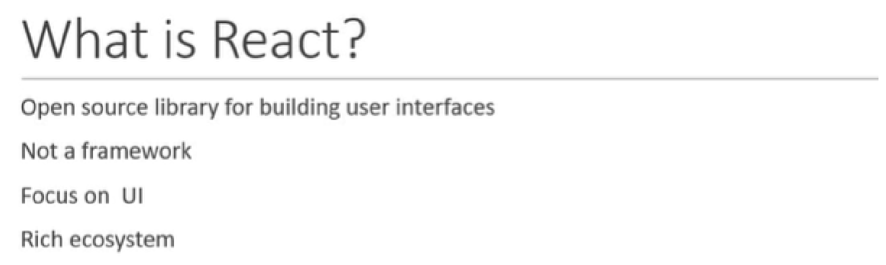
React JS Note

1. React



What is React?

Open source library for building user interfaces

Not a framework

Focus on UI

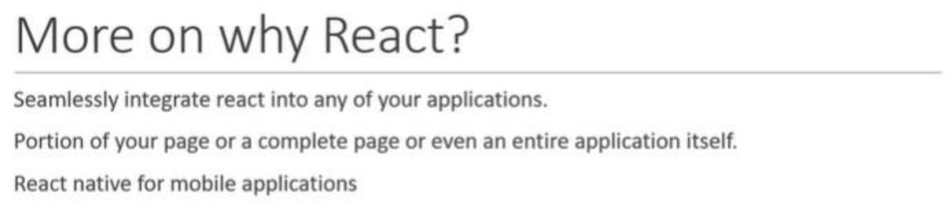
Rich ecosystem

React is declarative

Tell React what you want and React will build the actual UI

React will handle efficiently updating and rendering of the components

DOM updates are handles gracefully in React.

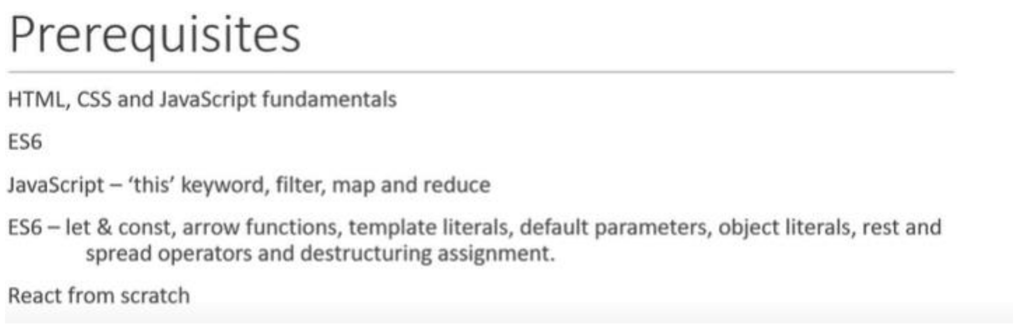


More on why React?

Seamlessly integrate react into any of your applications.

Portion of your page or a complete page or even an entire application itself.

React native for mobile applications



Prerequisites

HTML, CSS and JavaScript fundamentals

ES6

JavaScript – 'this' keyword, filter, map and reduce

ES6 – let & const, arrow functions, template literals, default parameters, object literals, rest and spread operators and destructuring assignment.

React from scratch

2. Create App

Create-react-app

npx	npm
npx create-react-app <project_name>	*npm install create-react-app -g*
npm package runner	*create-react-app<project_name>*

- npx create-react-app [project-name]

- npm install create-react-app -g <not recommand>

- npm start -> to start application

3. Project Structure

- package.json is all needs lib, start/build script, react version

4. React Component

- Component is Part of UI, reusable, can be nested inside of other Component

- Stateless Functional Componet -> JS Function return HTML

- Stateful Class Component -> ES6 Class -> Extends React.Component -> Render Method return HTML

5. Functional Component

- should only 1 Function in a Class ?

- import should be the same with function name or default export (recommend to easily understand)

- Use for -> Simple Function, Solution Without State, Responsible for UI

- Also Call Stateless/Dumb/Presentational component

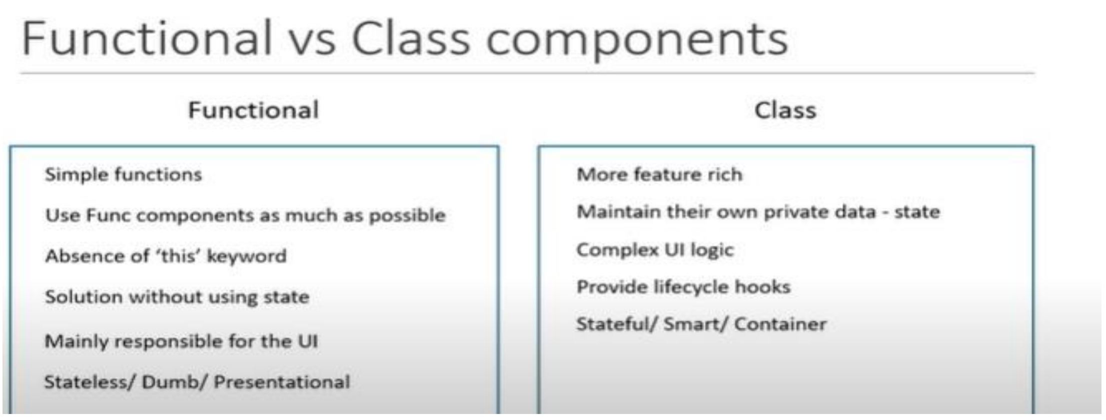
6. Class Component

- Use for -> Complex UI/logic

- Provide Lifecycle Hook

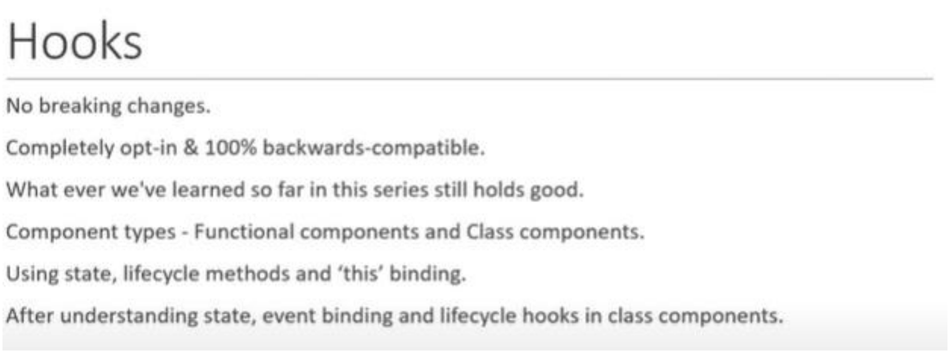
- Also call Stateful/Smart/Container component

7. Hook Update



Functional vs Class components

Functional	Class
Simple functions	More feature rich
Use Func components as much as possible	Maintain their own private data - state
Absence of 'this' keyword	Complex UI logic
Solution without using state	Provide lifecycle hooks
Mainly responsible for the UI	Stateful/ Smart/ Container
Stateless/ Dumb/ Presentational	



Hooks

No breaking changes.

Completely opt-in & 100% backwards-compatible.

What ever we've learned so far in this series still holds good.

Component types - Functional components and Class components.

Using state, lifecycle methods and 'this' binding.

After understanding state, event binding and lifecycle hooks in class components.

8. JSX

- Javascript XML

- JSX tag have tag name, attribute and children

- Make React Code to Simpler and elegant

- class replace with className

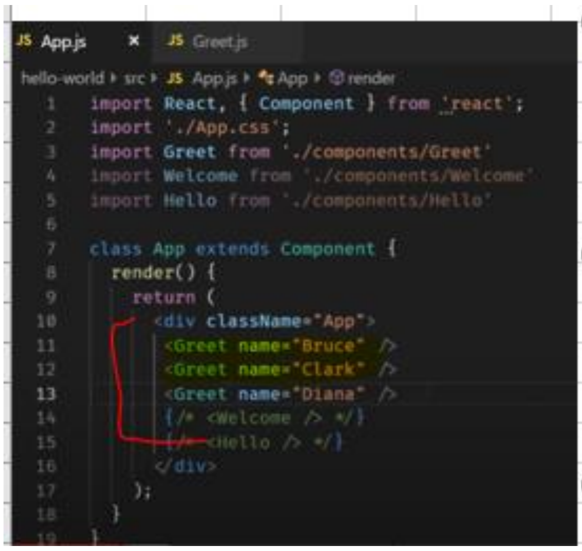
- use camelCase naming convention -> eg. onclick => onClick

9. Props

- data passing from **parent** to **child**, function parameter

- **immutable**

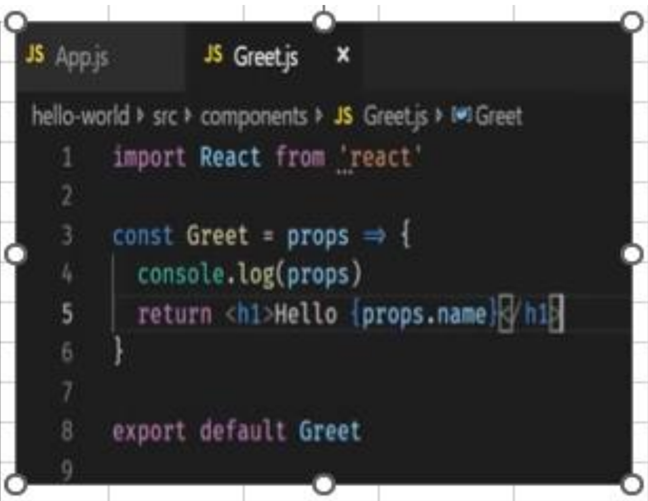
- props.name/ props.children



JS App.js × JS Greet.js

hello-world › src › JS App.js › App › render

```
import React, { Component } from 'react';
import './App.css';
import Greet from './components/Greet'
import Welcome from './components/Welcome'
import Hello from './components/Hello'

class App extends Component {
  render() {
    return (
      <div className="App">
        <Greet name="Bruce" />
        <Greet name="Clark" />
        <Greet name="Diana" />
        {/* <Welcome /> */}
        {/* <Hello /> */}
      </div>
    );
  }
}
```



JS App.js JS Greet.js ×

hello-world › src › components › JS Greet.js › Greet

```
import React from 'react'

const Greet = props => {
  console.log(props)
  return <h1>Hello {props.name}</h1>
}

export default Greet
```

10. State

- manage within the same components

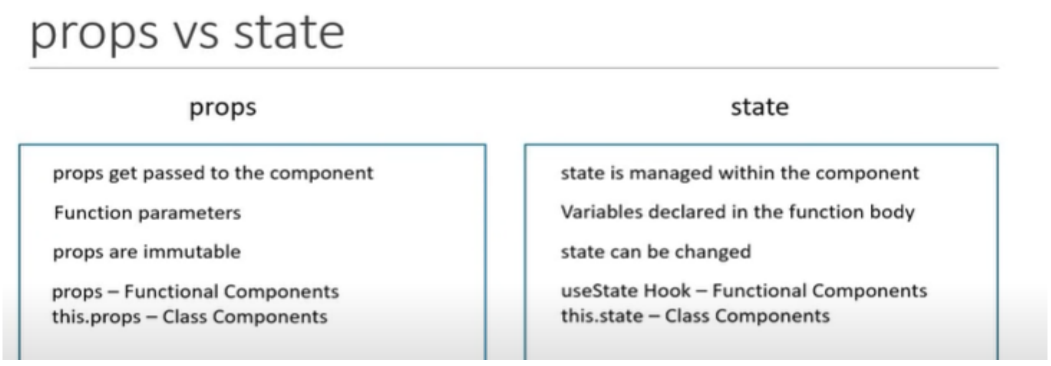
- variable declare within function body

- use **this.setState** to change state value

- **this.setState({ count: 1});**

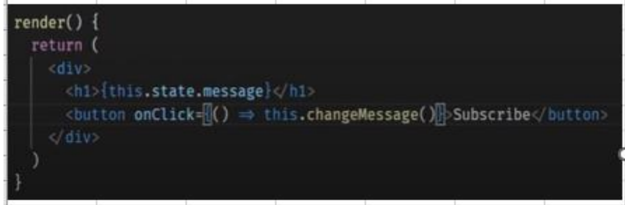
- use callback function for custom logic

- use prevState to get previous State

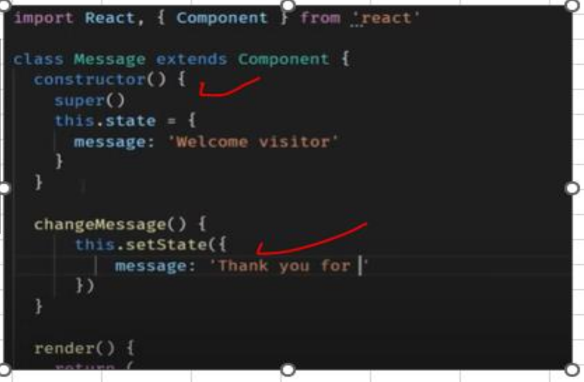


props vs state

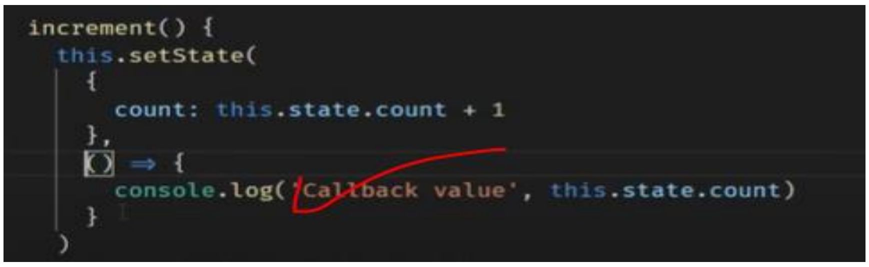
props	state
props get passed to the component	state is managed within the component
Function parameters	Variables declared in the function body
props are immutable	state can be changed
props – Functional Components this.props – Class Components	useState Hook – Functional Components this.state – Class Components



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



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

class Message extends Component {
  constructor() {
    super()
    this.state = {
      message: 'Welcome visitor'
    }
  }

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

  render() {
```



```
increment() {
  this.setState(
    {
      count: this.state.count + 1
    },
    () => {
      console.log('Callback value', this.state.count)
    }
  )
```

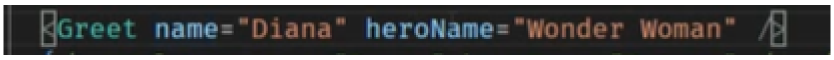
Added Custom logic in Call back

11. Destructure

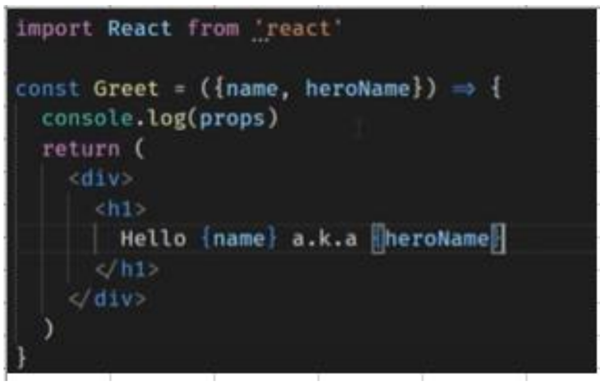
- Destructure is ES5 Feature

- function component => const Gteet = ({name, heroName})

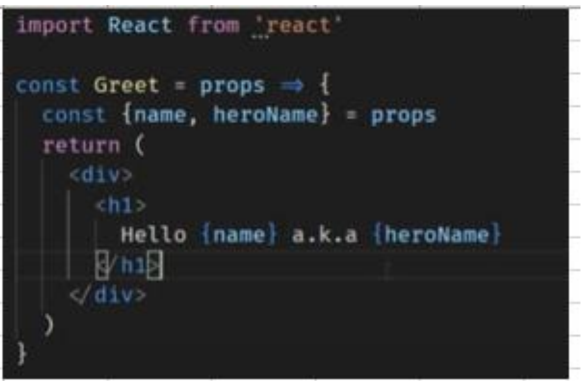
- class component => const {name, heroName} = this.props



```
<Greet name="Diana" heroName="Wonder Woman" />
```



```
import React from 'react'

const Greet = ({name, heroName}) => {
  console.log(props)
  return (
    <div>
      <h1>
        Hello {name} a.k.a {heroName}
      </h1>
    </div>
  )
}
```

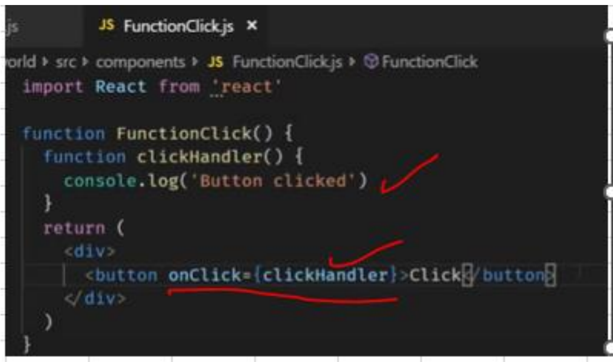


```
import React from 'react'

const Greet = props => {
  const {name, heroName} = props
  return (
    <div>
      <h1>
        Hello {name} a.k.a {heroName}
      </h1>
    </div>
  )
}
```

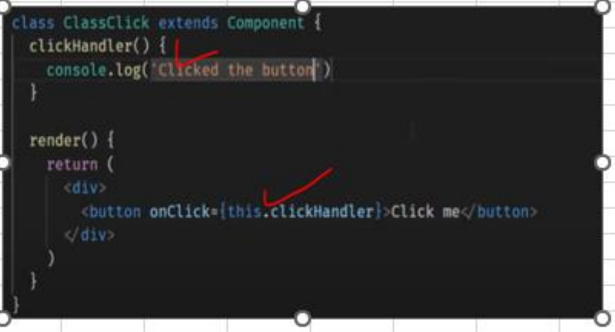
12. Event Handling

- functionalComponent -> onClick = {clickHanler} => *** do not add (), if add auto call function when init

- classComponet -> onClick = {**this**.clickHandler}



```
import React from 'react'

function FunctionClick() {
  function clickHandler() {
    console.log('Button clicked')
  }
  return (
    <div>
      <button onClick={clickHandler}>Click</button>
    </div>
  )
}
```



```
class ClassClick extends Component {
  clickHandler() {
    console.log('Clicked the button')
  }

  render() {
    return (
      <div>
        <button onClick={this.clickHandler}>Click me</button>
      </div>
    )
  }
}
```

13. Binding event Handler

- onClick = {this.clickHandler.bind(this)} // not good because of performance

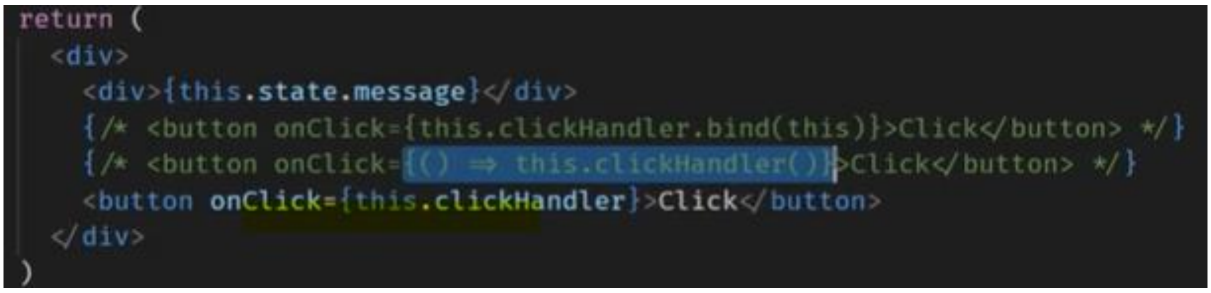
- onClick = {() => this.clickHandler()} =? *** call function and return, so need ()// not good

- // this.clickHandler = this.clickHander.bind(this)

- clickHandler = () => {

        this.setState({m : test})

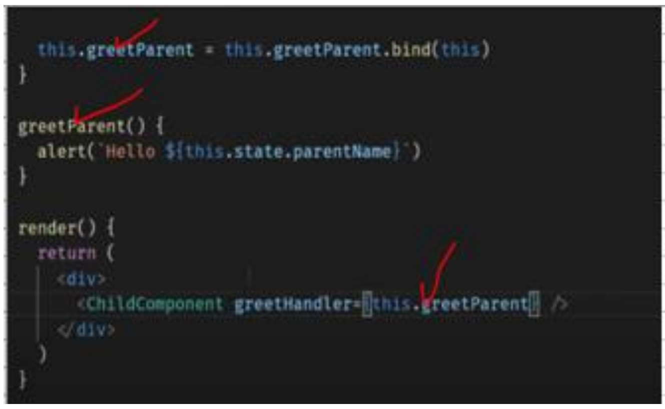
    }// better



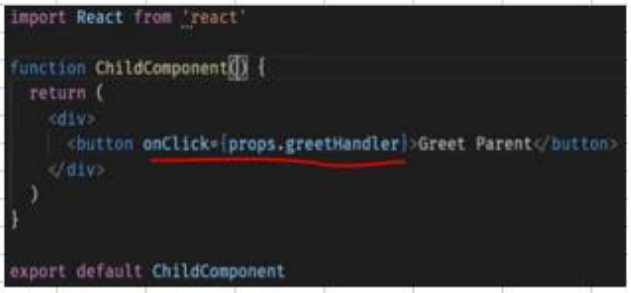
```
return (
  <div>
    <div>{this.state.message}</div>
    {/* <button onClick={this.clickHandler.bind(this)}>Click</button> */}
    {/* <button onClick={() => this.clickHandler()}>Click</button> */}
    <button onClick={this.clickHandler}>Click</button>
  </div>
)
```

14. Method as Prop

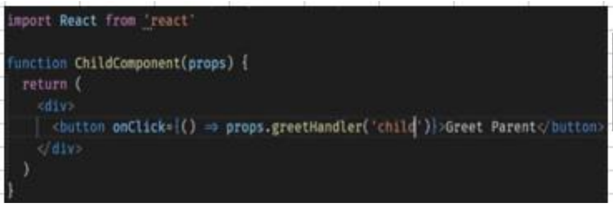
Pass Method as a prop



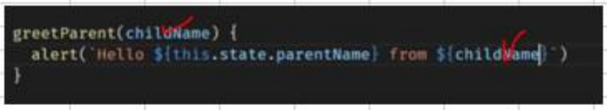
```
  this.greetParent = this.greetParent.bind(this)
}

greetParent() {
  alert(`Hello ${this.state.parentName}`)
}

render() {
  return (
    <div>
      <ChildComponent greetHandler={this.greetParent} />
    </div>
  )
}
```



```
import React from 'react'

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

export default ChildComponent
```

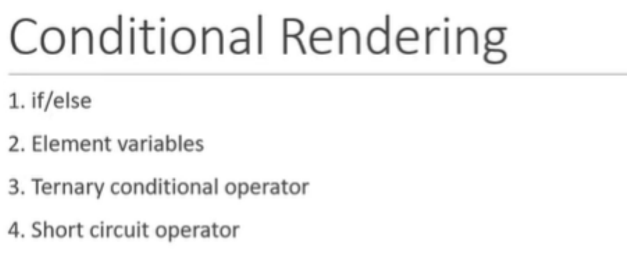


```
import React from 'react'

function ChildComponent(props) {
  return (
    <div>
      <button onClick={() => props.greetHandler('child')}>Greet Parent</button>
    </div>
  )
}
```



```
greetParent(childName) {
  alert(`Hello ${this.state.parentName} from ${childName}`)
}
```

15. <u>Conditional Rendering</u>



Conditional Rendering

1. if/else
2. Element variables
3. Ternary conditional operator
4. Short circuit operator

- if/else => add if/else condition in render return method and if change condition need to reload/re-render the jsx

- Element Variable   => declare variable and update value based on if/else condition in render method

- Ternary operator   => return (this.state.login ? <div>login</div> : <div>guest</div>)

- Short Circuit Operator => return (this.state.login && <div>login</div>)

16. <u>List Rendering</u>

- names.map(name => <h3>{name}</h3>)

- const nameList = names.map(name => <h3>{name]</h3>);

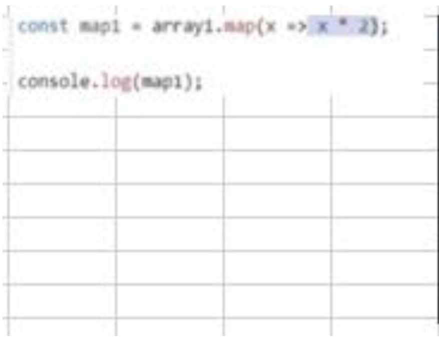
- return <div>{nameList}</div>

# for parent and child

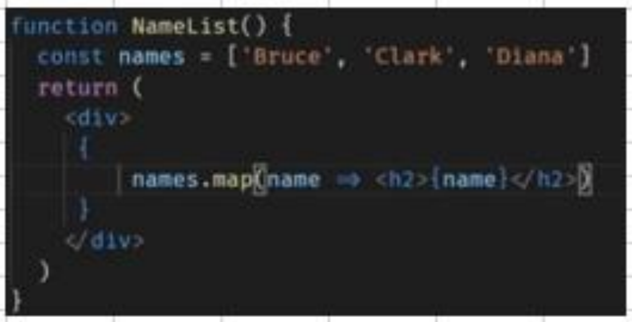
- parent => const personList = persons.map (person => <Person person={person} />

    return <div>{personList}</div>

- child => function Person(person) { return ({person.name}); // but key is missing for the list



```
const map1 = array1.map(x => x * 2);

console.log(map1);
```



```
function NameList() {
  const names = ['Bruce', 'Clark', 'Diana']
  return (
    <div>
      {
        names.map(name => <h2>{name}</h2>)
      }
    </div>
  )
}
```

17. <u>List and Key</u>

- handline ui efficient

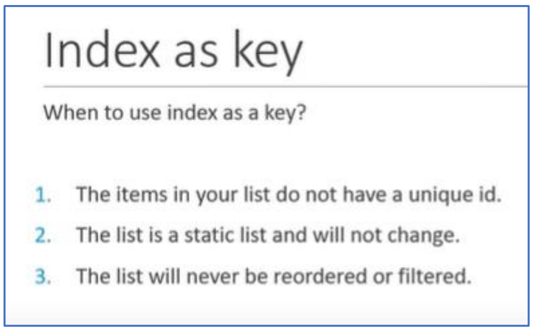
- to detect which element is added/remove

- parent => const personList = persons.map (person => <Person key={person.id} person={person} />

18. <u>Index as a key</u>

- const nameList = names.map((name, index) => <h3 key={index}>{name]</h3>);

- use for => 1. item don't have unique id, 2. list is static and will not change, 3. list will be never filter or re-order


Index as key

When to use index as a key?

1. The items in your list do not have a unique id.
2. The list is a static list and will not change.
3. The list will never be reordered or filtered.

19. Styling React JS

- 1. css stylesheet, 2. inline style, 3. CSS modules, 4. CSS in JS lib

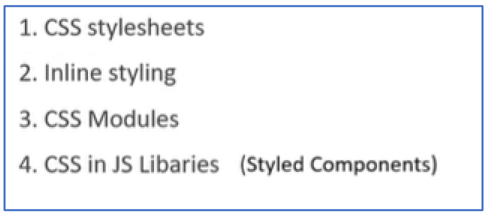
- import CSS class => import './style.css';

- let className = props.primary ? 'primary' : '';

- <h1 className = {className}> test </h1>

import styles from './appStyles.module.css'

<h1 className={styles.success}>success</h1>


1. CSS stylesheets
2. Inline styling
3. CSS Modules
4. CSS in JS Libaries (Styled Components)

20. Form Handling

- handleUsernameChange = (event) => {

    this.setState({ username: event.target.value });

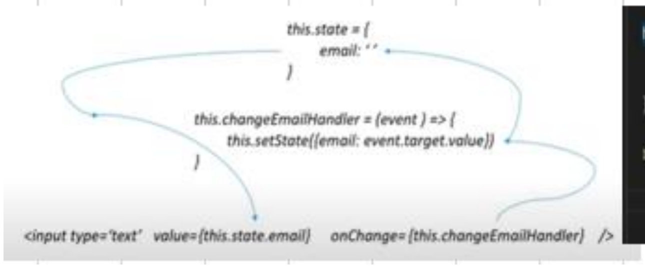
}

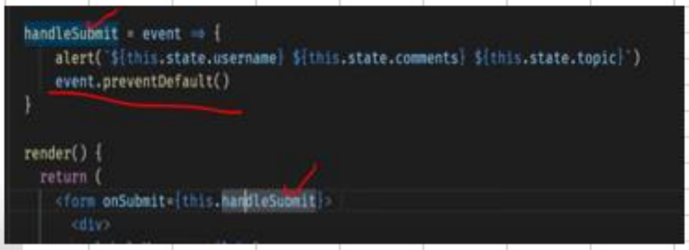
- <input value={this.state.username} onChange={this.handleUsernameChange} />

- submit button will trigger to default HTML form submit and need to disable

- <form onSubmit= {this.handleSubmit}>

- handleSubmit = event = { alert('${this.state.username}')}


```
this.state = {
    email: ''
}

this.changeEmailHandler = (event ) => {
    this.setState({email: event.target.value})
}

<input type='text'  value={this.state.email}   onChange= {this.changeEmailHandler}  />
```


```
handleSubmit = event => {
    alert(`${this.state.username} ${this.state.comments} ${this.state.topic}`)
    event.preventDefault()
}

render() {
  return (
    <form onSubmit={this.handleSubmit}>
      <div>
```

21. Lifecycle

- **Mounting** -> component is being created and inserted into DOM

  ->eg. constructor, static getDerivedStateFromProps, render and componentDidMount

- **Updating** -> component is being re-render as a result of changing props/state

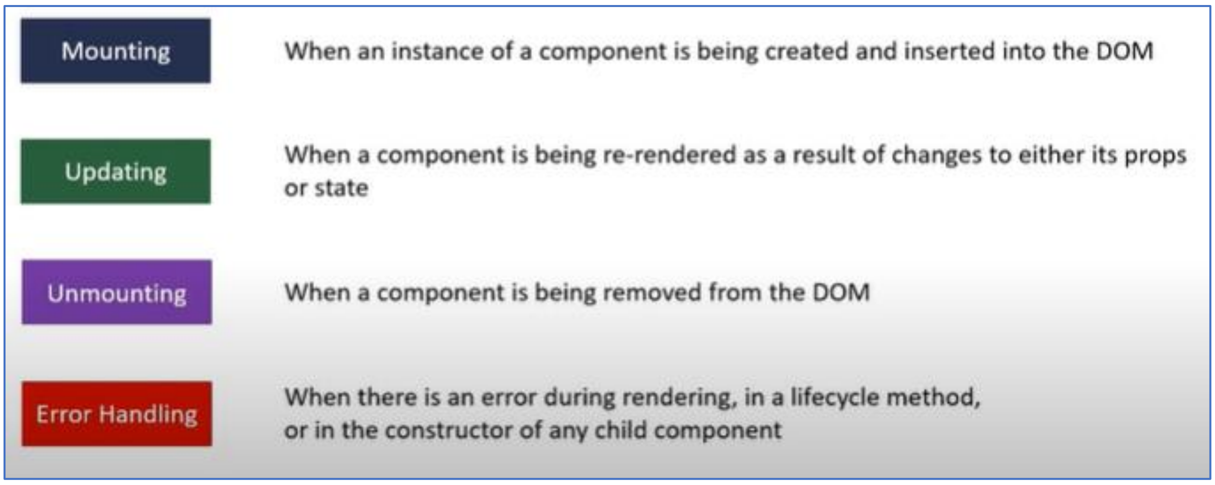
  ->eg. static getDerivedStateFromProps, shouldComponentUpdate, render, getSnapshotBeforeUpdate, componentDidUpdate

- **Unmounting** -> component is being removed from the DOM

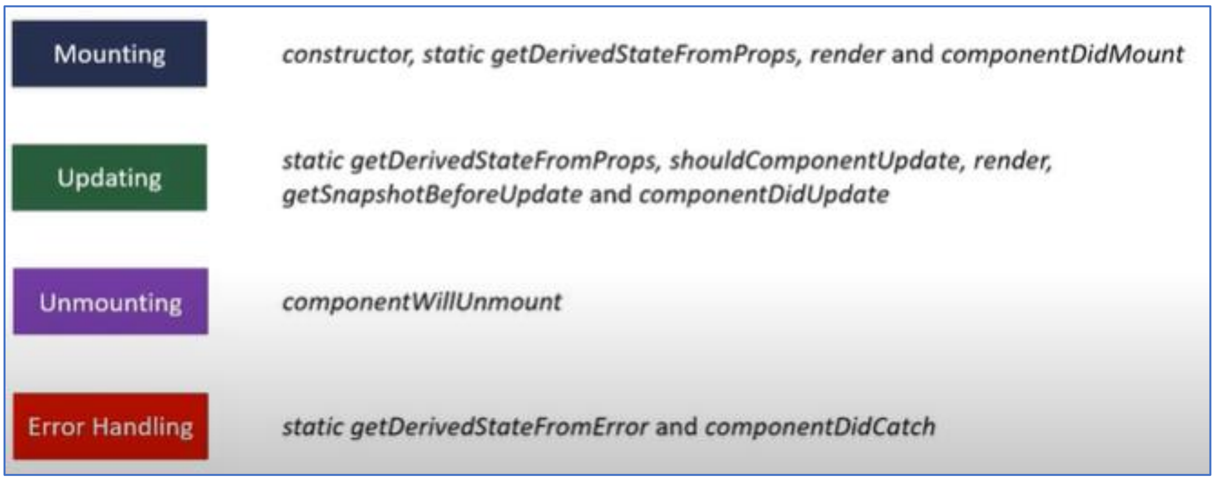
  -> eg. componentWillUnmount

- **Error Handling**-> Error during rendering, in a lifecycle method/ constructor or child component

  -> eg. static getDerivedStateFromError and componentDidCatch

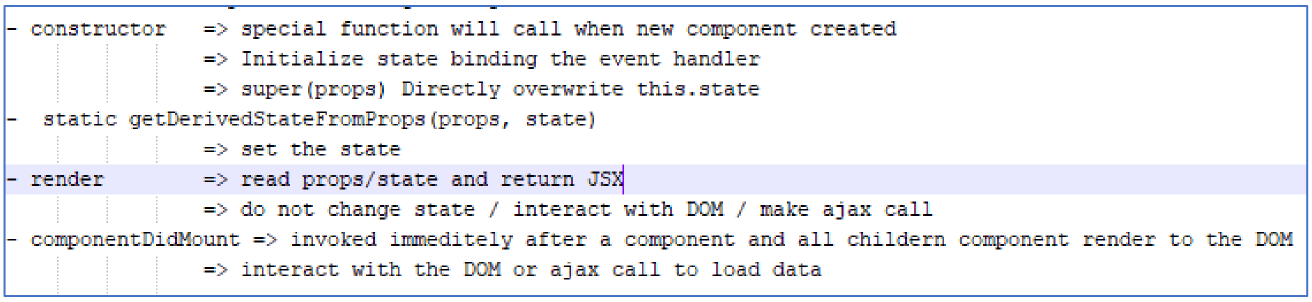


Mounting	When an instance of a component is being created and inserted into the DOM
Updating	When a component is being re-rendered as a result of changes to either its props or state
Unmounting	When a component is being removed from the DOM
Error Handling	When there is an error during rendering, in a lifecycle method, or in the constructor of any child component

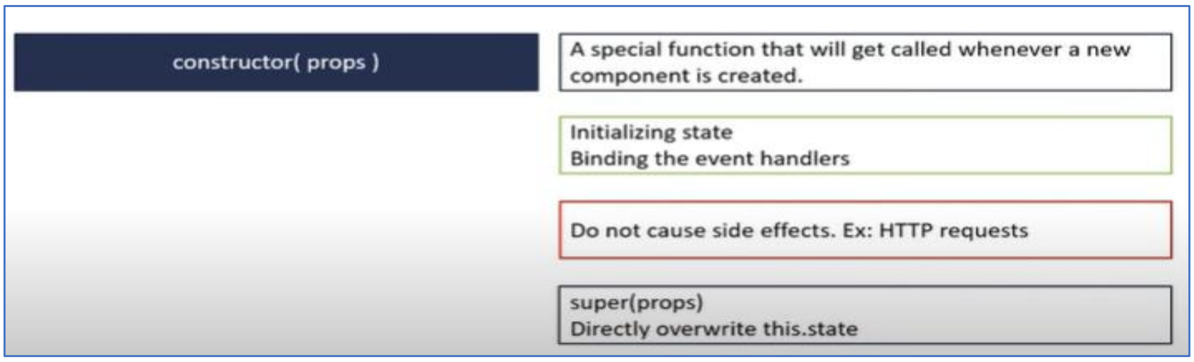


Mounting	*constructor, static getDerivedStateFromProps, render* and *componentDidMount*
Updating	*static getDerivedStateFromProps, shouldComponentUpdate, render, getSnapshotBeforeUpdate* and *componentDidUpdate*
Unmounting	*componentWillUnmount*
Error Handling	*static getDerivedStateFromError* and *componentDidCatch*

22. Component Mounting Lifecycle



```
- constructor   => special function will call when new component created
                => Initialize state binding the event handler
                => super(props) Directly overwrite this.state
-  static getDerivedStateFromProps(props, state)
                => set the state
- render        => read props/state and return JSX
                => do not change state / interact with DOM / make ajax call
- componentDidMount => invoked immeditely after a component and all childern component render to the DOM
                => interact with the DOM or ajax call to load data
```



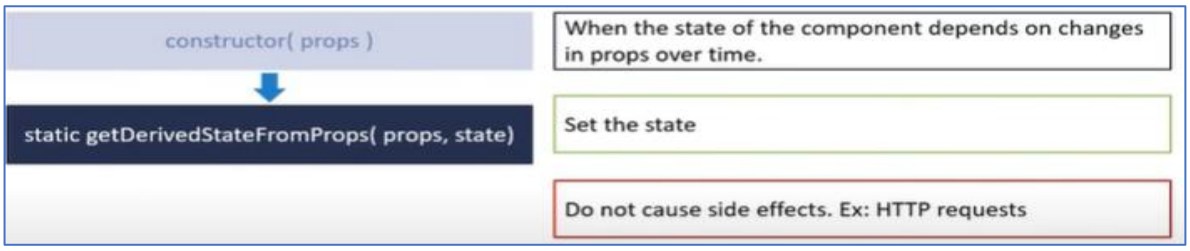
constructor(props)

A special function that will get called whenever a new component is created.

Initializing state
Binding the event handlers

Do not cause side effects. Ex: HTTP requests

super(props)
Directly overwrite this.state



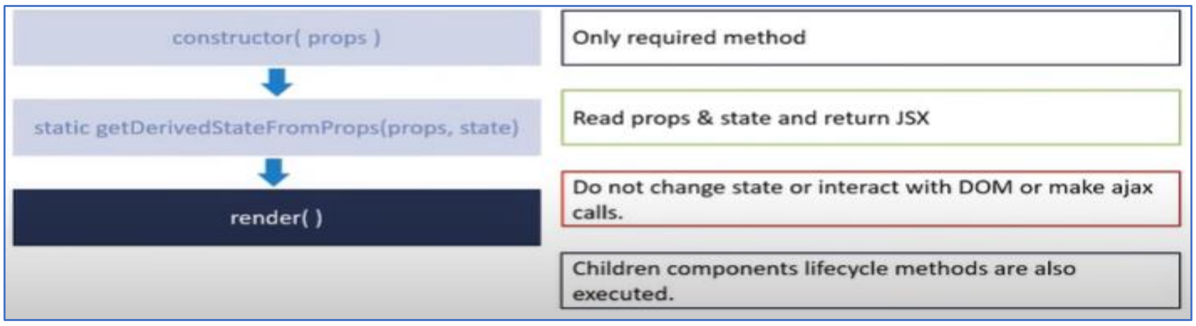
constructor(props)

static getDerivedStateFromProps(props, state)

When the state of the component depends on changes in props over time.

Set the state

Do not cause side effects. Ex: HTTP requests



constructor(props)

static getDerivedStateFromProps(props, state)

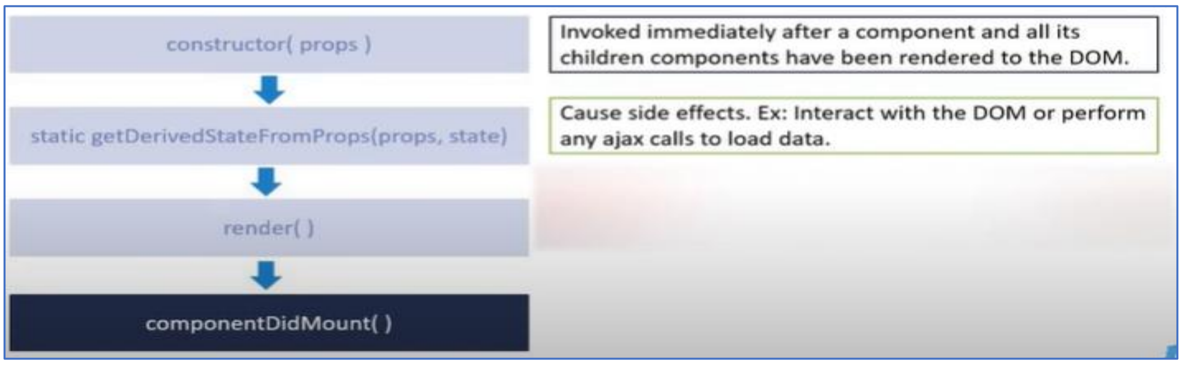
render( )

Only required method

Read props & state and return JSX

Do not change state or interact with DOM or make ajax calls.

Children components lifecycle methods are also executed.



constructor(props)

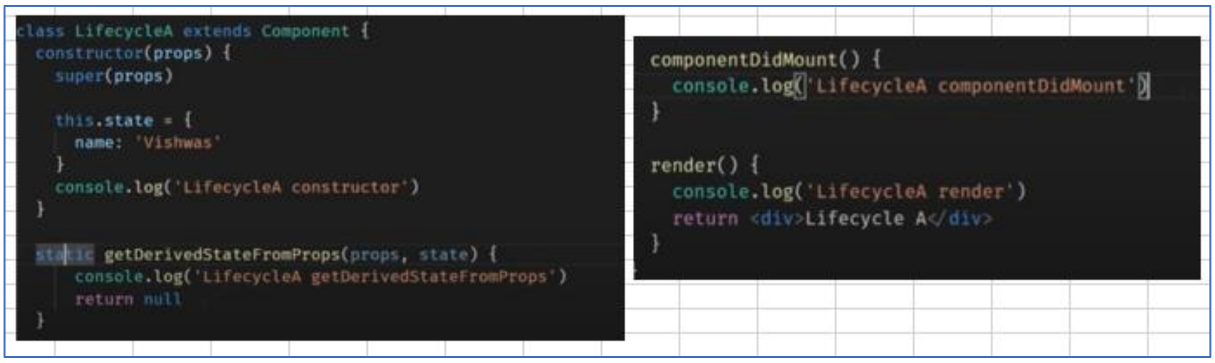
static getDerivedStateFromProps(props, state)

render( )

componentDidMount( )

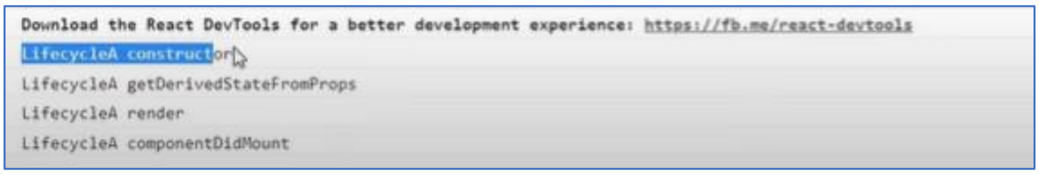
Invoked immediately after a component and all its children components have been rendered to the DOM.

Cause side effects. Ex: Interact with the DOM or perform any ajax calls to load data.



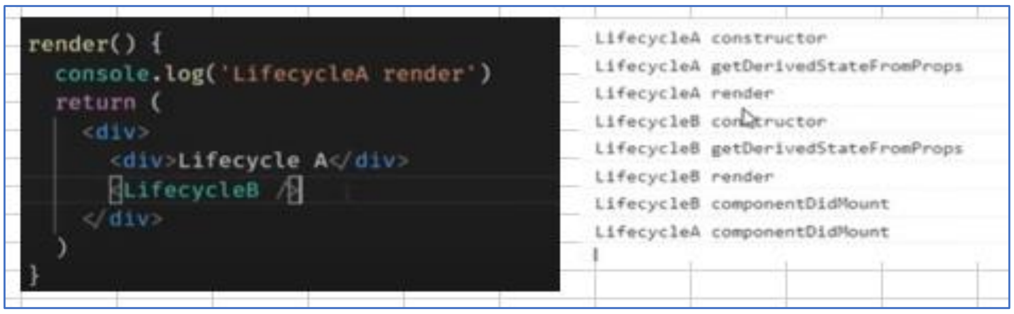
```
class LifecycleA extends Component {
  constructor(props) {
    super(props)

    this.state = {
      name: 'Vishwas'
    }
    console.log('LifecycleA constructor')
  }

  static getDerivedStateFromProps(props, state) {
    console.log('LifecycleA getDerivedStateFromProps')
    return null
  }
```

```
componentDidMount() {
  console.log('LifecycleA componentDidMount')
}

render() {
  console.log('LifecycleA render')
  return <div>Lifecycle A</div>
}
```



```
Download the React DevTools for a better development experience: https://fb.me/react-devtools
LifecycleA constructor
LifecycleA getDerivedStateFromProps
LifecycleA render
LifecycleA componentDidMount
```

<u>Parent & Child</u>

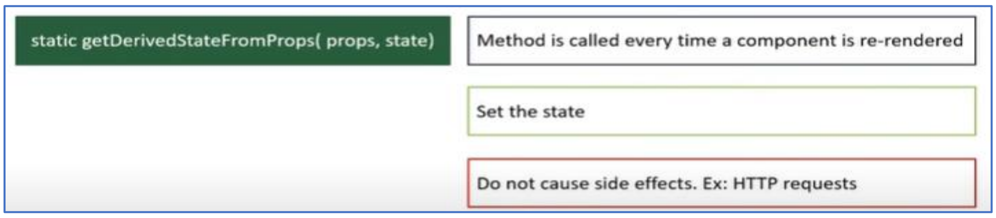


```
render() {
  console.log('LifecycleA render')
  return (
    <div>
      <div>Lifecycle A</div>
      <LifecycleB />
    </div>
  )
}
```

```
LifecycleA constructor
LifecycleA getDerivedStateFromProps
LifecycleA render
LifecycleB constructor
LifecycleB getDerivedStateFromProps
LifecycleB render
LifecycleB componentDidMount
LifecycleA componentDidMount
```

23. Updating Lifecycle

```
- static getDerivedStateFromProps(props, state) => method call everytime re-render component
            => set the state
            => don't cause side effect, HTTP request
- shouldComponentUpdate  => dictates the component should re-render or not
            => performance optimize
            => don't cause side effect, HTTP request and setstate method
- render          => read props/state and return JSX
                  => do not change state / interact with DOM / make ajax call
- getSnapshotBeforeUpdate(prevProps, prevState)
                  => called before the changes before virtual DOM are not be reflected on the DOM
                  => Capture information from DOM
- ComponentDidUpdate(prevProps, prevState, snapshot)
                  => called after the render is finished in the re-render cycles
                  => cause side effect
- ComponentWillUnmount
                  => method is invoked immediately before a component is unmounted and destroyed
                  => cancelling network request, cancelling subscription and invalidation timer
- getDerivedStateFromError(error)
- componentDidCatch
```

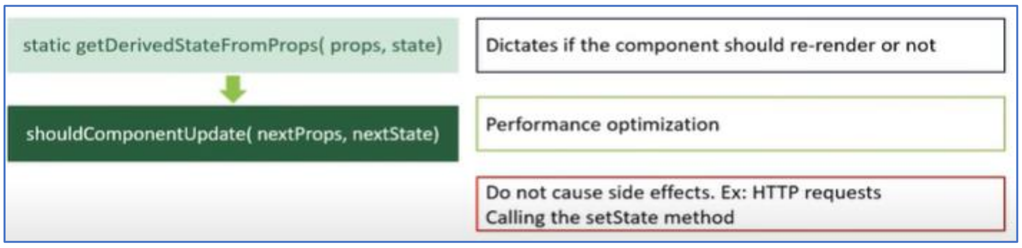


static getDerivedStateFromProps(props, state)

Method is called every time a component is re-rendered

Set the state

Do not cause side effects. Ex: HTTP requests



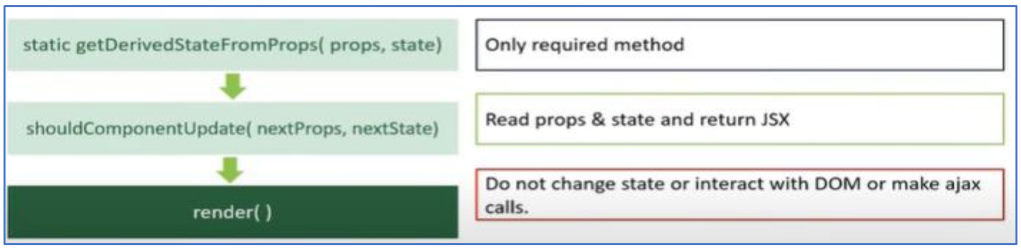
static getDerivedStateFromProps(props, state)

shouldComponentUpdate(nextProps, nextState)

Dictates if the component should re-render or not

Performance optimization

Do not cause side effects. Ex: HTTP requests
Calling the setState method



static getDerivedStateFromProps(props, state)

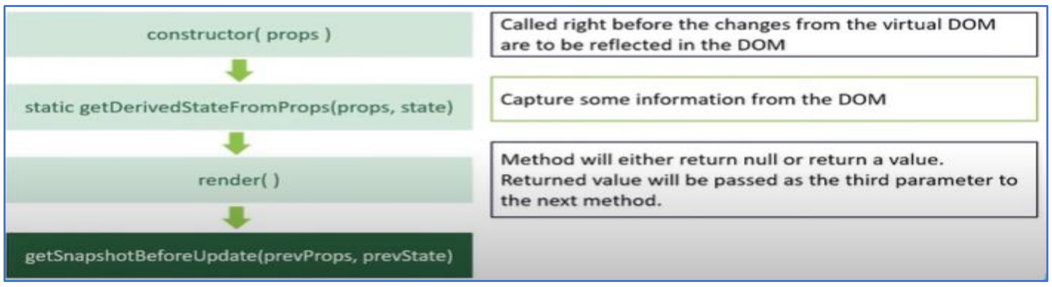
shouldComponentUpdate(nextProps, nextState)

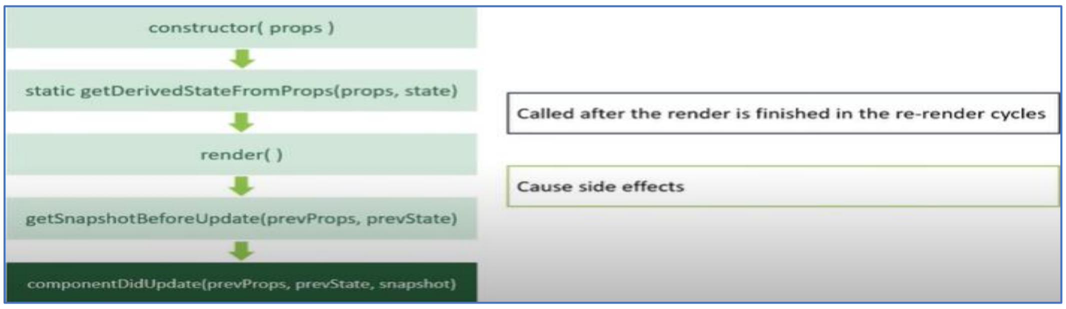
render( )

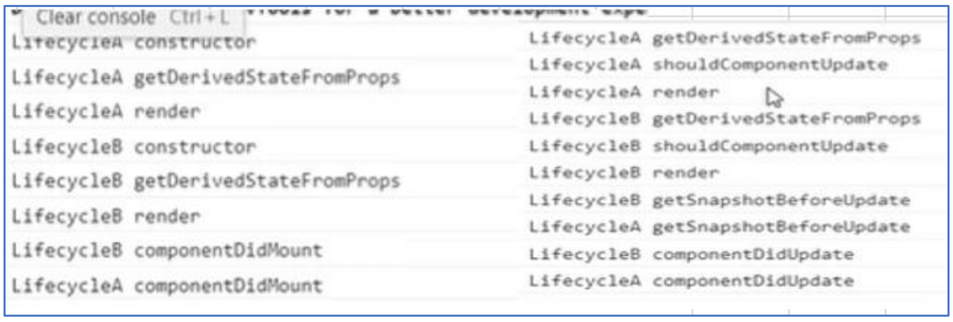
Only required method

Read props & state and return JSX

Do not change state or interact with DOM or make ajax calls.

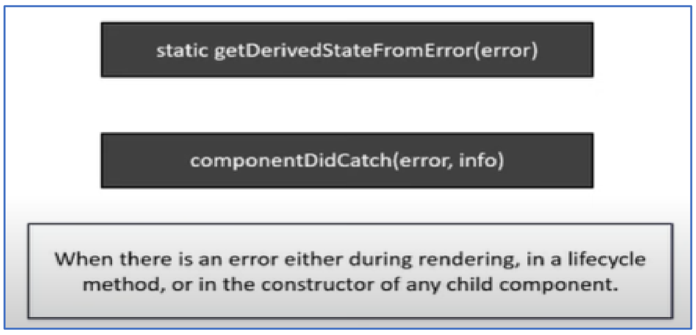

constructor(props)
static getDerivedStateFromProps(props, state)
render( )
getSnapshotBeforeUpdate(prevProps, prevState)
Called right before the changes from the virtual DOM are to be reflected in the DOM
Capture some information from the DOM
Method will either return null or return a value. Returned value will be passed as the third parameter to the next method.


constructor(props)
static getDerivedStateFromProps(props, state)
render( )
getSnapshotBeforeUpdate(prevProps, prevState)
componentDidUpdate(prevProps, prevState, snapshot)
Called after the render is finished in the re-render cycles
Cause side effects



```
LifecycleA constructor
LifecycleA getDerivedStateFromProps
LifecycleA render
LifecycleB constructor
LifecycleB getDerivedStateFromProps
LifecycleB render
LifecycleB componentDidMount
LifecycleA componentDidMount
```

```
LifecycleA getDerivedStateFromProps
LifecycleA shouldComponentUpdate
LifecycleA render
LifecycleB getDerivedStateFromProps
LifecycleB shouldComponentUpdate
LifecycleB render
LifecycleB getSnapshotBeforeUpdate
LifecycleA getSnapshotBeforeUpdate
LifecycleB componentDidUpdate
LifecycleA componentDidUpdate
```


static getDerivedStateFromError(error)
componentDidCatch(error, info)
When there is an error either during rendering, in a lifecycle method, or in the constructor of any child component.

24. Fragment

- return single element without DIV

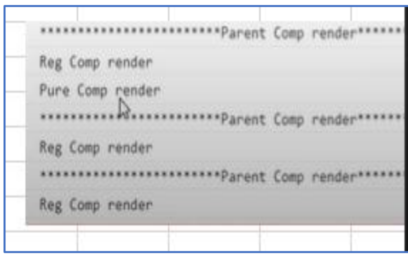
- <React.Fragment> <div></div><h3>test</h3></React.Fragment>

25. Pure Component

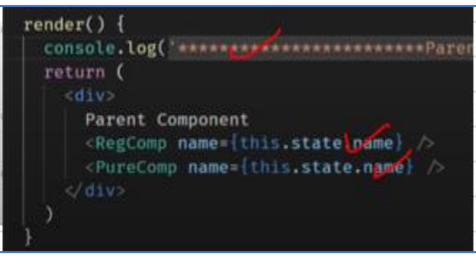
- pure component never re-render/ onetime only

- if parent component is Pure, child also pure

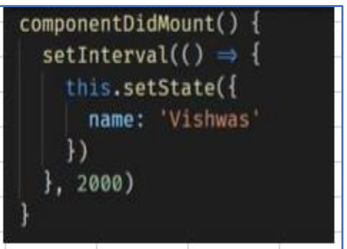
- never mutate state, always return new object state



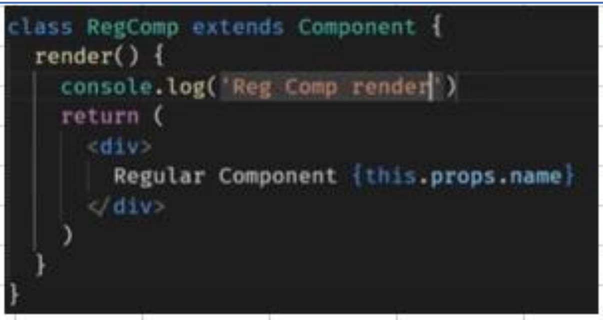
```
************************Parent Comp render******
Reg Comp render
Pure Comp render
************************Parent Comp render******
Reg Comp render
************************Parent Comp render******
Reg Comp render
```



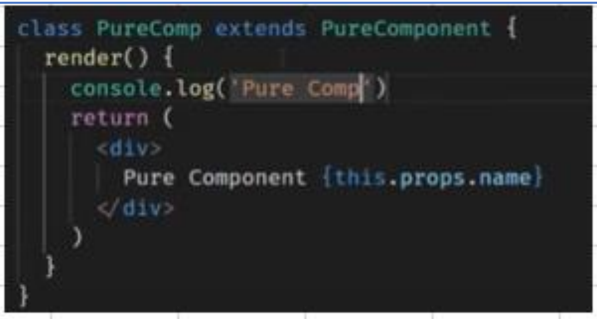
```
render() {
  console.log('************************Paren
  return (
    <div>
      Parent Component
      <RegComp name={this.state.name} />
      <PureComp name={this.state.name} />
    </div>
  )
}
```



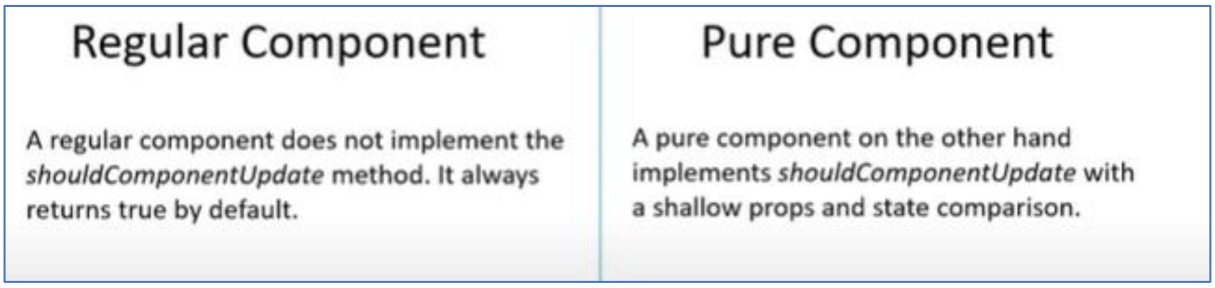
```
componentDidMount() {
  setInterval(() => {
    this.setState({
      name: 'Vishwas'
    })
  }, 2000)
}
```



```
class RegComp extends Component {
  render() {
    console.log('Reg Comp render')
    return (
      <div>
        Regular Component {this.props.name}
      </div>
    )
  }
}
```



```
class PureComp extends PureComponent {
  render() {
    console.log('Pure Comp')
    return (
      <div>
        Pure Component {this.props.name}
      </div>
    )
  }
}
```

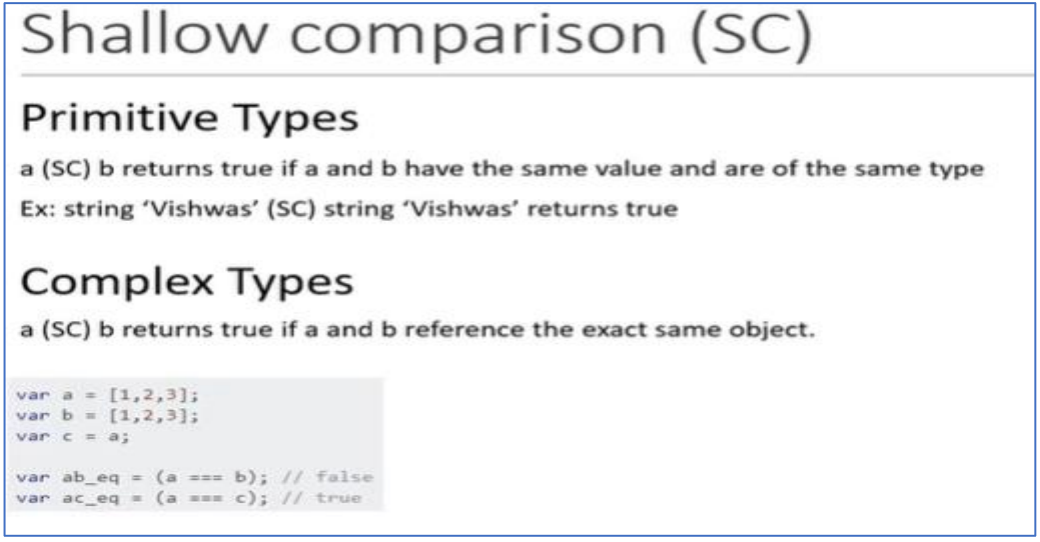


Regular Component

A regular component does not implement the *shouldComponentUpdate* method. It always returns true by default.

Pure Component

A pure component on the other hand implements *shouldComponentUpdate* with a shallow props and state comparison.



Shallow comparison (SC)

Primitive Types

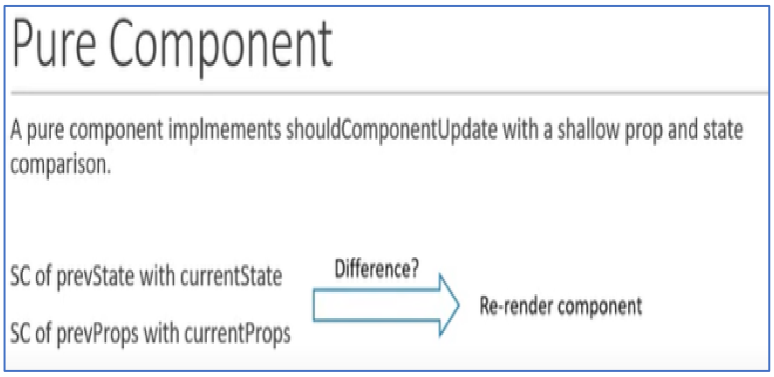
a (SC) b returns true if a and b have the same value and are of the same type

Ex: string 'Vishwas' (SC) string 'Vishwas' returns true

Complex Types

a (SC) b returns true if a and b reference the exact same object.

```
var a = [1,2,3];
var b = [1,2,3];
var c = a;

var ab_eq = (a === b); // false
var ac_eq = (a === c); // true
```

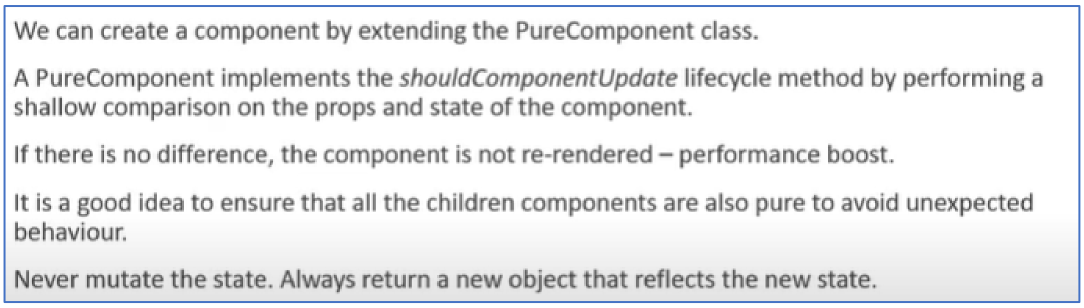


Pure Component

A pure component implmements shouldComponentUpdate with a shallow prop and state comparison.

SC of prevState with currentState

SC of prevProps with currentProps



We can create a component by extending the PureComponent class.

A PureComponent implements the *shouldComponentUpdate* lifecycle method by performing a shallow comparison on the props and state of the component.

If there is no difference, the component is not re-rendered – performance boost.

It is a good idea to ensure that all the children components are also pure to avoid unexpected behaviour.

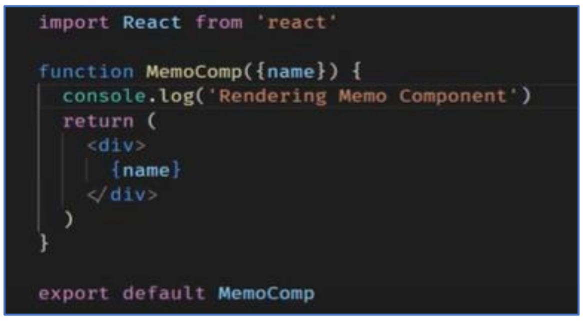
Never mutate the state. Always return a new object that reflects the new state.

26. <u>Memo Component</u>

- same like pure component

- component never re-render and no changes props

- react.purecomponent for class component and react.memo for function component



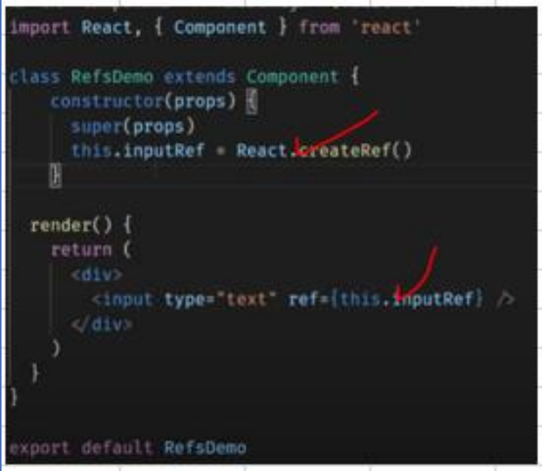
```
import React from 'react'

function MemoComp({name}) {
  console.log('Rendering Memo Component')
  return (
    <div>
      {name}
    </div>
  )
}

export default MemoComp
```

27. <u>Refs</u>

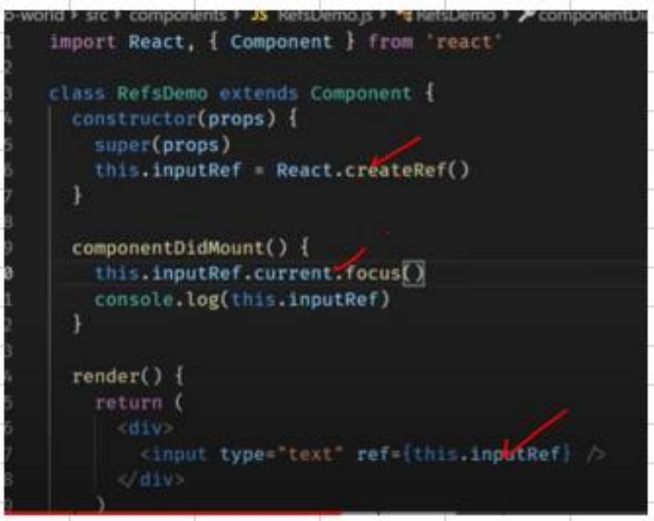
- this.inputRef = React.createRef();

- <input type=text ref={this.inputRef}/>

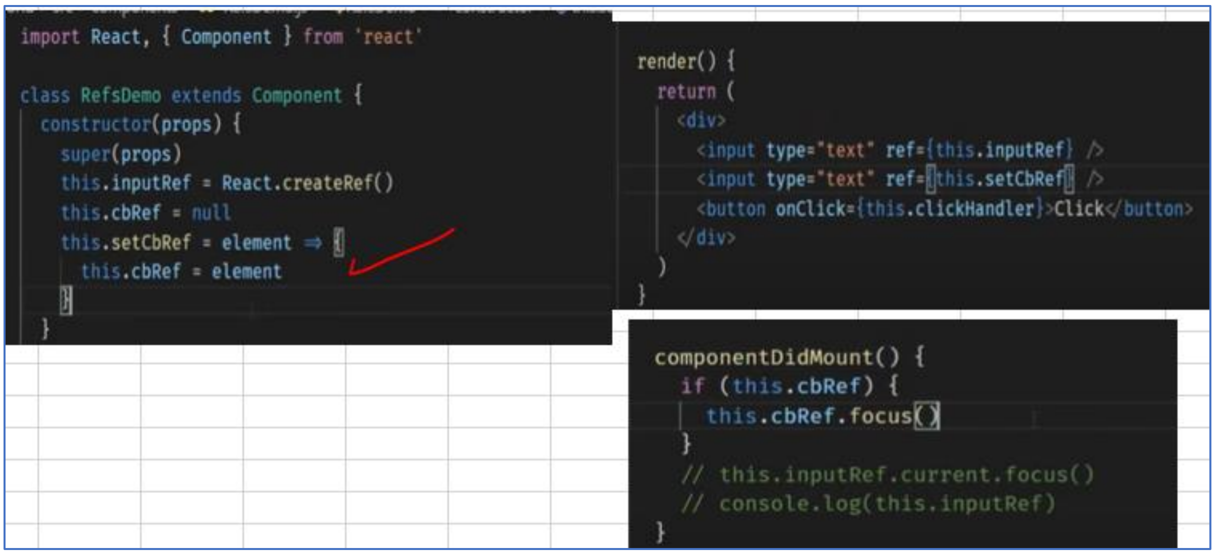
- this.inputRef.current.value



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

class RefsDemo extends Component {
    constructor(props) {
        super(props)
        this.inputRef = React.createRef()
    }

  render() {
    return (
      <div>
        <input type="text" ref={this.inputRef} />
      </div>
    )
  }
}

export default RefsDemo
```


o-world › src › components › JS RefsDemo.js › RefsDemo › componentDi

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

class RefsDemo extends Component {
  constructor(props) {
    super(props)
    this.inputRef = React.createRef()
  }

  componentDidMount() {
    this.inputRef.current.focus()
    console.log(this.inputRef)
  }

  render() {
    return (
      <div>
        <input type="text" ref={this.inputRef} />
      </div>
    )
```

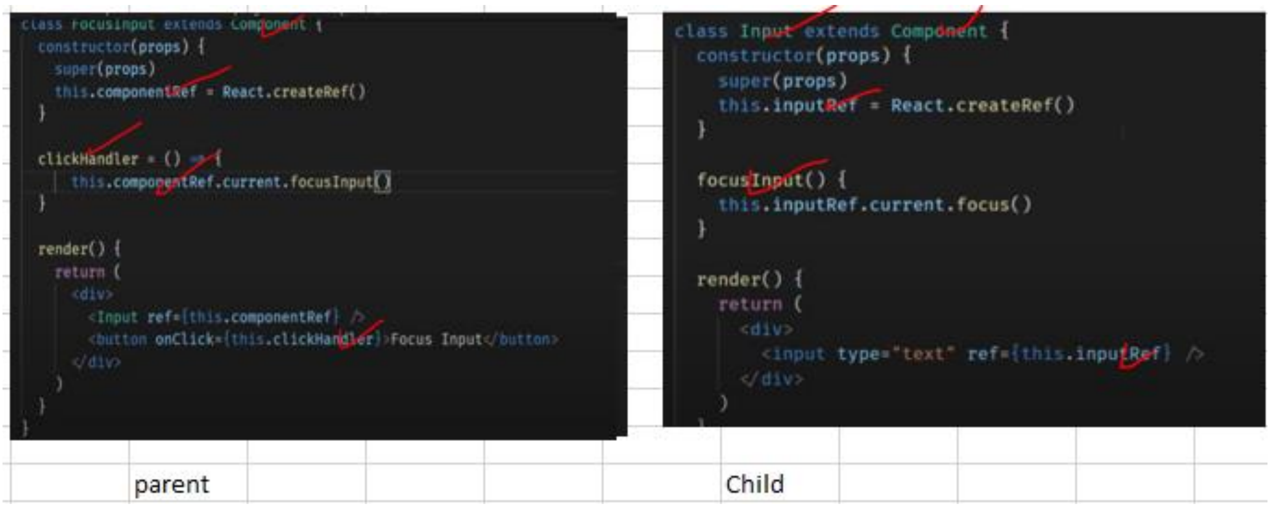


```
import React, { Component } from 'react'

class RefsDemo extends Component {
  constructor(props) {
    super(props)
    this.inputRef = React.createRef()
    this.cbRef = null
    this.setCbRef = element => {
      this.cbRef = element
    }
  }
```

```
render() {
  return (
    <div>
      <input type="text" ref={this.inputRef} />
      <input type="text" ref={this.setCbRef} />
      <button onClick={this.clickHandler}>Click</button>
    </div>
  )
}
```

```
componentDidMount() {
  if (this.cbRef) {
    this.cbRef.focus()
  }
  // this.inputRef.current.focus()
  // console.log(this.inputRef)
}
```

28. Ref vs Class Component



```
class FocusInput extends Component {
  constructor(props) {
    super(props)
    this.componentRef = React.createRef()
  }

  clickHandler = () => {
    this.componentRef.current.focusInput()
  }

  render() {
    return (
      <div>
        <Input ref={this.componentRef} />
        <button onClick={this.clickHandler}>Focus Input</button>
      </div>
    )
  }
}
```

```
class Input extends Component {
  constructor(props) {
    super(props)
    this.inputRef = React.createRef()
  }

  focusInput() {
    this.inputRef.current.focus()
  }

  render() {
    return (
      <div>
        <input type="text" ref={this.inputRef} />
      </div>
    )
```

29. Forward Ref

- forward ref to cild component

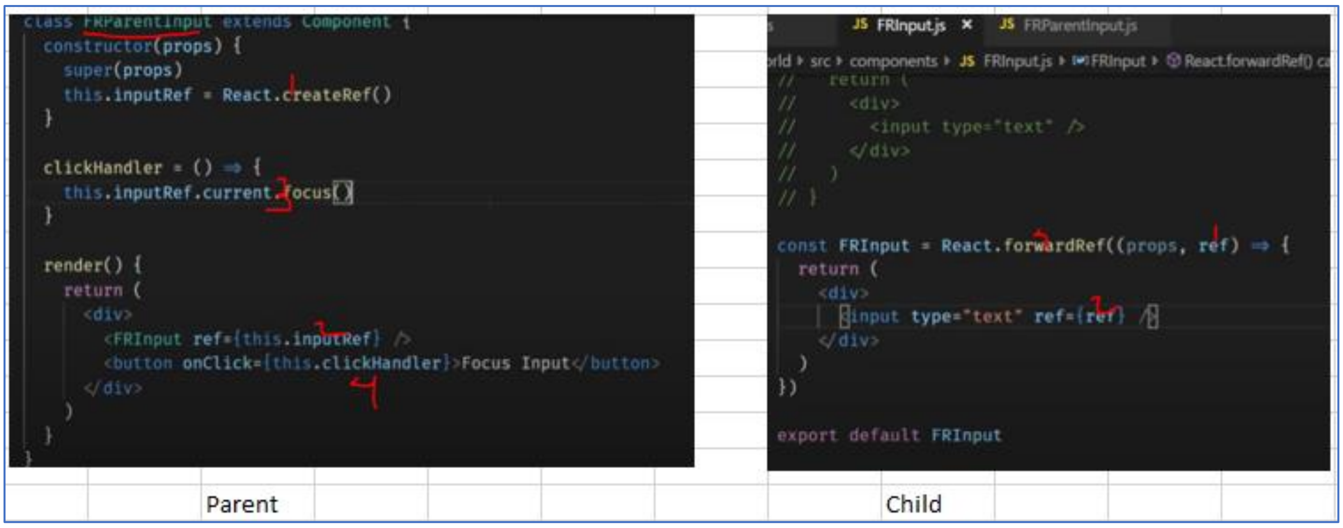
- 1. create ref in parent component and pass ref to child

eg. <FRInput ref={this.inputRef} />

    and child => const FRInput = React.forwardRef((props, ref) => {

        <input ref= {ref} />

    }



```
class FRParentInput extends Component {
  constructor(props) {
    super(props)
    this.inputRef = React.createRef()
  }

  clickHandler = () => {
    this.inputRef.current.focus()
  }

  render() {
    return (
      <div>
        <FRInput ref={this.inputRef} />
        <button onClick={this.clickHandler}>Focus Input</button>
      </div>
    )
  }
}
```

Parent

```
//   return (
//     <div>
//       <input type="text" />
//     </div>
//   )
// }

const FRInput = React.forwardRef((props, ref) => {
  return (
    <div>
      <input type="text" ref={ref} />
    </div>
  )
})

export default FRInput
```

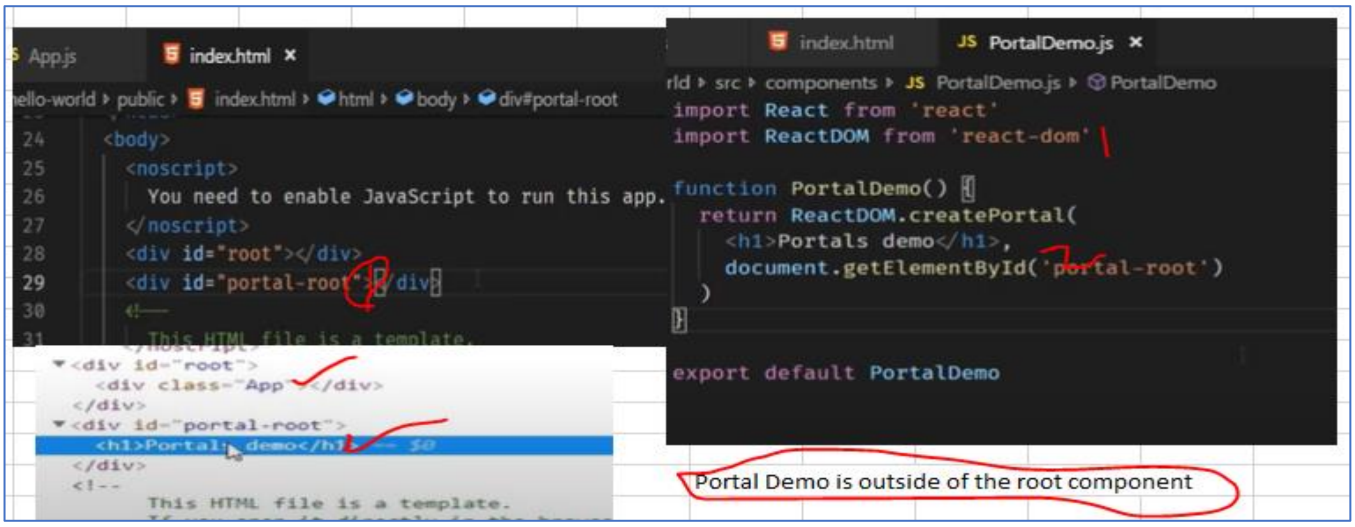
Child

30. React Portals

- create react outside root DOM node

- create another div id='portal' under index.html

- create ReactDOM.createPortal(<div>test</div> , document.getelementbyid('portal')) in js file



```
<body>
  <noscript>
    You need to enable JavaScript to run this app.
  </noscript>
  <div id="root"></div>
  <div id="portal-root"></div>
  <!--
    This HTML file is a template.
```

```
▼<div id="root">
   <div class="App"></div>
 </div>
▼<div id="portal-root">
   <h1>Portals demo</h1> == $0
 </div>
 <!--
   This HTML file is a template.
```

```
import React from 'react'
import ReactDOM from 'react-dom'

function PortalDemo() {
  return ReactDOM.createPortal(
    <h1>Portals demo</h1>,
    document.getElementById('portal-root')
  )
}

export default PortalDemo
```

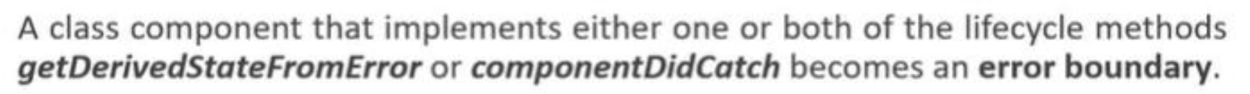
Portal Demo is outside of the root component

31. Error Boundaries

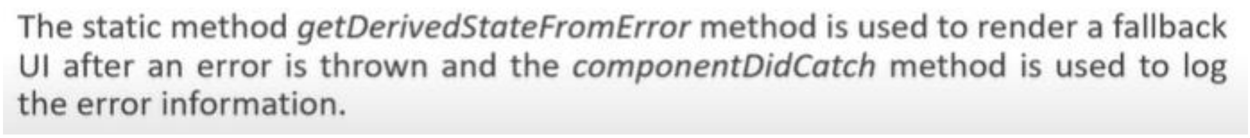
- react component that catch javascript error from child component, log these errors and show fallback UI

- method Name getDirectiveStateFromError and componentDidCatch

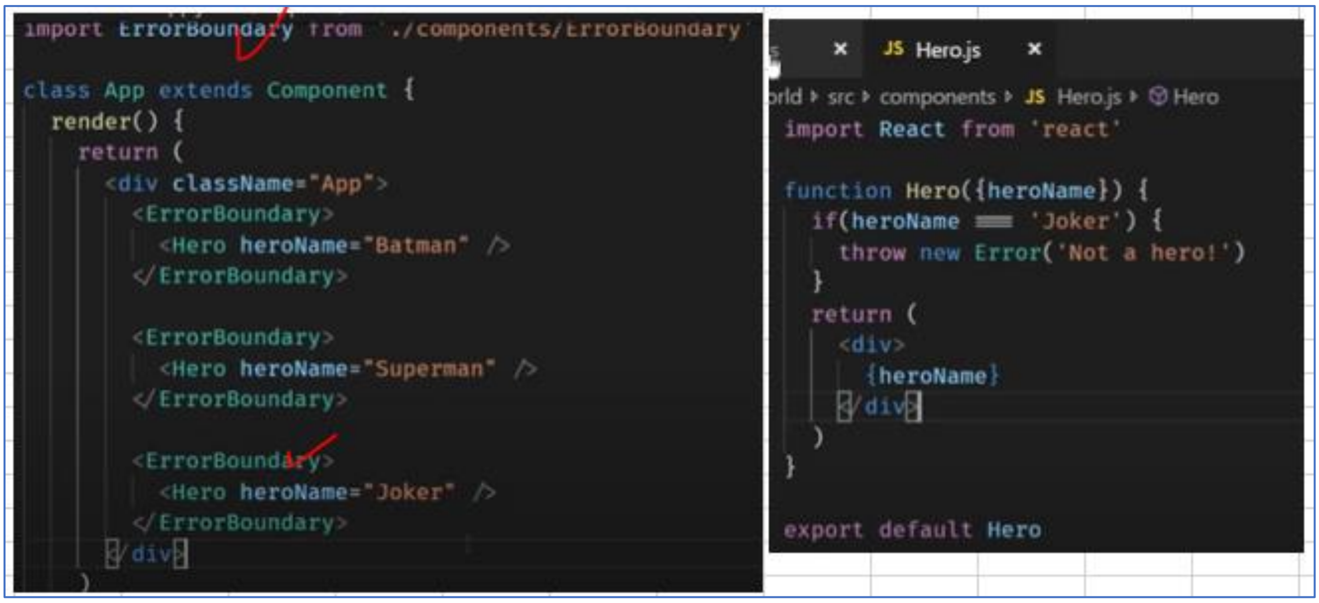
- Error Boundaries is only for production, for dev you will be still see all error log



A class component that implements either one or both of the lifecycle methods ***getDerivedStateFromError*** or ***componentDidCatch*** becomes an **error boundary**.

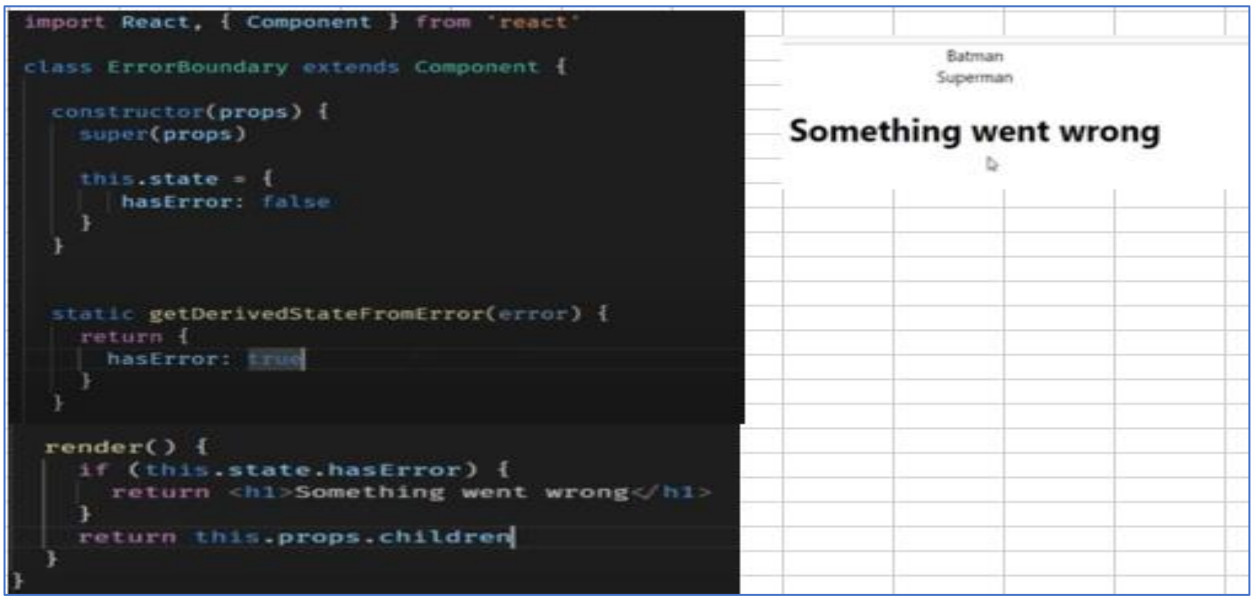


The static method *getDerivedStateFromError* method is used to render a fallback UI after an error is thrown and the *componentDidCatch* method is used to log the error information.



```
import ErrorBoundary from './components/ErrorBoundary'

class App extends Component {
  render() {
    return (
      <div className="App">
        <ErrorBoundary>
          <Hero heroName="Batman" />
        </ErrorBoundary>

        <ErrorBoundary>
          <Hero heroName="Superman" />
        </ErrorBoundary>

        <ErrorBoundary>
          <Hero heroName="Joker" />
        </ErrorBoundary>
      </div>
    )
```

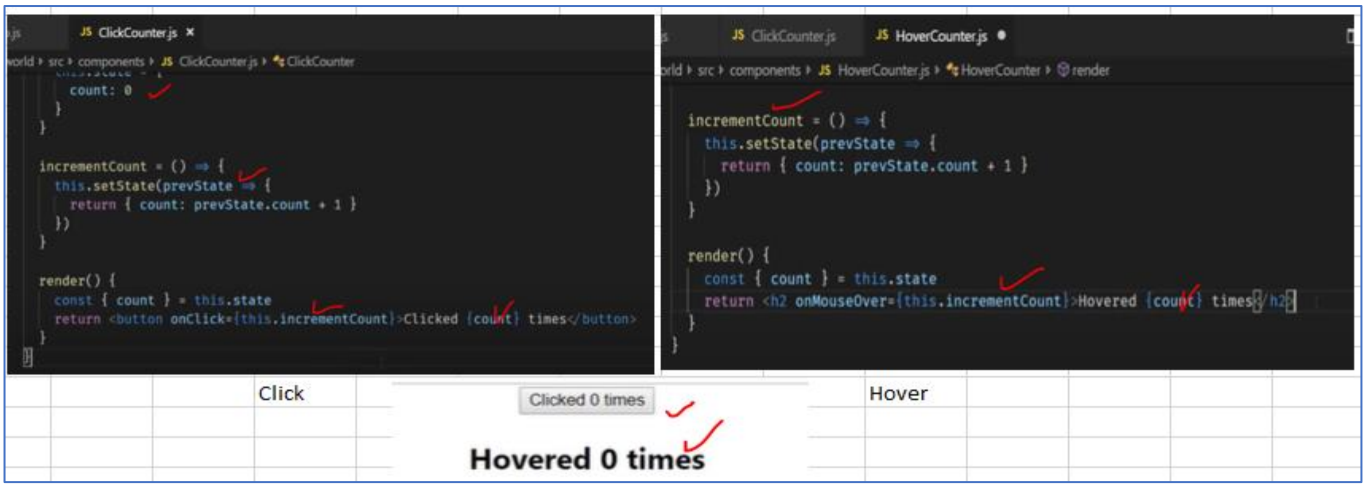
```
import React from 'react'

function Hero({heroName}) {
  if(heroName === 'Joker') {
    throw new Error('Not a hero!')
  }
  return (
    <div>
      {heroName}
    </div>
  )
}

export default Hero
```



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

class ErrorBoundary extends Component {

  constructor(props) {
    super(props)

    this.state = {
      hasError: false
    }
  }


  static getDerivedStateFromError(error) {
    return {
      hasError: true
    }
  }

  render() {
    if (this.state.hasError) {
      return <h1>Something went wrong</h1>
    }
    return this.props.children
  }
}
```

```
Batman
Superman

Something went wrong
```

32. Higher Order Component



ClickCounter.js
orld › src › components › ClickCounter.js › ClickCounter

```
      count: 0
    }
  }

  incrementCount = () => {
    this.setState(prevState => {
      return { count: prevState.count + 1 }
    })
  }

  render() {
    const { count } = this.state
    return <button onClick={this.incrementCount}>Clicked {count} times</button>
  }
}
```

ClickCounter.js
HoverCounter.js
rld › src › components › HoverCounter.js › HoverCounter › render

```
  incrementCount = () => {
    this.setState(prevState => {
      return { count: prevState.count + 1 }
    })
  }

  render() {
    const { count } = this.state
    return <h2 onMouseOver={this.incrementCount}>Hovered {count} times</h2>
  }
}
```

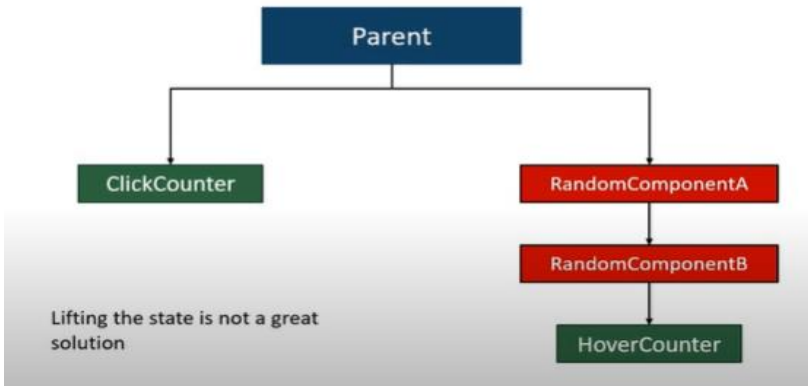
Click

Clicked 0 times

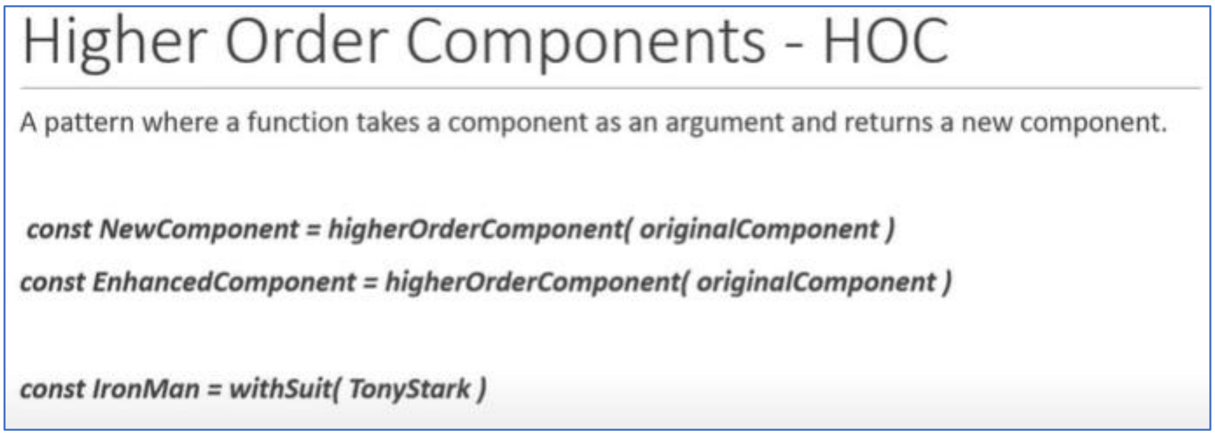
Hovered 0 times

Hover

Should be reusable code



Need common function for between cross component



Higher Order Components - HOC

A pattern where a function takes a component as an argument and returns a new component.

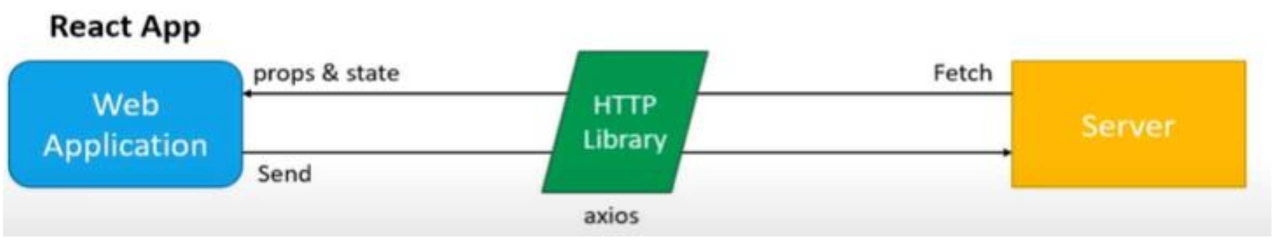
const NewComponent = higherOrderComponent(originalComponent)

const EnhancedComponent = higherOrderComponent(originalComponent)

const IronMan = withSuit(TonyStark)

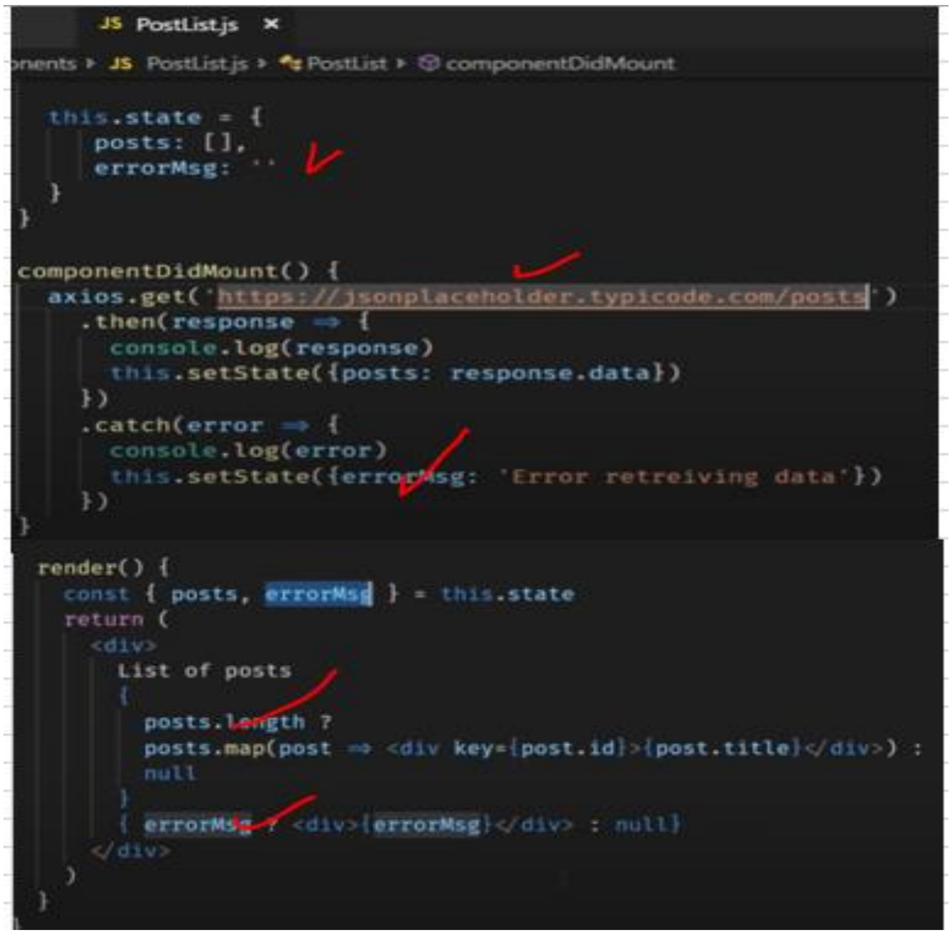
HOC is accept original component and return new component

33. React HTTP



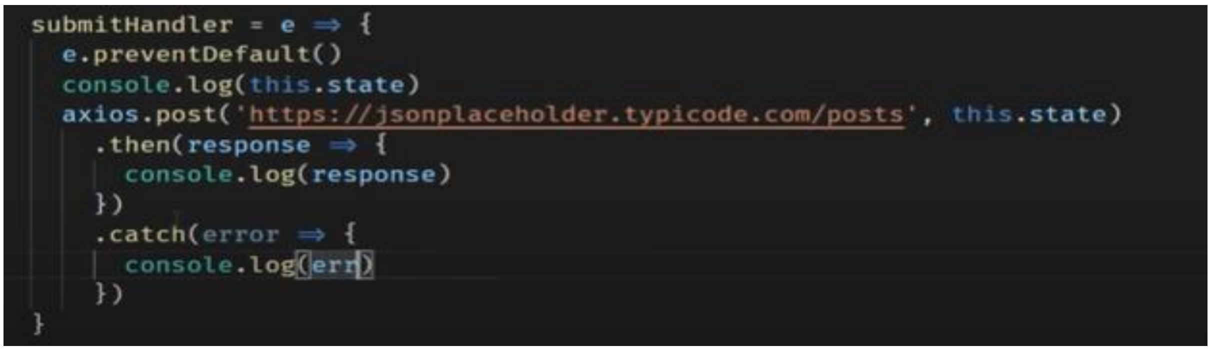
Npm install axios

https://jsonplaceholder.typicode.com/(to test)



```
JS PostList.js ×
onents ▸ JS PostList.js ▸ PostList ▸ componentDidMount

  this.state = {
    posts: [],
    errorMsg: ''
  }
}

componentDidMount() {
  axios.get('https://jsonplaceholder.typicode.com/posts')
    .then(response => {
      console.log(response)
      this.setState({posts: response.data})
    })
    .catch(error => {
      console.log(error)
      this.setState({errorMsg: 'Error retreiving data'})
    })
}
```

```
  render() {
    const { posts, errorMsg } = this.state
    return (
      <div>
        List of posts
        {
          posts.length ?
          posts.map(post => <div key={post.id}>{post.title}</div>) :
          null
        }
        { errorMsg ? <div>{errorMsg}</div> : null}
      </div>
    )
  }
```



```
submitHandler = e => {
  e.preventDefault()
  console.log(this.state)
  axios.post('https://jsonplaceholder.typicode.com/posts', this.state)
    .then(response => {
      console.log(response)
    })
    .catch(error => {
      console.log(err)
    })
}
```

34. HOOK

- min react version 16.8

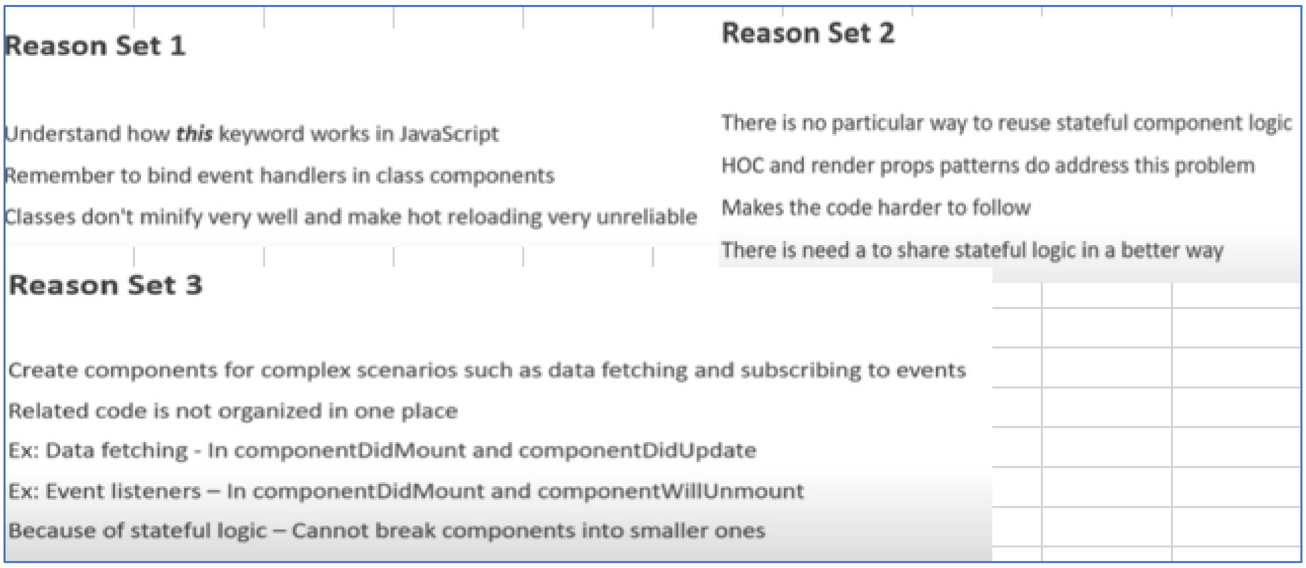
- state and other features without writing class

- hook don't work in class component

- avoid confustion with 'this' keywork

- allow to reuse stateful logic

- organize the logic inside a component onto reuse



Reason Set 1

Understand how ***this*** keyword works in JavaScript

Remember to bind event handlers in class components

Classes don't minify very well and make hot reloading very unreliable

Reason Set 2

There is no particular way to reuse stateful component logic

HOC and render props patterns do address this problem

Makes the code harder to follow

There is need a to share stateful logic in a better way

Reason Set 3

Create components for complex scenarios such as data fetching and subscribing to events

Related code is not organized in one place

Ex: Data fetching - In componentDidMount and componentDidUpdate

Ex: Event listeners – In componentDidMount and componentWillUnmount

Because of stateful logic – Cannot break components into smaller ones

35. <u>Rules of HOOK</u>

- only call in top level

- don't call hook in loop, conditions or nested functions

- only call hook in react function

- only call in react functional component

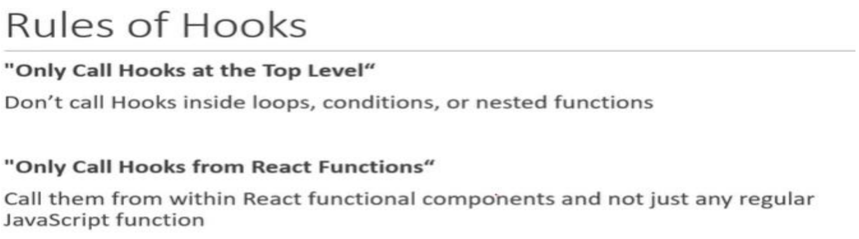
- state is always object in class

- with useState hook, the state don't have to be object

- use state hook return an array with 2 elements

- the first element is the current value of the state, the second element is state setter function

- when dealing with object or array, always make sure to spread your state variable and then call setter function



Rules of Hooks

"Only Call Hooks at the Top Level"

Don't call Hooks inside loops, conditions, or nested functions

"Only Call Hooks from React Functions"

Call them from within React functional components and not just any regular JavaScript function

36. <u>useState Hook</u>

- const [count, setCount] = useState(0); // cal, method and default

- <button onClick={() => setCount(count+1)}

- const incrementFive = () => {

for(let i=0; i<5; i++) (setCount(prevCount+1));

}

- const [name, setName] = useSate({fName:'', lName:''})

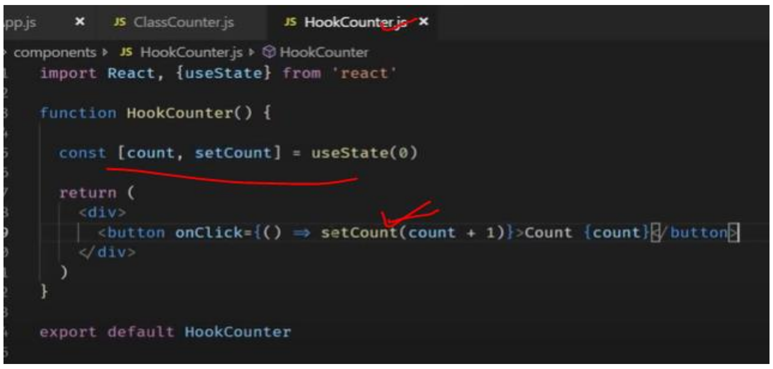
- <input value={name.fName} onChange={e => setName({...name, fName: e.target.value})}// clone from prev state

- const [items, setItems] = useState([])

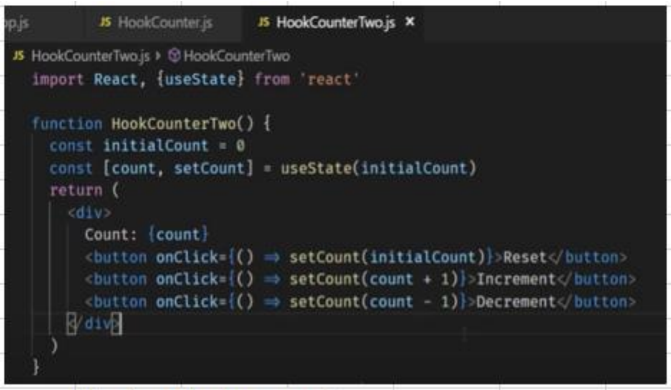
- const addItem = () => {

    setItem([...items, { id: items.length, value: 2}])

}

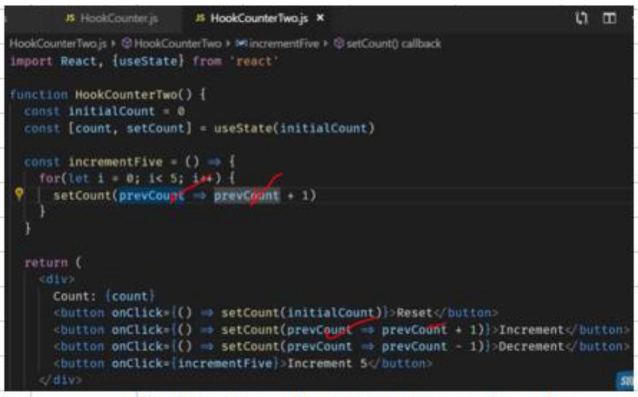


```
import React, {useState} from 'react'

function HookCounter() {

  const [count, setCount] = useState(0)

  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Count {count}</button>
    </div>
  )
}

export default HookCounter
```



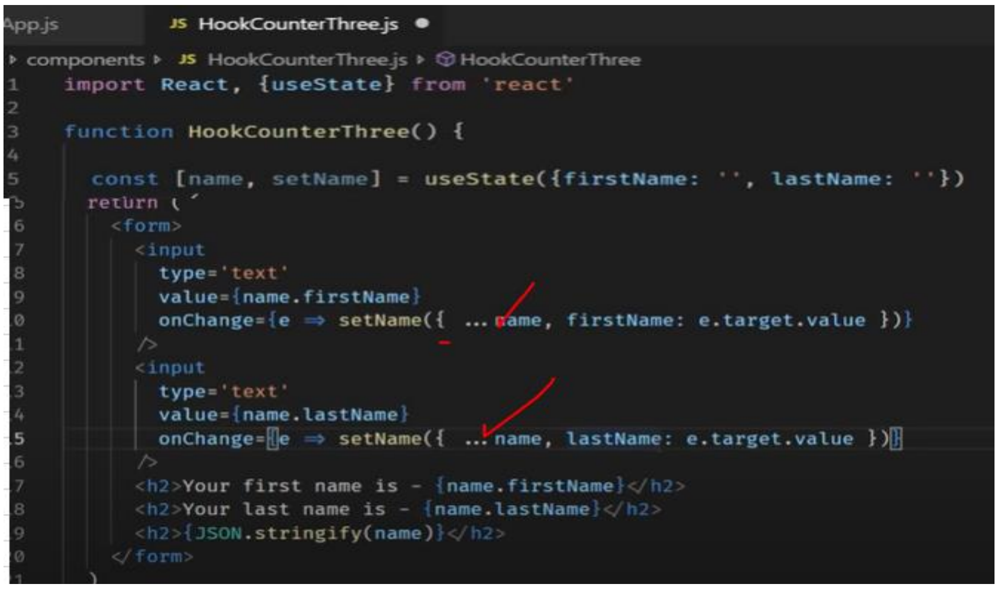
```
import React, {useState} from 'react'

function HookCounterTwo() {
  const initialCount = 0
  const [count, setCount] = useState(initialCount)
  return (
    <div>
      Count: {count}
      <button onClick={() => setCount(initialCount)}>Reset</button>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
    </div>
  )
}
```

Not Cover for increment 5



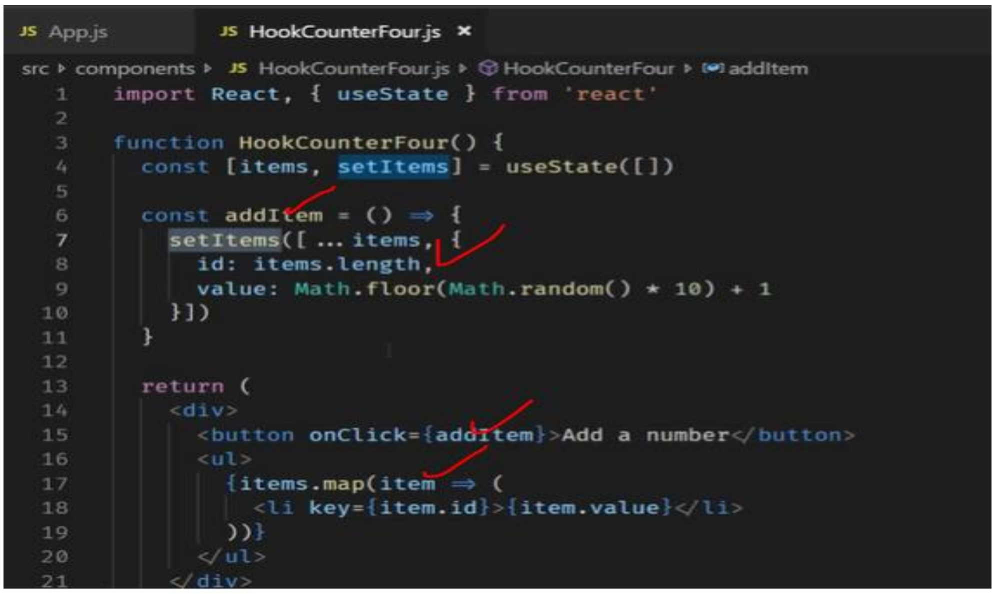
```
import React, {useState} from 'react'

function HookCounterTwo() {
  const initialCount = 0
  const [count, setCount] = useState(initialCount)

  const incrementFive = () => {
    for(let i = 0; i< 5; i++) {
      setCount(prevCount => prevCount + 1)
    }
  }

  return (
    <div>
      Count: {count}
      <button onClick={() => setCount(initialCount)}>Reset</button>
      <button onClick={() => setCount(prevCount => prevCount + 1)}>Increment</button>
      <button onClick={() => setCount(prevCount => prevCount - 1)}>Decrement</button>
      <button onClick={incrementFive}>Increment 5</button>
    </div>
```

Use PrevCount/ prev state to get prev value



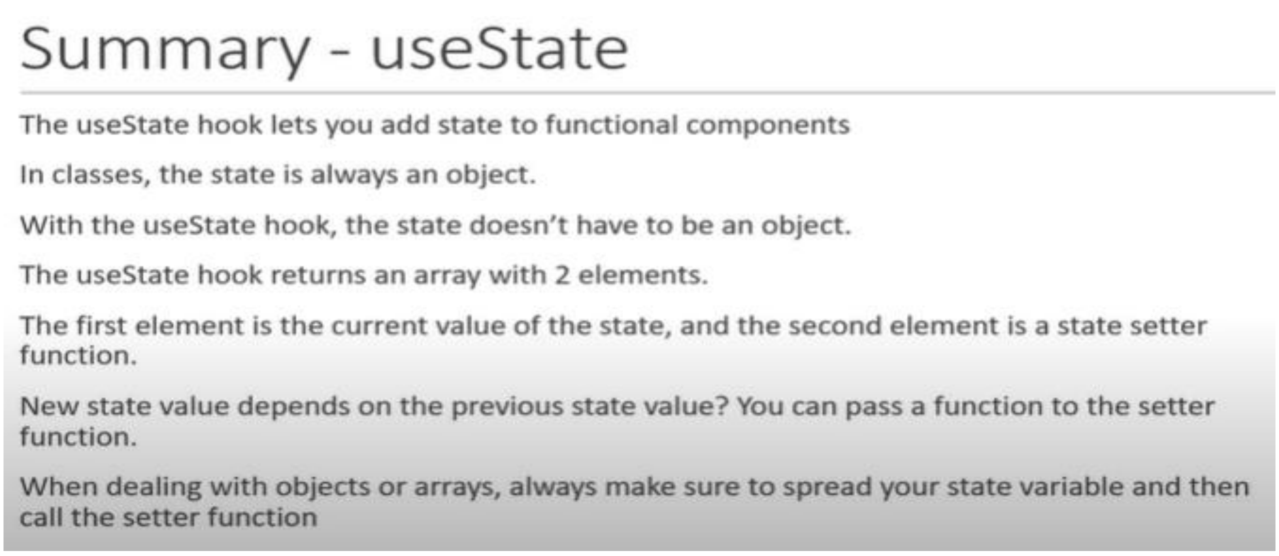
```
import React, {useState} from 'react'

function HookCounterThree() {

  const [name, setName] = useState({firstName: '', lastName: ''})
  return (
    <form>
      <input
        type='text'
        value={name.firstName}
        onChange={e => setName({ ...name, firstName: e.target.value })}
      />
      <input
        type='text'
        value={name.lastName}
        onChange={e => setName({ ...name, lastName: e.target.value })}
      />
      <h2>Your first name is - {name.firstName}</h2>
      <h2>Your last name is - {name.lastName}</h2>
      <h2>{JSON.stringify(name)}</h2>
    </form>
  )
```

... is clone the value from prevState


App.js   HookCounterFour.js

src › components › HookCounterFour.js › HookCounterFour › addItem

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

function HookCounterFour() {
  const [items, setItems] = useState([])

  const addItem = () => {
    setItems([ ...items, {
      id: items.length,
      value: Math.floor(Math.random() * 10) + 1
    }])
  }

  return (
    <div>
      <button onClick={addItem}>Add a number</button>
      <ul>
        {items.map(item => (
          <li key={item.id}>{item.value}</li>
        ))}
      </ul>
    </div>
```



Summary - useState

The useState hook lets you add state to functional components

In classes, the state is always an object.

With the useState hook, the state doesn't have to be an object.

The useState hook returns an array with 2 elements.

The first element is the current value of the state, and the second element is a state setter function.

New state value depends on the previous state value? You can pass a function to the setter function.

When dealing with objects or arrays, always make sure to spread your state variable and then call the setter function

37. useEffect after render

=====> useEffect after render

- useEffect will call after every render/re-render component(same componentDidMound/componentDidUpdate/ compomentWillUnmount)

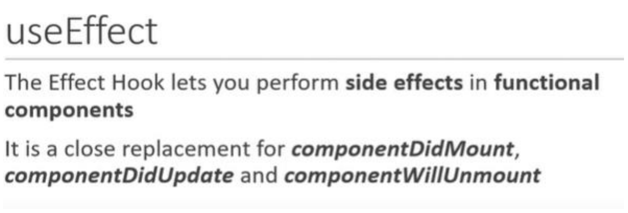
=====> Conditional run effect

- useEffect(() => {   }, [count]); // second parameter is condition to run useEffect method

=====> Run Effect one time only

- useEffect(() => {   }, []); // empty array to run 1 time only

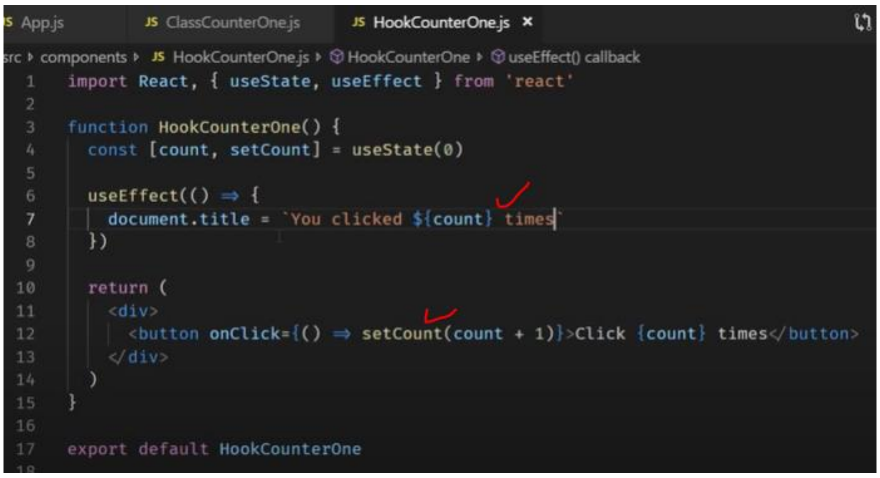
=====> UseEffect with cleanup



useEffect

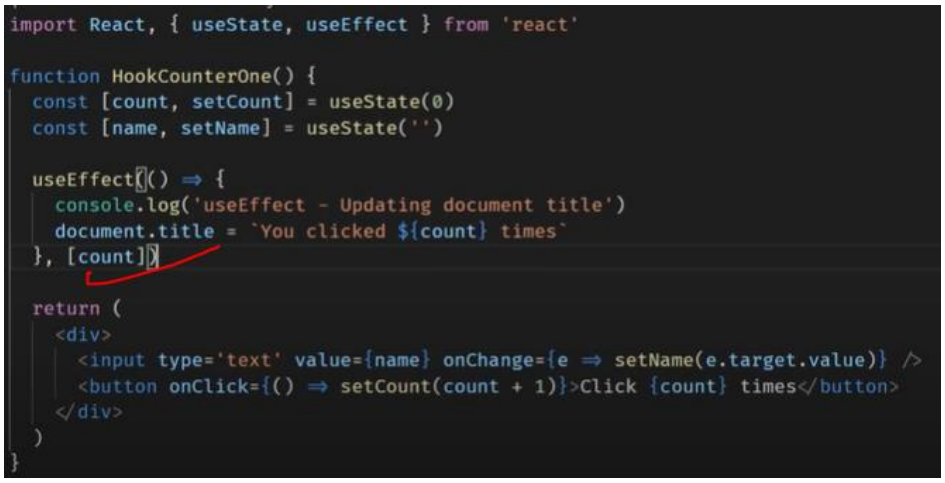
The Effect Hook lets you perform **side effects** in **functional components**

It is a close replacement for ***componentDidMount***, ***componentDidUpdate*** and ***componentWillUnmount***

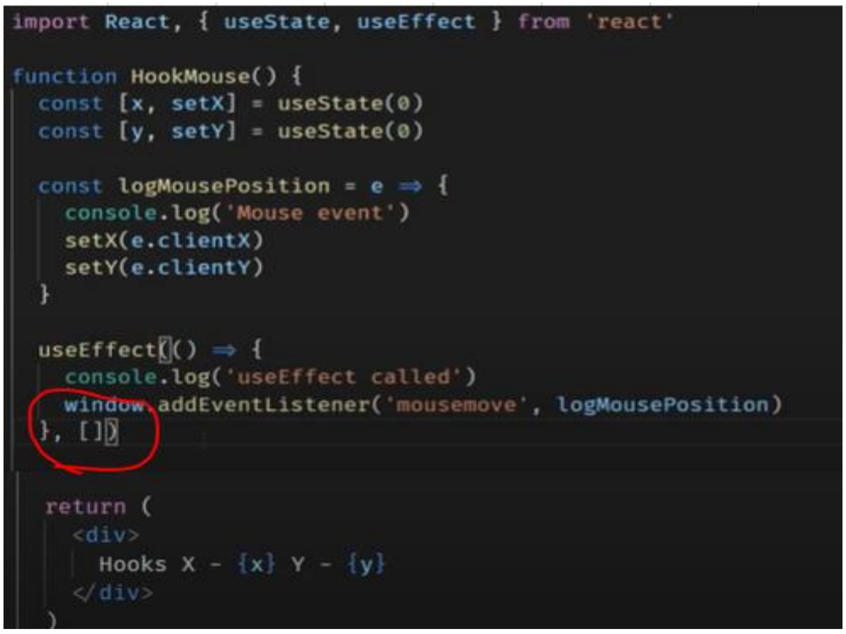

App.js
ClassCounterOne.js
HookCounterOne.js
src › components › HookCounterOne.js › HookCounterOne › useEffect() callback

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

function HookCounterOne() {
  const [count, setCount] = useState(0)

  useEffect(() => {
    document.title = `You clicked ${count} times`
  })

  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Click {count} times</button>
    </div>
  )
}

export default HookCounterOne
```

Fetching data useEffect (**should pass [] for 1 time load from server**)



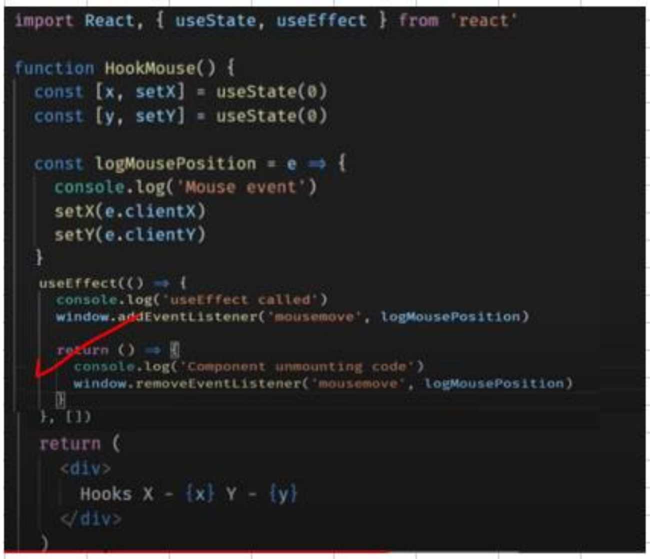
```
import React, { useState, useEffect } from 'react'

function HookCounterOne() {
  const [count, setCount] = useState(0)
  const [name, setName] = useState('')

  useEffect(() => {
    console.log('useEffect - Updating document title')
    document.title = `You clicked ${count} times`
  }, [count])

  return (
    <div>
      <input type='text' value={name} onChange={e => setName(e.target.value)} />
      <button onClick={() => setCount(count + 1)}>Click {count} times</button>
    </div>
  )
}
```



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

function HookMouse() {
  const [x, setX] = useState(0)
  const [y, setY] = useState(0)

  const logMousePosition = e => {
    console.log('Mouse event')
    setX(e.clientX)
    setY(e.clientY)
  }

  useEffect(() => {
    console.log('useEffect called')
    window.addEventListener('mousemove', logMousePosition)
  }, [])

  return (
    <div>
      Hooks X - {x} Y - {y}
    </div>
  )
```

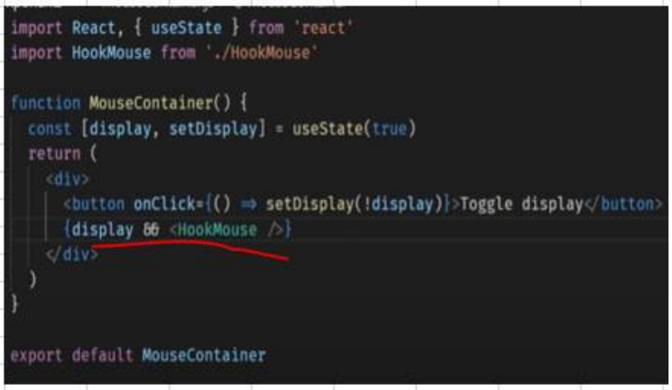
Only 1 time Load

User effect with Clean up function => mean unsubscribe function (use return fun



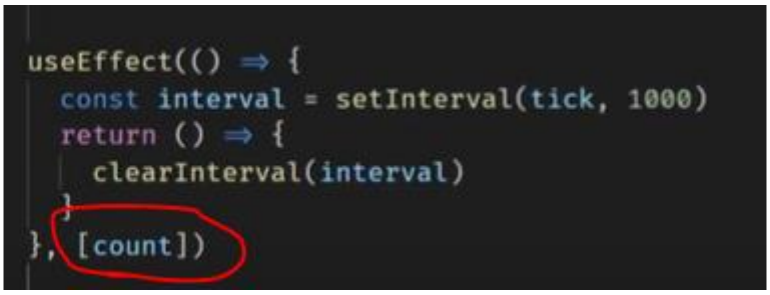
```
import React, { useState, useEffect } from 'react'

function HookMouse() {
  const [x, setX] = useState(0)
  const [y, setY] = useState(0)

  const logMousePosition = e => {
    console.log('Mouse event')
    setX(e.clientX)
    setY(e.clientY)
  }
  useEffect(() => {
    console.log('useEffect called')
    window.addEventListener('mousemove', logMousePosition)

    return () => {
      console.log('Component unmounting code')
      window.removeEventListener('mousemove', logMousePosition)
    }
  }, [])
  return (
    <div>
      Hooks X - {x} Y - {y}
    </div>
  )
```

Child



```
import React, { useState } from 'react'
import HookMouse from './HookMouse'

function MouseContainer() {
  const [display, setDisplay] = useState(true)
  return (
    <div>
      <button onClick={() => setDisplay(!display)}>Toggle display</button>
      {display && <HookMouse />}
    </div>
  )
}

export default MouseContainer
```

Parent



```
useEffect(() => {
  const interval = setInterval(tick, 1000)
  return () => {
    clearInterval(interval)
  }
}, [count])
```

38. <u>useContext</u>

- context is a way to **pass data through the component tree without passing props down manually**

- useContext

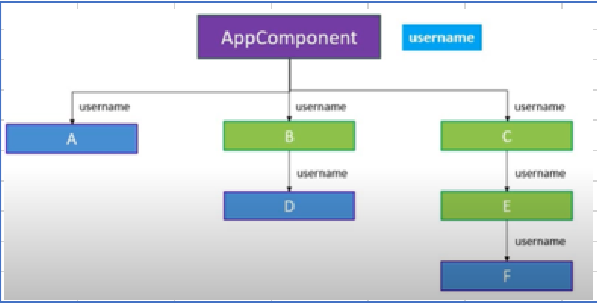
- useState - relative to state

- useEffect - relative to side effect

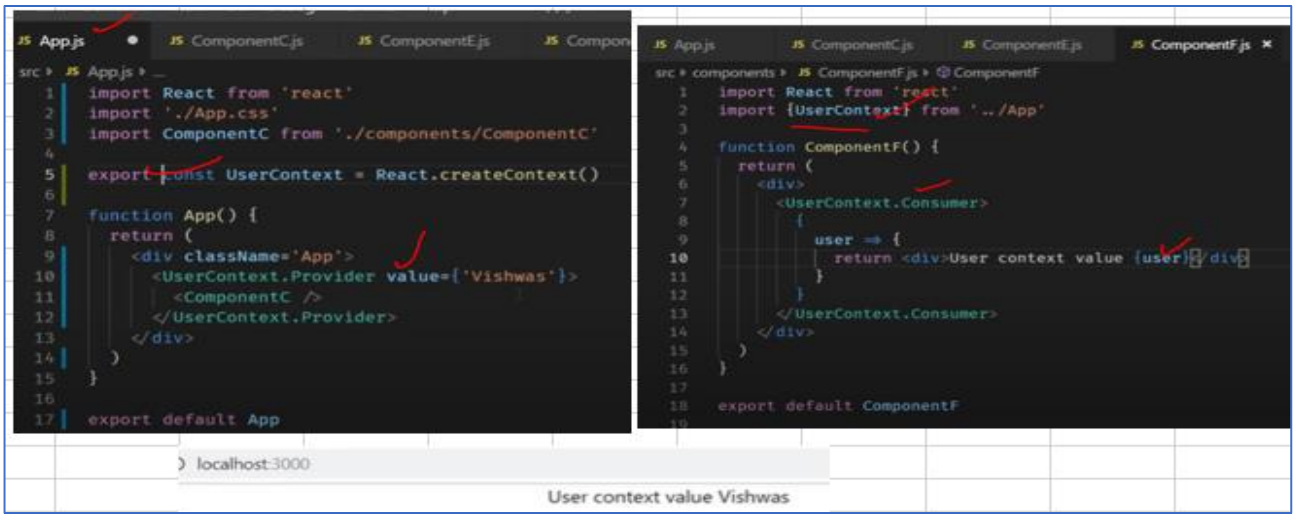
- useContext - context API

- useReducer - relative reducers => useReducer(reducer, initState)

- reducer(currentState, action)



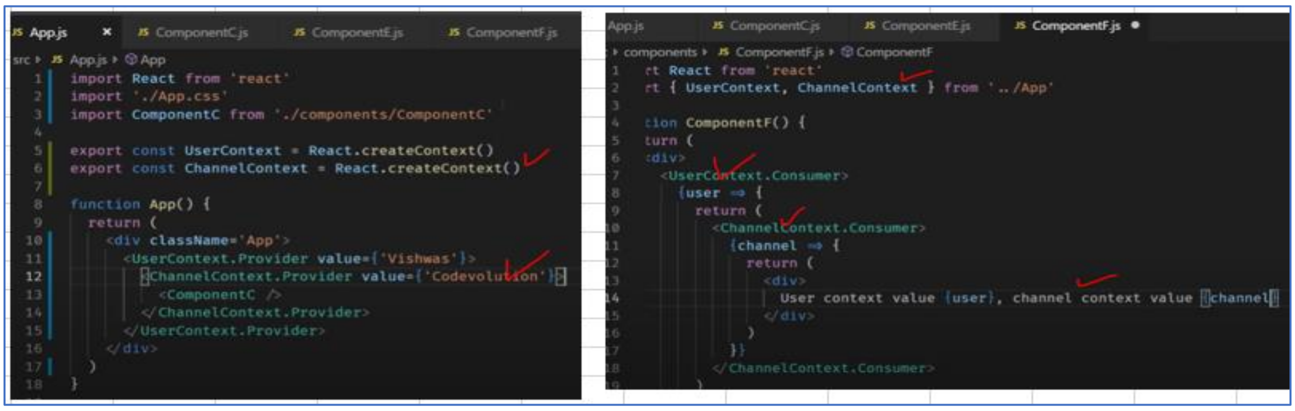
Context provides a way to pass data through the component tree without having to pass props down manually at every level.



```
import React from 'react'
import './App.css'
import ComponentC from './components/ComponentC'

export const UserContext = React.createContext()

function App() {
  return (
    <div className='App'>
      <UserContext.Provider value={'Vishwas'}>
        <ComponentC />
      </UserContext.Provider>
    </div>
  )
}

export default App
```

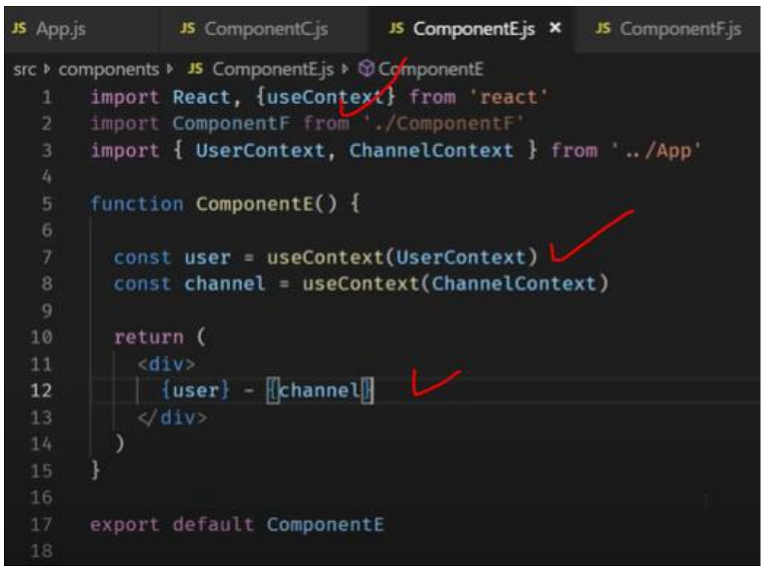
```
import React from 'react'
import {UserContext} from '../App'

function ComponentF() {
  return (
    <div>
      <UserContext.Consumer>
        {
          user => {
            return <div>User context value {user}</div>
          }
        }
      </UserContext.Consumer>
    </div>
  )
}

export default ComponentF
```

```
localhost:3000

User context value Vishwas
```



```
import React from 'react'
import './App.css'
import ComponentC from './components/ComponentC'

export const UserContext = React.createContext()
export const ChannelContext = React.createContext()

function App() {
  return (
    <div className='App'>
      <UserContext.Provider value={'Vishwas'}>
        <ChannelContext.Provider value={'Codevolution'}>
          <ComponentC />
        </ChannelContext.Provider>
      </UserContext.Provider>
    </div>
  )
}
```

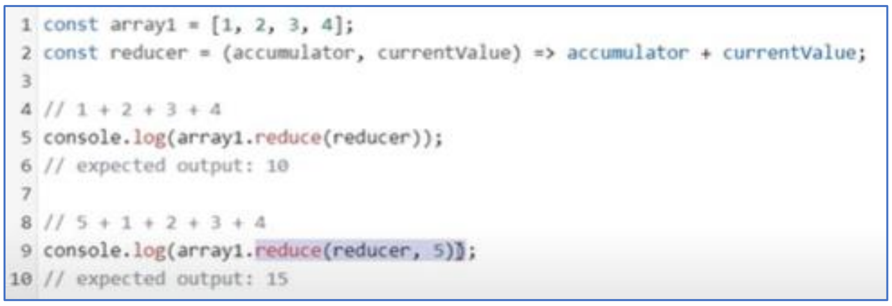
```
rt React from 'react'
rt { UserContext, ChannelContext } from '../App'

tion ComponentF() {
turn (
:div>
  <UserContext.Consumer>
    {user => {
      return (
        <ChannelContext.Consumer>
          {channel => {
            return (
              <div>
                User context value {user}, channel context value {channel}
              </div>
            )
          }}
        </ChannelContext.Consumer>
```



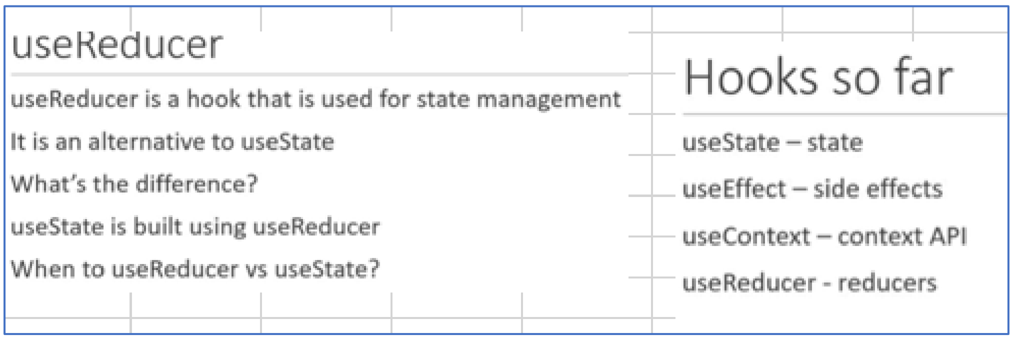
```
import React, {useContext} from 'react'
import ComponentF from './ComponentF'
import { UserContext, ChannelContext } from '../App'

function ComponentE() {

  const user = useContext(UserContext)
  const channel = useContext(ChannelContext)

  return (
    <div>
      {user} - {channel}
    </div>
  )
}

export default ComponentE
```

39. useReducer

- useReducer => is a hook that use for statemanagement, useReducer is related to reducer function



```
const array1 = [1, 2, 3, 4];
const reducer = (accumulator, currentValue) => accumulator + currentValue;

// 1 + 2 + 3 + 4
console.log(array1.reduce(reducer));
// expected output: 10

// 5 + 1 + 2 + 3 + 4
console.log(array1.reduce(reducer, 5));
// expected output: 15
```



useReducer

useReducer is a hook that is used for state management

It is an alternative to useState

What's the difference?

useState is built using useReducer

When to useReducer vs useState?

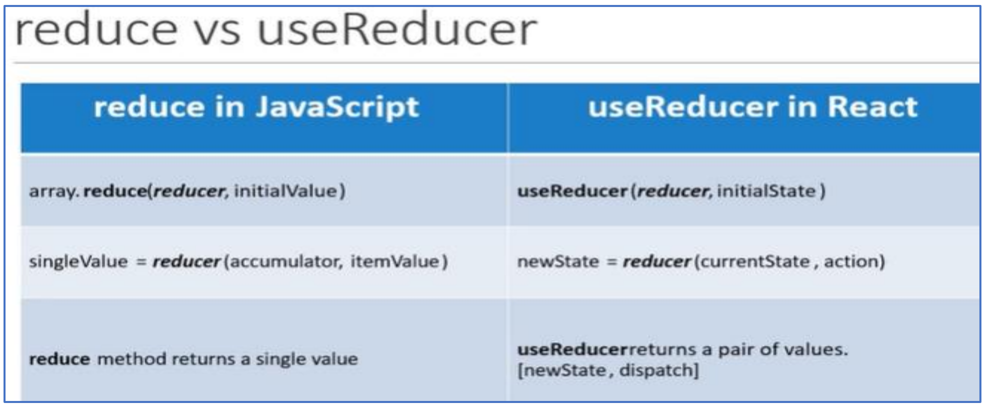
Hooks so far

useState – state

useEffect – side effects

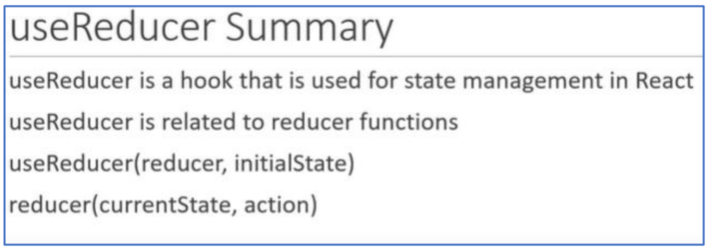
useContext – context API

useReducer - reducers



reduce vs useReducer

reduce in JavaScript	useReducer in React
array. **reduce(*reducer*,** initialValue)	**useReducer**(***reducer***, initialState)
singleValue = ***reducer***(accumulator, itemValue)	newState = ***reducer***(currentState , action)
reduce method returns a single value	**useReducer**returns a pair of values. [newState , dispatch]



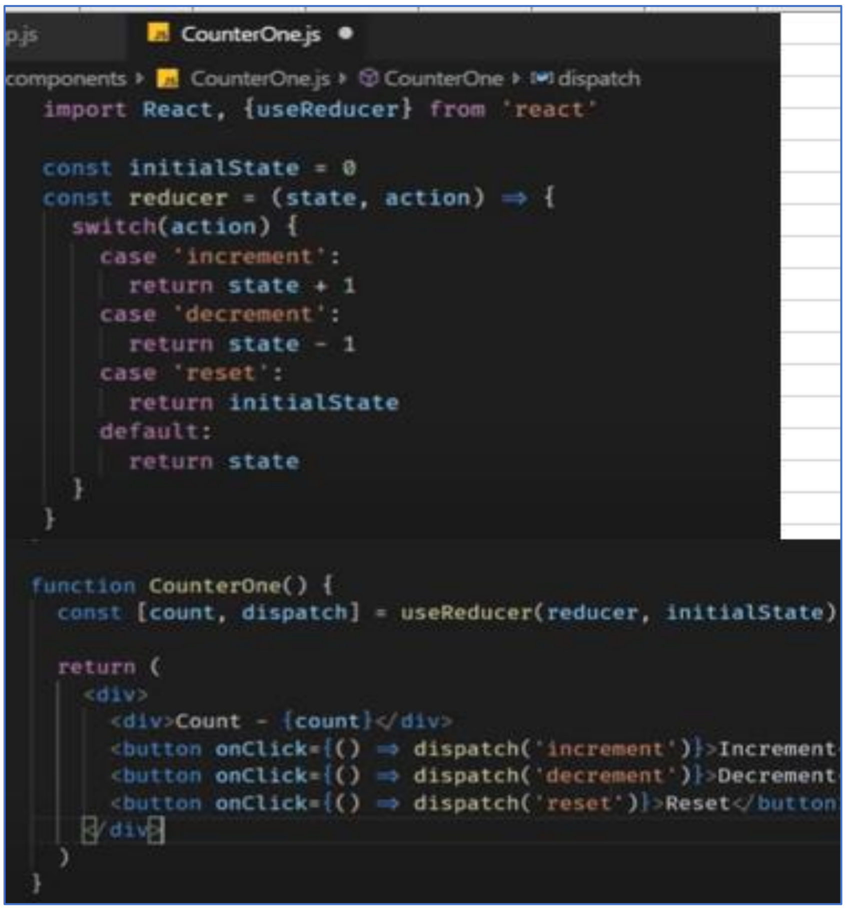
useReducer Summary

useReducer is a hook that is used for state management in React

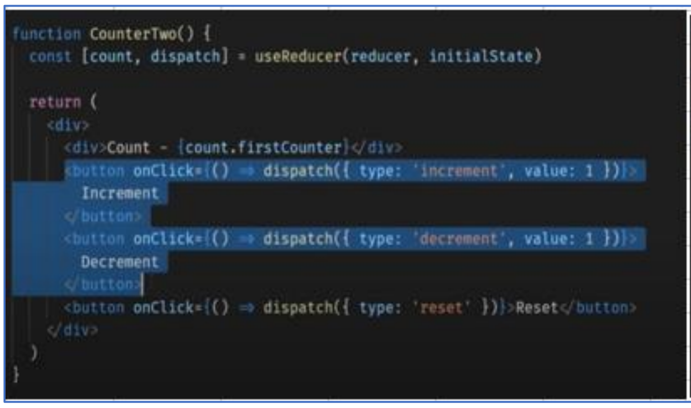
useReducer is related to reducer functions

useReducer(reducer, initialState)

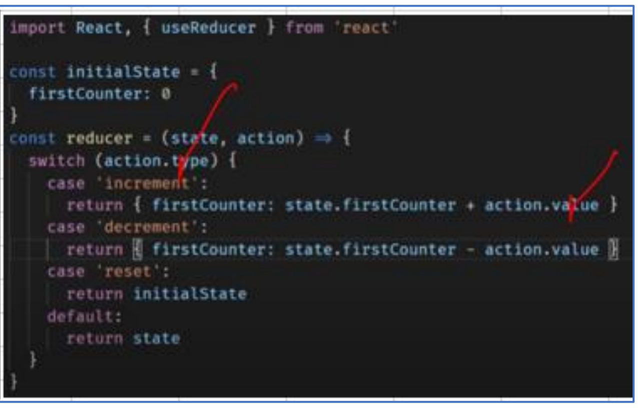
reducer(currentState, action)



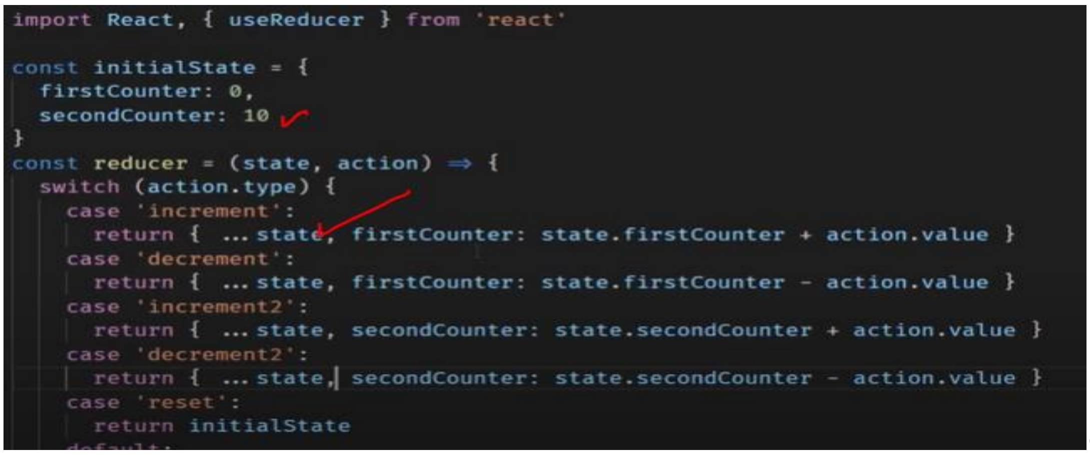
```
p.js    CounterOne.js ●
components > CounterOne.js > CounterOne > dispatch
import React, {useReducer} from 'react'

const initialState = 0
const reducer = (state, action) => {
  switch(action) {
    case 'increment':
      return state + 1
    case 'decrement':
      return state - 1
    case 'reset':
      return initialState
    default:
      return state
  }
}

function CounterOne() {
  const [count, dispatch] = useReducer(reducer, initialState)

  return (
    <div>
      <div>Count - {count}</div>
      <button onClick={() => dispatch('increment')}>Increment
      <button onClick={() => dispatch('decrement')}>Decrement
      <button onClick={() => dispatch('reset')}>Reset</button
    </div>
  )
}
```



```
function CounterTwo() {
  const [count, dispatch] = useReducer(reducer, initialState)

  return (
    <div>
      <div>Count - {count.firstCounter}</div>
      <button onClick={() => dispatch({ type: 'increment', value: 1 })}>
        Increment
      </button>
      <button onClick={() => dispatch({ type: 'decrement', value: 1 })}>
        Decrement
      </button>
      <button onClick={() => dispatch({ type: 'reset' })}>Reset</button>
    </div>
  )
}
```

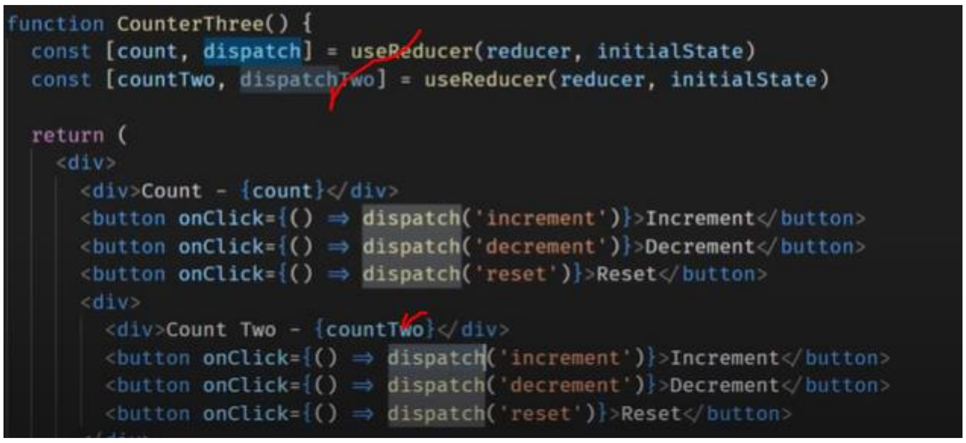


```
import React, { useReducer } from 'react'

const initialState = {
  firstCounter: 0
}
const reducer = (state, action) => {
  switch (action.type) {
    case 'increment':
      return { firstCounter: state.firstCounter + action.value }
    case 'decrement':
      return { firstCounter: state.firstCounter - action.value }
    case 'reset':
      return initialState
    default:
      return state
  }
}
```



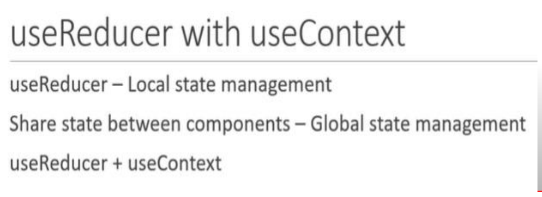
```
import React, { useReducer } from 'react'

const initialState = {
  firstCounter: 0,
  secondCounter: 10
}
const reducer = (state, action) => {
  switch (action.type) {
    case 'increment':
      return { ...state, firstCounter: state.firstCounter + action.value }
    case 'decrement':
      return { ...state, firstCounter: state.firstCounter - action.value }
    case 'increment2':
      return { ...state, secondCounter: state.secondCounter + action.value }
    case 'decrement2':
      return { ...state, secondCounter: state.secondCounter - action.value }
    case 'reset':
      return initialState
```

Clone



```
function CounterThree() {
  const [count, dispatch] = useReducer(reducer, initialState)
  const [countTwo, dispatchTwo] = useReducer(reducer, initialState)

  return (
    <div>
      <div>Count - {count}</div>
      <button onClick={() => dispatch('increment')}>Increment</button>
      <button onClick={() => dispatch('decrement')}>Decrement</button>
      <button onClick={() => dispatch('reset')}>Reset</button>
      <div>
        <div>Count Two - {countTwo}</div>
        <button onClick={() => dispatch('increment')}>Increment</button>
        <button onClick={() => dispatch('decrement')}>Decrement</button>
        <button onClick={() => dispatch('reset')}>Reset</button>
```

multiple reducers

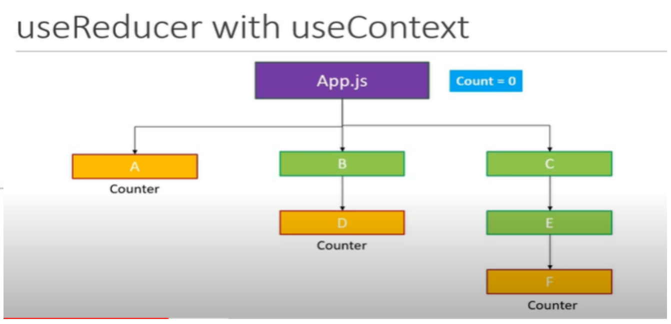


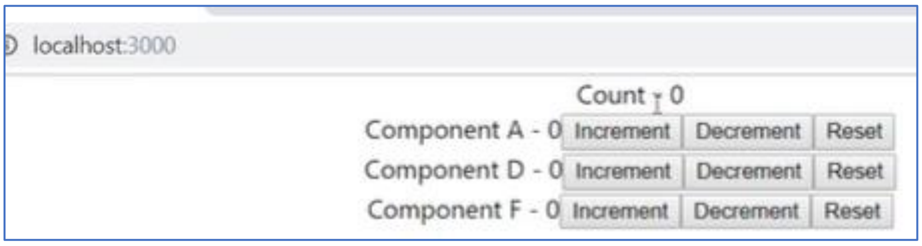
useReducer with useContext

useReducer – Local state management

Share state between components – Global state management

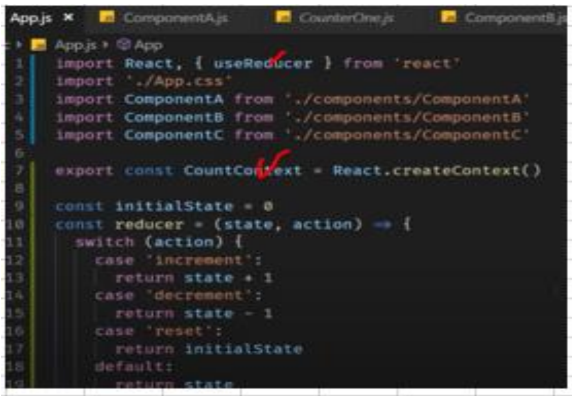
useReducer + useContext





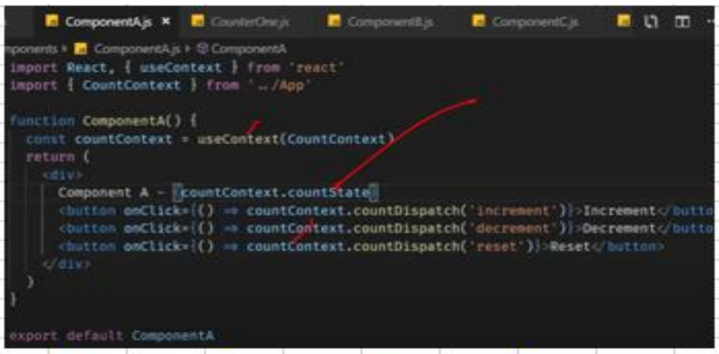
localhost:3000

Count - 0
Component A - 0 Increment Decrement Reset
Component D - 0 Increment Decrement Reset
Component F - 0 Increment Decrement Reset



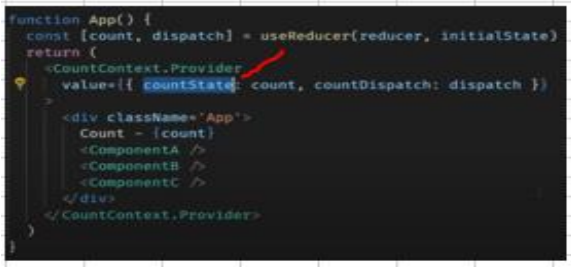
App.js

```
import React, { useReducer } from 'react'
import './App.css'
import ComponentA from './components/ComponentA'
import ComponentB from './components/ComponentB'
import ComponentC from './components/ComponentC'

export const CountContext = React.createContext()

const initialState = 0
const reducer = (state, action) => {
  switch (action) {
    case 'increment':
      return state + 1
    case 'decrement':
      return state - 1
    case 'reset':
      return initialState
    default:
      return state
```

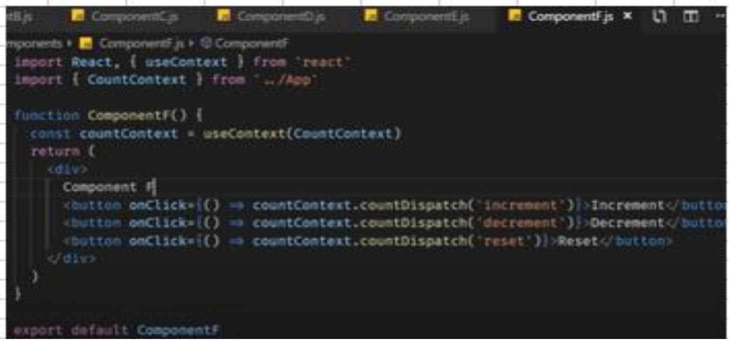


ComponentA.js

```
import React, { useContext } from 'react'
import { CountContext } from '../App'

function ComponentA() {
  const countContext = useContext(CountContext)
  return (
    <div>
      Component A - {countContext.countState}
      <button onClick={() => countContext.countDispatch('increment')}>Increment</butto
      <button onClick={() => countContext.countDispatch('decrement')}>Decrement</butto
      <button onClick={() => countContext.countDispatch('reset')}>Reset</button>
    </div>
  )
}

export default ComponentA
```

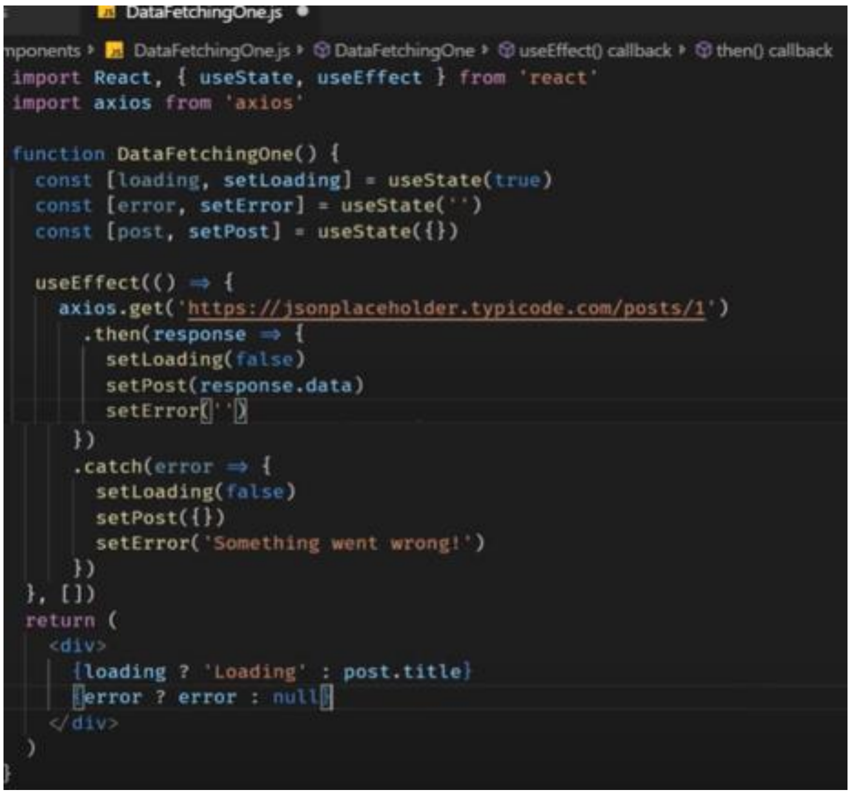


```
function App() {
  const [count, dispatch] = useReducer(reducer, initialState)
  return (
    <CountContext.Provider
      value={{ countState: count, countDispatch: dispatch }}
    >
      <div className='App'>
        Count - {count}
        <ComponentA />
        <ComponentB />
        <ComponentC />
      </div>
    </CountContext.Provider>
  )
}
```



ComponentF.js

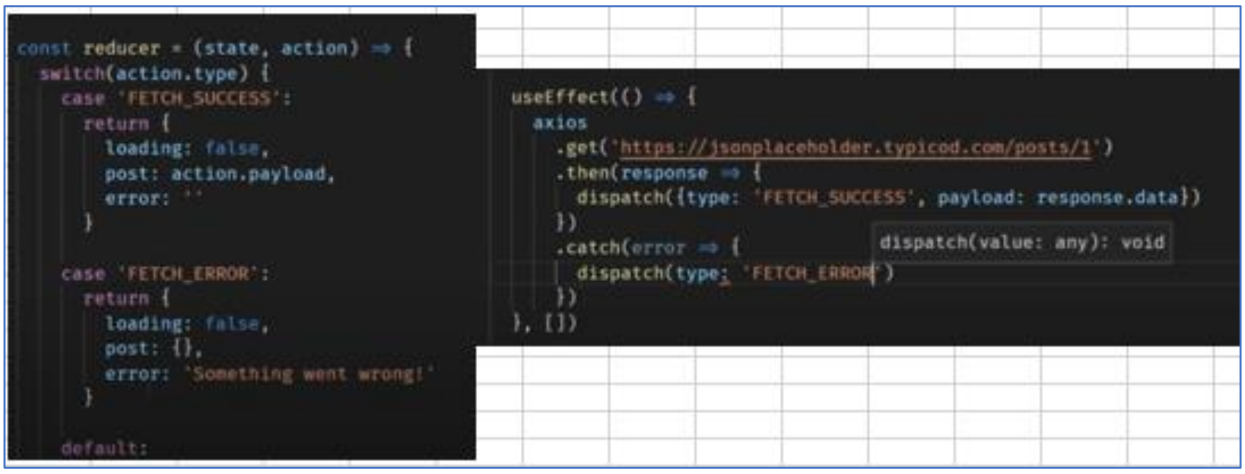
```
import React, { useContext } from 'react'
import { CountContext } from '../App'

function ComponentF() {
  const countContext = useContext(CountContext)
  return (
    <div>
      Component F
      <button onClick={() => countContext.countDispatch('increment')}>Increment</butto
      <button onClick={() => countContext.countDispatch('decrement')}>Decrement</butto
      <button onClick={() => countContext.countDispatch('reset')}>Reset</button>
    </div>
  )
}

export default ComponentF
```



DataFetchingOne.js
mponents › DataFetchingOne.js › DataFetchingOne › useEffect() callback › then() callback

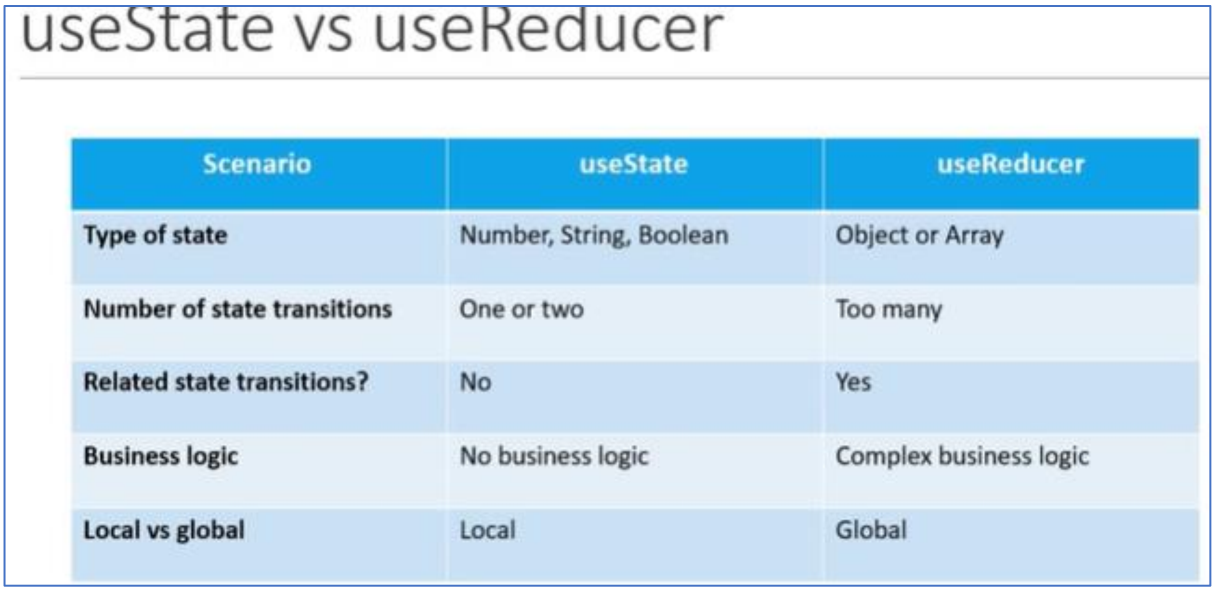
```
import React, { useState, useEffect } from 'react'
import axios from 'axios'

function DataFetchingOne() {
  const [loading, setLoading] = useState(true)
  const [error, setError] = useState('')
  const [post, setPost] = useState({})

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/posts/1')
      .then(response => {
        setLoading(false)
        setPost(response.data)
        setError('')
     })
     .catch(error => {
       setLoading(false)
       setPost({})
       setError('Something went wrong!')
     })
  }, [])
  return (
    <div>
      {loading ? 'Loading' : post.title}
      {error ? error : null}
    </div>
  )
}
```

- **Usestate** -> Number, string, boolean ,	**usereducer** -> Object or array
- **usestate** -> one or two number of tran ,	**usereducer** -> too many number of tran
- **useSatte** -> no read state tran ,	**useReducer** -> yes read state tran
- **useState** -> no business logic,	**useReducer** -> complex business logic
- **useState** -> local val,	**useReducer** -> global val



```
const reducer = (state, action) => {
  switch(action.type) {
    case 'FETCH_SUCCESS':
      return {
        loading: false,
        post: action.payload,
        error: ''
      }

    case 'FETCH_ERROR':
      return {
        loading: false,
        post: {},
        error: 'Something went wrong!'
      }

    default:
```

```
useEffect(() => {
  axios
    .get('https://jsonplaceholder.typicod.com/posts/1')
    .then(response => {
      dispatch({type: 'FETCH_SUCCESS', payload: response.data})
    })
    .catch(error => {                         dispatch(value: any): void
      dispatch(type: 'FETCH_ERROR')
    })
}, [])
```



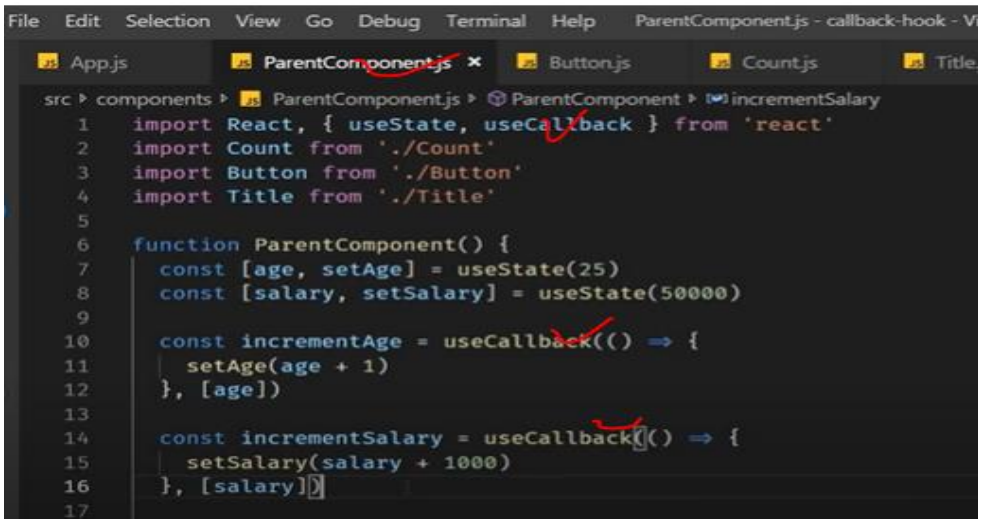
useState vs useReducer

Scenario	useState	useReducer
Type of state	Number, String, Boolean	Object or Array
Number of state transitions	One or two	Too many
Related state transitions?	No	Yes
Business logic	No business logic	Complex business logic
Local vs global	Local	Global

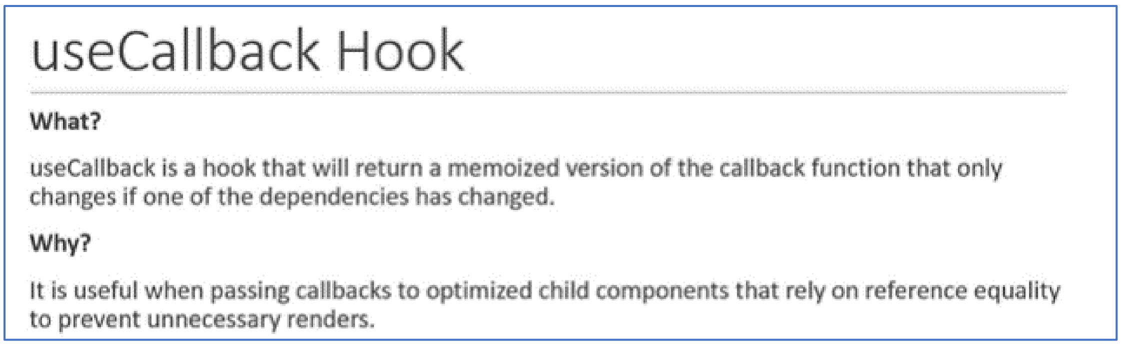
40. <u>UseCallback</u>

- only re-render specific component for better performance

- const incrementAge = useCallback(() => {setAge(age+1)}, [age]) // re-render based on age change


File Edit Selection View Go Debug Terminal Help ParentComponent.js - callback-hook - V
App.js ParentComponent.js Button.js Count.js Title
src › components › ParentComponent.js › ParentComponent › incrementSalary

```
import React, { useState, useCallback } from 'react'
import Count from './Count'
import Button from './Button'
import Title from './Title'

function ParentComponent() {
  const [age, setAge] = useState(25)
  const [salary, setSalary] = useState(50000)

  const incrementAge = useCallback(() => {
    setAge(age + 1)
  }, [age])

  const incrementSalary = useCallback(() => {
    setSalary(salary + 1000)
  }, [salary])
```



useCallback Hook

What?

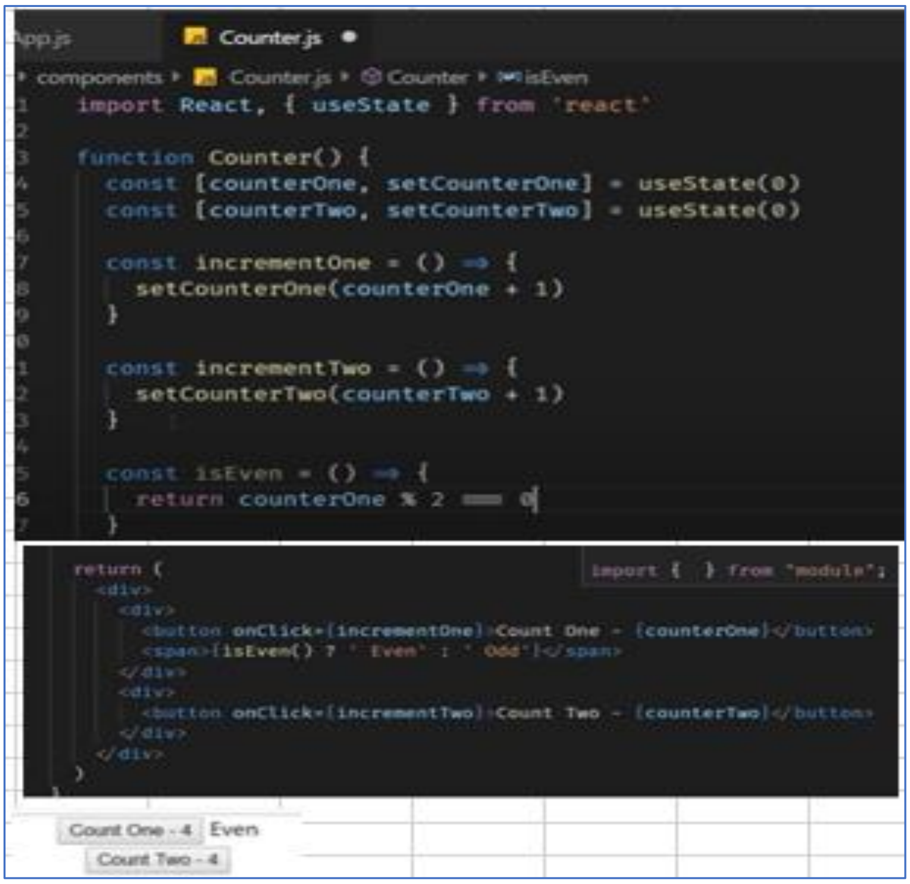
useCallback is a hook that will return a memoized version of the callback function that only changes if one of the dependencies has changed.

Why?

It is useful when passing callbacks to optimized child components that rely on reference equality to prevent unnecessary renders.

41. <u>UseMemo Hook</u>

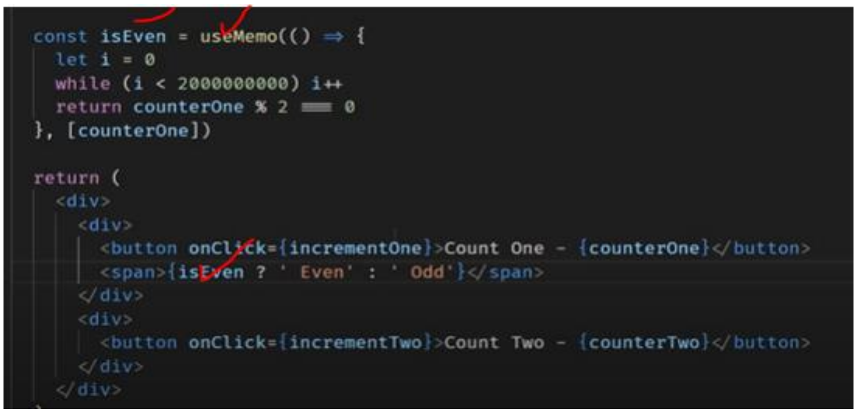
usMemo is a hook that will only re-compute that one of the dependancy changed


App.js
Counter.js
components › Counter.js › Counter › isEven

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

function Counter() {
  const [counterOne, setCounterOne] = useState(0)
  const [counterTwo, setCounterTwo] = useState(0)

  const incrementOne = () => {
    setCounterOne(counterOne + 1)
  }

  const incrementTwo = () => {
    setCounterTwo(counterTwo + 1)
  }

  const isEven = () => {
    return counterOne % 2 === 0
  }
```

```
  return (
    <div>
      <div>
        <button onClick={incrementOne}>Count One - {counterOne}</button>
        <span>{isEven() ? ' Even' : ' Odd'}</span>
      </div>
      <div>
        <button onClick={incrementTwo}>Count Two - {counterTwo}</button>
      </div>
    </div>
  )
}
```

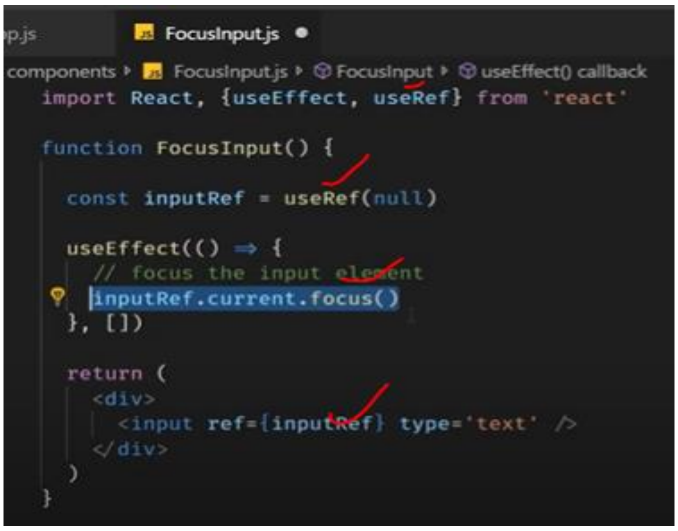
import { } from "module";
Count One - 4
Even
Count Two - 4



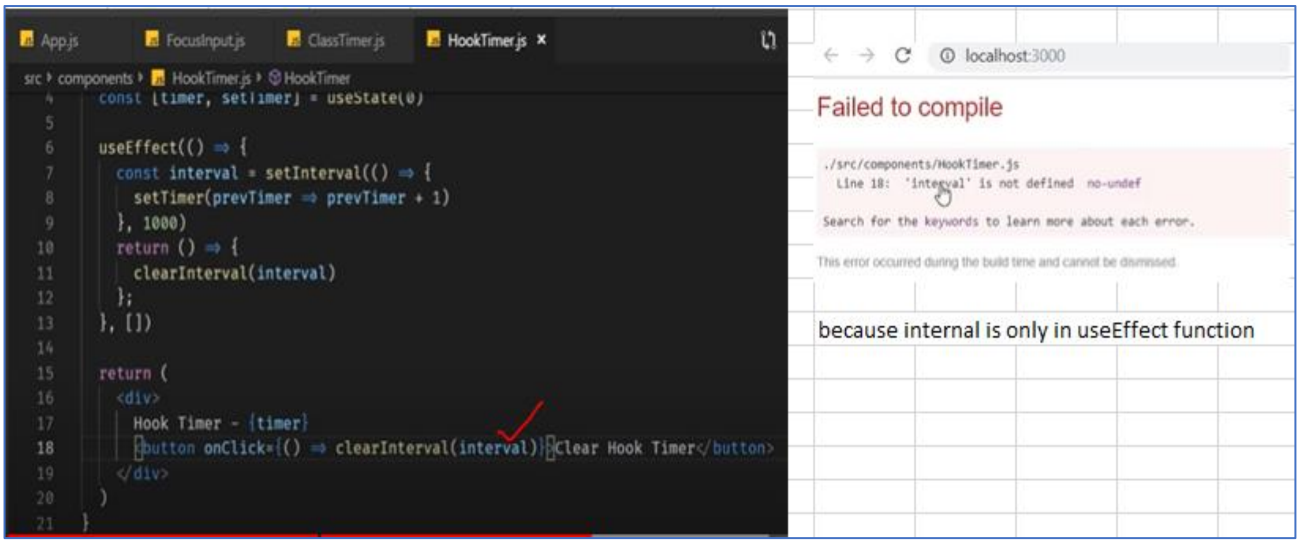
```
const isEven = useMemo(() => {
  let i = 0
  while (i < 2000000000) i++
  return counterOne % 2 === 0
}, [counterOne])

return (
  <div>
    <div>
      <button onClick={incrementOne}>Count One - {counterOne}</button>
      <span>{isEven ? ' Even' : ' Odd'}</span>
    </div>
    <div>
      <button onClick={incrementTwo}>Count Two - {counterTwo}</button>
    </div>
  </div>
```

42. <u>UseRef Hook</u>

useRef is hook to access DOM node directly from functional component


p.js
FocusInput.js
components › FocusInput.js › FocusInput › useEffect() callback

```
import React, {useEffect, useRef} from 'react'

function FocusInput() {

  const inputRef = useRef(null)

  useEffect(() => {
    // focus the input element
    inputRef.current.focus()
  }, [])

  return (
    <div>
      <input ref={inputRef} type='text' />
    </div>
  )
}
```



```
  const [timer, setTimer] = useState(0)

  useEffect(() => {
    const interval = setInterval(() => {
      setTimer(prevTimer => prevTimer + 1)
    }, 1000)
    return () => {
      clearInterval(interval)
    };
  }, [])

  return (
    <div>
      Hook Timer - {timer}
      <button onClick={() => clearInterval(interval)}>Clear Hook Timer</button>
    </div>
  )
}
```

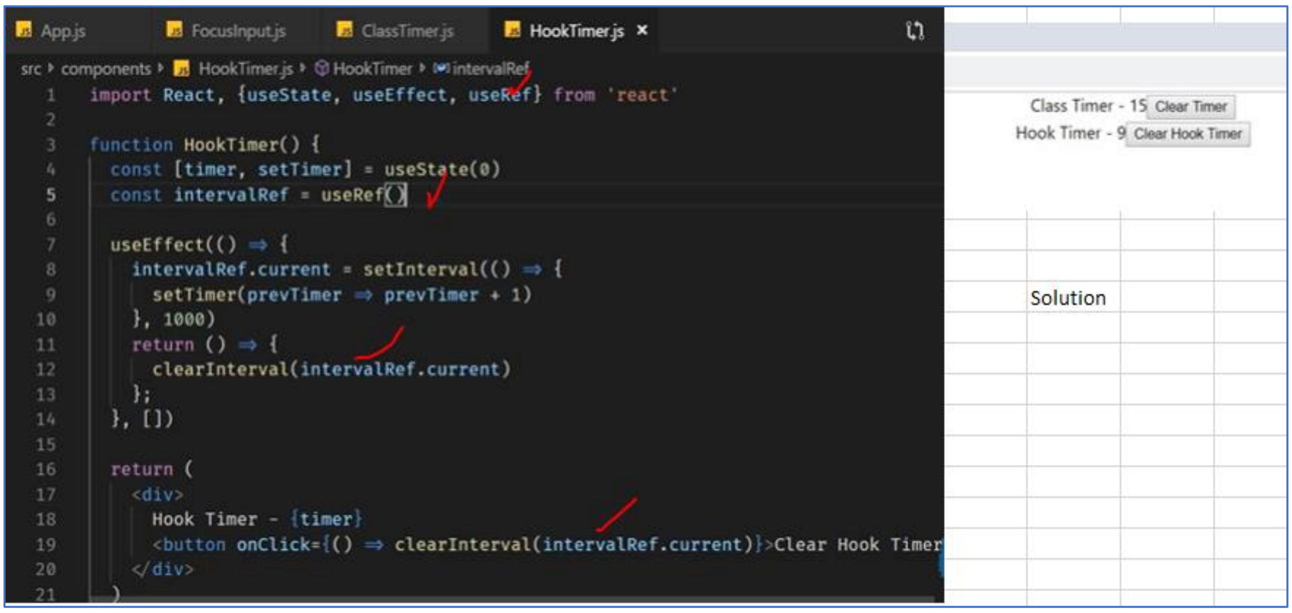
localhost:3000

Failed to compile

```
./src/components/HookTimer.js
  Line 18:  'interval' is not defined  no-undef

Search for the keywords to learn more about each error.
```

This error occurred during the build time and cannot be dismissed.

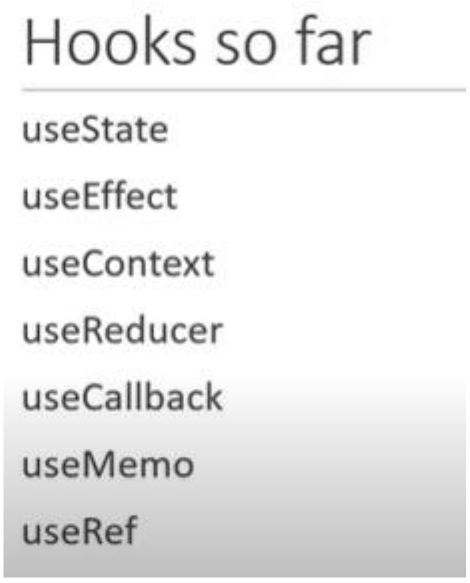
because internal is only in useEffect function



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

function HookTimer() {
  const [timer, setTimer] = useState(0)
  const intervalRef = useRef()

  useEffect(() => {
    intervalRef.current = setInterval(() => {
      setTimer(prevTimer => prevTimer + 1)
    }, 1000)
    return () => {
      clearInterval(intervalRef.current)
    };
  }, [])

  return (
    <div>
      Hook Timer - {timer}
      <button onClick={() => clearInterval(intervalRef.current)}>Clear Hook Timer
    </div>
  )
```

Class Timer - 15 Clear Timer
Hook Timer - 9 Clear Hook Timer

Solution



Hooks so far

useState
useEffect
useContext
useReducer
useCallback
useMemo
useRef

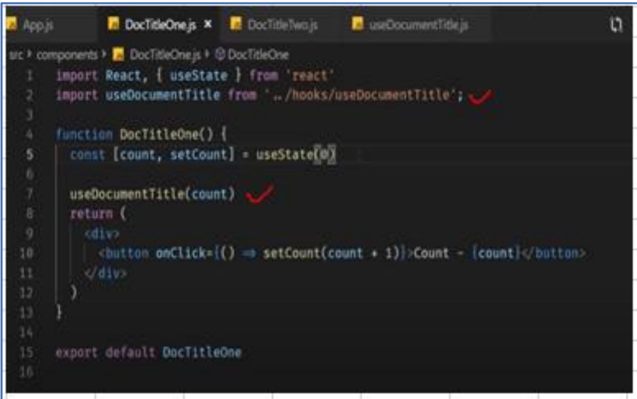
43. Custom Hook

Custom Hooks

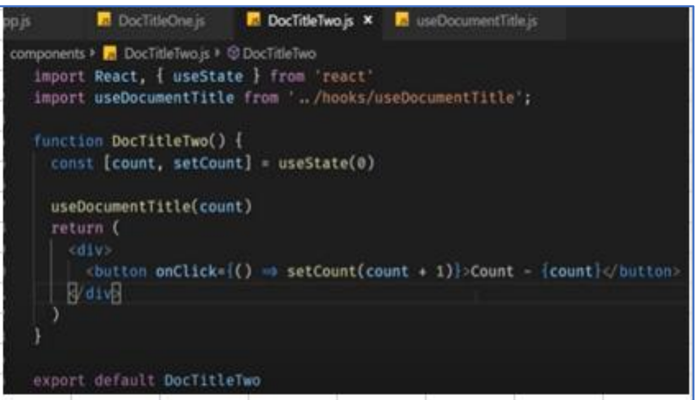
A custom Hook is basically a JavaScript function whose name starts with "use".
A custom hook can also call other Hooks if required.

Why?
Share logic – Alternative to HOCs and Render Props

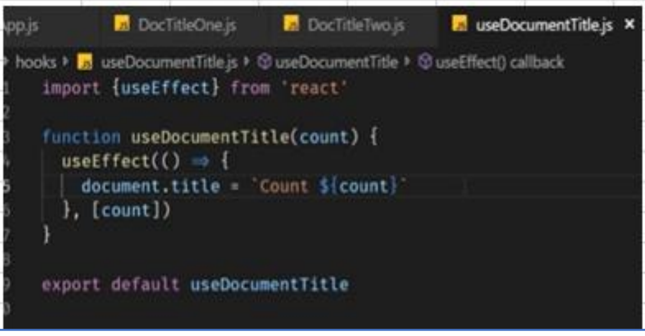
How to create custom hooks?



```
import React, { useState } from 'react'
import useDocumentTitle from '../hooks/useDocumentTitle';

function DocTitleOne() {
  const [count, setCount] = useState(0)

  useDocumentTitle(count)
  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Count - {count}</button>
    </div>
  )
}

export default DocTitleOne
```

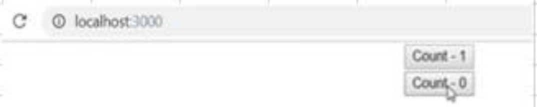


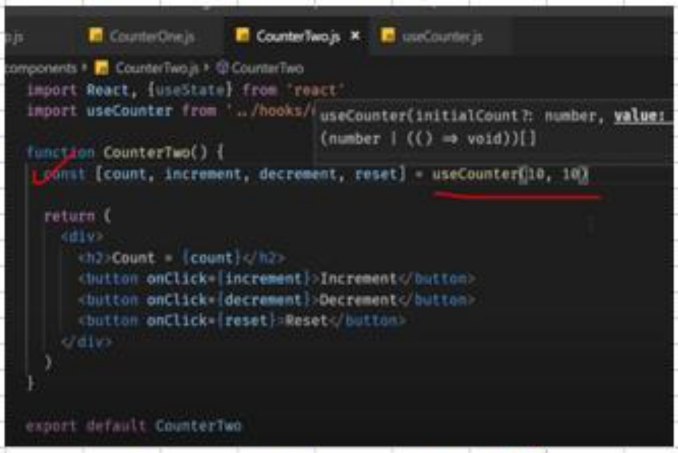
```
import React, { useState } from 'react'
import useDocumentTitle from '../hooks/useDocumentTitle';

function DocTitleTwo() {
  const [count, setCount] = useState(0)

  useDocumentTitle(count)
  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Count - {count}</button>
    </div>
  )
}

export default DocTitleTwo
```


pp.js
DocTitleOne.js
DocTitleTwo.js
useDocumentTitle.js
hooks › useDocumentTitle.js › useDocumentTitle › useEffect() callback

```
import {useEffect} from 'react'

function useDocumentTitle(count) {
  useEffect(() => {
    document.title = `Count ${count}`
  }, [count])
}

export default useDocumentTitle
```

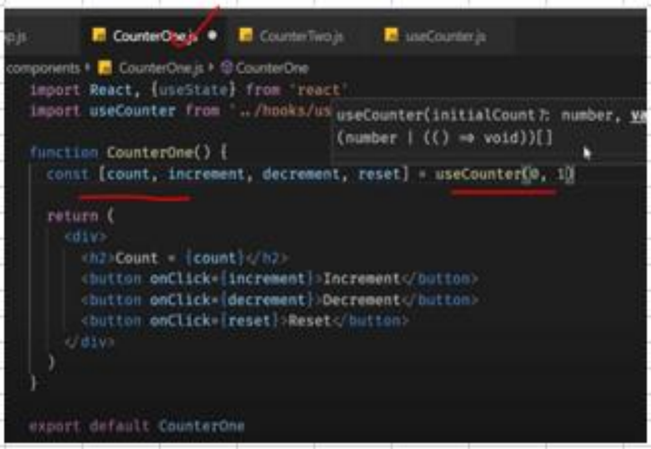
this is custom Hook





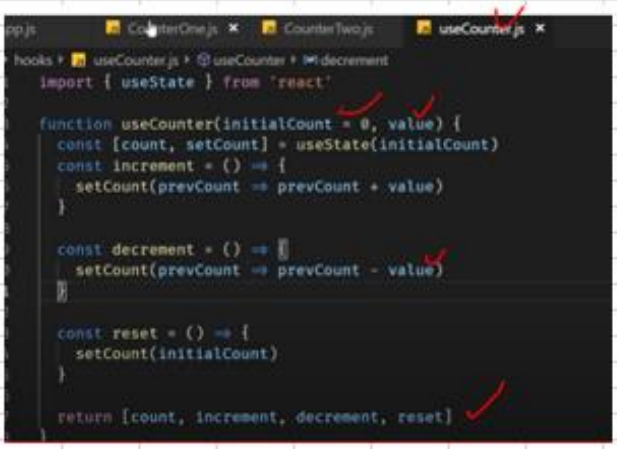
```
import React, {useState} from 'react'
import useCounter from '../hooks/useCounter'

function CounterTwo() {
  const [count, increment, decrement, reset] = useCounter(10, 10)

  return (
    <div>
      <h2>Count = {count}</h2>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
      <button onClick={reset}>Reset</button>
    </div>
  )
}

export default CounterTwo
```

```
useCounter(initialCount?: number, value:
(number | (() => void))[]
```

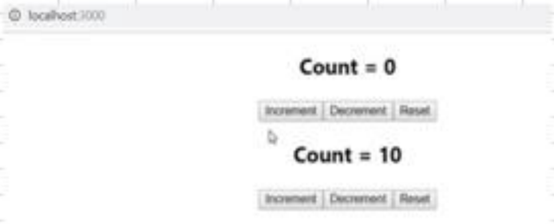


```
import React, {useState} from 'react'
import useCounter from '../hooks/useCounter'

function CounterOne() {
  const [count, increment, decrement, reset] = useCounter(0, 1)

  return (
    <div>
      <h2>Count = {count}</h2>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
      <button onClick={reset}>Reset</button>
    </div>
  )
}

export default CounterOne
```

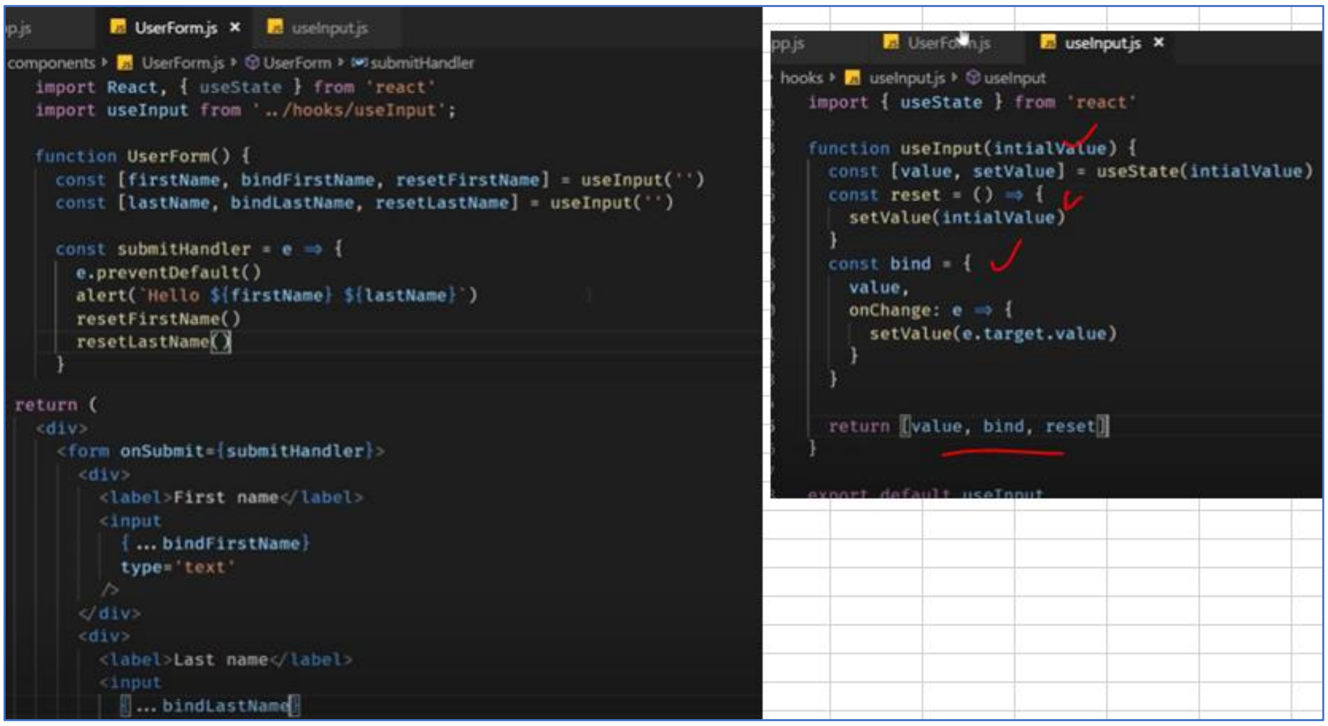
```
useCounter(initialCount?: number, va
(number | (() => void))[]
```



```
import { useState } from 'react'

function useCounter(initialCount = 0, value) {
  const [count, setCount] = useState(initialCount)
  const increment = () => {
    setCount(prevCount => prevCount + value)
  }

  const decrement = () => {
    setCount(prevCount => prevCount - value)
  }

  const reset = () => {
    setCount(initialCount)
  }

  return [count, increment, decrement, reset]
}
```



Custom Hook return Array and accept parameter

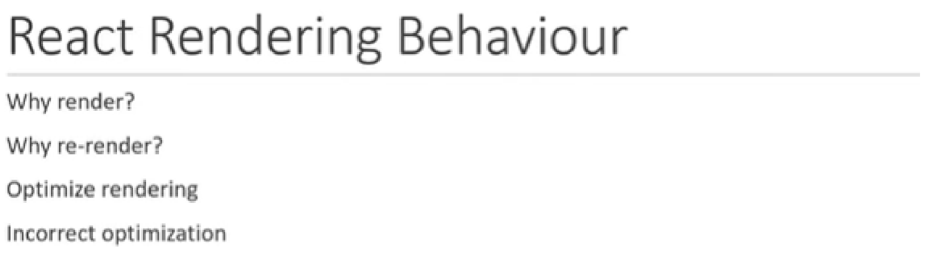

p.js
UserForm.js ×
useInput.js
components › UserForm.js › UserForm › submitHandler
pp.js
UserForm.js
useInput.js ×
hooks › useInput.js › useInput

```
import React, { useState } from 'react'
import useInput from '../hooks/useInput';

function UserForm() {
  const [firstName, bindFirstName, resetFirstName] = useInput('')
  const [lastName, bindLastName, resetLastName] = useInput('')

  const submitHandler = e => {
    e.preventDefault()
    alert(`Hello ${firstName} ${lastName}`)
    resetFirstName()
    resetLastName()
  }

return (
  <div>
    <form onSubmit={submitHandler}>
      <div>
        <label>First name</label>
        <input
          {...bindFirstName}
          type='text'
        />
      </div>
      <div>
        <label>Last name</label>
        <input
          {...bindLastName}
```

```
import { useState } from 'react'

function useInput(intialValue) {
  const [value, setValue] = useState(intialValue)
  const reset = () => {
    setValue(intialValue)
  }
  const bind = {
    value,
    onChange: e => {
      setValue(e.target.value)
    }
  }

  return [value, bind, reset]
}

export default useInput
```

User Input with custom Hook

44. React Render



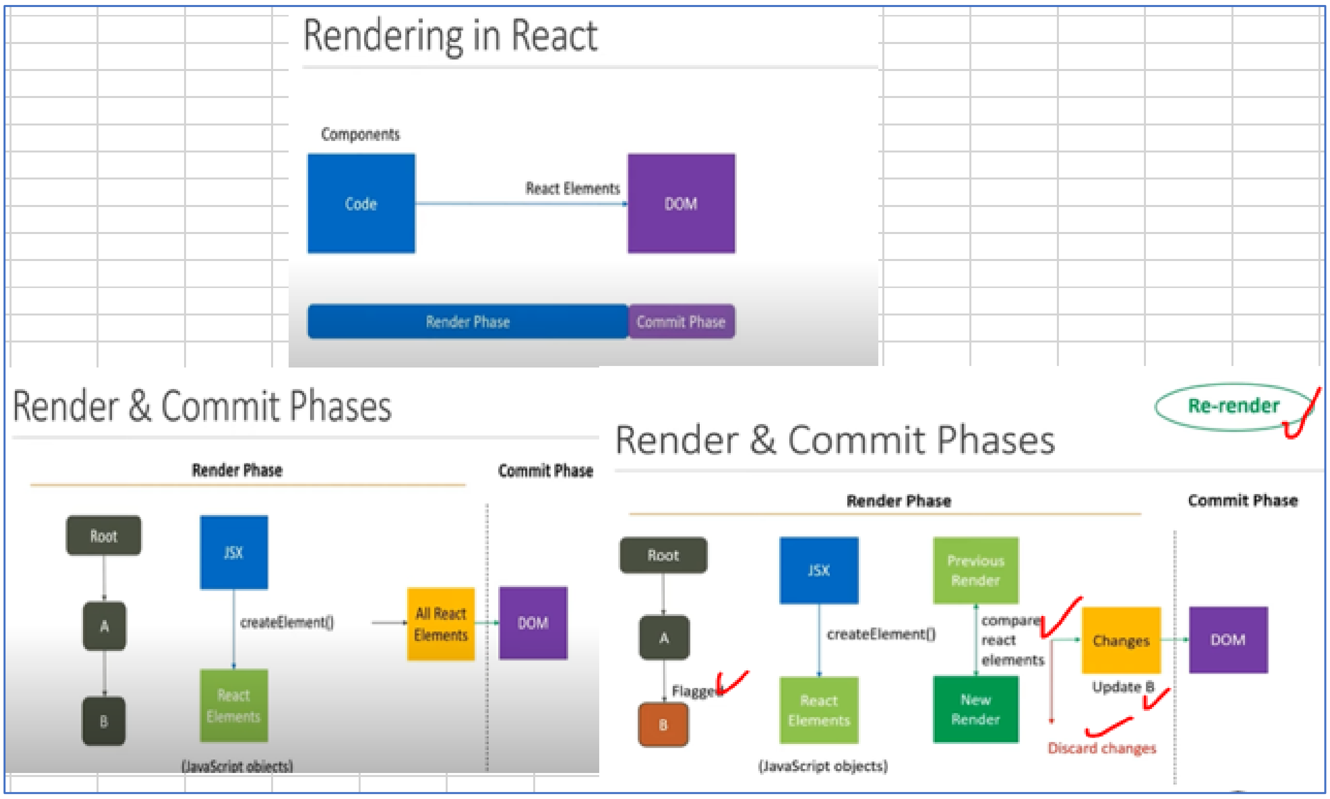
React Rendering Behaviour

Why render?

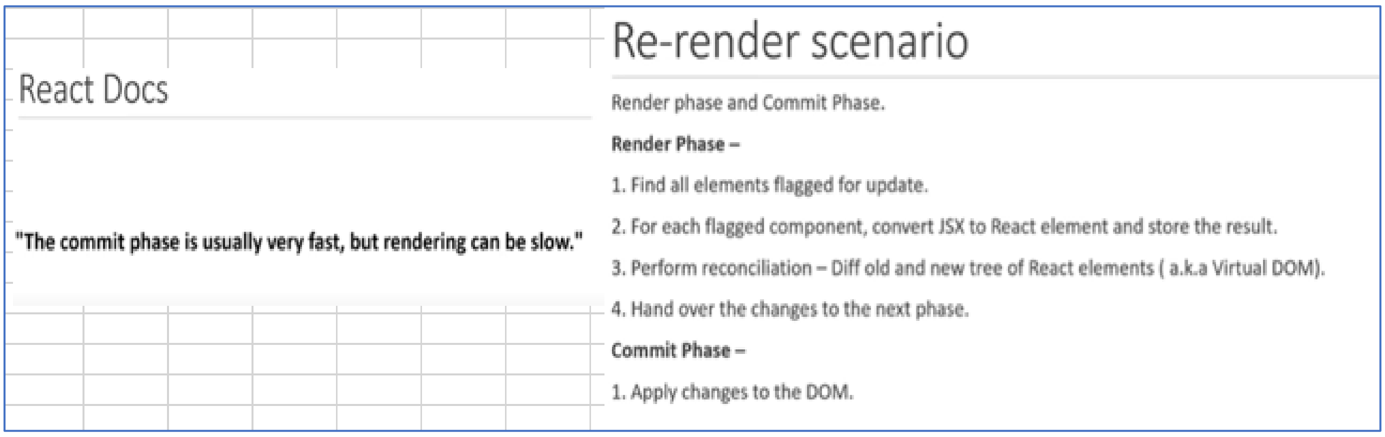
Why re-render?

Optimize rendering

Incorrect optimization



Compare the component are changed, if change update, or not for better performance



React Docs

"The commit phase is usually very fast, but rendering can be slow."

Re-render scenario

Render phase and Commit Phase.

Render Phase –

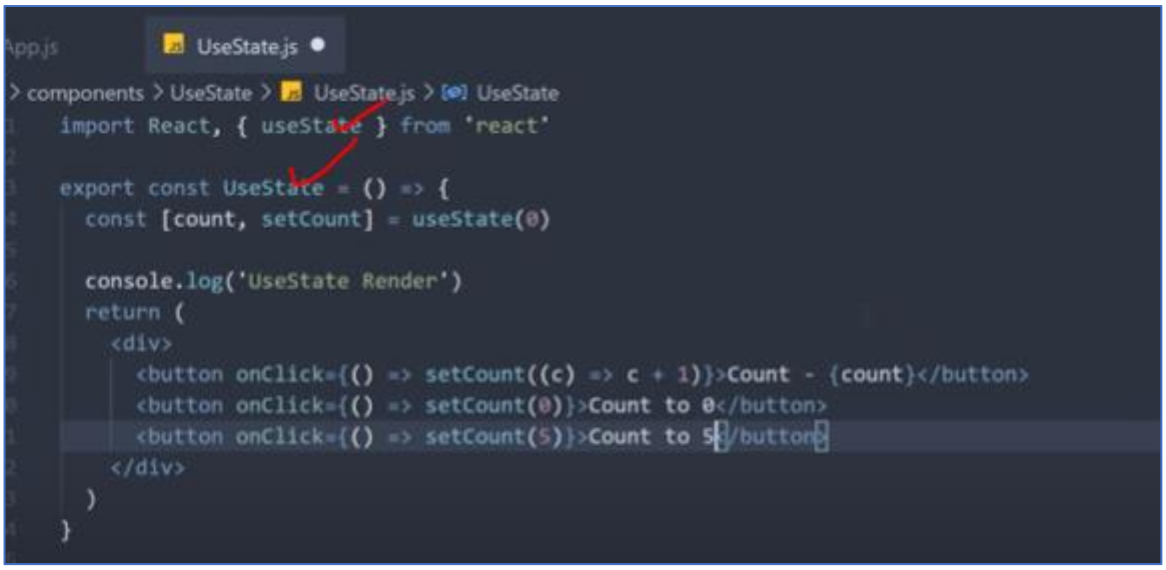
1. Find all elements flagged for update.
2. For each flagged component, convert JSX to React element and store the result.
3. Perform reconciliation – Diff old and new tree of React elements (a.k.a Virtual DOM).
4. Hand over the changes to the next phase.

Commit Phase –

1. Apply changes to the DOM.

45. <u>useState Re-render</u>

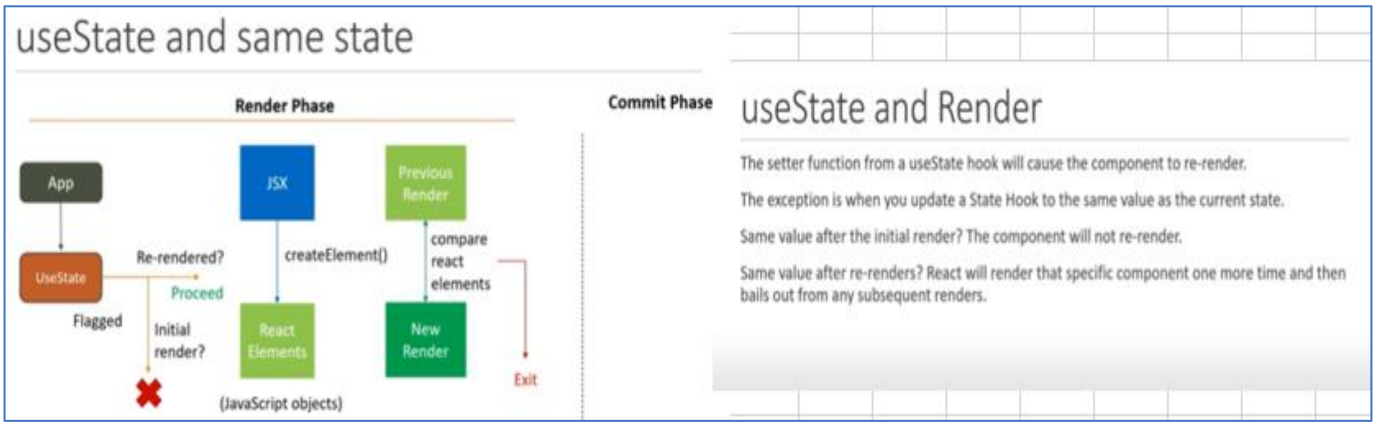
Check the state to re-render, if state is same not re-render



App.js   UseState.js

> components > UseState > UseState.js > UseState

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

export const UseState = () => {
  const [count, setCount] = useState(0)

  console.log('UseState Render')
  return (
    <div>
      <button onClick={() => setCount((c) => c + 1)}>Count - {count}</button>
      <button onClick={() => setCount(0)}>Count to 0</button>
      <button onClick={() => setCount(5)}>Count to 5</button>
    </div>
  )
}
```



useState and same state

useState and Render

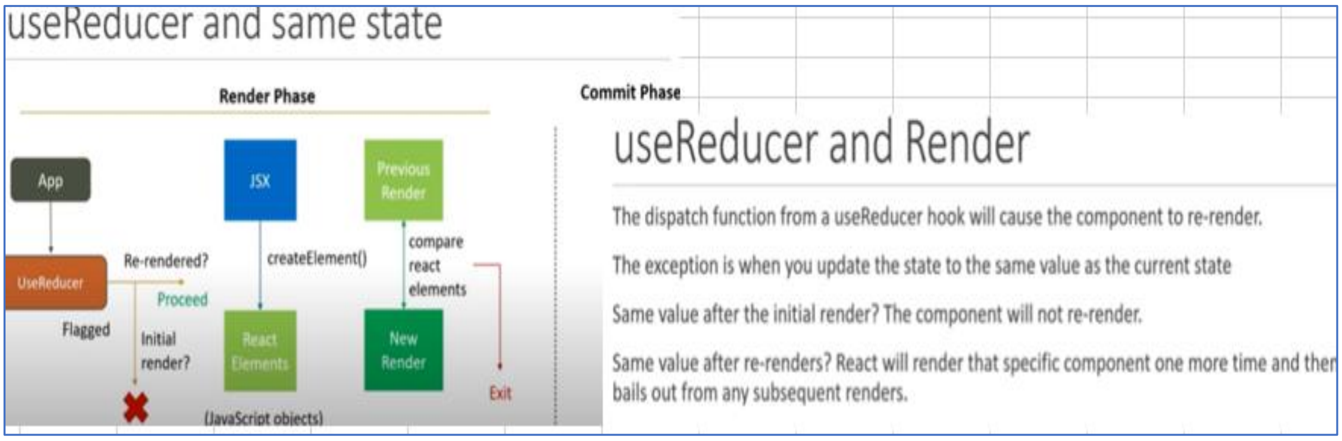
The setter function from a useState hook will cause the component to re-render.

The exception is when you update a State Hook to the same value as the current state.

Same value after the initial render? The component will not re-render.

Same value after re-renders? React will render that specific component one more time and then bails out from any subsequent renders.

46. <u>useReducer Re-Render</u>



useReducer and same state

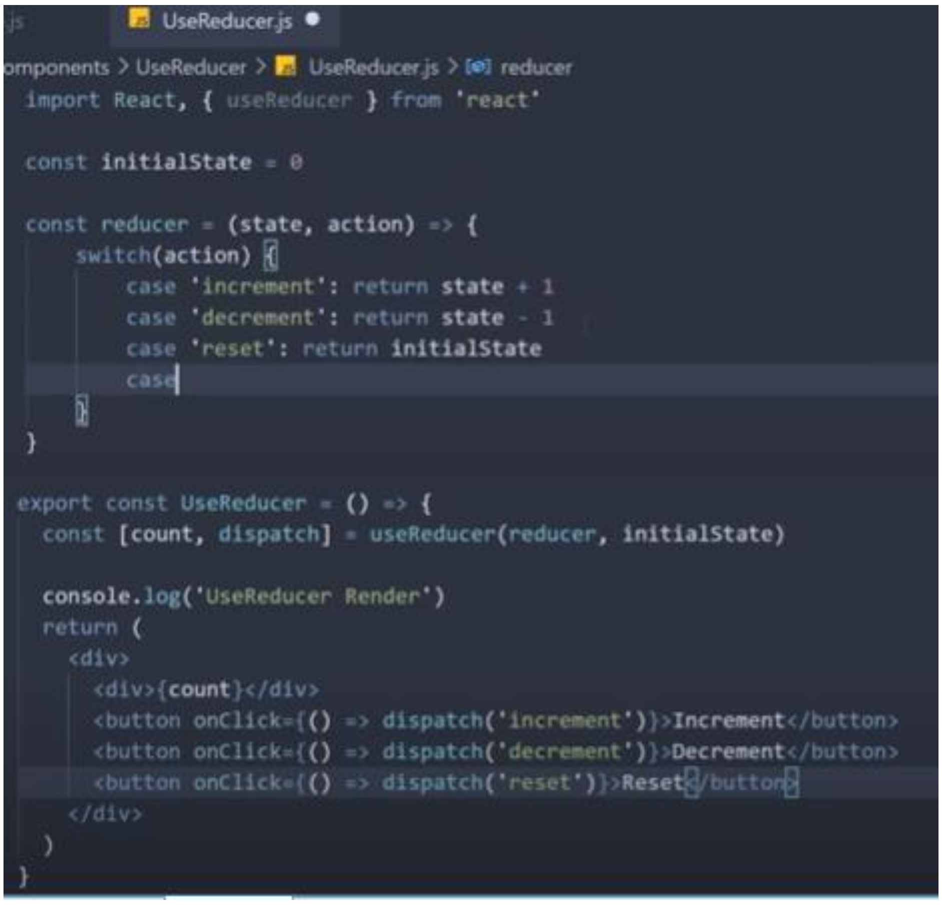
useReducer and Render

The dispatch function from a useReducer hook will cause the component to re-render.

The exception is when you update the state to the same value as the current state

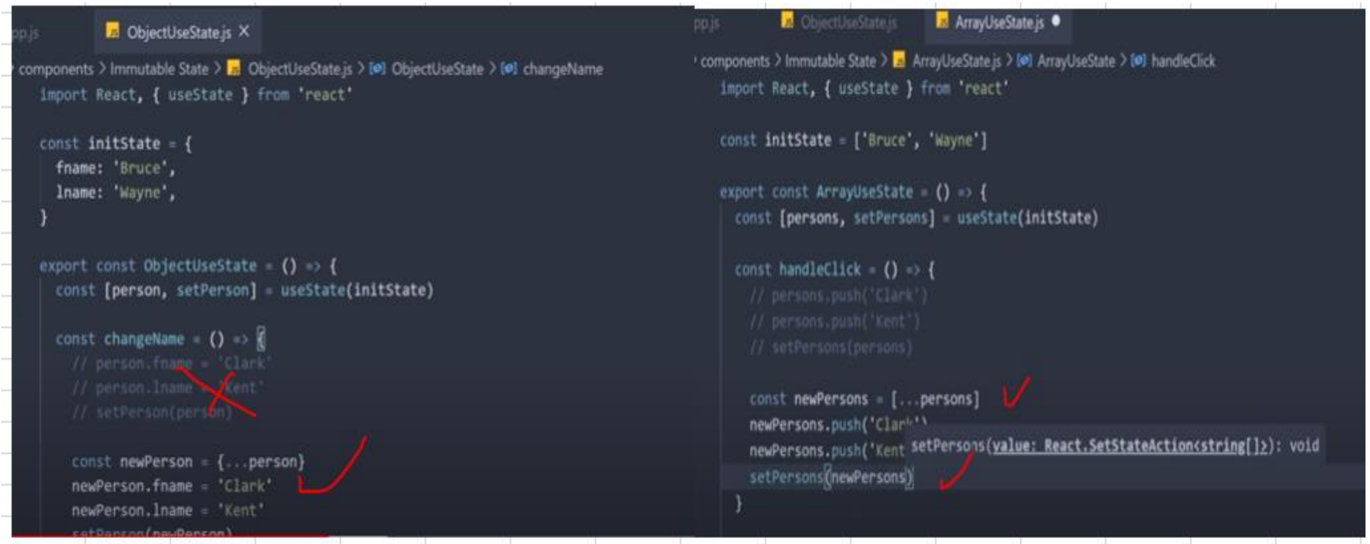
Same value after the initial render? The component will not re-render.

Same value after re-renders? React will render that specific component one more time and then bails out from any subsequent renders.



```
UseReducer.js
omponents > UseReducer > UseReducer.js > reducer
import React, { useReducer } from 'react'

const initialState = 0

const reducer = (state, action) => {
    switch(action) {
        case 'increment': return state + 1
        case 'decrement': return state - 1
        case 'reset': return initialState
        case
    }
}

export const UseReducer = () => {
  const [count, dispatch] = useReducer(reducer, initialState)

  console.log('UseReducer Render')
  return (
    <div>
      <div>{count}</div>
      <button onClick={() => dispatch('increment')}>Increment</button>
      <button onClick={() => dispatch('decrement')}>Decrement</button>
      <button onClick={() => dispatch('reset')}>Reset</button>
    </div>
  )
}
```

47. State Immutable Re-Render



```
ObjectUseState.js
components > Immutable State > ObjectUseState.js > ObjectUseState > changeName
import React, { useState } from 'react'

const initState = {
  fname: 'Bruce',
  lname: 'Wayne',
}

export const ObjectUseState = () => {
  const [person, setPerson] = useState(initState)

  const changeName = () => {
    // person.fname = 'Clark'
    // person.lname = 'Kent'
    // setPerson(person)

    const newPerson = {...person}
    newPerson.fname = 'Clark'
    newPerson.lname = 'Kent'
    setPerson(newPerson)
```

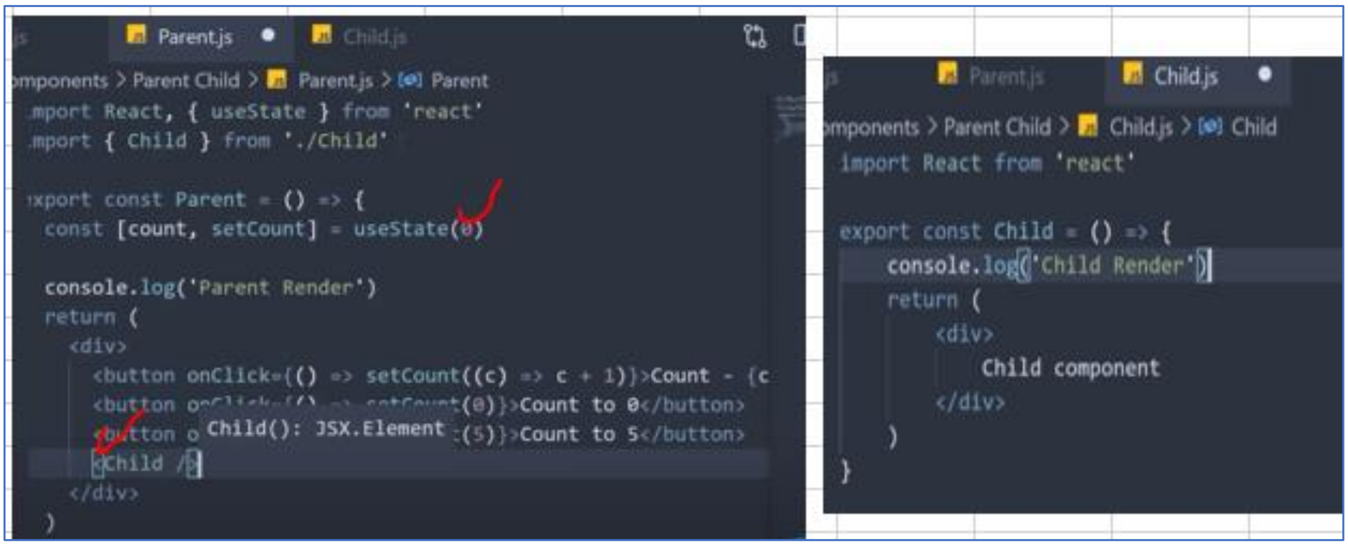
```
ArrayUseState.js
components > Immutable State > ArrayUseState.js > ArrayUseState > handleClick
import React, { useState } from 'react'

const initState = ['Bruce', 'Wayne']

export const ArrayUseState = () => {
  const [persons, setPersons] = useState(initState)

  const handleClick = () => {
    // persons.push('Clark')
    // persons.push('Kent')
    // setPersons(persons)

    const newPersons = [...persons]
    newPersons.push('Clark')
    newPersons.push('Kent setPersons(value: React.SetStateAction<string[]>): void
    setPersons(newPersons)
  }
```

Person Object do not changes and can't re-render bacause directly update to person obj

Object And Array are Value changed but Ref do not change so immutable

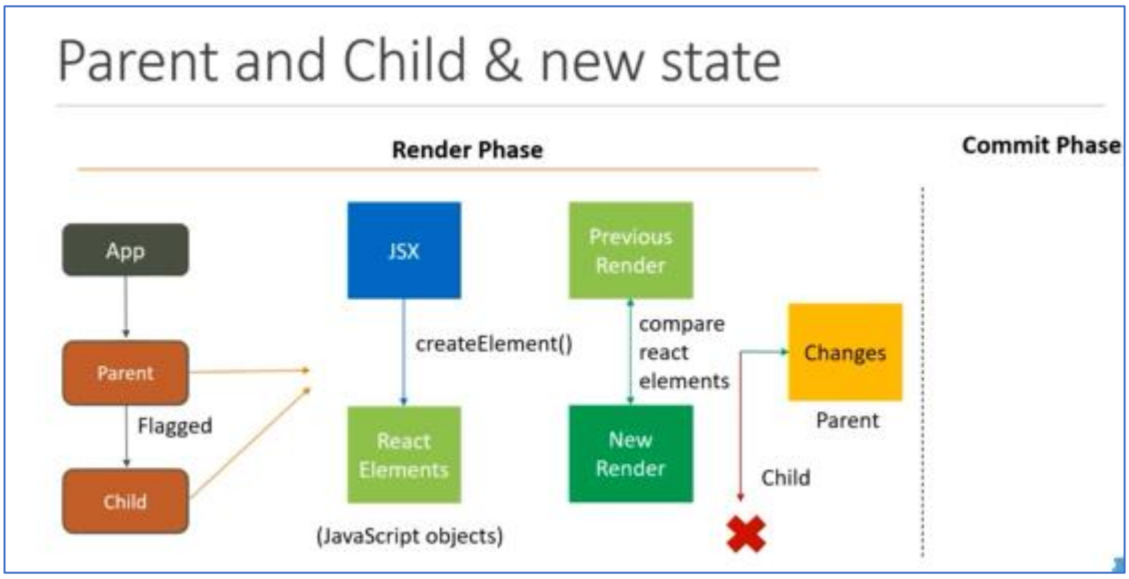
48. Parent Child Component Re-Render

if new state is same old state parent component re-render 1 more time to make sure but child component not re-render

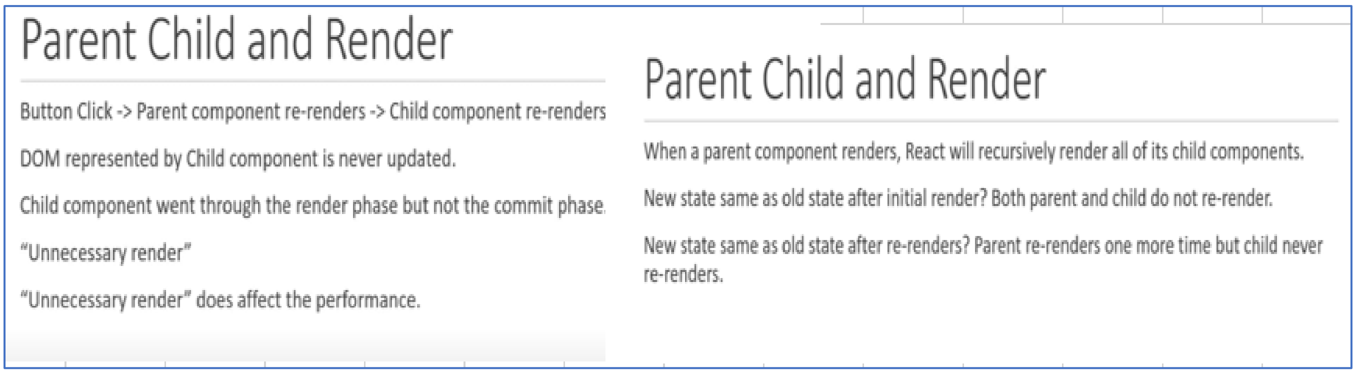


```
mponents > Parent Child > Parent.js > Parent
mport React, { useState } from 'react'
mport { Child } from './Child'

xport const Parent = () => {
  const [count, setCount] = useState(0)

  console.log('Parent Render')
  return (
    <div>
      <button onClick={() => setCount((c) => c + 1)}>Count - {c
      <button onClick={() => setCount(0)}>Count to 0</button>
      <button o Child(): JSX.Element (5)}>Count to 5</button>
      <Child />
    </div>
  )
```

```
mponents > Parent Child > Child.js > Child
import React from 'react'

export const Child = () => {
  console.log('Child Render')
  return (
    <div>
      Child component
    </div>
  )
}
```



Parent and Child & new state



Parent Child and Render

Button Click -> Parent component re-renders -> Child component re-renders

DOM represented by Child component is never updated.

Child component went through the render phase but not the commit phase.

"Unnecessary render"

"Unnecessary render" does affect the performance.

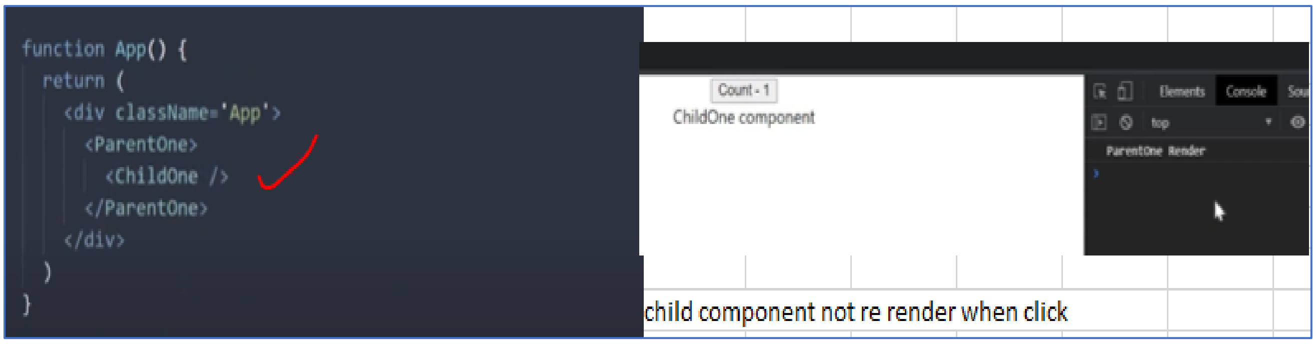
Parent Child and Render

When a parent component renders, React will recursively render all of its child components.

New state same as old state after initial render? Both parent and child do not re-render.

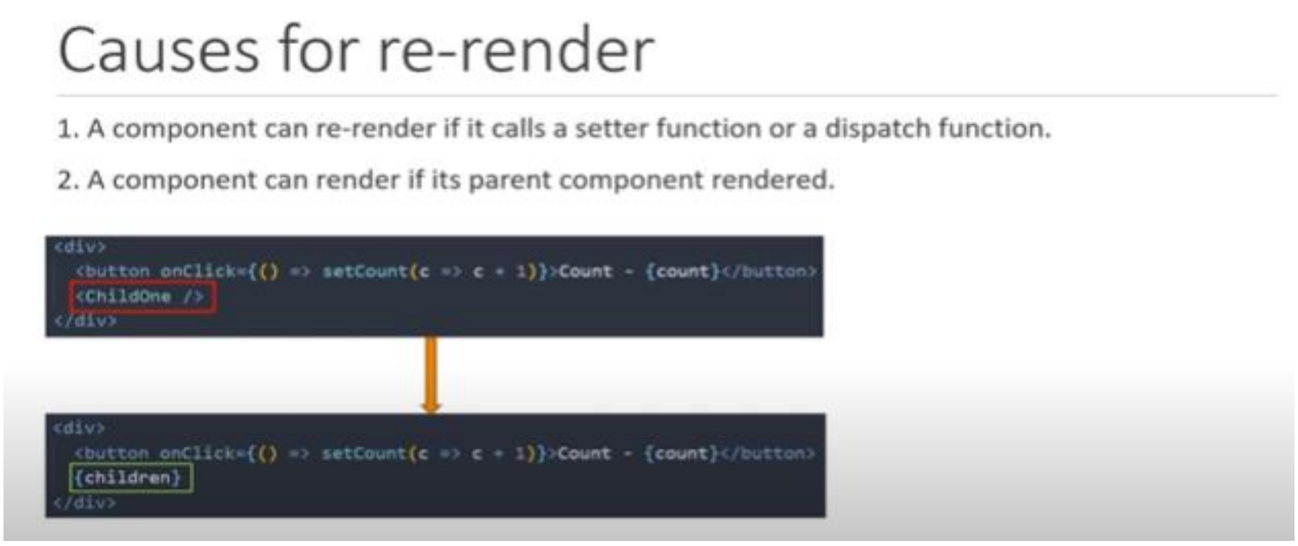
New state same as old state after re-renders? Parent re-renders one more time but child never re-renders.

49. Un-necessary re-render



```
function App() {
  return (
    <div className='App'>
      <ParentOne>
        <ChildOne />
      </ParentOne>
    </div>
  )
}
```

child component not re render when click

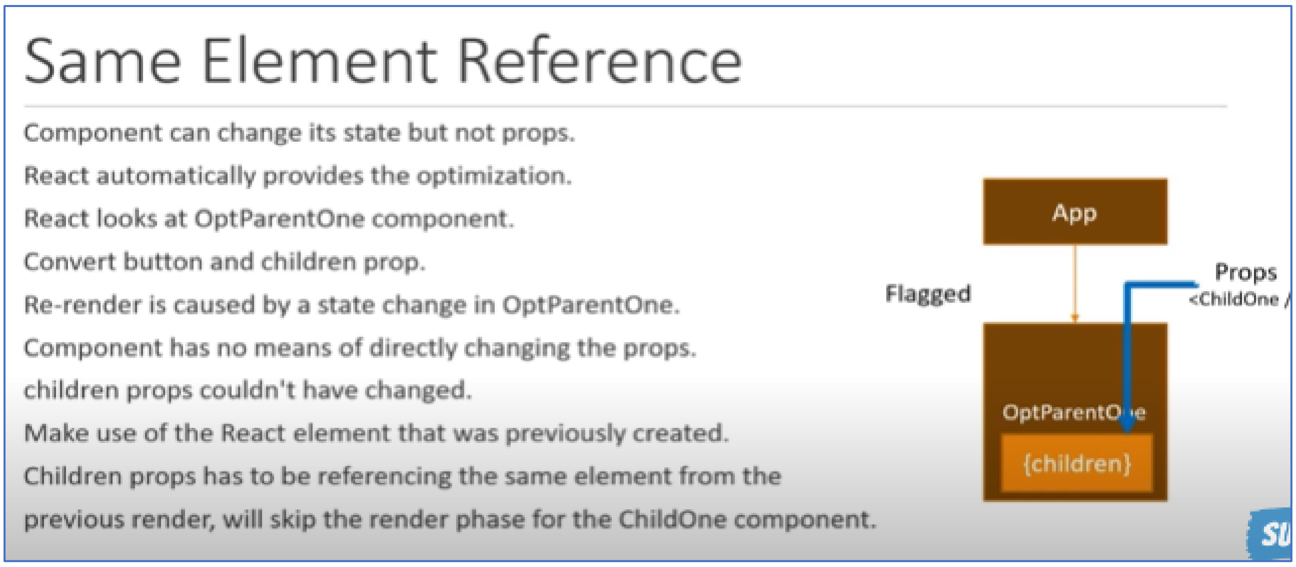


Causes for re-render

1. A component can re-render if it calls a setter function or a dispatch function.
2. A component can render if its parent component rendered.

```
<div>
  <button onClick={() => setCount(c => c + 1)}>Count - {count}</button>
  <ChildOne />
</div>
```

```
<div>
  <button onClick={() => setCount(c => c + 1)}>Count - {count}</button>
  {children}
</div>
```

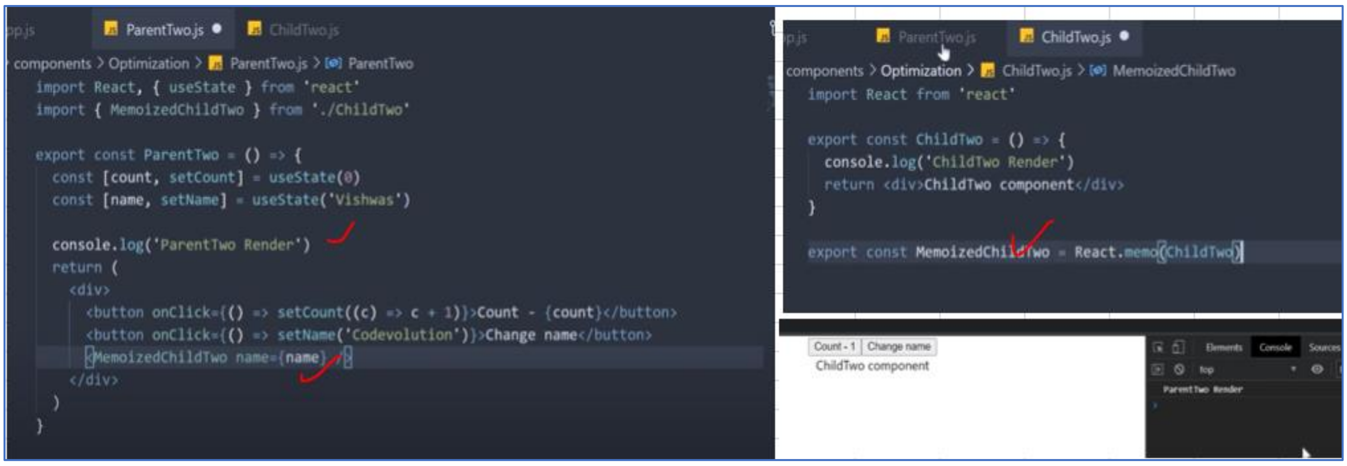


Same Element Reference

Component can change its state but not props.
React automatically provides the optimization.
React looks at OptParentOne component.
Convert button and children prop.
Re-render is caused by a state change in OptParentOne.
Component has no means of directly changing the props.
children props couldn't have changed.
Make use of the React element that was previously created.
Children props has to be referencing the same element from the previous render, will skip the render phase for the ChildOne component.

50. React Memo Re-Render

react.memo is shallow compare


ParentTwo.js
ChildTwo.js
components > Optimization > ParentTwo.js > ParentTwo

```
import React, { useState } from 'react'
import { MemoizedChildTwo } from './ChildTwo'

export const ParentTwo = () => {
  const [count, setCount] = useState(0)
  const [name, setName] = useState('Vishwas')

  console.log('ParentTwo Render')
  return (
    <div>
      <button onClick={() => setCount((c) => c + 1)}>Count - {count}</button>
      <button onClick={() => setName('Codevolution')}>Change name</button>
      <MemoizedChildTwo name={name} />
    </div>
  )
}
```

ParentTwo.js
ChildTwo.js
components > Optimization > ChildTwo.js > MemoizedChildTwo

```
import React from 'react'

export const ChildTwo = () => {
  console.log('ChildTwo Render')
  return <div>ChildTwo component</div>
}

export const MemoizedChildTwo = React.memo(ChildTwo)
```

```
ParentTwo Render
```

React.memo

In React, when a parent component renders, a child component might un-necessarily render.

To optimize this behaviour, you can use React.memo and pass in the child component.

React.memo will perform a shallow comparison of the previous and new props and re-render the child component only if the props have changed.

Questions on Optimization

When do I use the same element reference technique and when do I use React.memo?

Same Element Reference	React.memo
When your parent component re-renders because of state change in the parent component. This technique does not work if the parent component re-renders because of changes in its props state change? Yes props change? No	When your child component is being asked to re-render due to changes in the parent's state which do not affect the child component props in anyway.

Questions on Optimization

If React.memo provides the optimization by comparing the props, why not wrap every single component with React.memo?

Why doesnt React just internally memoize every component and not expose React.memo to the developers?

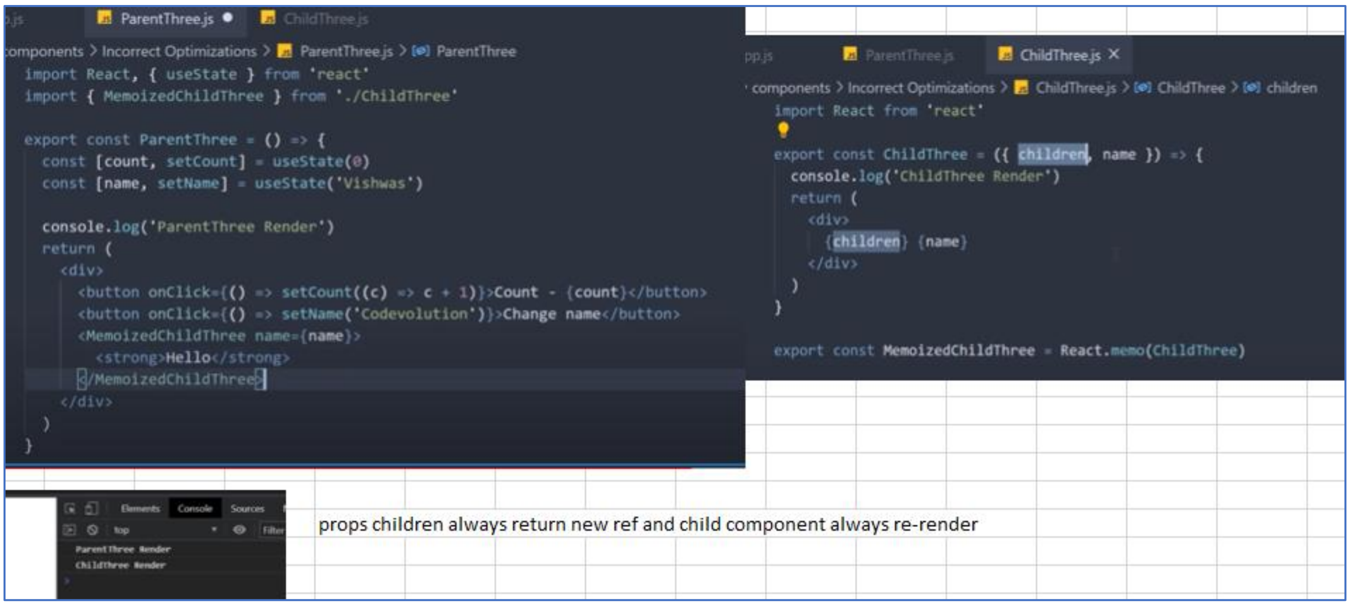
"Shallow comparisons aren't free. They're O(prop count). And they only buy something if it bails out.

All comparisons where we end up re-rendering are wasted. Why would you expect always comparing to be faster? Considering many components always get different props."

Render Optimization

When you optimize the rendering of one component, React will also skip rendering that component's entire subtree because it's effectively stopping the default ***"render children recursively"*** behavior of React.

51. Incorrect Memo with Child Component



```
import React, { useState } from 'react'
import { MemoizedChildThree } from './ChildThree'

export const ParentThree = () => {
  const [count, setCount] = useState(0)
  const [name, setName] = useState('Vishwas')

  console.log('ParentThree Render')
  return (
    <div>
      <button onClick={() => setCount((c) => c + 1)}>Count - {count}</button>
      <button onClick={() => setName('Codevolution')}>Change name</button>
      <MemoizedChildThree name={name}>
        <strong>Hello</strong>
      </MemoizedChildThree>
    </div>
  )
}
```

```
import React from 'react'

export const ChildThree = ({ children, name }) => {
  console.log('ChildThree Render')
  return (
    <div>
      {children} {name}
    </div>
  )
}

export const MemoizedChildThree = React.memo(ChildThree)
```

```
ParentThree Render
ChildThree Render
```

props children always return new ref and child component always re-render

52. Incorrect Memo with Impure Component