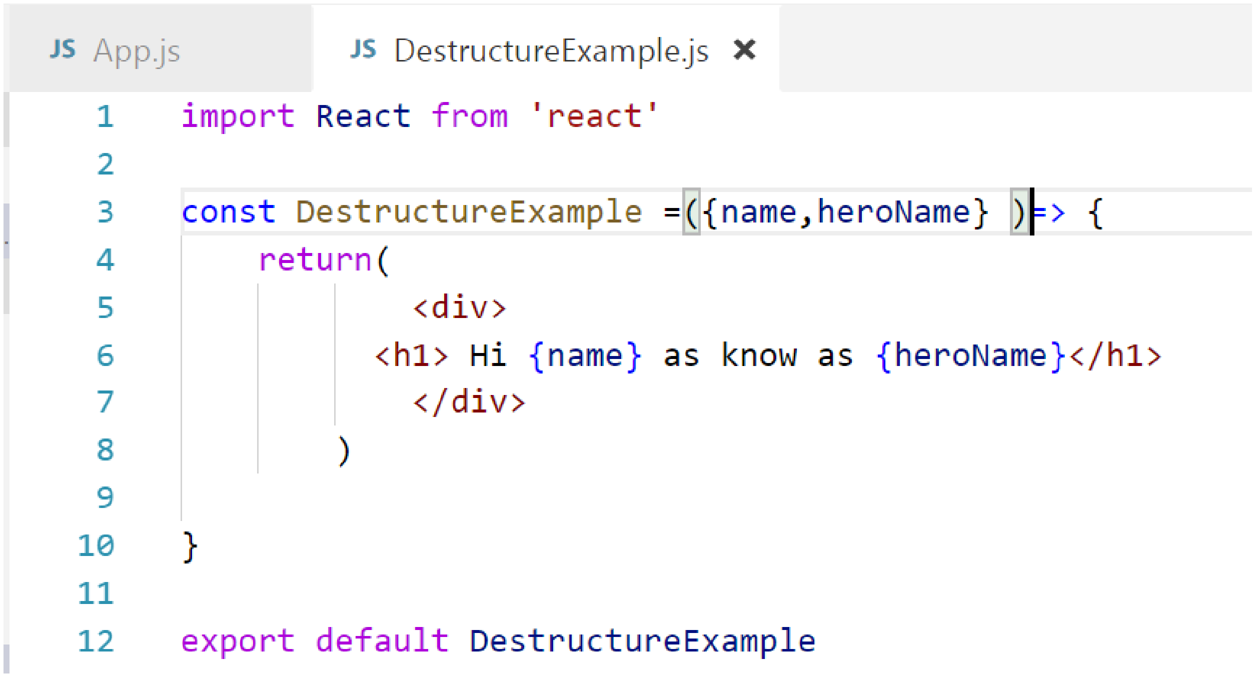
1. Destructuring props in functional components

There are two way to destructure props in a functional components

1. Destructure in functional parameter itself
2. Destructure in the functional body

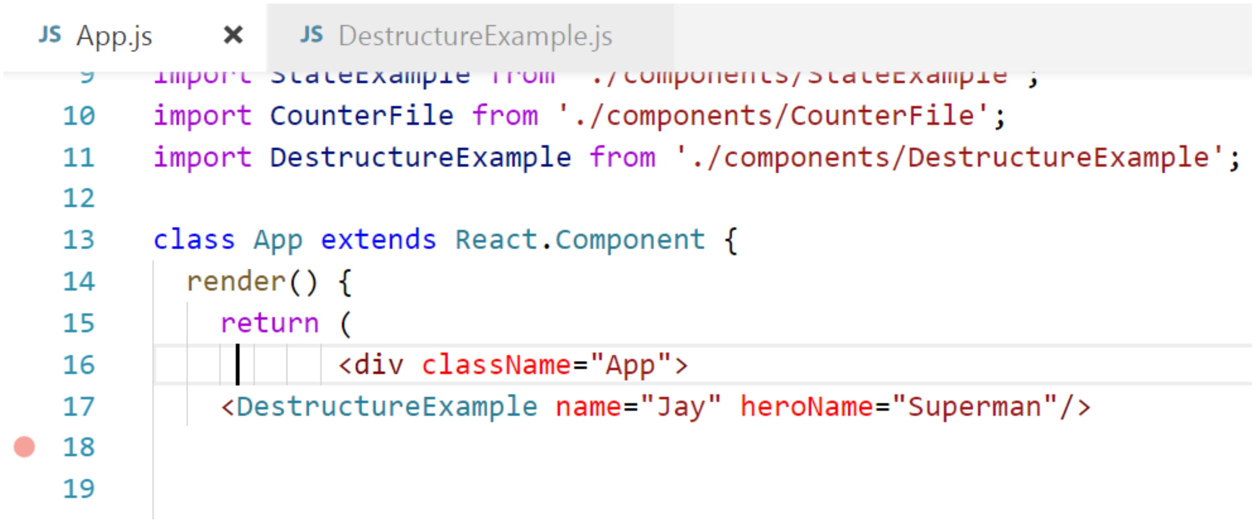
1. **Destructure in functional parameter itself**

Example:-



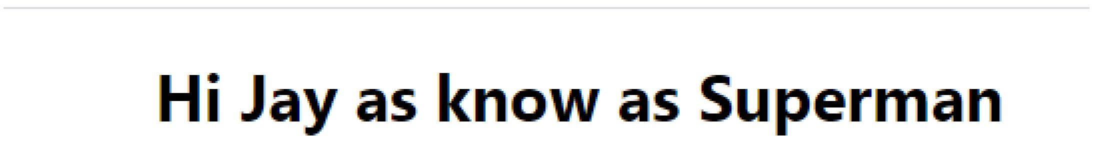
```
import React from 'react'

const DestructureExample =({name,heroName} )=> {
    return(
            <div>
        <h1> Hi {name} as know as {heroName}</h1>
            </div>
        )

}

export default DestructureExample
```

App.js



```
import StateExample from './components/StateExample';
import CounterFile from './components/CounterFile';
import DestructureExample from './components/DestructureExample';

class App extends React.Component {
  render() {
    return (
      |     <div className="App">
    <DestructureExample name="Jay" heroName="Superman"/>


```

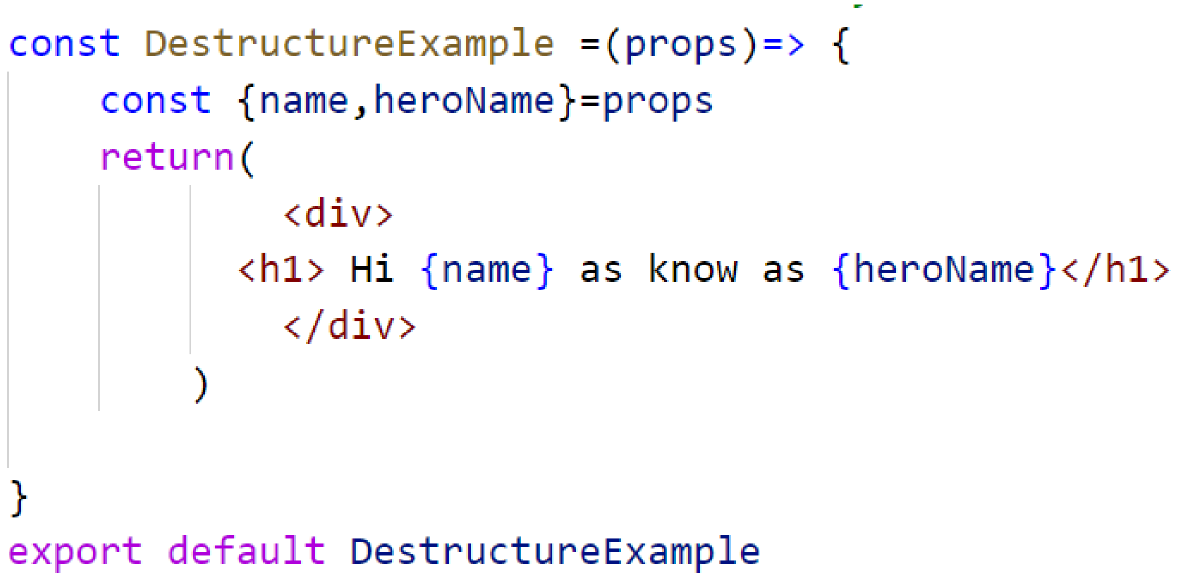
Output:



Hi Jay as know as Superman

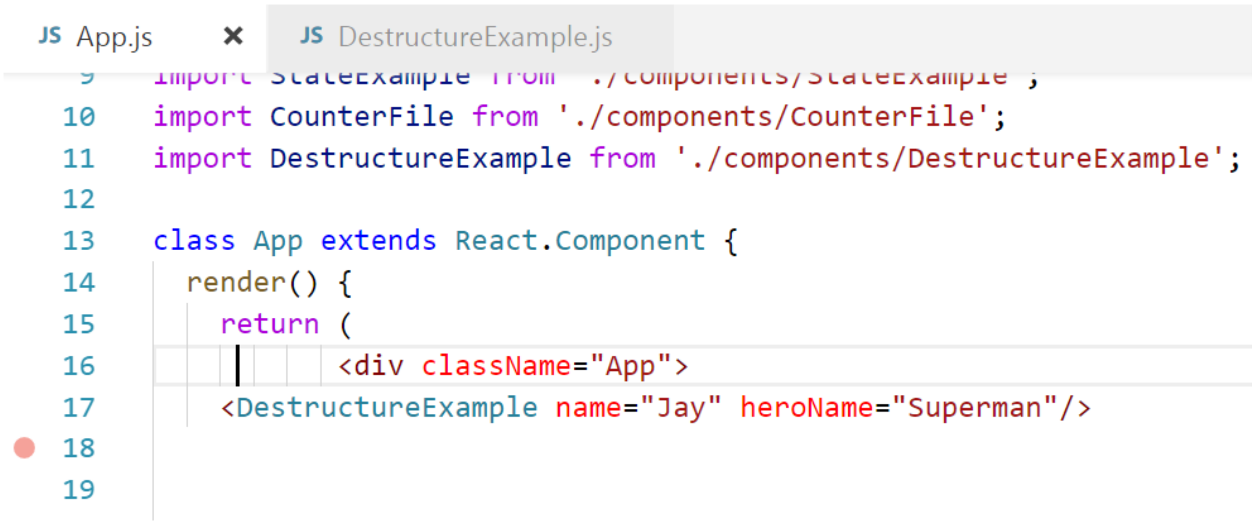
2. Destructure in the functional body

Example:



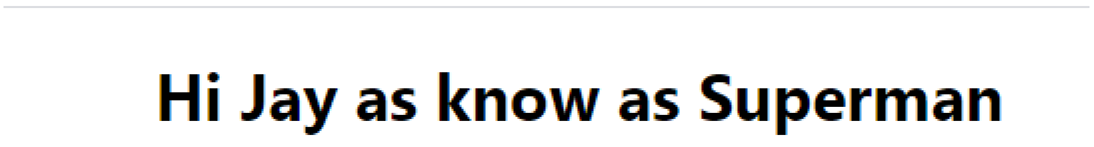
```
const DestructureExample =(props)=> {
     const {name,heroName}=props
     return(
             <div>
          <h1> Hi {name} as know as {heroName}</h1>
             </div>
         )

}
export default DestructureExample
```

App.js



```
import StateExample from './components/StateExample';
import CounterFile from './components/CounterFile';
import DestructureExample from './components/DestructureExample';

class App extends React.Component {
  render() {
    return (
        <div className="App">
    <DestructureExample name="Jay" heroName="Superman"/>


```

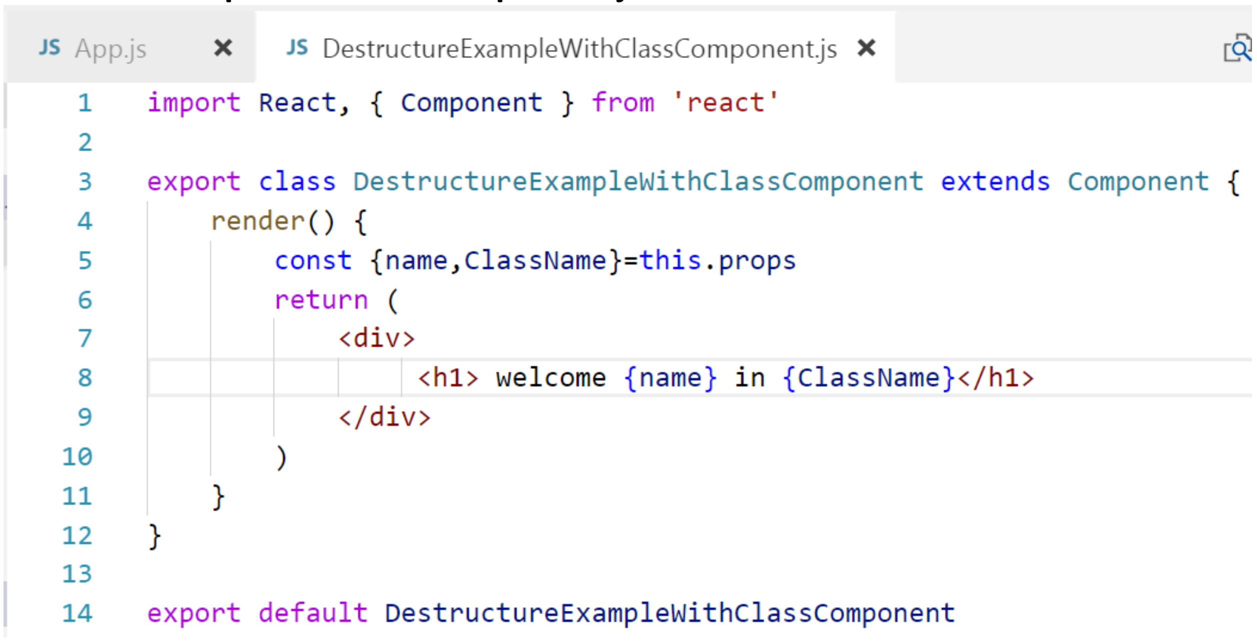
Output:-



```
Hi Jay as know as Superman
```

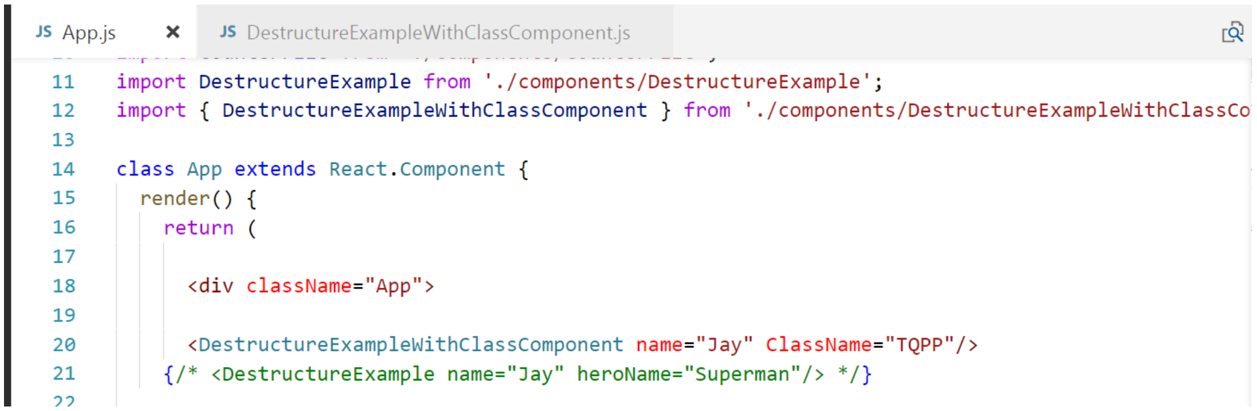
2. Destructuring props in Class components

DestructureExampleWithClassComponent.js



```
import React, { Component } from 'react'

export class DestructureExampleWithClassComponent extends Component {
    render() {
        const {name,ClassName}=this.props
        return (
            <div>
                <h1> welcome {name} in {ClassName}</h1>
            </div>
        )
    }
}

export default DestructureExampleWithClassComponent
```

App.js



```
import DestructureExample from './components/DestructureExample';
import { DestructureExampleWithClassComponent } from './components/DestructureExampleWithClassCo

class App extends React.Component {
  render() {
    return (

      <div className="App">

      <DestructureExampleWithClassComponent name="Jay" ClassName="TQPP"/>
    {/* <DestructureExample name="Jay" heroName="Superman"/> */}
```

Event Handling:-

Any web Application you create typically things to have user interaction. When user interaction with your application event are fired for example mouse-click, mouse-over, key-press, change event and so on

The application must handle such event and execute the necessary code

Handling events with React elements is very similar to handling events on DOM elements. There are some syntactic differences:

- React events are named using **camelCase, rather than lowercase.**
- With JSX you pass a function as the event handler, rather than a string.

In HTML

<button onclick=“handleClick()">Click Me</button>

In React

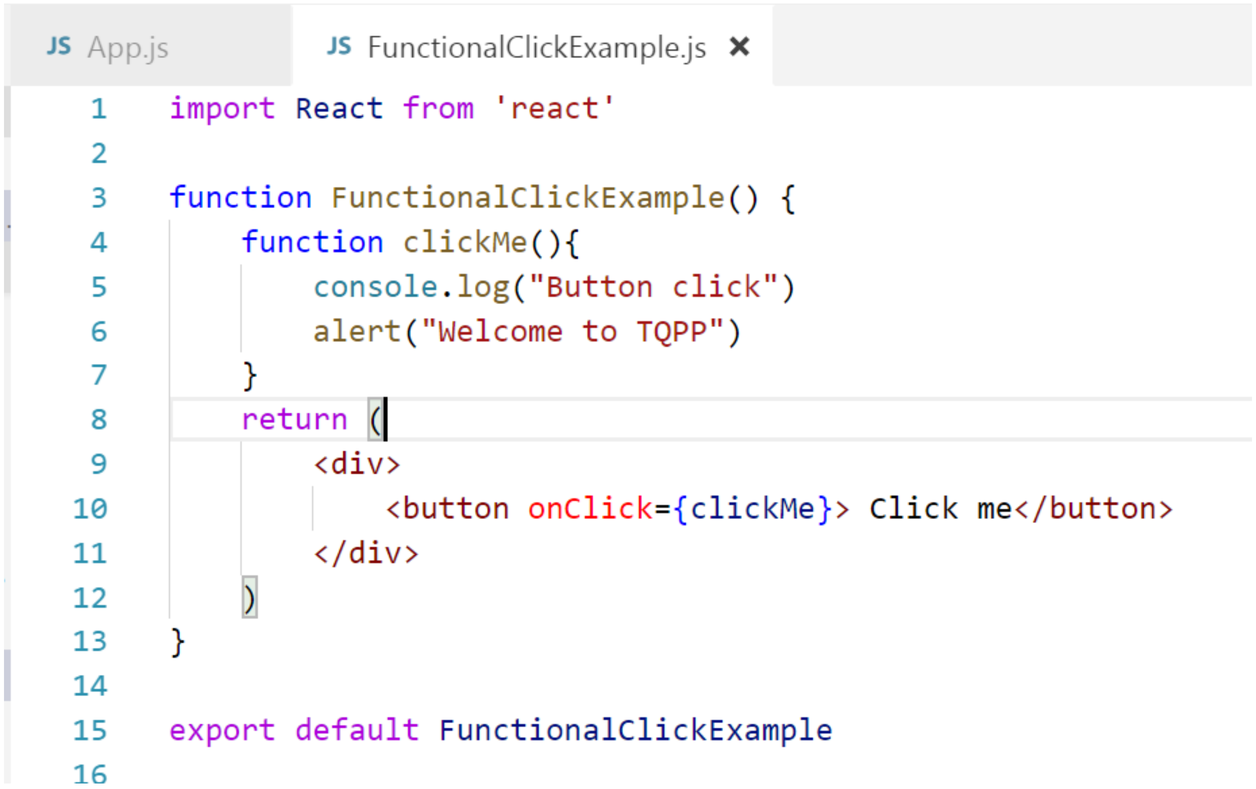
<button onClick={handleClick}>Click Me</button>// Function Component

<button onClick={this.handleClick}>Click Me</button> // Class Component

Example: - onClick event

Create Functional component use cmd **rfce**

FunctionalClickExample.js

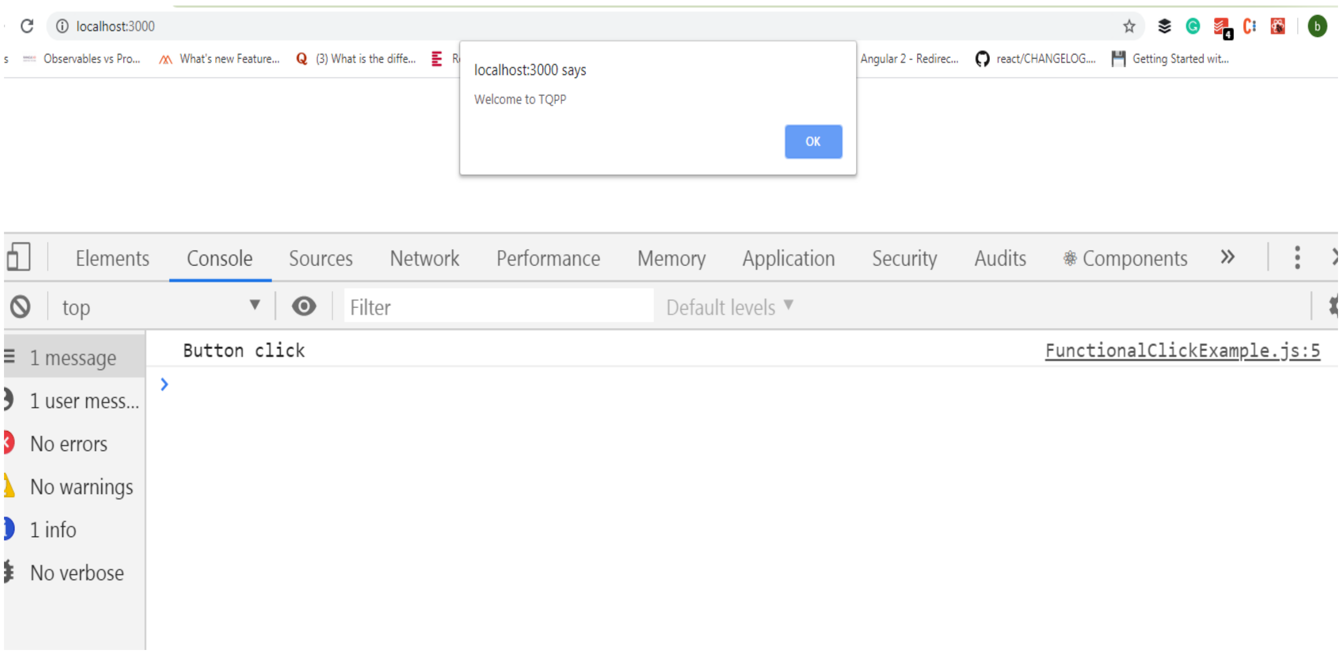


```
import React from 'react'

function FunctionalClickExample() {
    function clickMe(){
        console.log("Button click")
        alert("Welcome to TQPP")
    }
    return (
        <div>
            <button onClick={clickMe}> Click me</button>
        </div>
    )
}

export default FunctionalClickExample

```

App.js

```
import FunctionalClickExample from './components/FunctionalClickExample'

class App extends React.Component {
  render() {
    return (
          <div className="App">
      <FunctionalClickExample/>
```

Output:

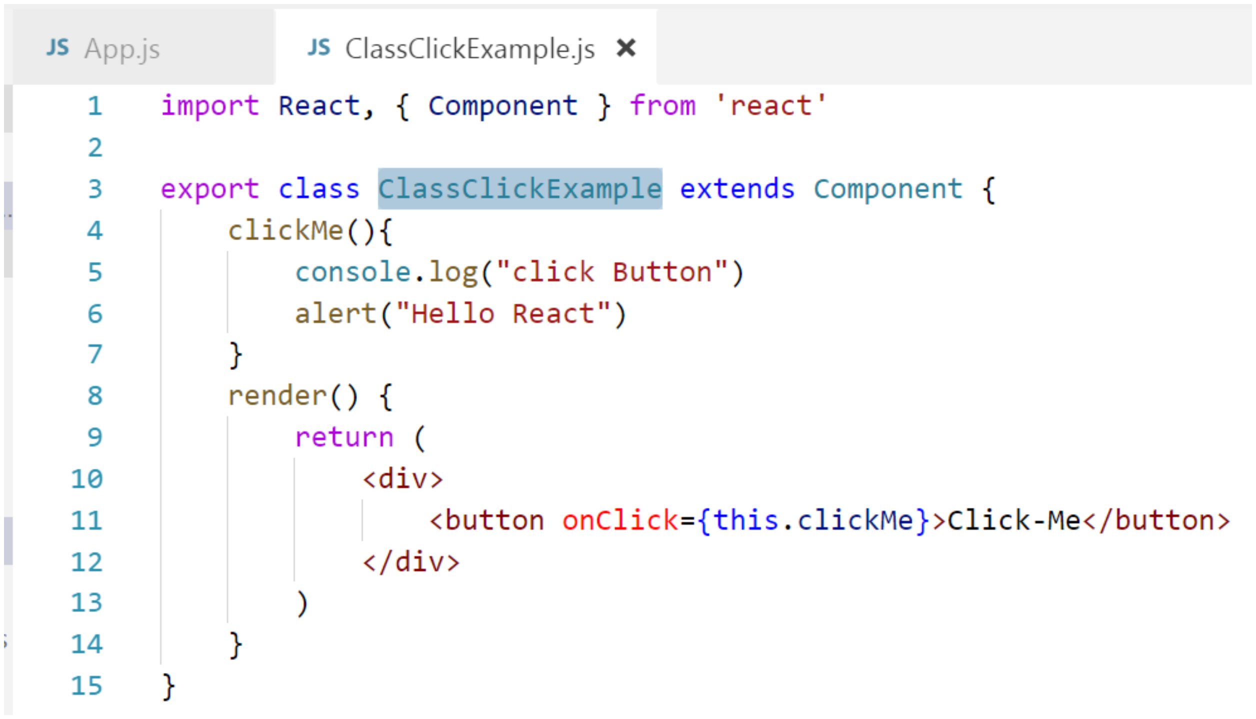


```
localhost:3000 says
Welcome to TQPP
```

```
Button click                                              FunctionalClickExample.js:5
```

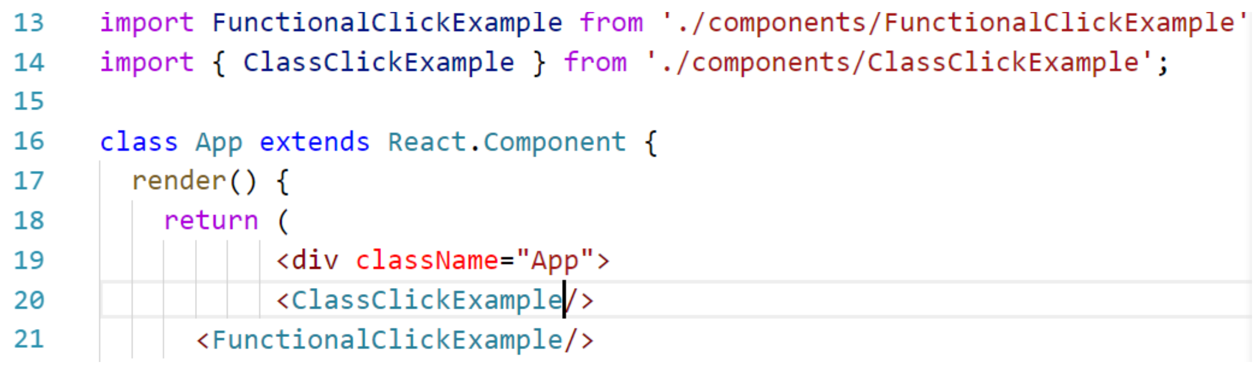
Create class component use cmd **rce**

ClassClickExample.js



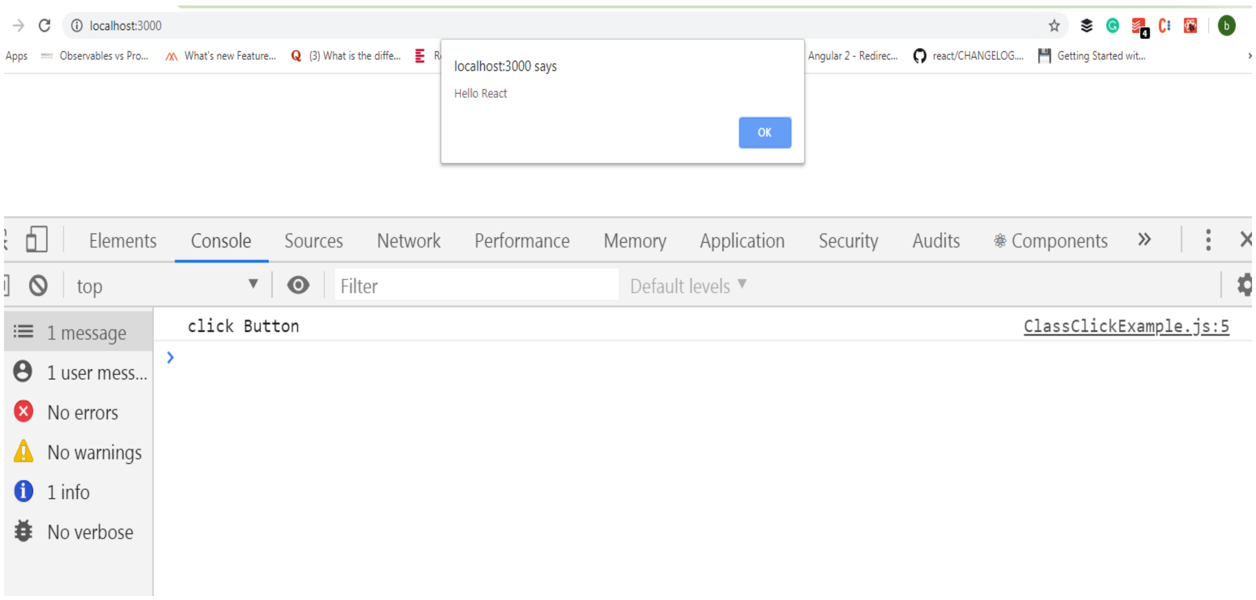
```
import React, { Component } from 'react'

export class ClassClickExample extends Component {
    clickMe(){
        console.log("click Button")
        alert("Hello React")
    }
    render() {
        return (
            <div>
                <button onClick={this.clickMe}>Click-Me</button>
            </div>
        )
    }
}
```

App.js



```
import FunctionalClickExample from './components/FunctionalClickExample'
import { ClassClickExample } from './components/ClassClickExample';

class App extends React.Component {
  render() {
    return (
        <div className="App">
        <ClassClickExample/>
      <FunctionalClickExample/>
```

Output:



```
localhost:3000 says
Hello React
```

```
click Button                                                ClassClickExample.js:5
```

Binding Event Handlers:-

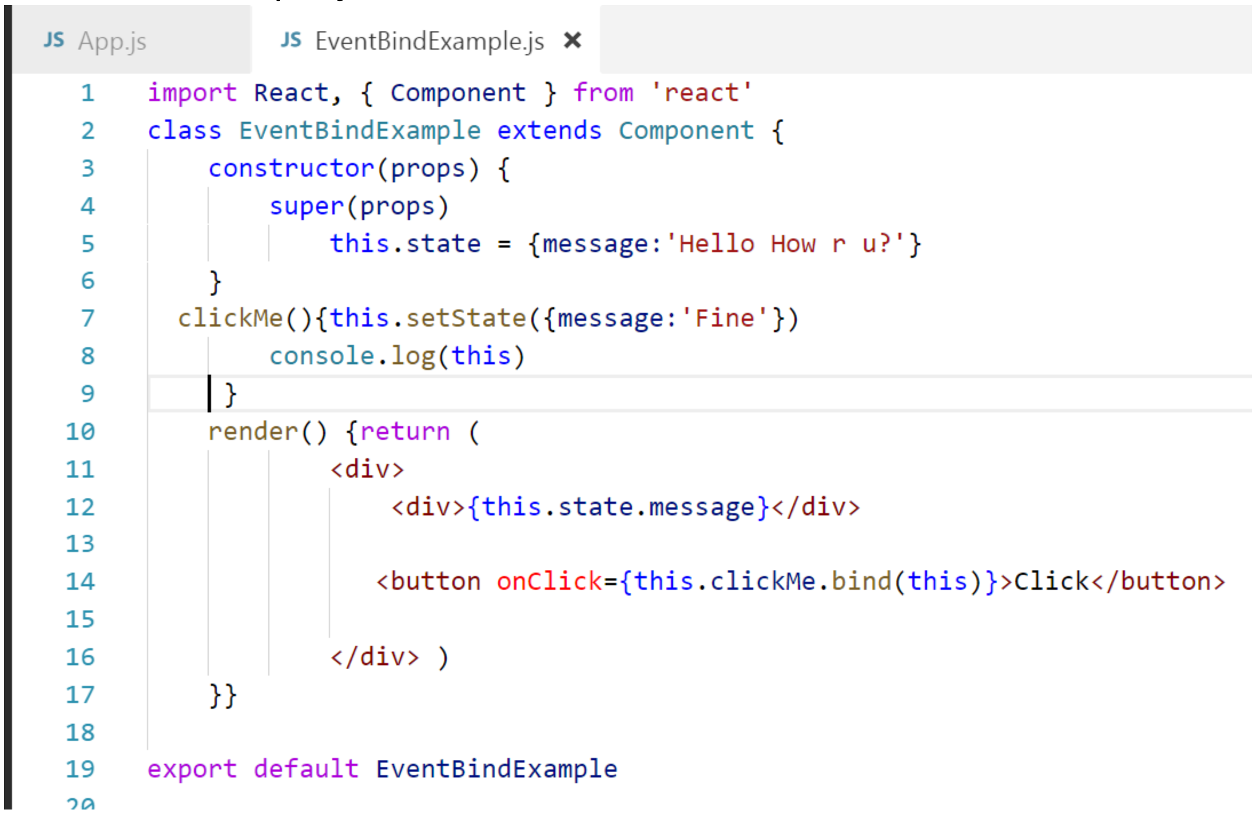
E.g. Onclick on button we simply change a message which is part of components state

There are four approaches to binding event handler

1. Binding in render ()
2. Arrow function in render ()
3. Binding event handler in class constructor
4. Class property as arrow function

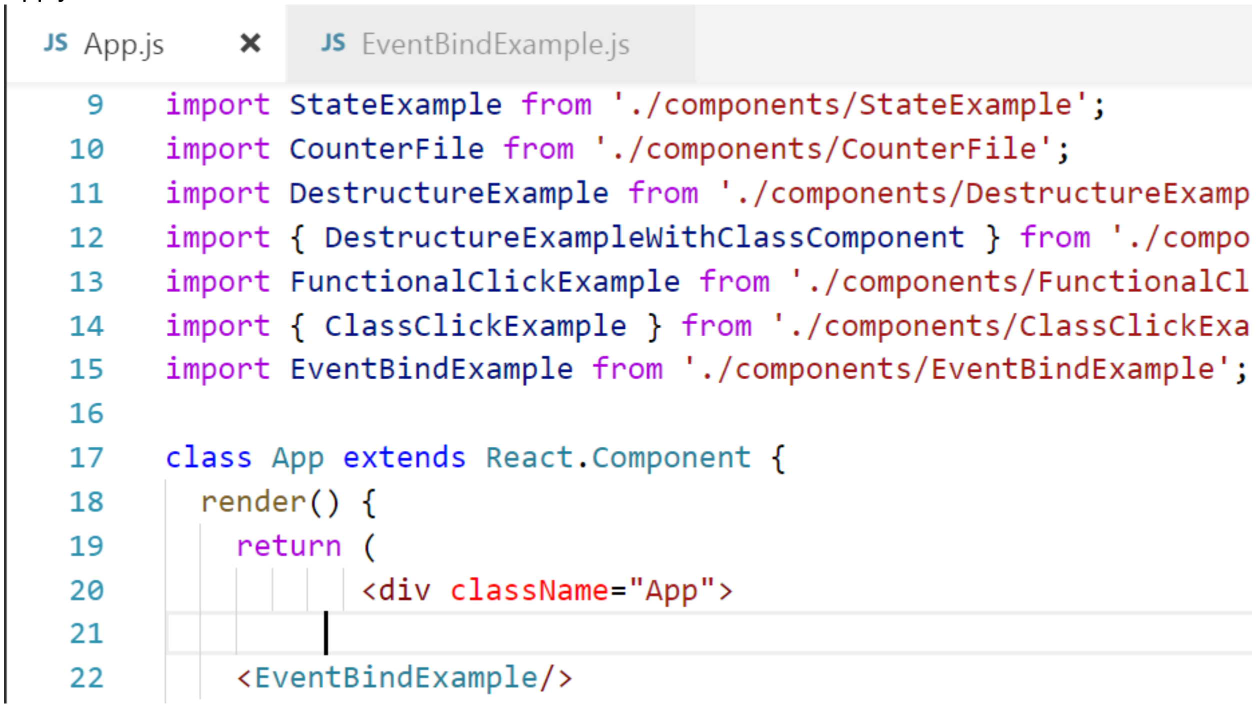
1. Binding in render ()

EventBindExample.js



```
import React, { Component } from 'react'
class EventBindExample extends Component {
    constructor(props) {
        super(props)
            this.state = {message:'Hello How r u?'}
    }
  clickMe(){this.setState({message:'Fine'})
        console.log(this)
    }
    render() {return (
            <div>
                <div>{this.state.message}</div>

               <button onClick={this.clickMe.bind(this)}>Click</button>

            </div> )
    }}

export default EventBindExample
```

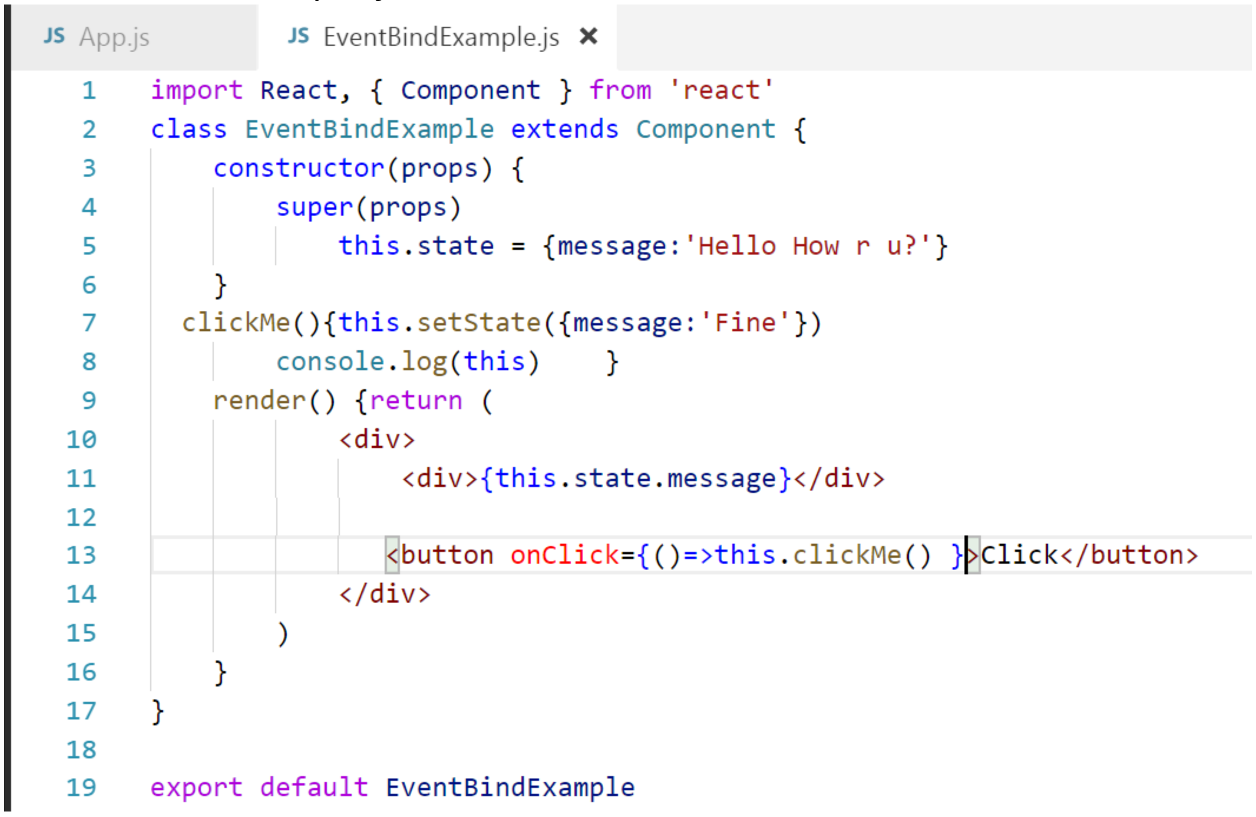
App.js



```
import StateExample from './components/StateExample';
import CounterFile from './components/CounterFile';
import DestructureExample from './components/DestructureExamp
import { DestructureExampleWithClassComponent } from './compo
import FunctionalClickExample from './components/FunctionalCl
import { ClassClickExample } from './components/ClassClickExa
import EventBindExample from './components/EventBindExample';

class App extends React.Component {
  render() {
    return (
          <div className="App">

    <EventBindExample/>
```

2. Arrow function in render ()

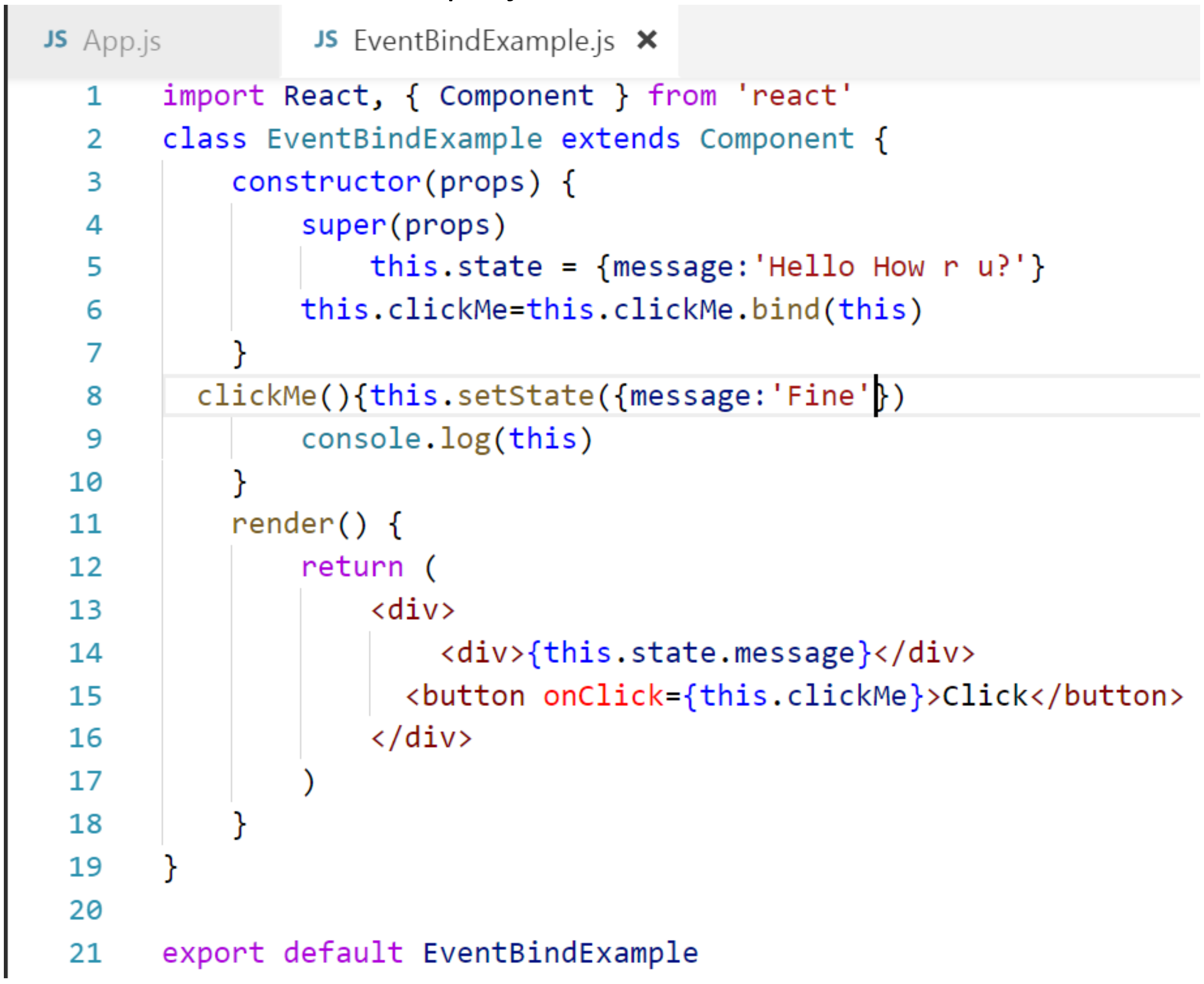
EventBindExample.js



```
import React, { Component } from 'react'
class EventBindExample extends Component {
    constructor(props) {
        super(props)
            this.state = {message:'Hello How r u?'}
    }
  clickMe(){this.setState({message:'Fine'})
        console.log(this)     }
    render() {return (
            <div>
                <div>{this.state.message}</div>

               <button onClick={()=>this.clickMe() }>Click</button>
            </div>
        )
    }
}

export default EventBindExample
```

3. Binding event handler in class constructor

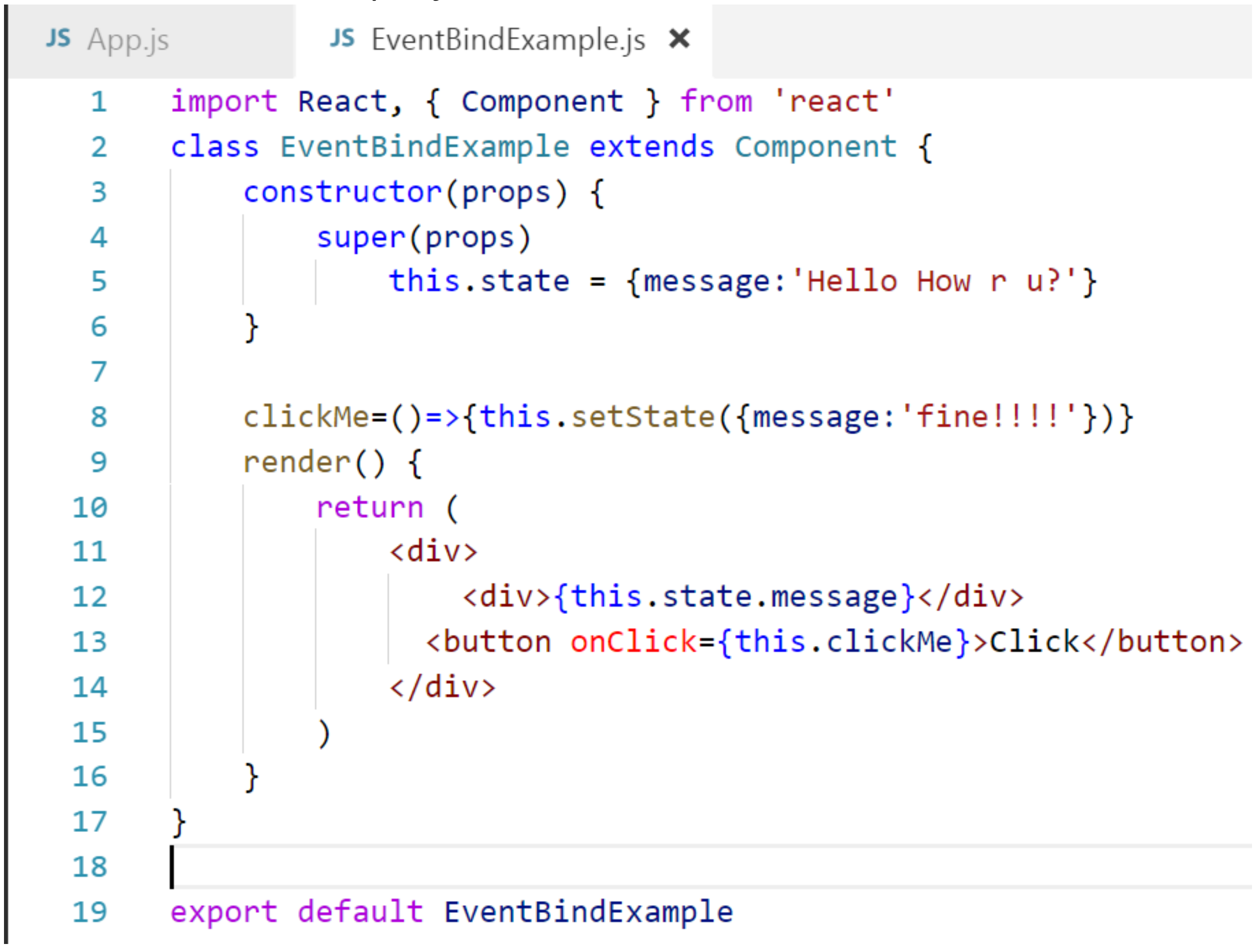
EventBindExample.js



```
import React, { Component } from 'react'
class EventBindExample extends Component {
    constructor(props) {
        super(props)
            this.state = {message:'Hello How r u?'}
        this.clickMe=this.clickMe.bind(this)
    }
  clickMe(){this.setState({message:'Fine'})
        console.log(this)
    }
    render() {
        return (
            <div>
                <div>{this.state.message}</div>
              <button onClick={this.clickMe}>Click</button>
            </div>
        )
    }
}

export default EventBindExample
```

4. Class property as arrow function

   EventBindExample.js



```
import React, { Component } from 'react'
class EventBindExample extends Component {
    constructor(props) {
        super(props)
            this.state = {message:'Hello How r u?'}
    }

    clickMe=()=>{this.setState({message:'fine!!!!'})}
    render() {
        return (
            <div>
                <div>{this.state.message}</div>
             <button onClick={this.clickMe}>Click</button>
            </div>
        )
    }
}

export default EventBindExample
```

Methods as props:

You can see pervious lecture how to pass parent data to child component using props

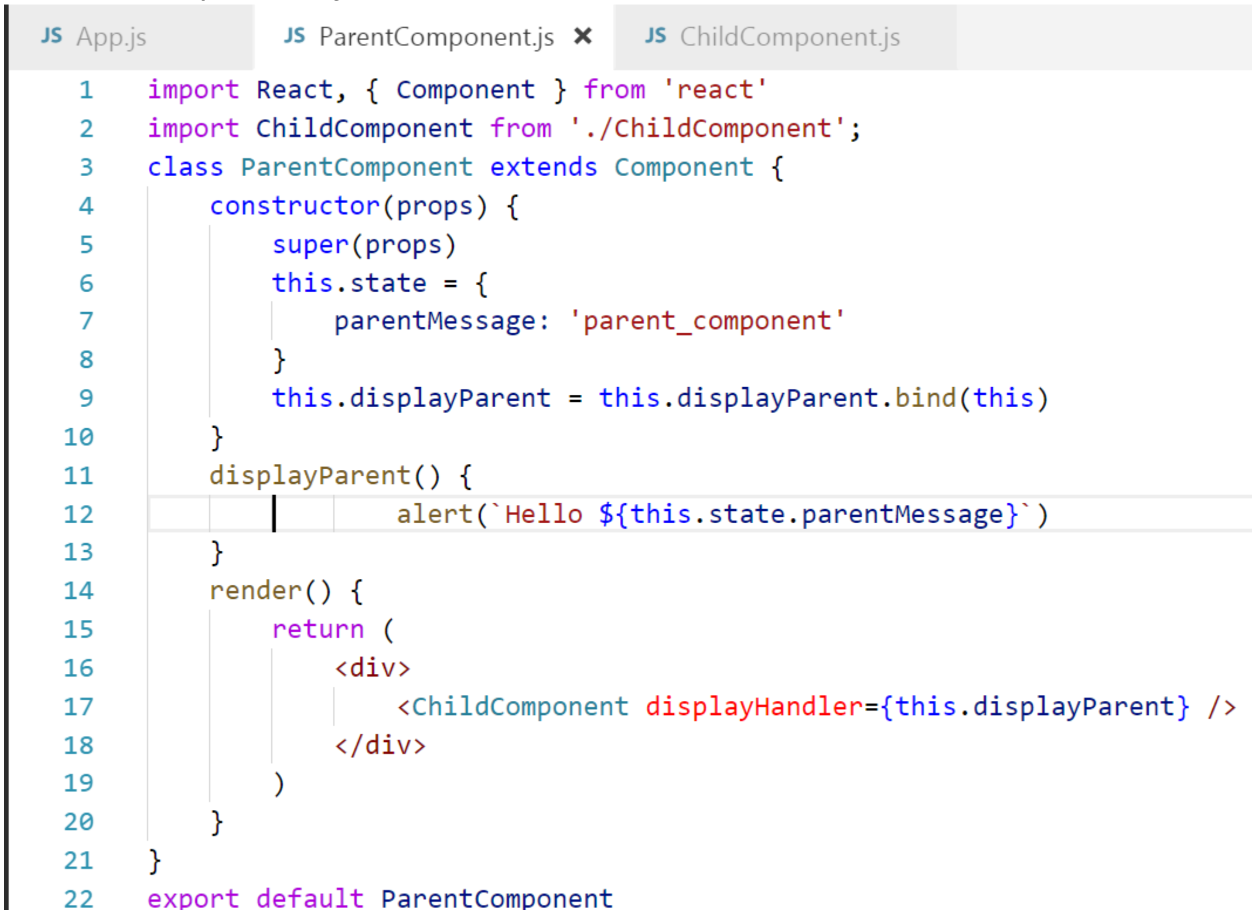
If you want pass data child component to parent component use props

Let's discuss how to use method as pops to pass data parent to child

Access parent method in child component use props

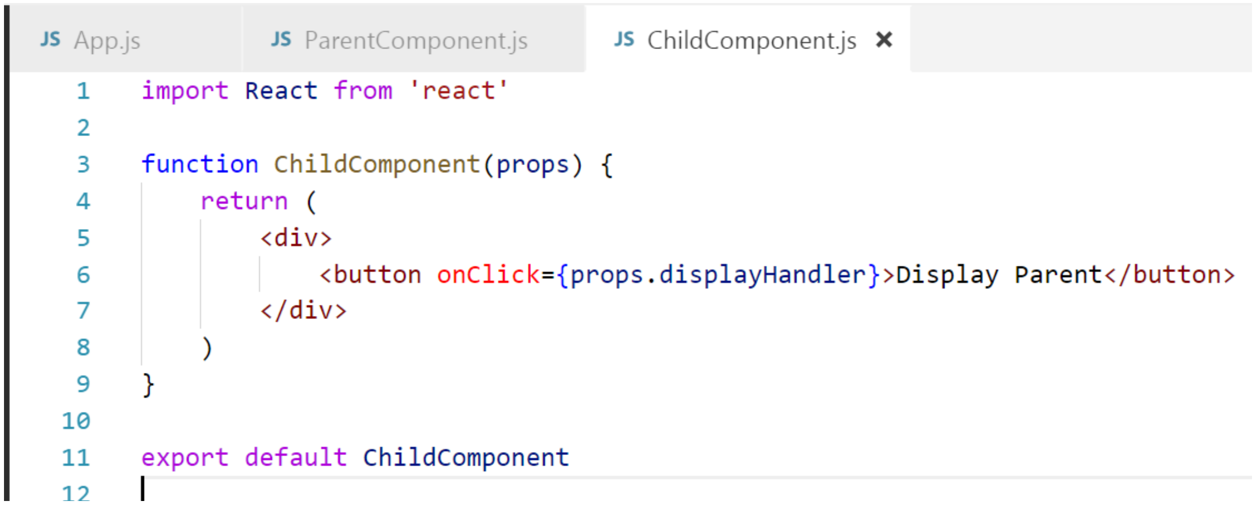
Pass data from parent to child show following example

ParentComponent.js



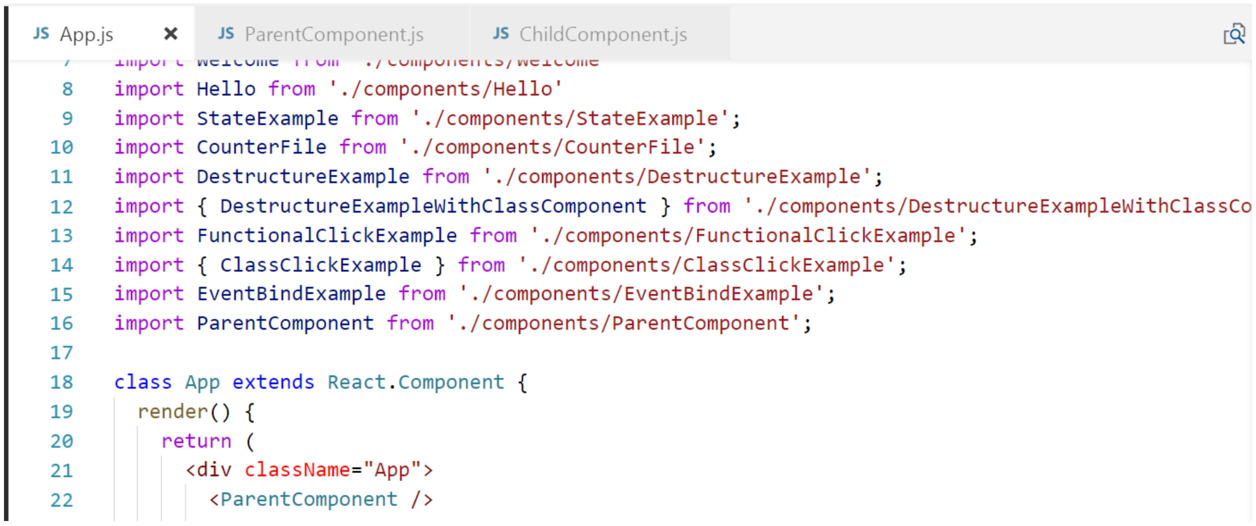
```
import React, { Component } from 'react'
import ChildComponent from './ChildComponent';
class ParentComponent extends Component {
    constructor(props) {
        super(props)
        this.state = {
            parentMessage: 'parent_component'
        }
        this.displayParent = this.displayParent.bind(this)
    }
    displayParent() {
                alert(`Hello ${this.state.parentMessage}`)
    }
    render() {
        return (
            <div>
                <ChildComponent displayHandler={this.displayParent} />
            </div>
        )
    }
}
export default ParentComponent
```

ChildComponent.js



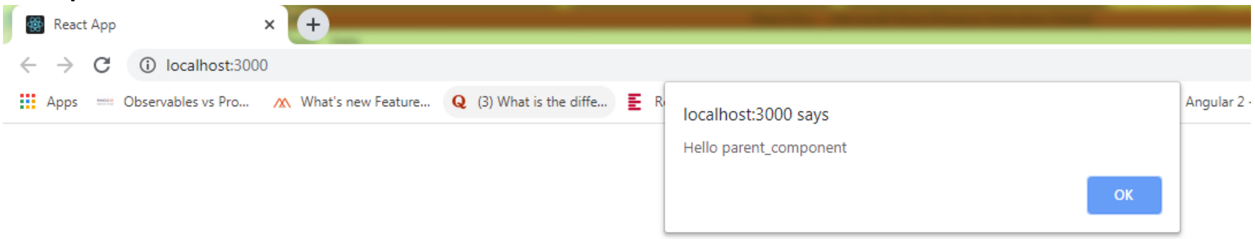
```
import React from 'react'

function ChildComponent(props) {
    return (
        <div>
            <button onClick={props.displayHandler}>Display Parent</button>
        </div>
    )
}

export default ChildComponent

```

App.js



```
import Welcome from './components/Welcome'
import Hello from './components/Hello'
import StateExample from './components/StateExample';
import CounterFile from './components/CounterFile';
import DestructureExample from './components/DestructureExample';
import { DestructureExampleWithClassComponent } from './components/DestructureExampleWithClassCo
import FunctionalClickExample from './components/FunctionalClickExample';
import { ClassClickExample } from './components/ClassClickExample';
import EventBindExample from './components/EventBindExample';
import ParentComponent from './components/ParentComponent';

class App extends React.Component {
  render() {
    return (
      <div className="App">
        <ParentComponent />
```

Output:

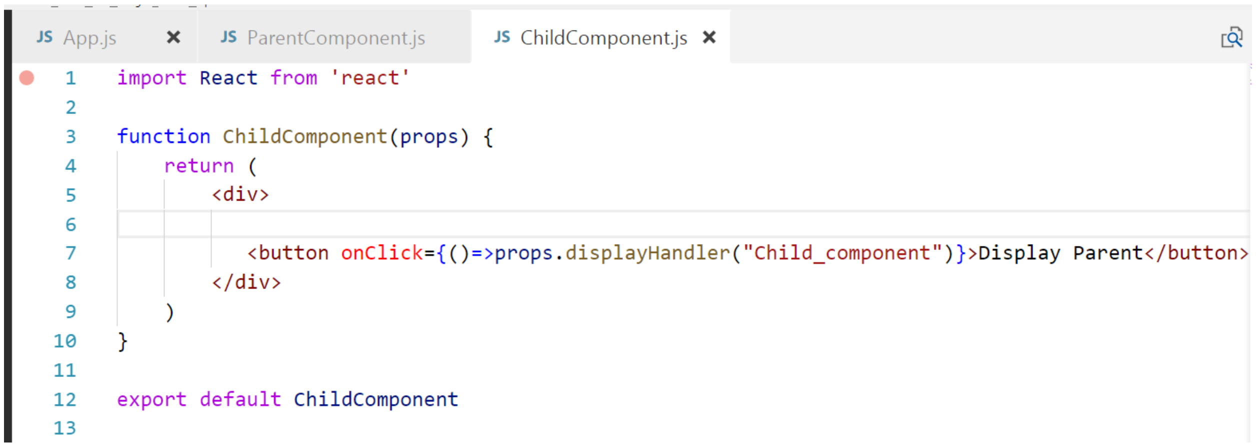


```
localhost:3000 says
Hello parent_component
```

Now pass data or parameter from child to parent in this arrow function in the return statement really useful. Arrow function syntax is simplest way to pass a parameter from the child component to parent component

Example:

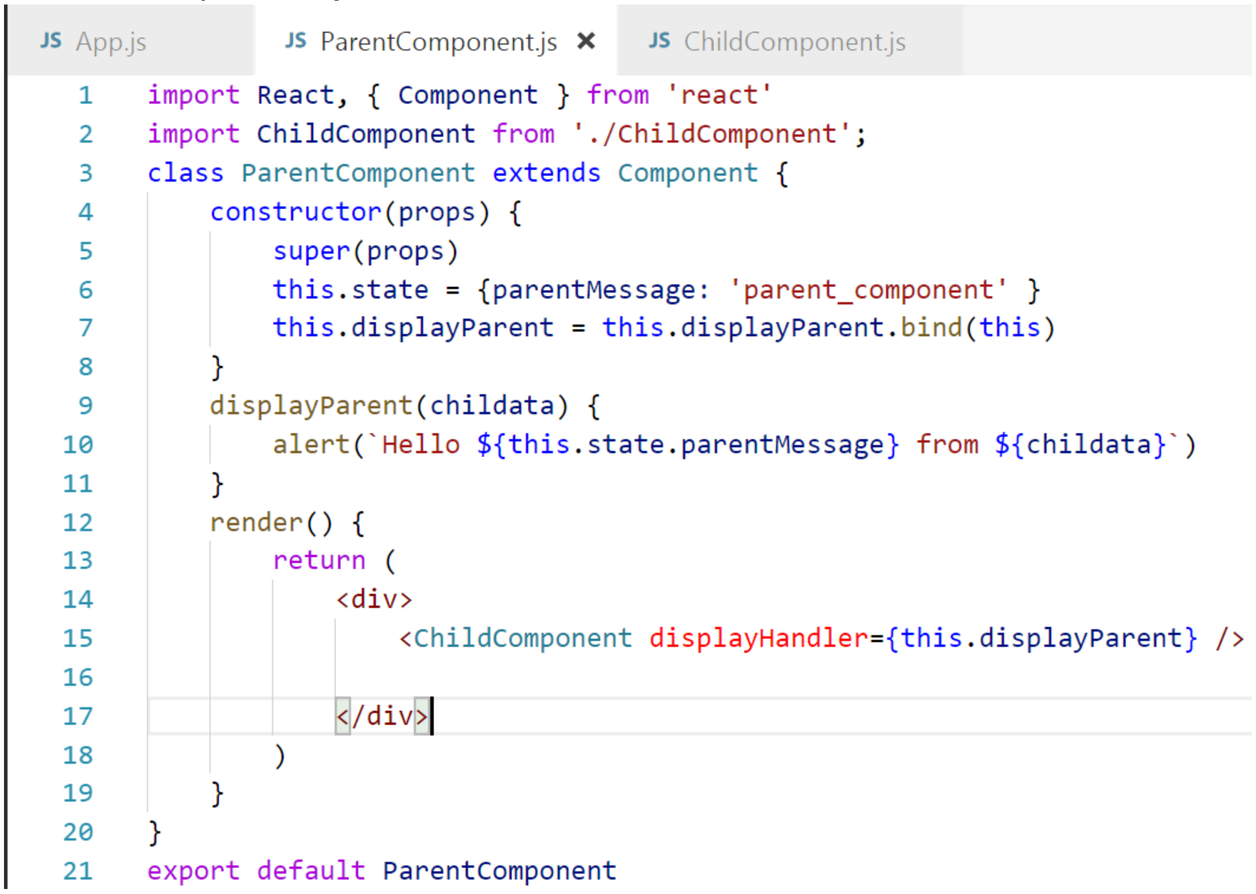
ChildComponent.js



App.js ParentComponent.js ChildComponent.js

```
import React from 'react'

function ChildComponent(props) {
    return (
        <div>

            <button onClick={()=>props.displayHandler("Child_component")}>Display Parent</button>
        </div>
    )
}

export default ChildComponent

```

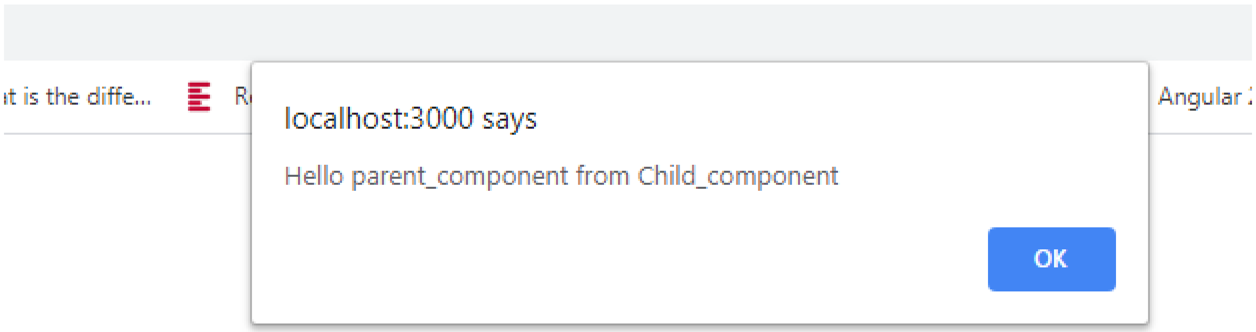
ParentComponent.js



App.js ParentComponent.js ChildComponent.js

```
import React, { Component } from 'react'
import ChildComponent from './ChildComponent';
class ParentComponent extends Component {
    constructor(props) {
        super(props)
        this.state = {parentMessage: 'parent_component' }
        this.displayParent = this.displayParent.bind(this)
    }
    displayParent(childata) {
        alert(`Hello ${this.state.parentMessage} from ${childata}`)
    }
    render() {
        return (
            <div>
                <ChildComponent displayHandler={this.displayParent} />

            </div>
        )
    }
}
export default ParentComponent
```

Output:



localhost:3000 says

Hello parent_component from Child_component