Conditional rendering

Conditional rendering in React allows you to control what content is displayed in your components based on certain conditions or criteria. You can use JavaScript expressions and conditional statements to determine what JSX (markup) to render. Here are some common techniques for conditional rendering in React:

1. \*\*Using the `if` statement:\*\*

You can use a standard `if` statement to conditionally render content within your component's `render` method:

Eg

import React, { Component } from 'react';

class ConditionalRenderingExample extends Component {

render() {

if (someCondition) {

return <p>Content to render when the condition is true</p>;

} else {

return <p>Content to render when the condition is false</p>;

}

}

}

2. Using the ternary operator:

The ternary operator (`? :`) is a concise way to conditionally render content. It's often used when you want to render one of two components based on a condition:

Eg

import React from 'react';

function ConditionalRenderingExample(props) {

const condition = props.someCondition;

return (

<div>

{condition ? (

<p>Content to render when the condition is true</p>

) : (

<p>Content to render when the condition is false</p>

)}

</div>

);

}

3. Using logical && operator:

You can use the logical `&&` operator to conditionally render a component. The component will only render when the condition on the left is `true`:

Eg

import React from 'react';

function ConditionalRenderingExample(props) {

const condition = props.someCondition;

return (

<div>

{condition && <p>Content to render when the condition is true</p>}

</div>

);

}

```

4. Using `map` for conditional rendering in lists:

When rendering a list of items conditionally, you can use the `map` function to iterate over the items and render them based on a condition:

Eg

import React from 'react';

function ConditionalRenderingList(props) {

const items = props.items;

return (

<ul>

{items.map((item) => (

item.condition && <li key={item.id}>{item.text}</li>

))}

</ul>

);

}

```

5. Using a function to encapsulate rendering logic:

You can create a separate function to encapsulate the rendering logic based on conditions. This can help keep your `render` method clean and maintainable:

Eg

import React from 'react';

function RenderContent(condition) {

if (condition) {

return <p>Content to render when the condition is true</p>;

} else {

return <p>Content to render when the condition is false</p>;

}

}

function ConditionalRenderingExample(props) {

const someCondition = props.someCondition;

return (

<div>

{RenderContent(someCondition)}

</div>

);

}