**Assignment-8**

**MODULE: 10 List and Hooks**

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

* Lifecycle management is a crucial aspect of building robust React applications, and it's handled differently in class components and functional components with Hooks.
* Class Components:
* Mounting Phase:
* constructor(): This is the first method called when a class component is instantiated. It's used for initialising state and binding methods.
* componentWillMount(): This method is called just before the initial render. However, it's rarely used in practice.
* render(): Renders the component's JSX to the DOM.
* componentDidMount(): Called after the component has been rendered in the DOM. It's commonly used for data fetching, setting up subscriptions, and initialising third-party libraries.
* Updating Phase:
* componentWillReceiveProps(): Deprecated. Previously used for updating state based on new props.
* shouldComponentUpdate(): Used to control if the component should re-render or not.
* componentWillUpdate(): Deprecated. Previously used for preparation before an update.
* render(): Renders the updated component.
* componentDidUpdate(): Called after the component updates. Useful for side-effects after an update.
* Unmounting Phase:
* componentWillUnmount(): Called just before the component is removed from the DOM. Used for cleanup tasks like unsubscribing from subscriptions and releasing resources.
* For Example:
* Class\_state.jsx

import React, { Component } from 'react'

import Img from './Img';

export default class Class\_state extends Component {

constructor(){

super();

this.state = {

course:"Frontend",

name: "Raj nagar",

age: "33",

number: 1,

isImage: true

}

}

render() {

return (

<div className='container mt-5'>

<button onClick={()=> this.setState({course:"React"})}>Change</button>

<h1>My Course is {this.state.course}</h1>

<hr />

<button onClick={()=> this.setState({name:"Rajesh nagar",age:"34"})}>Change</button>

<h1>Hi my name is {this.state.name} and my age is {this.state.age}</h1>

<hr />

<button onClick={()=> this.setState({number:this.state.number+1})}>+</button>

<h1>{this.state.number}</h1>

<button onClick={()=> this.setState({number:this.state.number-1})}>-</button>

<hr />

<button onClick={()=> this.setState({isImage:false})}>Hide</button>

<button onClick={() => this.setState({isImage: true })}>Show</button>

<button onClick={() => this.setState({isImage: !this.state.isImage})}>

Hide/Show

</button>

{this.state.isImage?<Img/>:null}

</div>

)

}

}

* Img.jsx

import React from "react";

function Img() {

return (

<div>

<img

src="https://bcciplayerimages.s3.ap-south-1.amazonaws.com/ipl/IPLHeadshot2023/57.png"

alt=""

width={"200px"}

/>

</div>

);

}

export default Img;

* App.js

import Class\_state from "./State/Class\_state";

function App() {

return (

<>

<Class\_state/>

</>

);

}

export default App;

* Functional Components with Hooks:
* Functional components, in contrast, leverage Hooks for lifecycle management, offering a more flexible and concise approach.
* useState: Manages component state.
* useEffect: Replicates the functionality of multiple class component lifecycle methods.
* Mounting Phase:
* Use the `useState` hook for state initialization.
* Use `useEffect` with an empty dependency array (`[]`) to mimic `componentDidMount`. Code inside this effect runs after the initial render.
* Updating Phase:
* Use `useEffect` with specific dependencies to replicate `componentDidUpdate`, allowing you to respond to changes in state or props.
* Unmounting Phase:
* Use `useEffect` to return a cleanup function, akin to `componentWillUnmount`. This cleanup function runs before the component unmounts and can be used for resource cleanup.
* Functional components with Hooks encourage a more functional and declarative approach to managing lifecycles, resulting in cleaner, more readable code. They also facilitate better code organisation by separating concerns based on logic rather than lifecycle phases. This approach has gained popularity in React development.
* For example:
* App.js

import Func\_state from "./State/Func\_state";

function App() {

return (

<>

<Func\_state/>

</>

);

}

export default App;

* Func\_state.jsx

import React, { useState } from "react";

import Img from "./Img";

function Func\_state() {

const [course, setCourse] = useState("Frontend");

const [myobj, setObj] = useState({

name: "Raj nagar",

age: "33",

number: 1,

isImage: true,

});

return (

<div className="container mt-5">

<button onClick={() => setCourse("React")}>Change</button>

<h1>My Course is {course}</h1>

<hr />

<button

onClick={() => setObj({ ...myobj, name: "Rajesh nagar", age: "34" })}

>

Change

</button>

<h1>

Hi my name is {myobj.name} and my age is {myobj.age}

</h1>

<hr />

<button onClick={() => setObj({ ...myobj, number: myobj.number + 1 })}>

+

</button>

<h1>{myobj.number}</h1>

<button onClick={() => setObj({ ...myobj, number: myobj.number - 1 })}>

-

</button>

<hr />

<button onClick={() => setObj({ ...myobj, isImage: false })}>Hide</button>

<button onClick={() => setObj({ ...myobj, isImage: true })}>Show</button>

<button onClick={() => setObj({ ...myobj, isImage: !myobj.isImage })}>

Hide/Show

</button>

{myobj.isImage ? <Img /> : null}

</div>

);

}

export default Func\_state;

* Img.jsx

import React from "react";

function Img() {

return (

<div>

<img

src="https://bcciplayerimages.s3.ap-south-1.amazonaws.com/ipl/IPLHeadshot2023/57.png"

alt=""

width={"200px"}

/>

</div>

);

}

export default Img;