Understanding Lifecycle Methods

* What are lifecycle Methods and what are the uses of it

Lifecycle methods are special methods built into React, used to operate on components throughout their duration in the DOM. For example, when the component mounts, renders, updates, or unmounts. You already know the most important lifecycle method, the render method. Soon you will know a few more, with a good understanding of when to use them and what tasks should be done within them.

One important thing to notice is that lifecycle methods can only be used in class components. In functional components we have other options. But more on that in the next lesson. Let’s begin.

* Constructor

The constructor() method is called before anything else, when the component is initiated, and it is the natural place to set up the initial state and other initial values.

The constructor() method is called with the props, as arguments, and you should always start by calling the super(props) before anything else, this will initiate the parent's constructor method and allows the component to inherit methods from its parent (React.Component).

**Example:**

The constructor method is called, by React, every time you make a component:

class Header extends React.Component {

constructor(props) {

super(props);

this.state = {favoritecolor: "red"};

}

render() {

return (

<h1>My Favorite Color is {this.state.favoritecolor}</h1>

);

}

}

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

* Render

The render() method is required, and is the method that actually outputs the HTML to the DOM.

**Example:**

A simple component with a simple render() method:

class Header extends React.Component {

render() {

return (

<h1>This is the content of the Header component</h1>

);

}

}

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

* Should component update

In the shouldComponentUpdate() method you can return a Boolean value that specifies whether React should continue with the rendering or not.

The default value is true.

The example below shows what happens when the shouldComponentUpdate() method returns false:

**Example:**

Stop the component from rendering at any update:

class Header extends React.Component {

constructor(props) {

super(props);

this.state = {favoritecolor: "red"};

}

shouldComponentUpdate() {

return false;

}

changeColor = () => {

this.setState({favoritecolor: "blue"});

}

render() {

return (

<div>

<h1>My Favorite Color is {this.state.favoritecolor}</h1>

<button type="button" onClick={this.changeColor}>Change color</button>

</div>

);

}

}

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

* Render

The render() method is of course called when a component gets updated, it has to re-render the HTML to the DOM, with the new changes.

The example below has a button that changes the favorite color to blue:

**Example:**

Click the button to make a change in the component's state:

class Header extends React.Component {

constructor(props) {

super(props);

this.state = {favoritecolor: "red"};

}

changeColor = () => {

this.setState({favoritecolor: "blue"});

}

render() {

return (

<div>

<h1>My Favorite Color is {this.state.favoritecolor}</h1>

<button type="button" onClick={this.changeColor}>Change color</button>

</div>

);

}

}

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

* ComponentDidUpdate

The componentDidUpdate method is called after the component is updated in the DOM.

The example below might seem complicated, but all it does is this:

When the component is mounting it is rendered with the favorite color "red".

When the component has been mounted, a timer changes the state, and the color becomes "yellow".

This action triggers the update phase, and since this component has a componentDidUpdate method, this method is executed and writes a message in the empty DIV element:

**Example:**

The componentDidUpdate method is called after the update has been rendered in the DOM:

class Header extends React.Component {

constructor(props) {

super(props);

this.state = {favoritecolor: "red"};

}

componentDidMount() {

setTimeout(() => {

this.setState({favoritecolor: "yellow"})

}, 1000)

}

componentDidUpdate() {

document.getElementById("mydiv").innerHTML =

"The updated favorite is " + this.state.favoritecolor;

}

render() {

return (

<div>

<h1>My Favorite Color is {this.state.favoritecolor}</h1>

<div id="mydiv"></div>

</div>

);

}

}

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