Conditional Rendering

In React, you can create distinct components that encapsulate behavior you need. Then, you can render only some of them, depending on the state of your application.

Conditional rendering in React works the same way conditions work in JavaScript. Use JavaScript operators like `if` or the conditional operator to create elements representing the current state, and let React update the UI to match them.

Consider these two components:

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

We'll create a `Greeting` component that displays either of these components depending on whether a user is logged in:

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}

ReactDOM.render(
  // Try changing to isLoggedIn={true}:
  <Greeting isLoggedIn={false} />,
  document.getElementById('root')
);
```

Try it on CodePen

This example renders a different greeting depending on the value of `isLoggedIn` prop.

Element Variables

You can use variables to store elements. This can help you conditionally render a part of the component while the rest of the output doesn't change.

Consider these two new components representing Logout and Login buttons:

```
function LoginButton(props) {
  return (
    <button onClick={props.onClick}>
      Login
    </button>
  );
}

function LogoutButton(props) {
  return (
    <button onClick={props.onClick}>
      Logout
    </button>
  );
}
```

In the example below, we will create a stateful component called `LoginControl`.

It will render either `<LoginButton />` or `<LogoutButton />` depending on its current state. It will also render a `<Greeting />` from the previous example:

```
class LoginControl extends React.Component {
  constructor(props) {
    super(props);
    this.handleLoginClick = this.handleLoginClick.bind(this);
    this.handleLogoutClick = this.handleLogoutClick.bind(this);
    this.state = {isLoggedIn: false};
  }
```

```
  handleLoginClick() {
    this.setState({isLoggedIn: true});
  }

  handleLogoutClick() {
    this.setState({isLoggedIn: false});
  }

  render() {
    const isLoggedIn = this.state.isLoggedIn;
    let button;

    if (isLoggedIn) {
      button = <LogoutButton onClick={this.handleLogoutClick} />;
    } else {
      button = <LoginButton onClick={this.handleLoginClick} />;
    }

    return (
      <div>
        <Greeting isLoggedIn={isLoggedIn} />
        {button}
      </div>
    );
  }
}

ReactDOM.render(
  <LoginControl />,
  document.getElementById('root')
);
```

Try it on CodePen

While declaring a variable and using an `if` statement is a fine way to conditionally render a component, sometimes you might want to use a shorter syntax. There are a few ways to inline conditions in JSX, explained below.

Inline If with Logical && Operator

You may embed any expressions in JSX by wrapping them in curly braces. This includes the JavaScript logical `&&` operator. It can be handy for conditionally including an element:

```
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Hello!</h1>
      {unreadMessages.length > 0 &&
        <h2>
          You have {unreadMessages.length} unread messages.
        </h2>
      }
    </div>
  );
}

const messages = ['React', 'Re: React', 'Re:Re: React'];
ReactDOM.render(
  <Mailbox unreadMessages={messages} />,
  document.getElementById('root')
);
```

Try it on CodePen

It works because in JavaScript, `true && expression` always evaluates to `expression`, and `false && expression` always evaluates to `false`.

Therefore, if the condition is `true`, the element right after `&&` will appear in the output. If it is `false`, React will ignore and skip it.

Inline If-Else with Conditional Operator

Another method for conditionally rendering elements inline is to use the JavaScript conditional operator `condition ? true : false`.

In the example below, we use it to conditionally render a small block of text.

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
```

```
    </div>
  );
}
```

It can also be used for larger expressions although it is less obvious what's going on:

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      {isLoggedIn ? (
        <LogoutButton onClick={this.handleLogoutClick} />
      ) : (
        <LoginButton onClick={this.handleLoginClick} />
      )}
    </div>
  );
}
```

Just like in JavaScript, it is up to you to choose an appropriate style based on what you and your team consider more readable. Also remember that whenever conditions become too complex, it might be a good time to extract a component.

Preventing Component from Rendering

In rare cases you might want a component to hide itself even though it was rendered by another component. To do this return `null` instead of its render output.

In the example below, the `<WarningBanner />` is rendered depending on the value of the prop called `warn`. If the value of the prop is `false`, then the component does not render:

```
function WarningBanner(props) {
  if (!props.warn) {
    return null;
  }

  return (
    <div className="warning">
```

```
      </div>
    );
  }

  class Page extends React.Component {
    constructor(props) {
      super(props);
      this.state = {showWarning: true};
      this.handleToggleClick = this.handleToggleClick.bind(this);
    }

    handleToggleClick() {
      this.setState(state => ({
        showWarning: !state.showWarning
      }));
    }

    render() {
      return (
        <div>
          <WarningBanner warn={this.state.showWarning} />
          <button onClick={this.handleToggleClick}>
            {this.state.showWarning ? 'Hide' : 'Show'}
          </button>
        </div>
      );
    }
  }

  ReactDOM.render(
    <Page />,
    document.getElementById('root')
  );
```

Try it on CodePen

Returning `null` from a component's `render` method does not affect the firing of the component's lifecycle methods. For instance `componentDidUpdate` will still be called.

Edit this page

Previous article
Handling Events

Next article
Lists and Keys