



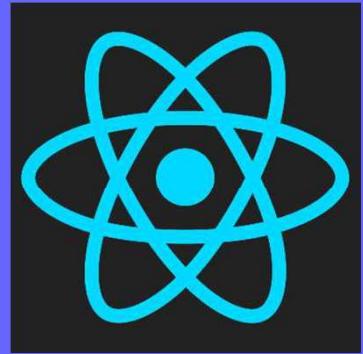
# State in React

# Objectives

After completing this lesson, you should be able to do the following:

- Basics in State
- useState
- State, Events and Managed Controls





# Introduction to State

- As we have seen so far every react component returns JSX which describes the user interface.
- It is possible to influence what is rendered on the screen using props as it turns out
- There is a second way to influence what is rendered on the screen and that is the **State of the component**.

- **State in React components** is essential to manage and communicate data in your application.
- It is represented as a **JavaScript object** and has **component level scope**, it can be thought of as the **Private Data of Your Component**.

- Both **props** and **State** hold information that influences the UI in the browser

## props vs state

---

### props

props get passed to the component  
Function parameters  
props are immutable  
props – Functional Components  
this.props – Class Components

### state

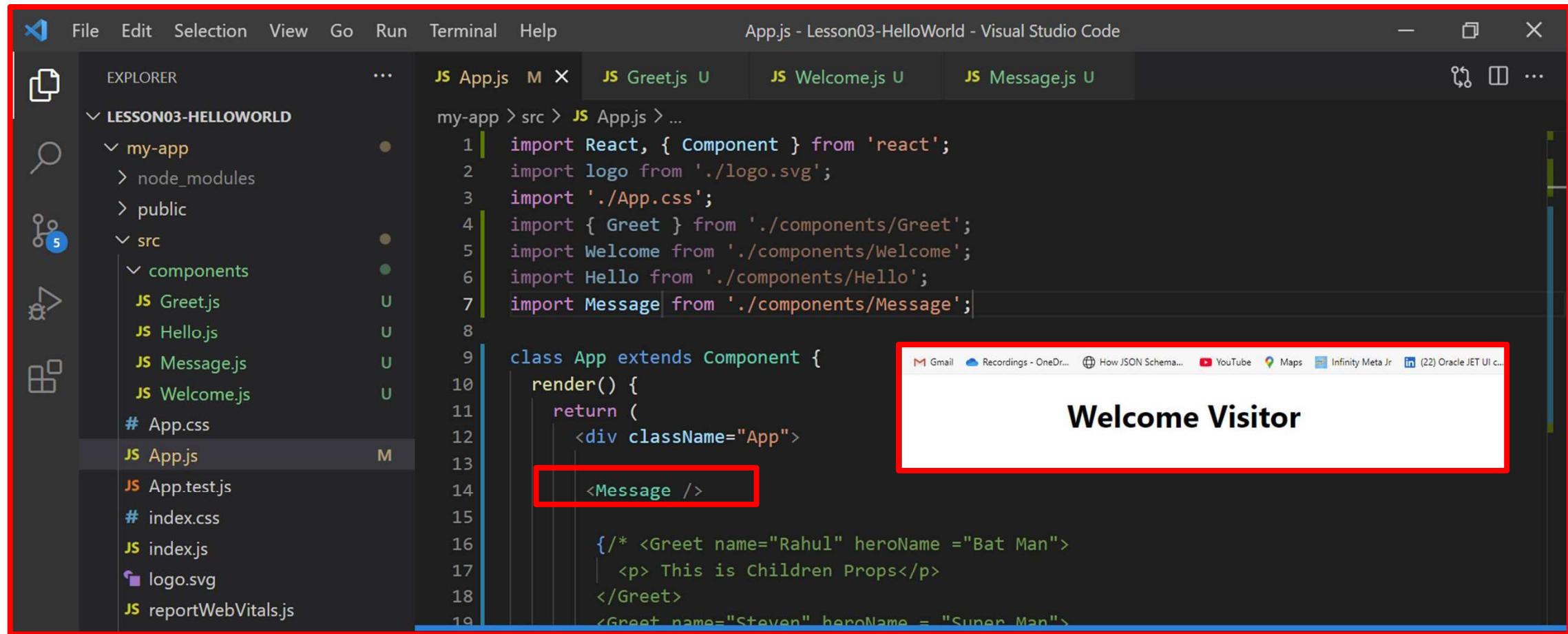
state is managed within the component  
Variables declared in the function body  
state can be changed  
useState Hook – Functional Components  
this.state – Class Components

# We shall Create Message.js

The screenshot shows the Visual Studio Code interface with a red border around the main window. The title bar reads "Message.js - Lesson03-HelloWorld - Visual Studio Code". The left sidebar has icons for Explorer, Search, Problems (with 5), and Editor. The Explorer view shows a project structure under "LESSON03-HELLOWORLD": my-app, node\_modules, public, src, components, Greet.js, Hello.js, Message.js (highlighted with a red box), Welcome.js, App.css, App.js, and App.test.js. The main editor area displays the following code:

```
my-app > src > components > JS Message.js > [?] default
1 import React, { Component } from 'react';
2
3 class Message extends Component {
4   render() {
5     return <h1> Welcome Visitor </h1>
6   }
7 }
8
9 export default Message;
```

# Including Message Component in App Component



File Edit Selection View Go Run Terminal Help App.js - Lesson03-HelloWorld - Visual Studio Code

EXPLORER

LESSON03-HELLOWORLD

- my-app
  - node\_modules
  - public
  - src
    - components
      - Greet.js
      - Hello.js
      - Message.js
      - Welcome.js
    - App.css
    - App.js M
    - App.test.js
    - index.css
    - index.js
    - logo.svg
    - reportWebVitals.js

JS App.js M X JS Greet.js U JS Welcome.js U JS Message.js U

```
my-app > src > JS App.js > ...
1 | import React, { Component } from 'react';
2 | import logo from './logo.svg';
3 | import './App.css';
4 | import { Greet } from './components/Greet';
5 | import Welcome from './components/Welcome';
6 | import Hello from './components/Hello';
7 | import Message from './components/Message';

8 |
9 | class App extends Component {
10|   render() {
11|     return (
12|       <div className="App">
13|         <Message />
14|
15|         {/* <Greet name="Rahul" heroName ="Bat Man">
16|           <p> This is Children Props</p>
17|         </Greet>
18|         <Greet name="Steven" heroName = "Super Man">
19|           <p> This is Children Props</p>
20|         </Greet>
21|       </div>
22|     );
23|   }
24| }

25| export default App;
```

Gmail Recordings - OneDr... How JSON Schema... YouTube Maps Infinity Meta Jr in (22) Oracle JET UI c...

Welcome Visitor

The screenshot shows the Visual Studio Code interface with the file 'App.js' open. The code imports components from 'components' and defines an 'App' class that renders a 'Message' component. A red box highlights the line '<Message />'. To the right, a preview window displays the text 'Welcome Visitor'. The sidebar shows the project structure under 'LESSON03-HELLOWORLD'.

## Real Need ?

- Now here is our new requirement we need to have a subscribe button right below the text
- When we click on the button the text being displayed should change from **Welcome Visitor** to **Thank U for Subscribing**
- Now if the message were to be sent from App.js as a property it will not work as props are immutable once the message is said to Welcome Visitor
- It can never be changed from the message component and so the solution is to use
  - **Component State.**

## Steps Involved

1. Create a State Object and Initialize it. This is done usually in the Class Constructor



The screenshot shows a code editor with several tabs at the top: JS App.js, JS Greet.js, JS Welcome.js, and JS Message.js. The JS Message.js tab is active. Below the tabs, a file path is displayed: my-app > src > components > JS Message.js > Message > constructor > message. The code itself is a class-based React component named Message. The constructor is highlighted with a red box. It initializes the state with a message: 'Welcome Visitor'.

```
JS App.js M X JS Greet.js U JS Welcome.js U JS Message.js U X
my-app > src > components > JS Message.js > Message > constructor > message
1 import React, { Component } from 'react';
2
3 class Message extends Component {
4
5   constructor() {
6     super()
7     this.state = {
8       message: 'Welcome Visitor'
9     }
10  }
11
12  render() {
13    return <h1> Welcome Visitor </h1>
14  }
15}
16
17 export default Message
```

- Now remember we are inside a class so don't forget to use the this keyword
- State is reserved keyword in React
- [ When we use this.state React Understands Our Intentions ]

## Step 2

- Bind this state value in the render function and we do this very similar to props instead of this.props we simply

The image shows a code editor interface with several tabs at the top: JS App.js, JS Greet.js, JS Welcome.js, and JS Message.js. The JS Message.js tab is active. Below the tabs, a file structure is displayed: my-app > src > components > JS Message.js. The code editor displays the following code:

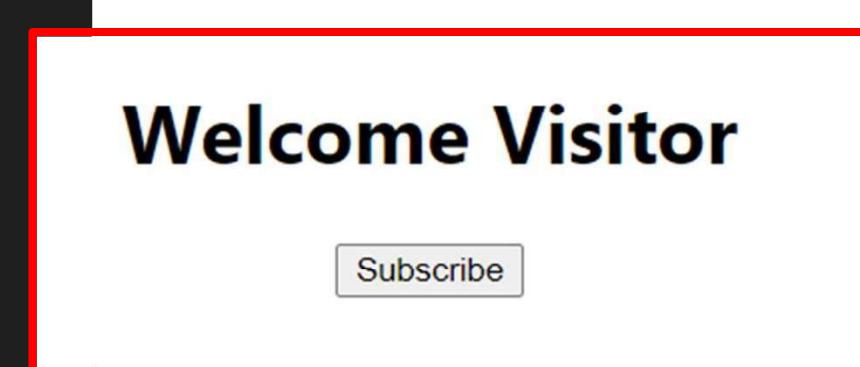
```
my-app > src > components > JS Message.js > Message > render
1 import React, { Component } from 'react';
2
3 class Message extends Component {
4
5     constructor() {
6         super()
7         this.state = {
8             message: 'Welcome Visitor'
9         }
10    }
11
12    render() {
13        return <h1> {this.state.message} </h1>
14    }
15}
16
17 export default Message
```

To the right of the code editor, a browser window is open, displaying the rendered output: "Welcome Visitor". A red box highlights the line of code `<h1> {this.state.message} </h1>` in the render function.

## Step 3:

3. We are now using state to render the message we now have the ability to change this message

```
class Message extends Component {  
  
  constructor() {  
    super()  
    this.state = {  
      message: 'Welcome Visitor'  
    }  
  }  
  
  render() {  
    return (  
      <div>  
        <h1> { this.state.message } </h1>  
        <button>Subscribe</button>  
      </div>  
    )  
  }  
}
```



- We need an enclosing div tag because react wants only one element returned

## Step 4 :

4. To listen to the click event on this button and change the message

The screenshot shows a code editor on the left and two browser previews on the right.

**Code Editor:**

```
changeMessage() {
  this.setState({
    message: 'Thank you for Subscribing !'
  })
}

render() {
  return (
    <div>
      <h1> { this.state.message } </h1>
      <button onClick={()=> this.changeMessage()}>Subscribe</button>
    </div>
  )
}
```

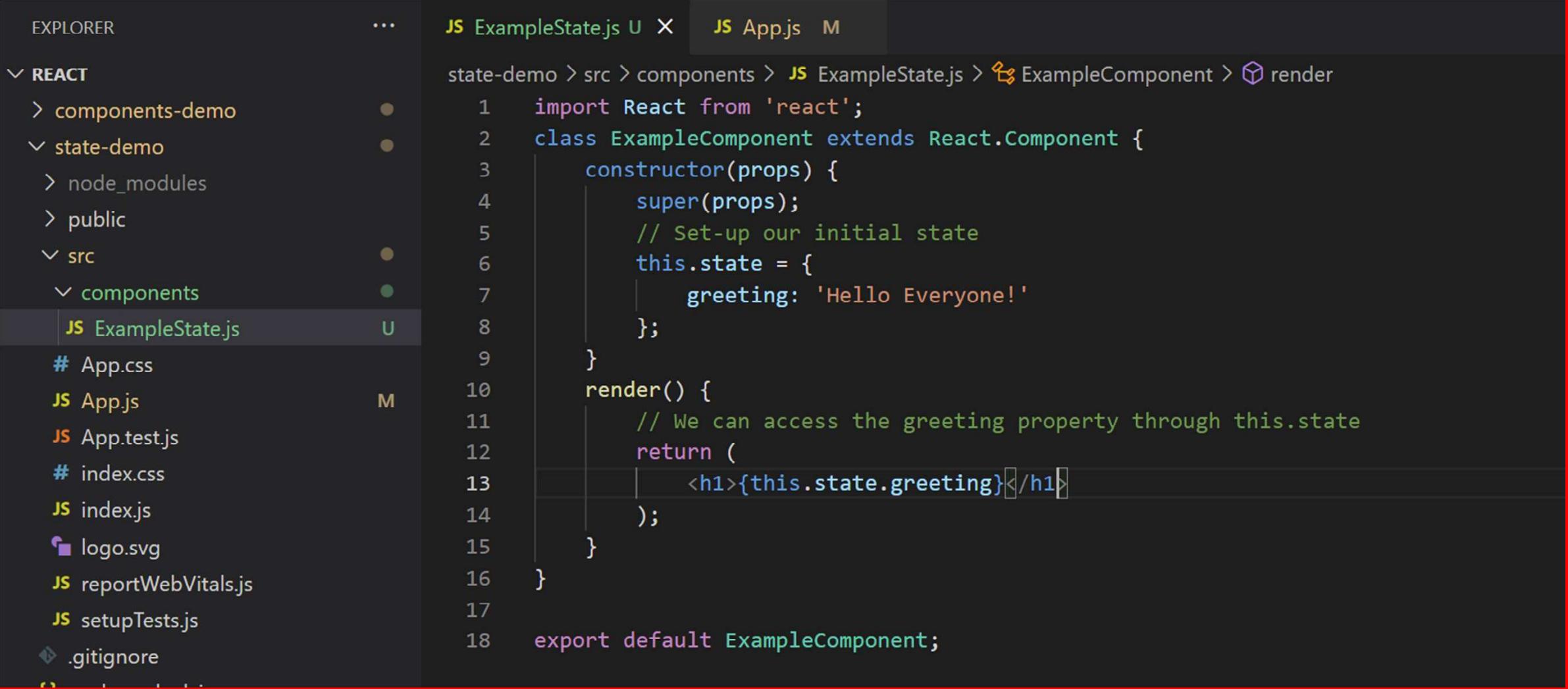
**Browser Previews:**

- Top Preview:** Displays the text "Welcome Visitor" above a "Subscribe" button.
- Bottom Preview:** Displays the text "Thank you for Subscribing !" above a "Subscribe" button.

Both the code editor and the browser previews have red boxes highlighting the `changeMessage()` method and the `onClick` event handler in the `render()` method.

- **setState** method accepts an object which is nothing but the new state of the component in the new state

1. State in react is nothing but an object that is privately maintained inside a component.
2. State can influence what is rendered in the browser
3. Lastly state can be changed within the component **[ Mutable ]**



The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists project files under a 'REACT' category. The 'ExampleState.js' file is selected in the Explorer. The main pane displays the code for 'ExampleComponent.js'. The code defines a class component that initializes state with a greeting and renders it as an `<h1>` element.

```
state-demo > src > components > JS ExampleState.js > ExampleComponent > render
1 import React from 'react';
2 class ExampleComponent extends React.Component {
3     constructor(props) {
4         super(props);
5         // Set-up our initial state
6         this.state = {
7             greeting: 'Hello Everyone!'
8         };
9     }
10    render() {
11        // We can access the greeting property through this.state
12        return (
13            <h1>{this.state.greeting}</h1>
14        );
15    }
16}
17
18 export default ExampleComponent;
```

The screenshot shows a development environment with a red border around the main workspace. On the left is the Explorer sidebar with a tree view of a React project structure:

- REACT
- components-demo
- state-demo
- node\_modules
- public
- src
  - components
    - ExampleState.js
  - App.css
  - App.js
  - App.test.js
  - index.css
  - index.js
- logo
- repo
- setup

The App.js file is open in the editor tab, showing the following code:

```
state-demo > src > JS App.js > ...
1 import logo from './logo.svg';
2 import './App.css';
3 import ExampleComponent from './components/ExampleState';
4 function App() {
5   return (
6     <div className="App">
7       <ExampleComponent />
8     </div>
9   );
10 }
11 }
12 }
13
14 export default App;
```

The browser preview window at the bottom shows the output of the code: "Hello Everyone!"

# Hello Everyone!

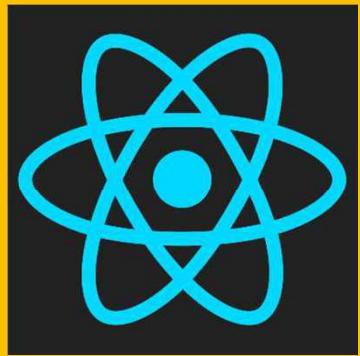
The screenshot shows a code editor interface with a red border around the main workspace. On the left is the Explorer sidebar, which lists the project structure under the 'REACT' category. The 'src' folder contains 'components' and 'state-demo'. Inside 'state-demo' are files like 'ExampleState.js' and 'StateDemo02.js', both of which are selected and highlighted with a green background. The main workspace shows the content of 'StateDemo02.js'. The code defines a class 'MyComponent' that extends 'React.Component'. It has a constructor that initializes state with a 'days' key set to an empty string. It binds its 'onChange' event to a method that updates the state with the new value from the input. The render method returns a div containing an input field with a default value of '2' and an onChange handler that calls the bound 'onChange' method. The URL for the input is constructed by concatenating the prop 'url' with '/days?=' and the current state 'days'.

```
1 import React from 'react';
2 export default class MyComponent extends React.Component {
3     constructor() {
4         super();
5         this.state = {
6             days: ''
7         }
8         this.onChange = this.onChange.bind(this);
9     }
10    onChange(e) {
11        this.setState({
12            days: e.target.value
13        });
14    }
15    render() {
16        return (
17            <div>
18                <input defaultValue={2} onChange={this.onChange} />
19                URL: {this.props.url + '/days?=' + this.state.days}
20            </div>
21        )
22    }
23 }
```

The screenshot shows a code editor interface with a red border around the main workspace. On the left is the Explorer sidebar, also with a red border, displaying a file tree for a 'REACT' project. The 'App.js' file is selected in the Explorer. The main workspace contains two tabs: 'StateDemo02.js' (highlighted in green) and 'App.js'. The 'App.js' tab is active, showing the following code:

```
state-demo > src > JS App.js > App
1 import logo from './logo.svg';
2 import './App.css';
3 import ExampleComponent from './components/ExampleState';
4 import MyComponent from './components/StateDemo02';
5 function App() {
6   return (
7     <div className="App">
8       /* <ExampleComponent /> */
9       <MyComponent />
10    </div>
11  );
12}
13
14
15
16
17 export default App;
18
```

A red box highlights the URL bar in a browser window at the bottom right, which displays 'URL: undefined/days?=2'. The browser window also shows other tabs like Recordings - OneDr..., YouTube, Maps, Infinity Meta Jr, and Oracle JET UI c... with their respective icons.



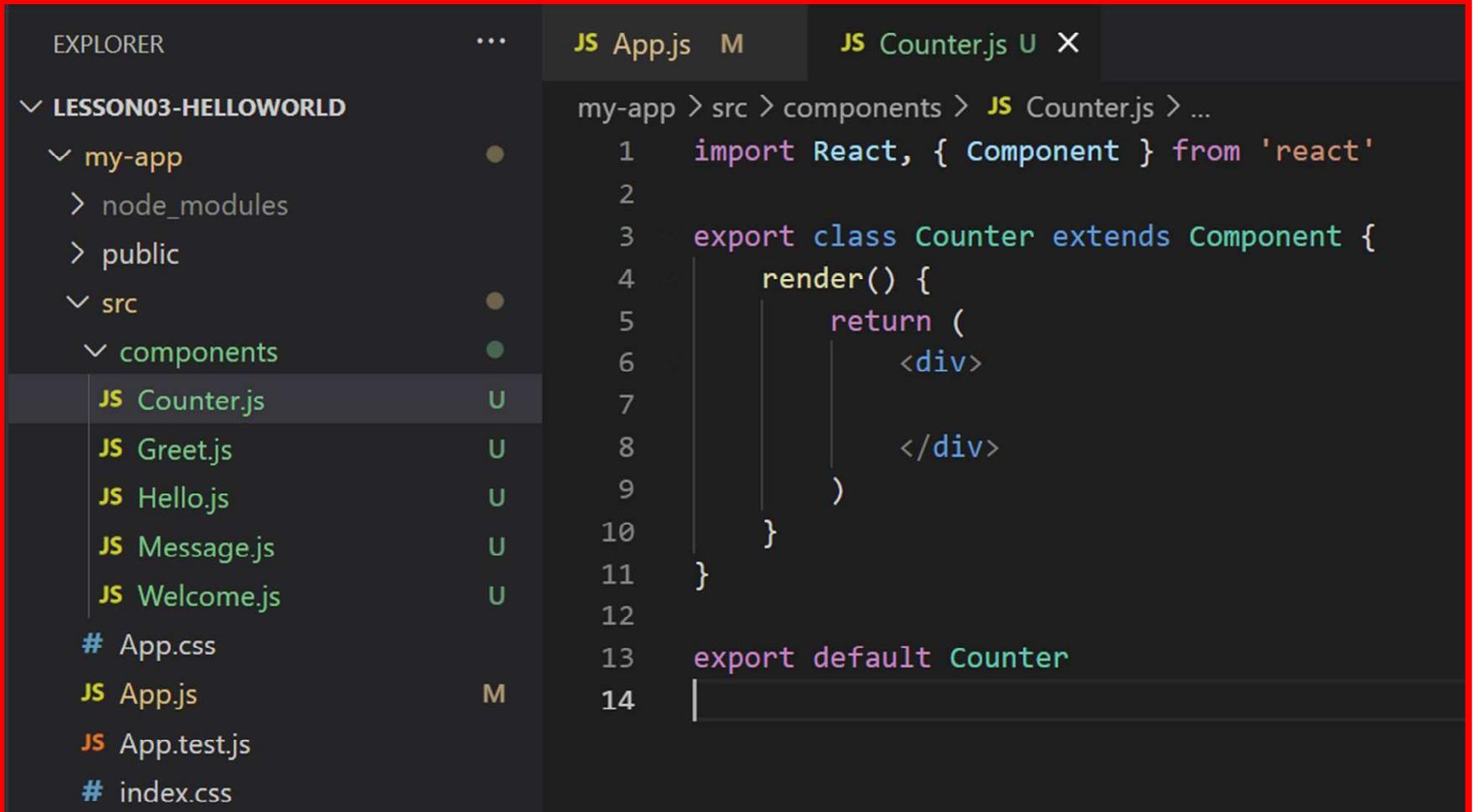
setState

## setState()

- The primary way that you make UI updates to your React applications is through a call to the `setState()` function.
- This function will perform a *shallow merge* between the new state that you provide and the previous state, and will trigger a re-render of your component and all decedents

# Counter Component

- Use Emmet Abbr [ rce ]



The screenshot shows a dark-themed VS Code interface. The Explorer sidebar on the left lists a project structure under 'LESSON03-HELLOWORLD'. The 'src' folder contains 'components' and several JavaScript files: Counter.js, Greet.js, Hello.js, Message.js, and Welcome.js. The Counter.js file is currently selected. The Editor pane on the right displays the following code:

```
my-app > src > components > JS Counter.js > ...
1 import React, { Component } from 'react'
2
3 export class Counter extends Component {
4   render() {
5     return (
6       <div>
7
8       </div>
9     )
10  }
11}
12
13 export default Counter
14 |
```

JS App.js M JS Counter.js U X

```
my-app > src > components > JS Counter.js > [0] default
1 import React, { Component } from 'react'
2
3 class Counter extends Component {
4     render() {
5         return (
6             <div>
7                 Count
8             </div>
9         )
10    }
11 }
12
13 export default Counter
14
```

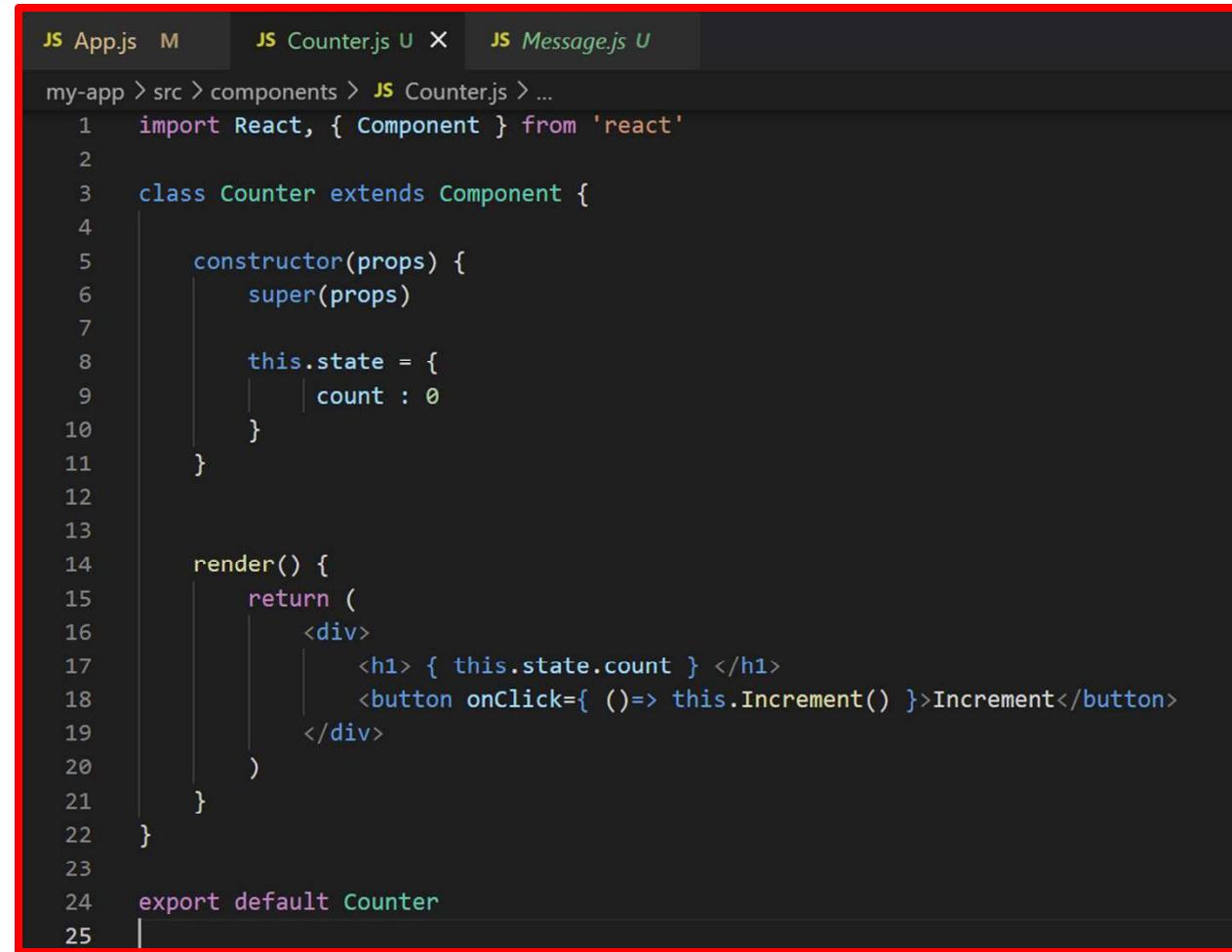
JS App.js M X JS Counter.js U

```
my-app > src > JS App.js > [0] App > [0] render
1 import React, { Component } from 'react';
2 import logo from './logo.svg';
3 import './App.css';
4 import { Greet } from './components/Greet';
5 import Welcome from './components/Welcome';
6 import Hello from './components/Hello';
7 import Message from './components/Message';
8 import Counter from './components/Counter';
9
10 class App extends Component {
11     render() {
12         return (
13             <div className="App">
14                 Count
15             </div>
16         )
17     }
18 }
19
20 export default App
21
```

dings - OneDr... How JSON Schema... YouTube Maps Infinity Meta...

Count

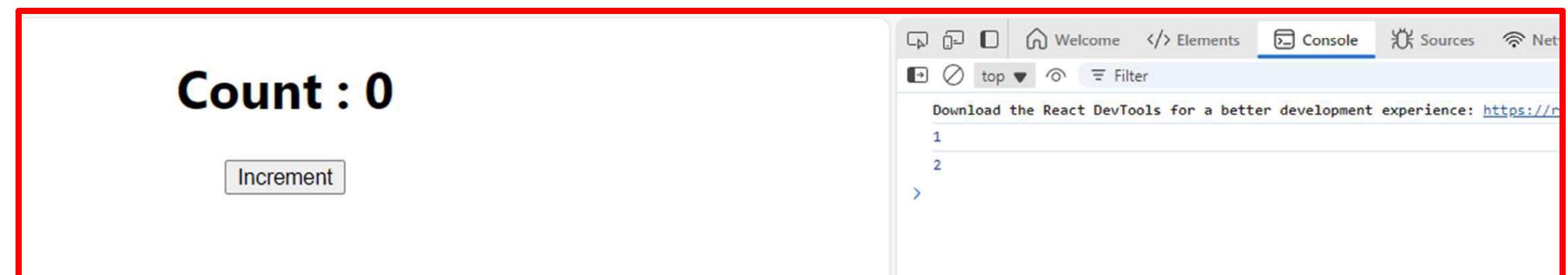
- We need is a count state to keep track of the counter value
  - Use rconst in Emmet Abbr



```
JS App.js M JS Counter.js U X JS Message.js U
my-app > src > components > JS Counter.js > ...
1 import React, { Component } from 'react'
2
3 class Counter extends Component {
4
5     constructor(props) {
6         super(props)
7
8         this.state = {
9             count : 0
10        }
11    }
12
13
14    render() {
15        return (
16            <div>
17                <h1> { this.state.count } </h1>
18                <button onClick={ ()=> this.Increment() }>Increment</button>
19            </div>
20        )
21    }
22
23
24    export default Counter
25 |
```

## What Happens if we Don't Use setState to Update and Directly Update it

```
increment() {  
    this.state.count = this.state.count + 1  
    console.log(this.state.count);  
}
```



- This is the main reason Why we should not modify State Directly
- The only place where you can assign this.state is the constructor any other time to change the state set state method has to be used

```
increment() {
  this.setState({
    count: this.state.count + 1
  })
  console.log(this.state.count);
}
```

The screenshot shows a browser window with two main parts. On the left, a React component is displayed with the text "Count : 3" and a button labeled "Increment". On the right, the browser's developer tools are open, specifically the "Console" tab. The console output shows the following sequence of numbers: 0, 1, 2, and then a blue arrow pointing down, indicating more entries.

```
0
1
2
```

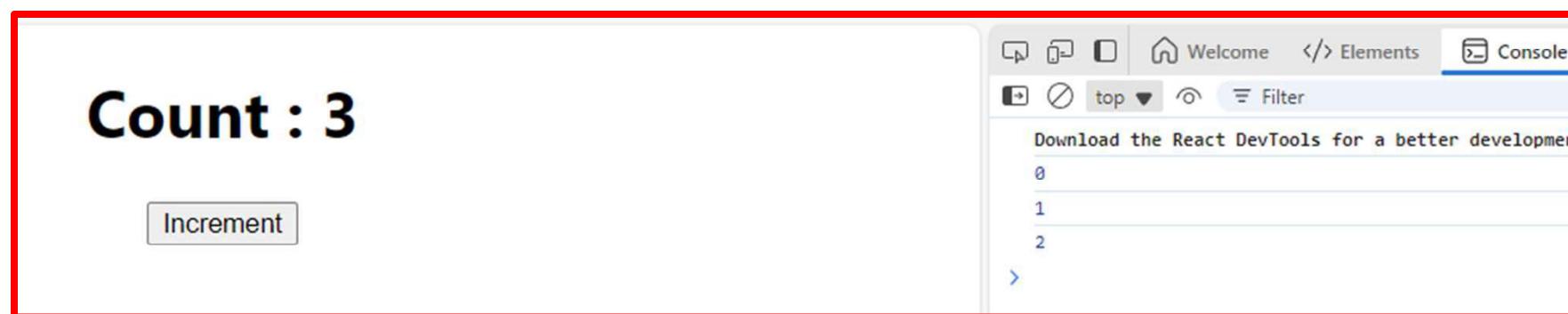
## Parameters

1. **updater:** It can be an object with a number of key-value pairs that should be merged into the state or a function that returns such an object.
2. **callback (optional):** a function which will be executed after `setState()` has been executed successfully.

Due to the fact that calls to `setState()` are not guaranteed by React to be atomic, this can sometimes be useful if you want to perform some action after you are positive that `setState()` has been executed successfully.

## Observe ....

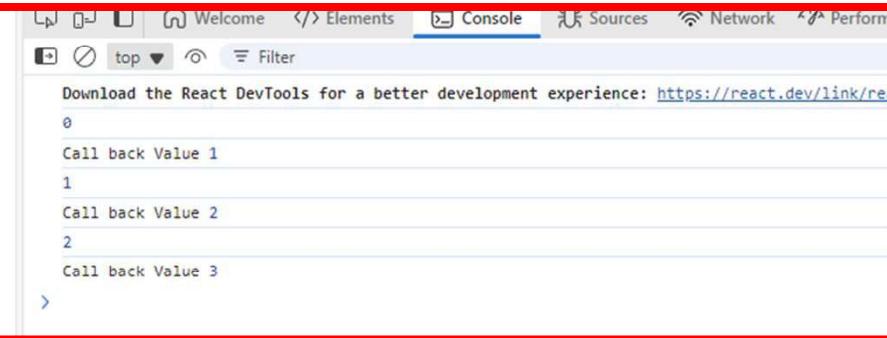
- There is one detail to observe with said state in the browser you can see that when we click on increment the value is 1 but if you take a look at the console the value is 0
- So the console value is 1 less than the rendered value and this is because calls to set state are asynchronous so what is happening is `console.log` over here is being called before the state is actually set
- Many a times in your application you might want to execute some code only after the state has been updated to handle such a situation you can pass in a callback function as the second parameter to the set state method so the set state method has two parameters the **first parameter** is the **state object** and **the second parameter** now is the **callback function** the callback function will be an arrow function



```
increment() {
  this.setState({
    count: this.state.count + 1
  },
  () => {
    console.log('Call back Value', this.state.count)
  }
)
console.log(this.state.count);
}
```

Count : 3

Increment



- whenever you need to execute some code after the state has been changed do not place that code right after the said state method instead place that code within the callback function that is passed as a second parameter to the said state method.

## Another UseCase

```
23
24     incrementFive() {
25         this.increment()
26         this.increment()
27         this.increment()
28         this.increment()
29         this.increment()
30     }
31     render() {
32         return (
33             <div>
34                 <h1> Count : {this.state.count} </h1>
35                 <button onClick={() => this.incrementFive()}>Increment</button>
36             </div>
37         )
38     }
}
```

Count : 2

Increment

Download the React DevTools for a better development experience: <https://react.devtools.dev>

0
0
0
0
0
⑤ Call back Value 1
1
1
1
1
1
⑥ Call back Value 2
>

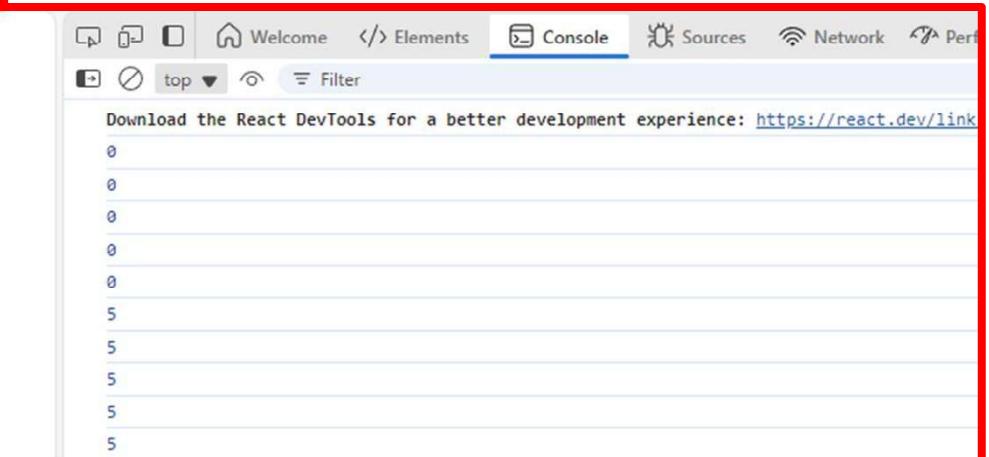
## Very Important Point to Note

- when we click on the button you can see that the value changes to one instead of changing to five and in the console strangely zero is logged five times and even the callback value of one is logged five times.
- Now why is this behavior is because react may group multiple set state calls into a single update for better performance so what happens in our scenario is that all the five set state calls are done in one single go and the updated value does not carry over between the different calls so whenever you have to update the state based on the previous state we need to pass a function as an argument to set state method instead of passing in an object so this dot said state and as the argument we pass in a function

```
increment() {
    // this.setState({
    //     count: this.state.count + 1
    // },
    // () => {
    //     console.log('Call back Value', this.state.count)
    // }
    this.setState(prevState => ({
        count: prevState.count + 1
    }))
    console.log(this.state.count);
}
```

Count : 10

Increment



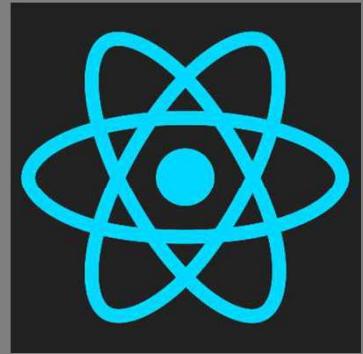
- When you have to update the state based on the previous state make sure to pass in a function as an argument instead of the regular object the function has access to the previous state which can be used to calculate the new state

### setState

Always make use of setState and never modify the state directly.

Code has to be executed after the state has been updated ? Place that code in the call back function which is the second argument to the setState method.

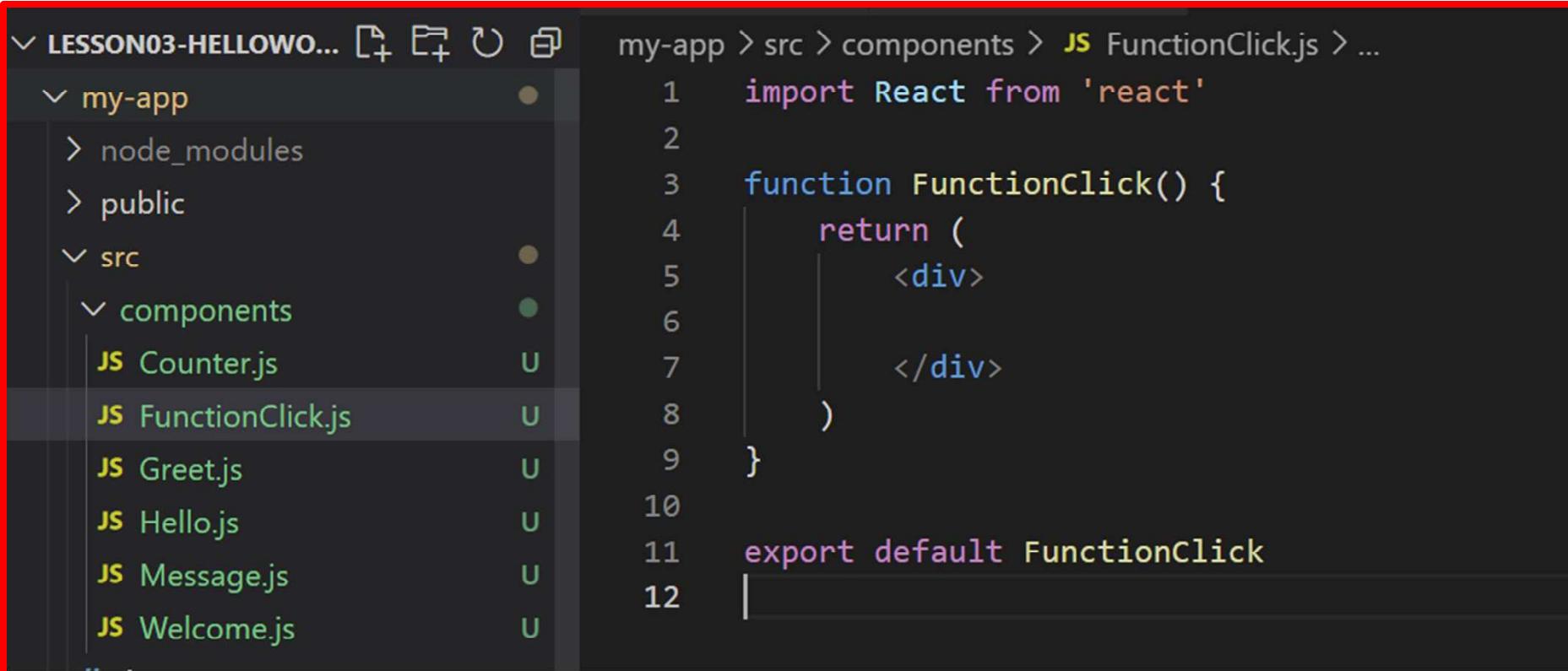
When you have to update state based on the previous state value, pass in a function as an argument instead of the regular object.



# Event Handling

- Any web application you create typically tends to have user interaction when the user interacts with your application events are fired
  - Eg: mouse clicks, mouse over, key press change event and so on the application must handle such events and execute the necessary code

➤ Create a Functional Component [ rfce is Emmet Abbr ]



The screenshot shows a code editor with a red border around the left sidebar and top header. The sidebar displays the project structure:

- LESSON03-HELLOWORLD
- my-app
  - node\_modules
  - public
  - src
    - components
      - Counter.js
      - FunctionClick.js**
      - Greet.js
      - Hello.js
      - Message.js
      - Welcome.js

The current file is **FunctionClick.js**, which contains the following code:

```
1 import React from 'react'
2
3 function FunctionClick() {
4     return (
5         <div>
6             </div>
7     )
8 }
9
10
11 export default FunctionClick
12 |
```

```
JS App.js M JS Counter.js U JS FunctionClick.js  
my-app > src > components > JS FunctionClick.js > ...  
1 import React from 'react'  
2  
3 function FunctionClick() {  
4     return (  
5         

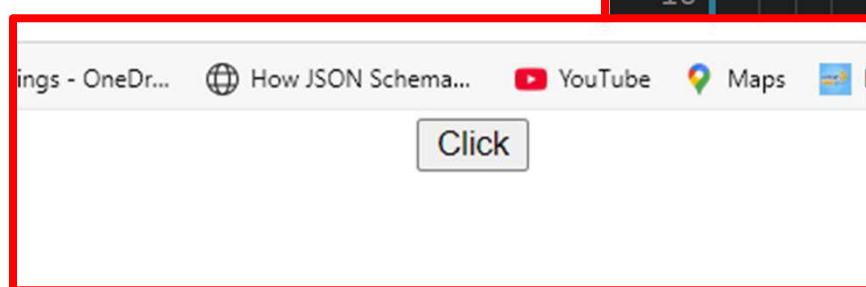
6             Click  
7

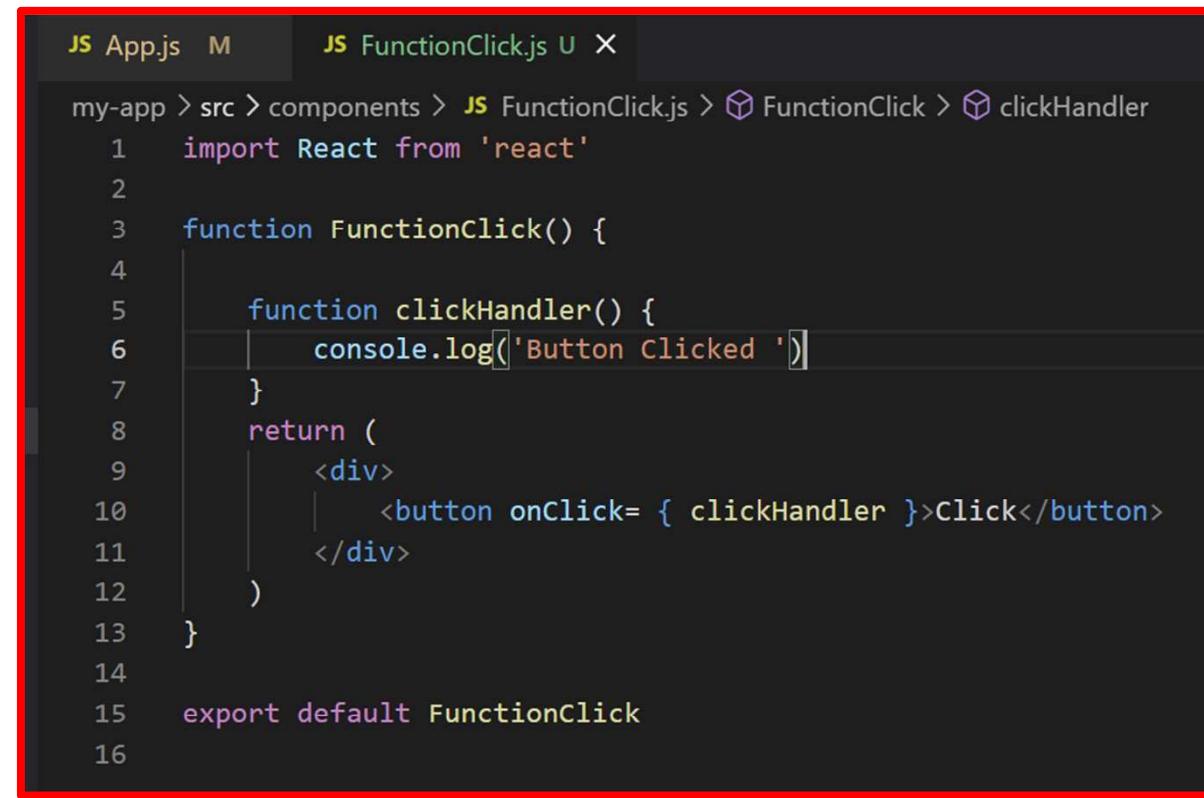
  
8     )  
9 }  
10  
11 export default FunctionClick  
12 |
```

```
JS App.js M X JS Counter.js U JS FunctionClick.js U JS Message.js U  
my-app > src > JS App.js > App > render  
1 import React, { Component } from 'react';  
2 import logo from './logo.svg';  
3 import './App.css';  
4 import { Greet } from './components/Greet';  
5 import Welcome from './components/Welcome';  
6 import Hello from './components/Hello';  
7 import Message from './components/Message';  
8 import Counter from './components/Counter';  
9 import FunctionClick from './components/FunctionClick';  
10 class App extends Component {  
11     render() {  
12         return (  
13             

14                 <FunctionClick />  
15                 /* <Counter /> */  
16

  
17         );  
18     }  
19 }
```

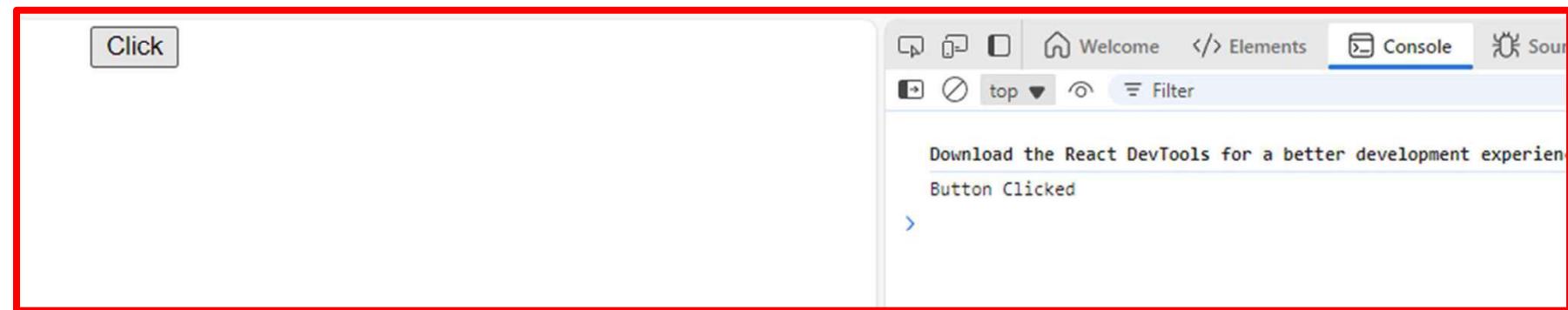




JS App.js M JS FunctionClick.js U X

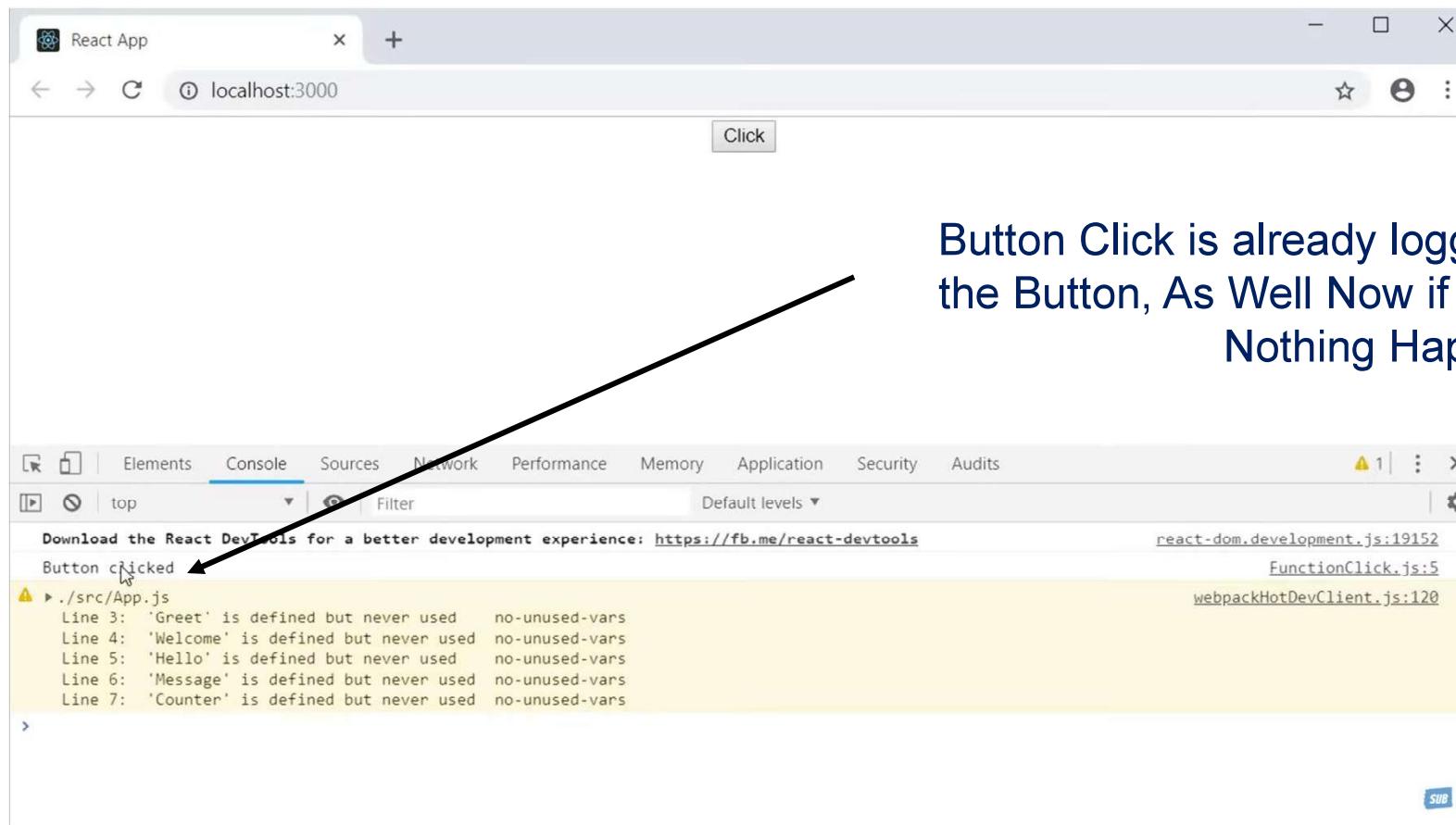
my-app > src > components > JS FunctionClick.js > FunctionClick > clickHandler

```
1 import React from 'react'
2
3 function FunctionClick() {
4
5     function clickHandler() {
6         console.log('Button Clicked ')
7     }
8     return (
9         <div>
10            <button onClick= { clickHandler }>Click</button>
11        </div>
12    )
13}
14
15 export default FunctionClick
16
```



## Never Use () in the Handler Function

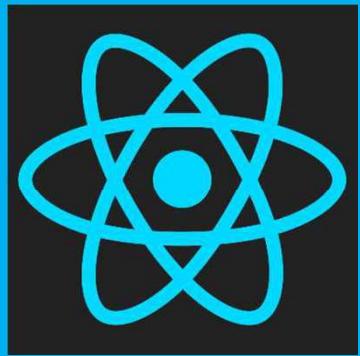
- If you do add parentheses it becomes a function call and that is not what we want we want the handler to be a function and not a function call.



Button Click is already logged Even Prior Click  
the Button, As Well Now if We Click the Button  
Nothing Happens

# Event's in Class Component

```
JS App.js      JS FunctionClick.js    JS ClassClick.js ×
hello-world ▶ src ▶ components ▶ JS ClassClick.js ▶ ClassClick ▶ render
1  import React, { Component } from 'react'
2
3  class ClassClick extends Component {
4    clickHandler() {
5      console.log('Clicked the button')
6    }
7
8    render() {
9      return (
10        <div>
11          <button onClick={this.clickHandler}>Click me</button>
12        </div>
13      )
14    }
15  }
16
17  export default ClassClick
18
```



# Event Binding

## Event Binding

- How to bind event handlers in react components ?
  - The reason we bind event handlers in react purely because of the way this keyword works in JavaScript it is not because of how react works

The screenshot shows a code editor with two tabs: "JS App.js" and "JS EventBind.js". Both tabs have a red border around them.

**JS App.js**

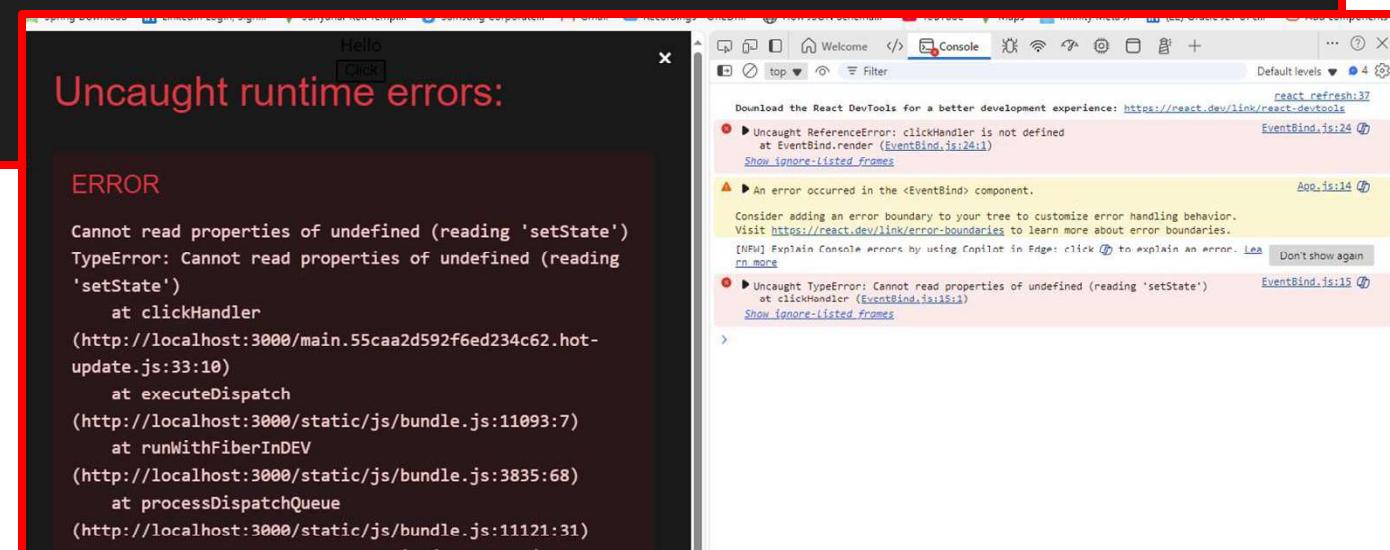
```
hello-world > src > components > JS EventBind.js > EventBind > render
1 import React, { Component } from 'react'
2
3 class EventBind extends Component {
4     render() {
5         return (
6             <div>
7                 <button>Click</button>
8             </div>
9         )
10    }
11 }
12
13 export default EventBind
14
```

**JS EventBind.js**

```
hello-world > src > JS App.js > App > render
4 import Welcome from './components/Welcome'
5 import Hello from './components/Hello'
6 import Message from './components/Message'
7 import Counter from './components/Counter'
8 import FunctionClick from './components/FunctionClick'
9 import ClassClick from './components/ClassClick'
10 import EventBind from './components/EventBind'
11
12 class App extends Component {
13     render() {
14         return (
15             <div className="App">
16                 <EventBind />
17                 {/* <FunctionClick /> */}
18             </div>
19         )
20     }
21 }
22
23 export default App
24
```

```
JS App.js JS EventBind.js x
hello-world > src > components > JS EventBind.js > EventBind > render
  4  constructor(props) {
  5    super(props)
  6    this.state = {
  7      message: 'Hello'
  8    }
  9  }
10 }
11
12 render() {
13   return (
14     <div>
15       <div>{this.state.message}</div>
16       <button>Click</button>
17     </div>
18   )
19 }
20 }
```

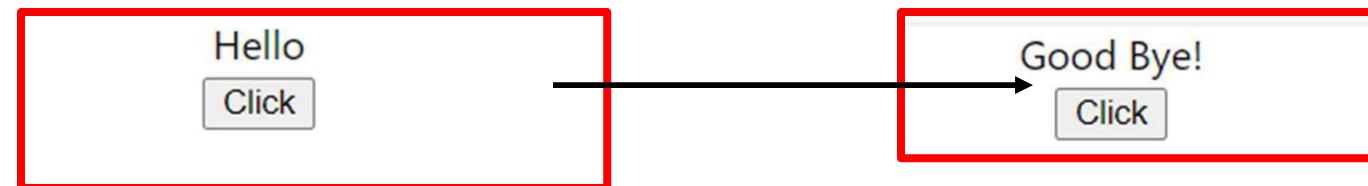
```
14     clickHandler() {
15         this.setState({
16             message : 'Good Bye!'
17         })
18     }
19
20     render() {
21         return (
22             <div>
23                 <div> { this.state.message }</div>
24                 <button onClick= { this.clickHandler }>Click</button>
25             </div>
26         )
27     }
28 }
```



## Different variants of binding the Event Handler

- The first option we have is to use the bind keyword and bind the handler in the render method

```
clickHandler() {  
  this.setState({  
    message : 'Good Bye!'  
  })  
  console.log(this)  
}  
  
render() {  
  return (  
    <div>  
      <div> { this.state.message }</div>  
      <button onClick= { this.clickHandler.bind(this) }>Click</button>  
    </div>  
  )  
}
```



Good Bye!

Click

The screenshot shows a browser's developer tools console tab. A red box highlights the entire content area. In the top bar, there are icons for back, forward, search, and refresh, followed by "Welcome" and "Console". Below the bar are buttons for "top", "Filter", and "Default levels". The console output starts with a message about downloading React DevTools. It then displays the state of a component named "EventBind". The state object has properties: context, props, refs, state, and updater. The state property contains a message: "Good Bye!". The updater property is a function object with methods: enqueueSetState, enqueueReplaceState, and enqueueForceUpdate. The \_reactInternalInstance property is a FiberNode object with properties: isMounted and replaceState. The \_reactInternals property is a FiberNode object with properties: tag, key, stateNode, elementType, type, and ... (ellipsis). The [[Prototype]] property points to a Component object.

```
Download the React DevTools for a better development experience: https://react.dev/link/react-devtools
EventBind.js:1
▶ EventBind {props: {...}, context: {...}, refs: {...}, updater: {...}, state: {...}, ...} ⓘ
▶ context: {}
▶ props: {}
▶ refs: {}
▶ state: {message: 'Good Bye!'}
▶ updater: {enqueueSetState: f, enqueueReplaceState: f, enqueueForceUpdate: f}
▶ _reactInternalInstance: {}
▶ _reactInternals: FiberNode {tag: 1, key: null, stateNode: EventBind, elementType: f, type: f, ...}
  isMounted: ...
  replaceState: ...
▶ [[Prototype]]: Component
>
```

## Second Approach for Binding Events [ Using Arrow Function ]

- It could be troublesome in large applications and components that contain nested children components so approach one binding in the render method the second approach is to use arrow functions in the render method

```
render() {
  return (
    <div>
      <div>{this.state.message}</div>
      /* <button onClick={this.cl} (method) EventBind.clickHandler(): void >
      <button onClick={() => this.clickHandler()}>Click</button>
    </div>
  )
}
```

The screenshot shows a code editor interface with a red border around the main workspace. On the left is the Explorer sidebar, which lists the project structure under the REACT folder. The current file being edited is SetStateDemo.js, located in the components folder of the state-demo directory. The code editor displays the following code:

```
state-demo > src > components > JS SetStateDemo.js > Greeting > render
1 import React from 'react'
2 class Greeting extends React.Component {
3     constructor(props) {
4         super(props);
5         this.click = this.click.bind(this);
6         // Set initial state (ONLY ALLOWED IN CONSTRUCTOR)
7         this.state = {
8             greeting: 'Hello!'
9         };
10    }
11    click(e) {
12        this.setState({
13            greeting: 'Hello World!'
14        });
15    }
16    render() {
17        return (
18            <div>
19                <p>{this.state.greeting}</p>
20                <button onClick={this.click}>Click me</button>
21            </div>
22        );
23    }
24}
25 export default Greeting;
```

- A React component with a "managed" input field. Whenever the value of the input field
- changes, an event handler is called which updates the state of the component with the new value of the input field.
- The call to `setState` in the event handler will trigger a call to render updating the component in the dom.

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists project files and folders under the REACT category. The main area displays the code for `ManagedControlDemo`. The code uses a class-based component structure with a constructor, state management via `this.state`, and event handling through `onChange` on the input field.

```
state-demo > src > components > JS StateEventsManagedControls.js > [?] default
1 import React from 'react';
2
3 class ManagedControlDemo extends React.Component {
4     constructor(props) {
5         super(props);
6         this.state = { message: "" };
7     }
8     handleChange(e) {
9         this.setState({ message: e.target.value });
10    }
11    render() {
12        return (
13            <div>
14                <legend>Type something here</legend>
15                <input
16                    onChange={this.handleChange.bind(this)}
17                    value={this.state.message}
18                    autoFocus />
19                <h1>{this.state.message}</h1>
20            </div>
21        );
22    }
23 }
24 export default ManagedControlDemo
```

The image shows a development environment with a code editor and a browser window. The code editor on the left displays the file `App.js` with the following content:

```
1 import logo from './logo.svg';
2 import './App.css';
3 import ExampleComponent from './components/ExampleState';
4 import MyComponent from './components/StateDemo02';
5 import Greeting from './components/SetStateDemo';
6 import ManagedControlDemo from './components/StateEventsManagedControls';

7 function App() {
8   return (
9     <div className="App">
10      {/* <ExampleComponent /> */}
11      {/* <MyComponent /> */}
12      {/* <Greeting /> */}
13      <ManagedControlDemo />
14    </div>
15  );
16}
17
18export default App;
```

The browser window on the right displays the rendered output of the application. It shows a search bar with the placeholder "Type something here" and a text input field containing "React App". Below the search bar, the text "React App" is displayed in large, bold, black font.

- It's very important to note the runtime behavior. Every time a user changes the value in the input field
  1. handleChange will be called and so
  2. setState will be called and so
  3. render will be called

## Summary

In this lesson, you should have learned how to:

- Basics in State
- setState
- State, Events and Managed Controls



