**Updating Components**

Here is some method for re-rendering the component

* Re-render component when state changes
* Re-render component when props change
* Re-render with key prop

***Re-render component when state changes***

Any time a React component state has changed, React has to run the [*render()*](https://linguinecode.com/post/4-techniques-conditional-render-react-props-state) method.

*Example:*

import React, { Component } from "react";

class App extends React.Component {

  state = {

    mssg: ""

  };

  handleClick = () => {

    this.setState({ mssg: "Hi there!" });

  };

  render() {

    console.log("render() method");

    return (

      <>

        <button onClick={this.handleClick}>Say something</button>

        <div>{this.state.mssg}</div>

      </>

    );

  }

}

export default App;

***Re-render component when props change***

*Example:*

*Child.js*

import React, { Component } from "react";

class Child extends React.Component {

  render() {

    console.log('Child component: render()');

    return {this.props.message};

  }

}

export default Child;

*App.js*

import React, { Component } from "react";

class App extends React.Component {

  state = {

    mssg: ""

  };

  handleClick = () => {

    this.setState({ mssg: "Hi there!" });

  };

  render() {

    return (

      <>

        <button onClick={this.handleClick}>Say something</button>

        <Child message={this.state.mssg} />

      </>

    );

  }

}

export default App;

In the example above, *<Child />* component does not have state, but it does have a custom prop that it accepts, message.

When the button gets clicked on it will update the *<Child />* component, and cause it to run the *render()* lifecycle again.

***Re-render with key prop***

By changing the value of the key prop, it will make React unmount the component and re-mount it again, and go through the *render()* lifecycle.

*Example:*

*ChildComp.js*

import React, { Component } from "react";

class ChildComp extends React.Component {

  componentDidMount() {

    console.log('componentDidMount() lifecycle')

  }

  render() {

    console.log('render() lifecycle')

    return <h1>{this.props.number} times</h1>;

  }

}

export default ChildComp;

*App.js*

import React, { Component } from "react";

class App extends React.Component {

  state = {

    num: 0,

  };

  handleClick() {

    this.setState(state => ({ num: state.num + 1 }));

  }

  render() {

    return (

      <>

        <button onClick={this.handleClick.bind(this)}>Increment</button>

        <ChildComp number={this.state.num} />

      </>

    )

  }

}

export default App;

**First ReactJS Components**

Components are like functions that return HTML elements. Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML. Components come in two types, Class components and Function components.

***Class Component***

A class component must include the extends React.Component statement. This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions. The component also requires a render() method, this method returns HTML.

*Example:*

import React, { Component } from "react";

class ReactClassComponent extends Component {

  render() {

    return (

      <div>

        <h3>React Class Component</h3>

      </div>

    );

  }

}

***Function Component***

Here is the same example as above, but created using a Function component instead.

A Function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand, and will be preferred in this tutorial.

*Example:*

import React from 'react'

function ReactFunctionalComponent() {

  return (

      <div>

        <h3>React Functional Component</h3>

      </div>

  )

}

***Mounting Components***

In React, every instance of a component goes through a lifecycle that consists of creation (mounting), updating, and deletion (unmounting).

Mounting is the initial phase in which the instance of the component is created and inserted into the DOM. When the component gets successfully inserted into the DOM, the component is said to be mounted. In the updating phase, the component gets updated while being present in the DOM. In the unmounting phase, the component is removed from the DOM.

React provides us various methods called the lifecycle methods which get triggered during different phases of the lifecycle of that component. We will use some of these lifecycle methods to see what happens in the mounting phase

*Example:*

*ReactMounting.js*

import React, { Component } from "react";

export class ReactMounting extends Component {

  constructor(props) {

    super(props);

    this.state = {

      num: 0,

    };

    this.buttonHandler = this.buttonHandler.bind(this);

  }

  buttonHandler() {

    this.setState((state) => ({ num: state.num + 1 }));

  }

  componentDidMount() {

    console.log("Mounted");

  }

  componentDidUpdate() {

    console.log("Updated");

  }

  render() {

    console.log("Rendered");

    return (

      <div>

        <h3>React Mounting</h3>

        <h4>Number: {this.state.num}</h4>

        <button onClick={this.buttonHandler}>Count</button>

      </div>

    );

  }

}

export default ReactMounting;

*App.js*

import "./App.css";

import ReactMounting from "./Components/ReactMounting";

function App() {

  return (

    <div className="App">

      <ReactMounting />

    </div>

  );

}

export default App;