React Components

Relevel
by Unacademy

Topics covered

Introduction to Components

Details of Components

Types of Components

Introduction to Props Includes Reusability of Components

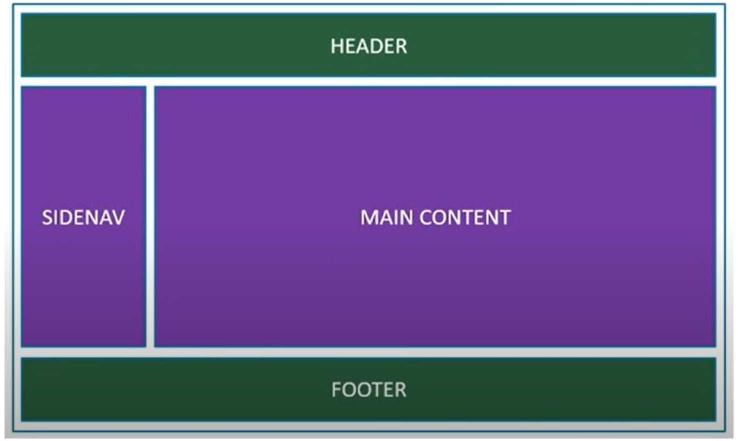
How to Merge Two Components to One Page

Conditional Rendering Of Components and its types

Types Of Conditional Rendering In Detail

Introduction to components

In React, a component is referred to as an isolated piece of code which can be reused in one or the other module. The React application contains a root component in which other subcomponents are included; for example - in a single page, the application is subdivided into 3 parts - Header, Main Content, and Footer. So, there is a single App Component having 3 subcomponents - Header Component, Main Content Component, and Footer Component.



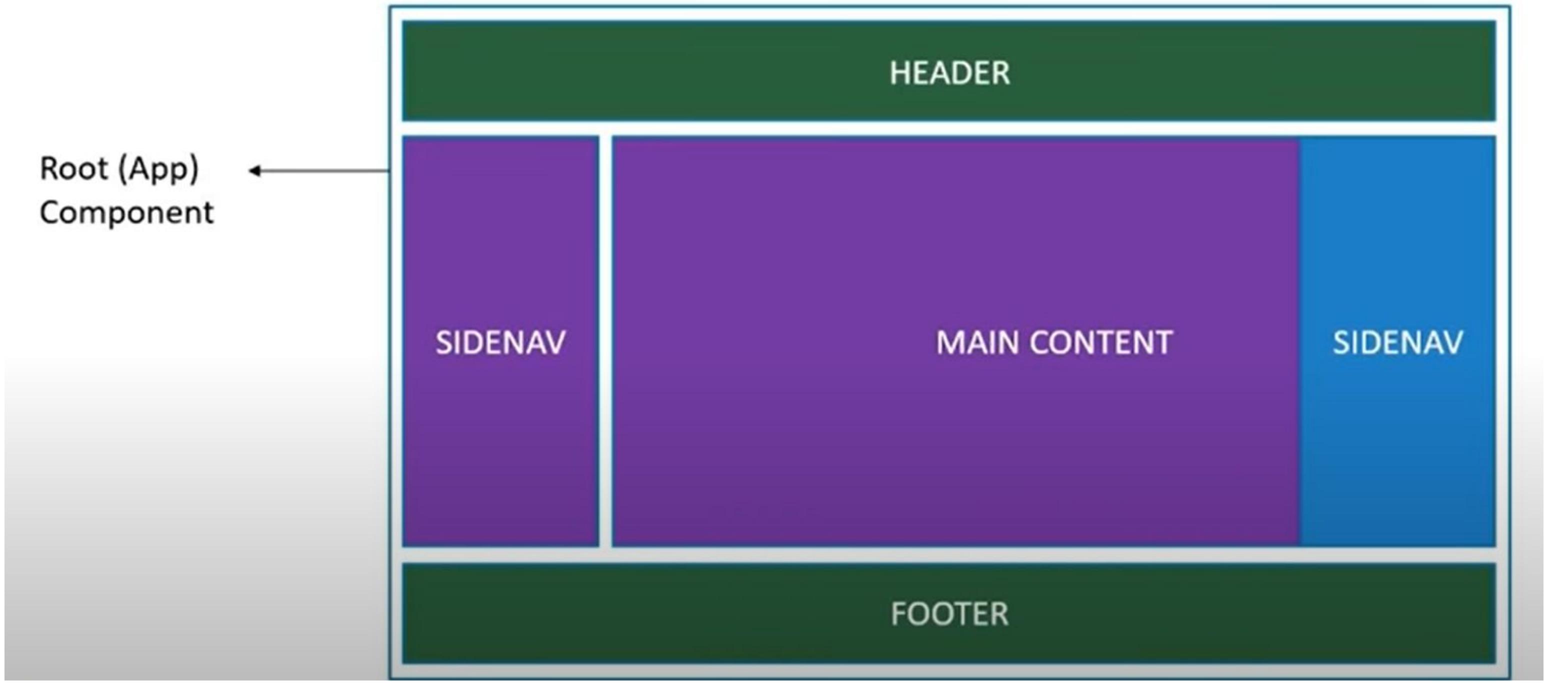
Details of Components

- In react a component represents a part of the user interface.
- Components are the building blocks of any react application.
- It takes an optional input and returns a React element which is rendered on the screen.
- Going back to the example from the earlier example the application has five components one for header, one for sidenav, one for the main content, one for the footer and finally one component to contain every other component, the containing component is the root component and is usually named as app component in the application, each of the four nested components describe only a part of the user interface.
- However all the components come together to make up the entire application
- Components are also reusable the same components can be used with different properties to display different information.

For Example:

- The side nav component can be the left side nav as well as the right side nav and also mentioned components also contain other components (image is shown in next slide)

For Example: app component contains the other components

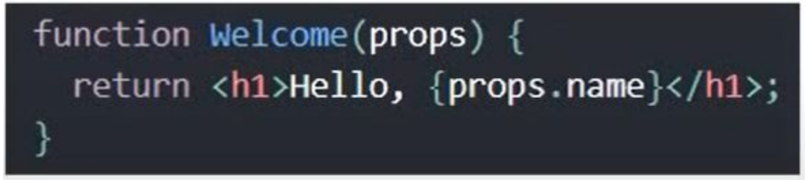

Root (App) Component
HEADER
SIDENAV
MAIN CONTENT
SIDENAV
FOOTER

Details of Components

- In react we have two component types can be either “stateful” or “stateless”
- Stateless (functional component)
- Stateful (class component)
- Functional components are literally JavaScript functions they return the HTML which describes the UI
- These components include immutable properties, i.e., the value for properties cannot be changed. We need to use Hooks (will be discussed in upcoming classes) to achieve functionality for making changes in properties using JS.

For example:

- A function call welcome which returns an h1 tag that says Hello John
- It doesn’t say that but for simplicity just assume we are returning regular HTML



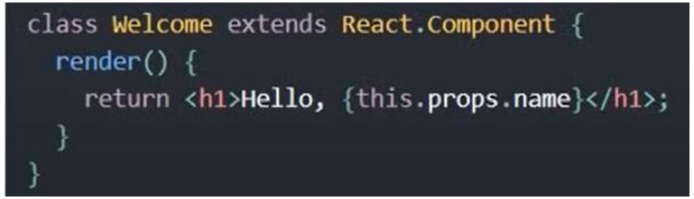
```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Details of Components

On the other hand **Class Components** are the regular ES6 classes that extend the component class from the react library they must contain a render method which in turn returns HTML.

For example:

- Class welcome extends React. Component and the class contains a render method which in returns an h1 tag that says Hello John



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

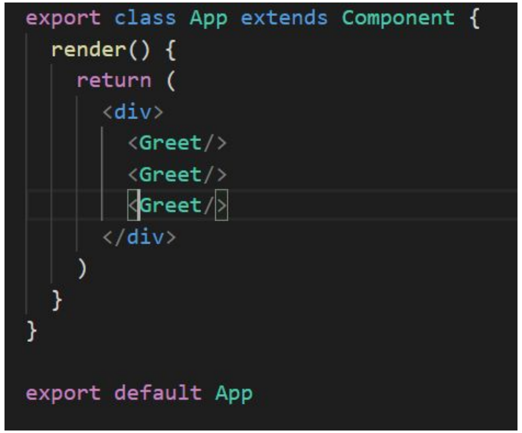
Components Reusability

Components are reusable, so you can create a component that returns any JSX you want to and include it in any part of your application

For example:

- Create a new component Greet.js and return Welcome John
- Let's say we need to reuse the Greet component, all you have to do is include the Greet tag as many times as you want

Duplicate it twice save the file and take a look at the browser you can see Hello John displayed 2 times.



```
export class App extends Component {
  render() {
    return (
      <div>
        <Greet/>
        <Greet/>
        <Greet/>
      </div>
    )
  }
}

export default App
```

Components Reusability

Output in Browser

Welcome John
Welcome John

This isn’t really helpful
What would be great is if we could pass in the name of the person we wanted to greet
that way reusing the same component we could greet three different people instead of greeting John two times that is where Props comes into picture

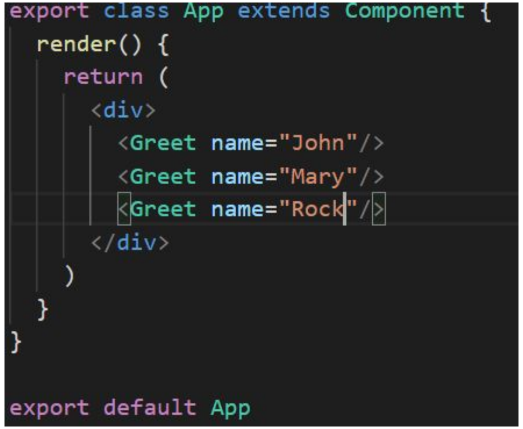
Introduction To Props

Props Short for properties is the optional input that your component can accept it also allows the component to be dynamic.

Now let’s understand how props work below:
Create a new component Greet.js
Our intention here is to pass a name from app component to the Greet component and render that name in the browser.

To specify props for a component specify them as attributes, to specify a name property simply add the name attribute,
To the attribute assign the value lets go with the name “John”
Similarly lets add the attribute on the other two components as well name=”Mary”
And name =”Rock”

Now we are sending some information to the greet component but to retrieve these values in the greet component is a quick two-step process.



```
export class App extends Component {
  render() {
    return (
      <div>
        <Greet name="John"/>
        <Greet name="Mary"/>
        <Greet name="Rock"/>
      </div>
    )
  }
}

export default App
```

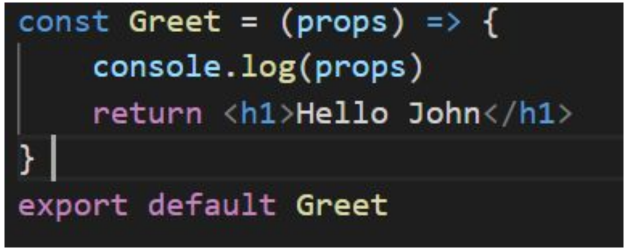
Props

The first step add a parameter to the functional component you are going to call it props. You can actually name this anything you want to but the convention is to name it props.

The second step is to use this parameter in the function body. Now in the function body first log the props parameter in the console. The function body is going to take up more than one line so add curly braces and also the return statement .

console.log(props) and then return the h1 element

Again the necessity of curly braces and the return keyword is an ES6 arrow function concept .
Go back to the browser and open dev tools.
You can see that there is an object logged in the console.
If you expand you can see that the object has a property called name and a value of “John” similarly “Mary” and “Rock” three objects corresponding to the three components



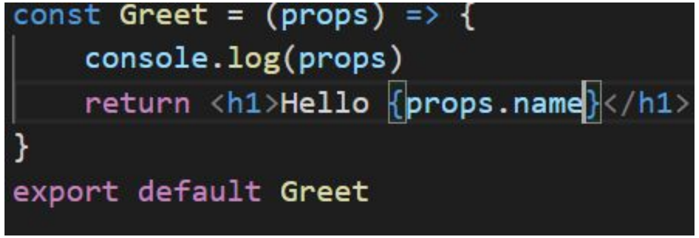
```
const Greet = (props) => {
    console.log(props)
    return <h1>Hello John</h1>
}
export default Greet
```

Props

So props is just an object that contains the attributes and their values which have been passed from the parent component so if we want to display the name that has been passed to the Greet component we need to use props.name.

In VS code instead of “name”
We need to use props.name, we need to ask react to evaluate the JSX expression and we do that by wrapping the expression with curly braces {props.name}
The curly braces is a feature of JSX which is really helpful and is used a lot in react applications

If we save the file and take a look at the browser you should be able to see
Hello John
Hello Mary
Hello Rock



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

Props

The reusability of components makes much more sense
Each component can have HTML template and we can pass in the data we want the component to use

Now there is one strict rule though, that is props are immutable or in simpler words their value cannot be changed
in our example If i try to assign a value to props.name

For example:

- Props.name=”XYZ” save and take a look at the browser you can see that the application breaks
- TYPE ERROR: cannot assign to read only property ‘name’ of object ‘#<Object’
- React components have to act like pure functions with respect to their props parameter you cannot change it under any circumstance
- nevertheless props is a great feature in react and any practical react application will definitely make use of props

How to merge components to create one page

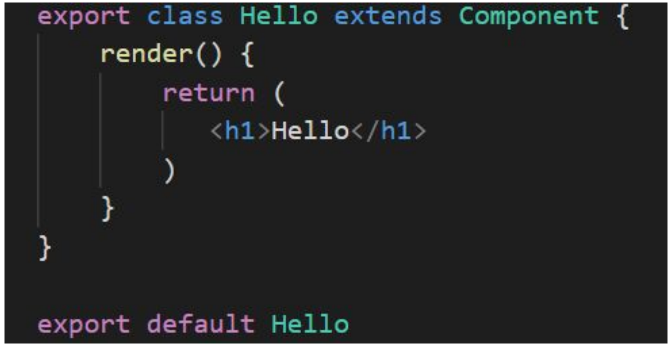
Even if we have multiple elements to render, there can only be a single root element.
Means if we want to render two or more elements, we have to wrap them in another element or component.

Let's understand with an example

Follow the below steps:

Create first component Hello

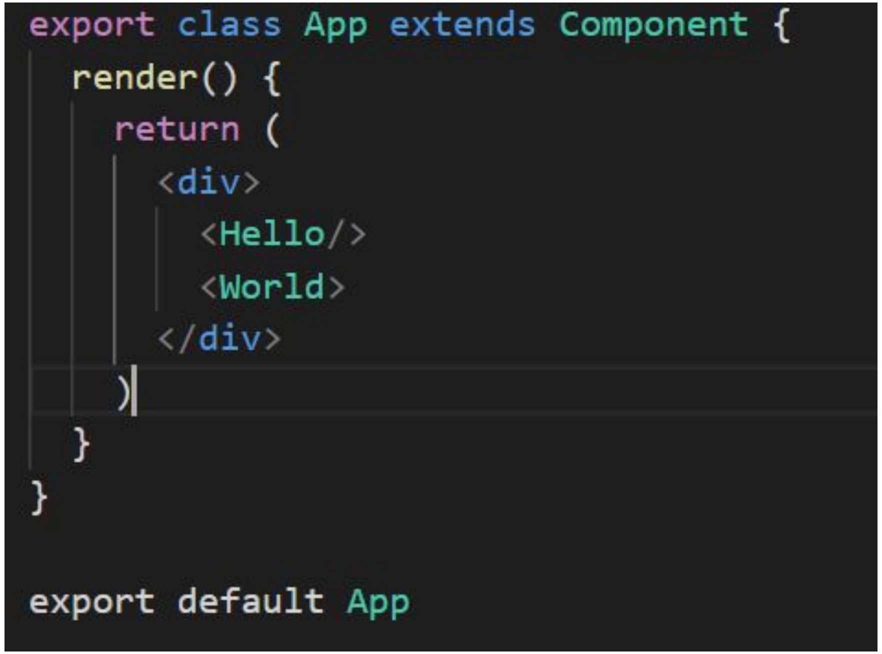
Similarly create second component World and return the text "World", try by yourself



```
export class Hello extends Component {
    render() {
        return (
            <h1>Hello</h1>
        )
    }
}

export default Hello
```

The Output will be “Hello World”



```
export class App extends Component {
  render() {
    return (
      <div>
        <Hello/>
        <World>
      </div>
    )
  }
}

export default App
```

Conditional rendering of components

- When building react applications you may often need to show or hide some HTML based on a certain condition
- Conditional rendering in react works the same way conditions work in JavaScript
- We have four different approaches and we will take a detailed look at each one of them
- if/else
- Element variables
- Ternary conditional operator
- Short circuit operator

if/else approach

Follow the below steps:

1. Create a new file UserGreeting.js
2. Within the file create a class component
3. In JSX simply return Welcome John
4. In the App component include the User Greeting component and make sure to import it at the top
5. Now save the files and take a look at the browser
6. You should be able to see the message Welcome John
7. Now, Go back to User Greeting component and
8. Begin by adding a constructor, within the constructor call super and then define the state
9. Create one state property called isLoggedIn and initialize it to false
10. In the JSX add another message that says welcome Guest
11. Now the message to be conditionally rendered based on the isLoggedIn state
12. If logged in message welcome John should be displayed and
13. If not logged in the message Welcome Guest should be displayed

if/else approach

Lets see how to achieve that with the first approach that is using the if/else condition

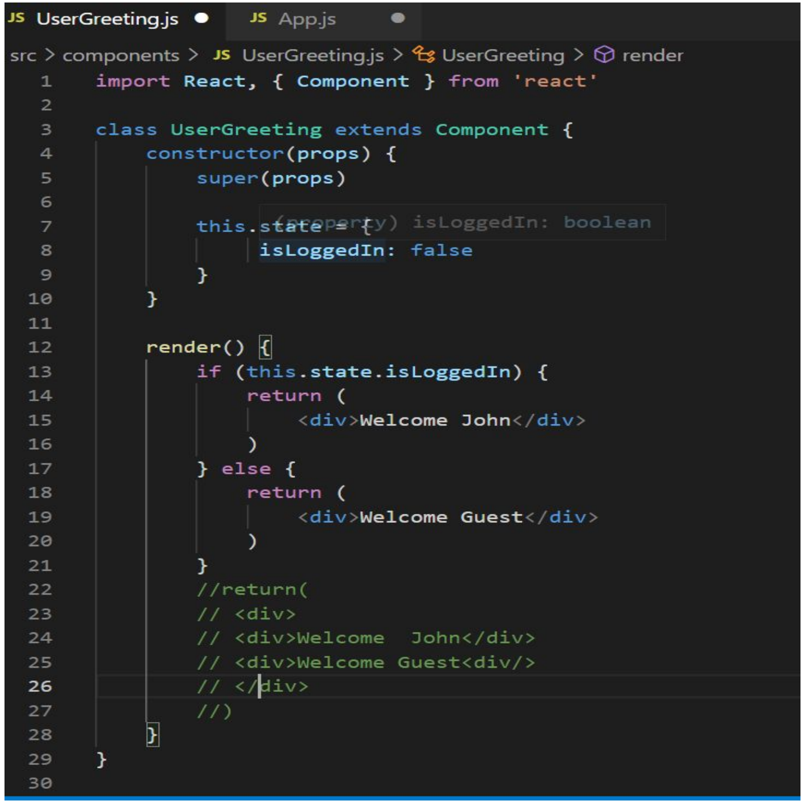
- In the render method let’s add the if-else condition
- If **this.state.isLoggedIn** we need to render Welcome John
- So we are going to have a return statement
- A div tag then Welcome John
- And the else condition that is if it is not logged in
- So else we are going to return parentheses div tag Welcome Guests
- And comment out the return statement we added before.
- Now you can see the if-else rendering

Now if you take a look at browser you can see the message. Welcome Guests is displayed that is because isLoggedIn is set to false change it to true and take a look at the browser you can see that the message Welcome John is displayed.

But, here you can see lot of repetition and the render method looks crowded

Code for if/else approach is added in next slide

if/else:



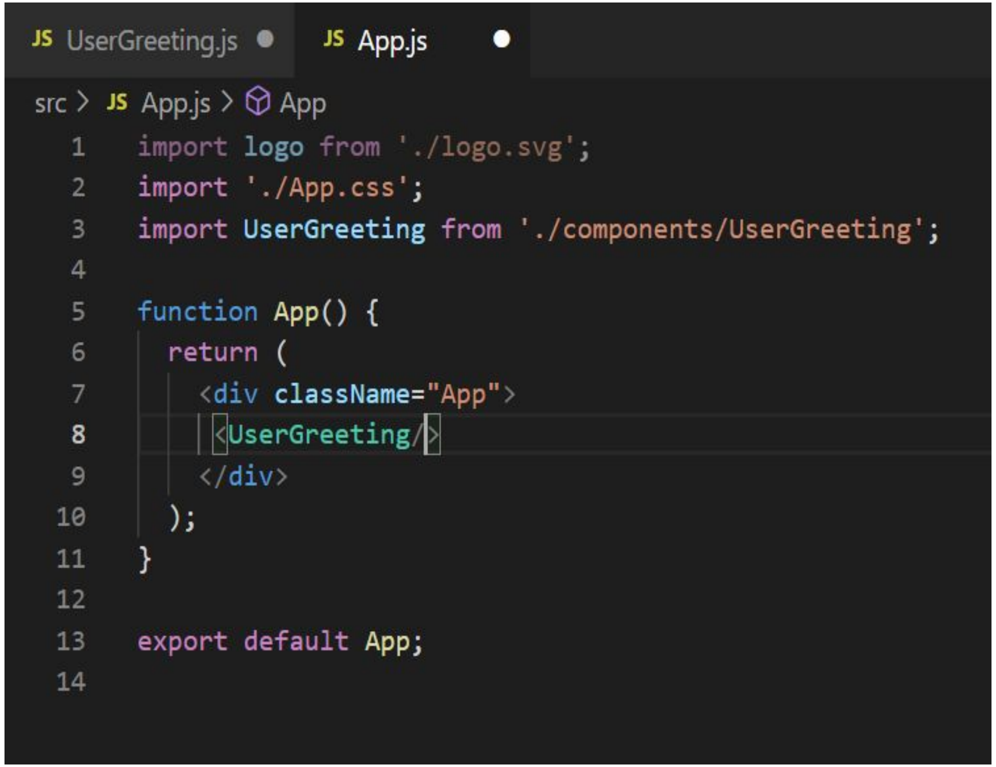
```
import React, { Component } from 'react'

class UserGreeting extends Component {
    constructor(props) {
        super(props)

        this.state = {
            isLoggedIn: false
        }
    }

    render() {
        if (this.state.isLoggedIn) {
            return (
                <div>Welcome John</div>
            )
        } else {
            return (
                <div>Welcome Guest</div>
            )
        }
        //return(
        // <div>
        // <div>Welcome  John</div>
        // <div>Welcome Guest<div/>
        // </div>
        //)
    }
}
```

if/else:



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

function App() {
  return (
    <div className="App">
      <UserGreeting/>
    </div>
  );
}

export default App;

```

#180DaysofPurpose

Element Variable

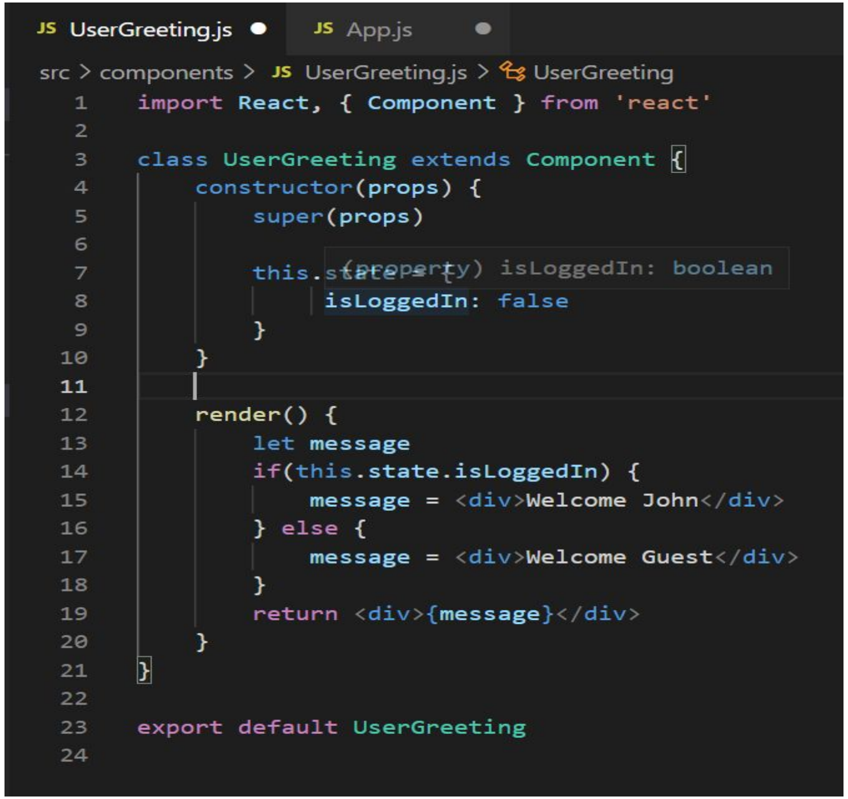
The better approach is using **element variables.** In this approach we use JavaScript variables to store elements this will also help you conditionally render the entire component or only a part of the component as well.

Steps to be followed:

- Declare a variable inside the render method (let message)
- Next store the appropriate element in this variable based on the condition
- So, If (this.state.isLoggedIn)
- message is going to be equal to a div tag that contains the text Welcome John
- else
- Message is equal to div tag that contains the text Welcome Guest
- And finally we return this message variable in the JSX
- So return div tag and within curly braces the variable {message}
- Save the file and now take a look at the browser
- You can see Welcome Guest is displayed
- So message is the variable which stores the element to be rendered and hence this is the element variable approach

Code for if/else approach is added in next slide

Element Variable:


UserGreeting.js
App.js
src > components > UserGreeting.js > UserGreeting
(property) isLoggedIn: boolean

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

class UserGreeting extends Component {
    constructor(props) {
        super(props)

        this.state = {
            isLoggedIn: false
        }
    }

    render() {
        let message
        if(this.state.isLoggedIn) {
            message = <div>Welcome John</div>
        } else {
            message = <div>Welcome Guest</div>
        }
        return <div>{message}</div>
    }
}

export default UserGreeting
```

Ternary Conditional operator

This method looks much better
The third approach is even more simpler and this approach uses the ternary conditional operator
The benefit of this approach is that it can be used inside the JSX

Now with in the render method add the return statement and with in parentheses use the conditional operator this.state.isLoggedIn ? a div tag says Welcome John a colon : and then again a div tag which says Welcome Guest

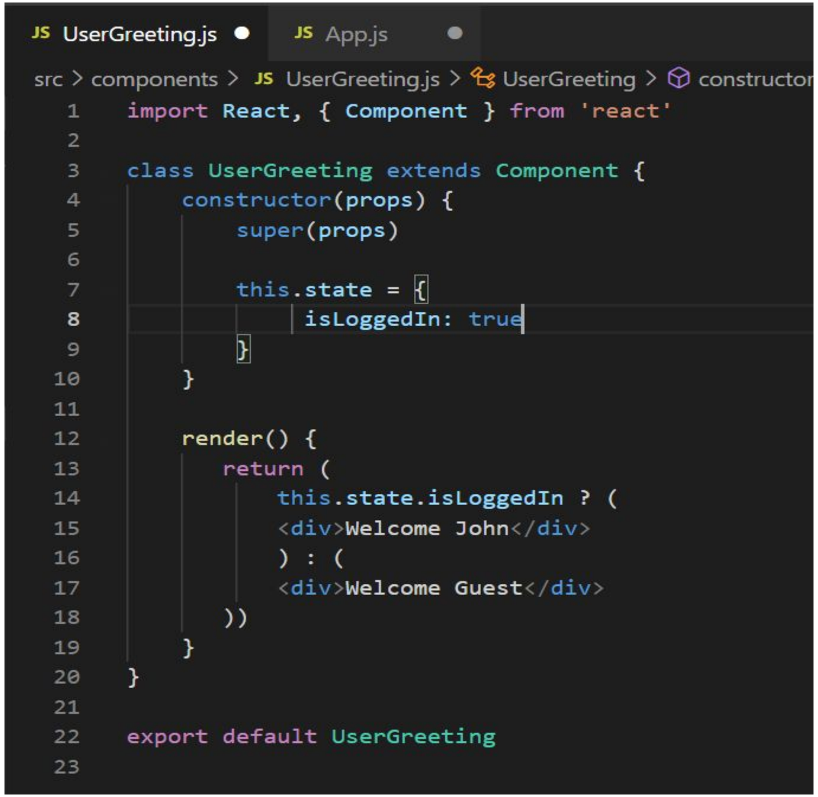
```
 this.state.isLoggedIn ? (
           <div>Welcome John</div>
           ) : (
           <div>Welcome Guest</div>
```

Now the first operator **this.state.isLoggedIn** is evaluated to either true or false if it is true the second operator is returned in our case the div tag Welcome John if the first operator turns out to be false then the third operator is returned in our case a div tag that says Welcome Guest

Go back to the browser you should be able to see the text

Code for element approach is added in next slide

Ternary Conditional Operator:


UserGreeting.js
App.js
src > components > UserGreeting.js > UserGreeting > constructor

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

class UserGreeting extends Component {
    constructor(props) {
        super(props)

        this.state = {
            isLoggedIn: true
        }
    }

    render() {
        return (
            this.state.isLoggedIn ? (
            <div>Welcome John</div>
            ) : (
            <div>Welcome Guest</div>
        ))
    }
}

export default UserGreeting
```

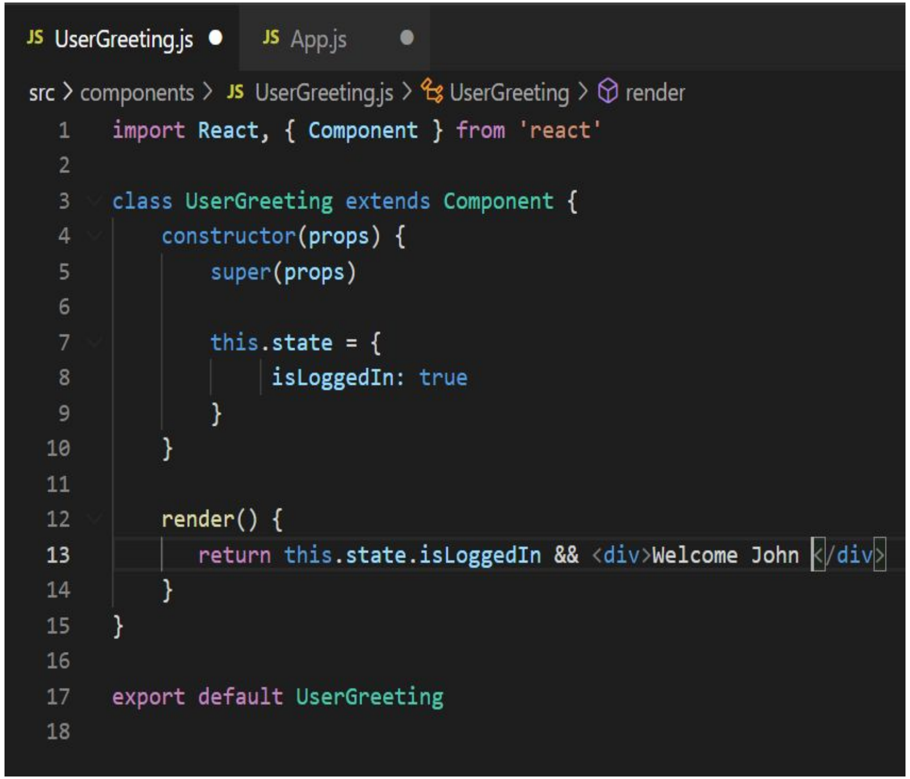
Short Circuit Operator

Final approach is short-circuit operator approach. This approach is just a specific case of this ternary operator approach, when you want to render either something or nothing you make use of the short-circuit operator

For example:
- Right now we return either Welcome John or Welcome Guest based on the isLoggedIn value
- Now let's say if the user is logged in and want to display Welcome John and if the user is not logged in want to render nothing onto the screen
- So based on isLoggedIn render either Welcome John or nothing
- To do that simply return this.state.isLoggedIn && div tag Welcome John
- So what happens here is the expression first evaluates the left hand side of the operator this.state.isLoggedIn if it is true it displays Welcome John
- It also evaluates the right hand side which in our case is the JSX that will be rendered in the browser
- However if at all the left hand side evaluates to false the right hand side will never be evaluated as it does not affect the final value of the whole expression and make sure to add the return keyword at the beginning of the statement
- So return this.state.isLoggedIn and Welcome John
- So if isLoggedIn is set to true in the browser we should have Welcome John being displayed and if i change it to false nothing is rendered in the browser
- So these are the four approaches to conditionally render UI in react

Code for element approach is added in next slide

Short Circuit Operator:



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

class UserGreeting extends Component {
    constructor(props) {
        super(props)

        this.state = {
            isLoggedIn: true
        }
    }

    render() {
      return this.state.isLoggedIn && <div>Welcome John </div>
    }
}

export default UserGreeting

```

Assignment

Create a new Component and try to pass more than one props from the App.js and access those in the newly created component and try to use conditional operator and render the HTML

Hint:

```
<Greet name='Mary' messageCount={10} isLoggedIn={false}/>
```

Thank You