React Component State:

React components has a built-in state object.

The state object is where you store property values that belong to the component.

When the state object changes, the component re-renders.

Creating the state Object

The state object is initialized in the constructor:

Example:

Specify the state object in the constructor method:

class Car extends React.Component {

constructor(props) {

super(props);

this.state = {brand: "Ford"};

}

render() {

return (

<div>

<h1>My Car</h1>

</div>

);

}

}

The state object can contain as many properties as you like:

Example:

Specify all the properties your component need:

class Car extends React.Component {

constructor(props) {

super(props);

this.state = {

brand: "Ford",

model: "Mustang",

color: "red",

year: 1964

};

}

render() {

return (

<div>

<h1>My Car</h1>

</div>

);

}

}

Using the state Object

Refer to the state object anywhere in the component by using the this.state.propertyname syntax:

Example:

Refer to the state object in the render() method:

class Car extends React.Component {

constructor(props) {

super(props);

this.state = {

brand: "Ford",

model: "Mustang",

color: "red",

year: 1964

};

}

render() {

return (

<div>

<h1>My {this.state.brand}</h1>

<p>

It is a {this.state.color}

{this.state.model}

from {this.state.year}.

</p>

</div>

);

}

}

Changing the state Object

To change a value in the state object, use the this.setState() method.

When a value in the state object changes, the component will re-render, meaning that the output will change according to the new value(s).

Example:

Add a button with an onClick event that will change the color property:

import React from 'react';

import ReactDOM from 'react-dom/client';

class Car extends React.Component {

constructor(props) {

super(props);

this.state = {

brand: "Ford",

model: "Mustang",

color: "red",

year: 1964

};

}

changeColor = () => {

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

}

render() {

return (

<div>

<h1>My {this.state.brand}</h1>

<p>

It is a {this.state.color}

{this.state.model}

from {this.state.year}.

</p>

<button

type="button"

onClick={this.changeColor}

>Change color</button>

</div>

);

}

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Car />);

Always use the setState() method to change the state object, it will ensure that the component knows its been updated and calls the render() method (and all the other lifecycle methods).

In our example in Footer React Class component we are managing the name field state as follws:

import React,{Component} from 'react'

class Footer extends Component

{

    state = {

    name:'inital value'   /\* this is initializing the state \*/

    }

    changed =(evnt)=>{

       // console.log(evnt.target.value)

        this.setState({name:evnt.target.value})  /\*changing the stae by geting key typeved event\*/

        console.log(this.state.name)

    }

    render()

    {

        return (

            <div>

            <h2>{this.props.trademark}</h2>

            <input onChange={this.changed} type="text" value={this.state.name}/>

            </div>

        )

    }

}

export default Footer

In the above example first we initialize the state in constructor and then we defined **changed** arrow function by calling **setState** function