**React Module 10 List And Hooks**

**1.Explain Life cycle in Class Component and functional component with Hooks**

Life Cycle Method using Class Componant

==>**Mounting** constructor(props): This is the first method that gets called when a component is created. It's used to initialize the component's state and bind methods.

Example:

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

this.handleClick = this.handleClick.bind(this);

}

handleClick() {

this.setState({ count: this.state.count + 1 });

}

render() {

return (

<div>

<p>Count: {this.state.count}</p>

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

</div>

);

}

**static getDerivedStateFromProps(props, state):** This method is called after the constructor() method and before the render() method. It's used to update the component's state based on changes to props.

Example:

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { count: props.initialCount };

}

static getDerivedStateFromProps(props, state) {

if (props.initialCount !== state.count) {

return { count: props.initialCount };

}

return null;

}

render() {

return (

<div>

<p>Count: {this.state.count}</p>

</div>

)

}

}

**render():** This method is called after the static getDerivedStateFromProps() method and is used to generate the component's JSX.

Example:

class MyComponent extends React.Component {

render() {

return (

<div>

<p>Hello, {this.props.name}!</p>

</div>

);

}

}

**componentDidMount():** This method is called after the component is mounted to the DOM and is used to perform any actions that require the component to be in the DOM, such as fetching data from an API.

Example:

class MyComponent extends React.Component {

componentDidMount() {

fetch('https://api.example.com/data')

.then(response => response.json())

.then(data => {

this.setState({ data });

});

}

render() {

return (

<div>

<p>{this.state.data}</p>

</div>

);

}

}

**Updating**

static getDerivedStateFromProps(props, state): This method is called before the render() method when a component's props have changed. It's used to update the component's state based on changes to props.

Example:

class MyComponent extends React.Component {

static getDerivedStateFromProps(props, state) {

if (props.name !== state.name) {

return { name: props.name };

}

return null;

}

render() {

return (

<div>

<p>Hello, {this.state.name}!</p>

</div>

);

}

}

**shouldComponentUpdate(nextProps, nextState):**

This method is called after static getDerivedStateFromProps() and before render(). It's used to determine whether the component should re-render.

Example:

class MyComponent extends React.Component {

shouldComponentUpdate(nextProps, nextState) {

if (nextProps.count === this.props.count

Functional Componant Life Cycles And All Hook

**useEffect(() => { ... }, []):** This method is called when the component is mounted to the DOM. The function passed as the first argument will be executed after the first render. The second argument, an array, contains the dependencies of the effect. If an empty array is passed, the effect will only be called once when the component mounts.

Example:

import React, { useState, useEffect } from 'react';

function MyComponent(props) {

const [count, setCount] = useState(0);

useEffect(() => {

console.log('Component mounted');

}, []);

return (

<div>

<p>Count: {count}</p>

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

}

**useEffect(() => { ... }):** This method is called when the component is mounted and whenever any state or prop changes. The function passed as the first argument will be executed after every render.

Example:

import React, { useState, useEffect } from 'react';

function MyComponent(props) {

const [count, setCount] = useState(0);

useEffect(() => {

console.log('Component updated');

});

return (

<div>

<p>Count: {count}</p>

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

}

**useEffect(() => { ... }, [stateDependency1, stateDependency2]):** This method is called when the component is mounted and whenever the specified dependencies change. The function passed as the first argument will be executed after every render if any of the specified dependencies change.

Example:

import React, { useState, useEffect } from 'react';

function MyComponent(props) {

const [count, setCount] = useState(0);

const [message, setMessage] = useState('');

useEffect(() => {

console.log('Count updated');

}, [count]);

useEffect(() => {

console.log('Message updated');

}, [message]);

return (

<div>

<p>Count: {count}</p>

<button onClick={() => setCount(count + 1)}>Increment</button>

<input type="text" value={message} onChange={(e) => setMessage(e.target.value)} />

</div>

);

}

**useEffect(() => { ... }, [propsDependency1, propsDependency2]):** This method is called when the component is mounted and whenever the specified dependencies change. The function passed as the first argument will be executed after every render if any of the specified dependencies change.

Example:

import React, { useState, useEffect } from 'react';

function MyComponent(props) {

const [data, setData] = useState([]);

useEffect(() => {

fetch(`https://api.example.com/data/${props.id}`)

.then((response) => response.json())

.then((data) => {

setData(data);

});

}, [props.id]);

return (

<div>

<ul>

{data.map((item) => (

<li key={item.id}>{item.name}</li>

))}

</ul>

</div>

);

}