Conditional Rendering 2

* Switch case operator continuation

Now there might be cases where you have multiple conditional renderings. The conditional rendering could apply based on different states.

Let's imagine a notification component that can render an error, warning or info component based on the input state. You can use a switch case operator to handle the conditional rendering of these multiple states.

function Notification({ text, state }) {

switch(state) {

case 'info':

return <Info text={text} />

case 'warning':

return <Warning text={text} />

case 'error':

return <Error text={text} />

default:

return null

}

}

Please note that you always have to use the default for the switch case operator. In React a component has always to return an element or null.

As a little information​, when a component has a conditional rendering based on a state, it makes sense to describe the interface of the component with React.PropTypes.

function Notification({ text, state }) {

switch(state) {

case 'info':

return <Info text={text} />

case 'warning':

return <Warning text={text} />

case 'error':

return <Error text={text} />

default:

return null

}

}

Notification.propTypes = {

text: React.PropTypes.string,

state: React.PropTypes.oneOf(['info', 'warning', 'error'])

}

Now you have one generic component to show different kinds of notifications. Based on the state prop the component could have different looks. An error should be red, a warning should be yellow and an info should be blue.

An alternative way would be to inline the switch case. Therefore you would need a self invoking JavaScript function.

function Notification({ text, state }) {

return (

<div>

{(() => {

switch(state) {

case 'info':

return <Info text={text} />

case 'warning':

return <Warning text={text} />

case 'error':

return <Error text={text} />

default:

return null

}

})()}

</div>

)}

* Conditional Rendering with enums

 If there are more than 3 components available to render conditionally, if-else becomes complex. So, developers should use enums to keep code clean.

In this file, we will add some basic React code to render on the webpage. The user needs to add the following code to the ‘first.js’ file.

**Filename: first.js**

|  |
| --- |
| import React, { Component } from 'react';  // Some basic code to render first component  class First extends Component {    render() {      return (        <div>          <h2>This is a first component</h2>        </div>      );    }  }  export default First; |

In this file, we will add some basic React code that is different from the first component. So, we can know which component is rendering on the screen. Edit the ‘second.js’ file and add the below code inside it.

**Filename: second.js**

|  |
| --- |
| import React, { Component } from 'react';  // some basic code to render second component  class Second extends Component {    render() {      return (        <div>          <h2>This is a second component.</h2>        </div>      );    }  }  export default Second; |

**Rendering component using enum**

**Step 1:** In javascript, we can create an object with key-value pairs and use it as an enum. Below, you can see the demo of a javascript object with key-value pair.

**Syntax:**

const Enumobj = {

key: value,

};

Example:

const Enumobj = {

first: <First />,

second: <Second />

};

**Step 2:** Now, we will make a javascript function that takes a state as a parameter and return a React component based on the state.

**Syntax:**

function Enum({state}){

return {object[state]};

}

**Example:**

function Enum({ state }) {

return <div>{Enumobj[state]}</div>;

}

**Step 3:**Let’s embed the ‘Enum‘ function in our ‘App‘ component. While calling the ‘Enum‘ function, we will add state value as props.

**Syntax:**

return (

<div>

<Enum state="Value"></Enum>

</div>

);

**Example:**

return (

<div>

<Enum state="first"></Enum>

<Enum state="second"></Enum>

</div>

);

**Filename: App.js**

In the App.js file, we will create an enum object first. After that, we will add an ‘enum’ function to render components according to state value. At last, we will edit the ‘App‘ component and call the ‘enum‘ function inside the component to render it conditionally. The user needs to add the below code to the ‘App.js‘ file.

**Filename: App.js**

|  |
| --- |
| import React, { Component } from 'react';  import Second from './components/second'  import First from './components/first'  // Creating enum object  const Enumobj = {    first: <First />,    second: <Second />  };  // Creating enum function to return  // Particular component according to state value  function Enum({ state }) {    return <div>{Enumobj[state]}</div>;  }  // Call enum function inside the App component  class App extends Component {    render() {      return (        <div>          <Enum state="first"></Enum>          <Enum state="second"></Enum>        </div>      );    }  }  export default App; |

**Steps to run:**The user needs to run beneath the command to the terminal in the current directory to see the output.

npm start

**Output:**

This is a first component.

This is a second component.