**React JS**

* **What is React JS?**
* React JS is a JavaScript framework or library for building a User Interface.
* It is used to create a single page application.
* React was developed by Facebook. Jordan Walke one of the facebook’s software engineers, created React and first it is used in Facebook newsfeed.
* **What is node js and how it requires for React?**
* NodeJS is a framework of JavaScript which is mainly used for working with the backend of our application or building the backend using JavaScript
* With the combination of Node and React, developers don't require learning complex back-end languages like Python or Ruby. They can use Node for server-side development and React for front-end code building without switching between frameworks and programming languages. And it saves resources, money, and time.
* Node js provides features for creating single page application using React js.
* Check the node version the command is – node -v.
* **What are the two node js platform that is used to create and update react js Application?**
* There are two platform provides node js to create and update React App 1) npx and npm

1. npx – npx stands for node package execute when we want to create react app then we use npx and other thing npx is only use in react only one time. There is a command to create a react app

npx create-react-app my-app

* Here my-app is a app name you write any app name instead of these.
* Check the version of npx command is- npx --v

1. npm – npm stands for node package manager after creating the app if we want to update the app or add packages using npm we can do easily.

* We can run our react app using npm start.
* Check the version using npm –v.

Note – Both npx and npm version are same after react app run successfully.

Simple React Snippet it is an extension for react JS shortcut.

* **What is React Router Dom and how it is installed?**
* React Router DOM is an npm package that enables you to implement dynamic routing in a web app,
* It allows you to display pages and allow users to navigate them.
* React Router Dom is used to build single-page applications i.e. applications that have many pages or components but the page is never refreshed instead the content is dynamically fetched based on the URL. This process is called Routing and it is made possible with the help of React Router Dom.
* Install command npm install react-router-dom@6.
* In react router dom there are many other items comes **BrowserRouter, Router, Routes**
* BrowserRouter- If suppose when we want to change the component from Url that is called BrowserRouter.
* Routes- There are many components for routing that is called Routes.
* Route- There are two element to pass the component one is path that is path is path of the component and another is element call the particular component.
* **What is MDB and how it is installed?**
* MDB stands for Material Design for Bootstrap it is same as Bootstrap that we use in css but here we use MDB in React.
* npm i mdb-react-ui-kit
* This is for font Awesome – npm i @fontawesome/fontawesome-free.
* import 'mdb-react-ui-kit/dist/css/mdb.min.css';
* import "@fortawesome/fontawesome-free/css/all.min.css";
* **Nested Router**

  <BrowserRouter>

        <Routes>

          <Route path="/" element={<Header />}>

            <Route path="home" element={<Home />} />

            <Route path="about" element={<About />} />

            <Route path="contact" element={<Contact />} />

          </Route>

        </Routes>

</BrowserRouter>

* Outlet always written in React fragment and it is used to display content in particular component.
* **Create own Routes?**
* When we want to create our own router createBrowserRouter is used.
* To parameters are used path of the route and element component we have to call.

Ex

const Mainroutes = createBrowserRouter([

    {

        path: "/",

        element :<Header/>

    },

    {

       path: "/home",

       element :<><Header/><Home/></>

    },

    {

        path:"/about",

        element:<><Header/><About/></>

    },

    {

        path:"/contact",

        element: <><Header/><Contact/></>

    }

* **Index.js**

import React from 'react';

import ReactDOM from 'react-dom/client';

import Mainroutes from './Mainroutes';

// import './index.css

// import App from './App';

// import reportWebVitals from './reportWebVitals';

import { RouterProvider } from 'react-router-dom';

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

root.render(

  // <React.StrictMode>

  //   <App />

  // </React.StrictMode>

  <RouterProvider router={Mainroutes}/>

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

// reportWebVitals();

* **Folder Structure of our own routers**



* **Lazy loading**
* Lazy loading basically used when we use to display some part when component is load when internet speed is low.

const Classcomporoutes = React.lazy(()=> import("./Classcompo/Classcomporoutes"));

const Functioncomporoutes = React.lazy(()=> import("./FunctionCompo/Functioncomporoutes"));

    {

        path:"/example",

        element:<><Header/><Example/></>,

        children:[

            {

                path:"Classcompo/\*",

                element: <Suspense fallback={<div>Loading...</div>}><Classcomporoutes/></Suspense>

            }

        ]

    },

    {

        path:"/example",

        element:<><Header/><Example/></>,

        children:[

            {

                path:"FunctionCompo/\*",

                element:<Suspense fallback={<div>Loading...</div>}><Functioncomporoutes/></Suspense>

            }

        ]

    }

* **Example.jsx**

import React from "react";

import { Link, Outlet } from "react-router-dom";

const Example = () => {

    return (<>

        <div className="container mt-3">

            <div className="row">

                <div className="col-6">

                    <h3>

                        <Link to="Classcompo">Class Component</Link>

                    </h3>

                </div>

                <div className="col-6">

                    <h3>

                        <Link to="FunctionCompo">Function Component</Link>

                    </h3>

                </div>

                    <div className="my-3">

                        <div className="col-md-12">

                            <Outlet></Outlet>

                        </div>

                    </div>

            </div>

        </div>

    </>);

}

export default Example;

* Here we have pass the folder name in the Link
* Outlet means the output display for class component menu. If we remove the outlet the class component menu doesn’t show
* **Create Classcomporoutes.jsx**

import React, { Component } from "react";

import { Route, Routes } from "react-router-dom";

import Classcompomenu from "./Classcompomenu";

import Classcompointro from "./Classcompointro";

class Classcomporoutes extends Component{

    render(){

        return(

            <Routes>

                <Route path="/" element={<Classcompomenu/>}>

                    <Route path="Classcompointro" element={<Classcompointro/>}/>

                </Route>

            </Routes>

        )

    }

}

export default Classcomporoutes;

* Here render means when we use class component if we want to return anything we write render function. This is class component so class component doesn’t return function expect function component but when we write render it allows returning in class component. Without render it gives error.
* There are two component pass. In / the default component is Classcompomenu and another in Classcompointro.
* **Classcompomenu.jsx**

import React, { Component } from "react";

import { Link, Outlet } from "react-router-dom";

class Classcompomenu extends Component {

    render() {

        return (

            <>

                <ul>

                    <li>

                        <Link to="Classcompointro">Classcompointro</Link>

                    </li>

                </ul>

                <Outlet></Outlet>

            </>

        )

    }

}

export default Classcompomenu;

* Here we create a menu for class component.
* If we remove Outlet here then the content of Class component Intro doesn’t show.
* **Classcompointro.jsx**

import React, { Component } from "react";

class Classcompointro extends Component{

    render() {

        return (

         <>

            <h2>Class Component Intro</h2>

         </>

        )

    }

}

export default Classcompointro;

* Here finally display content of Classcompointro after click on that.
* **JSX**
* JSX stands for JavaScript Syntax Extension.
* Basically with the help of JSX we can easily bind HTML and JavaScript code.

import React, { Component } from "react";

class Classcompojsx extends Component {

    render() {

        let a = 10;

        let b=20;

        let c=30;

        return (<>

            <h2>{a}</h2>

            <h2>{b}</h2>

            <h2>{c}</h2>

        </>);

    }

}

export default Classcompojsx;

* **Props**
* Props are used to pass the data from one component to another (Parent component to child component).
* Props are only display when our data is loaded in particular component.
* Below example shows that our Childcompoforclass is loaded in Classcompoprops show it shows the data in Claascompoprops.
* Basically props are nothing but when our component is loaded the data it shows the data that component.

import React, { Component } from "react";

import Childcompoforclass from "./Childcompoforclass";

class Classcompoprops extends Component {

    render() {

        return (<>

            <h1>Main Compo</h1>

            <Childcompoforclass data="dishank" email="dishank@gmail.com" />

        </>);

    }

}

export default Classcompoprops;

import React,{ Component } from "react";

class Childcompoforclass extends Component {

    render() {

        return (<>

            <h1>Child compo</h1>

            <h4>{this.props.data}</h4>

            <h4>{this.props.email}</h4>

        </>);

    }

}

export default Childcompoforclass;

* **State**
* State is used to change the component data run time.

import React, { Component } from "react";

class Classcompostate extends Component {

    constructor() {

        super()

        this.state = {

            count: 0,

            name: ""

        }

    }

    Increment = () => {

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

        this.setState({ name: this.state.name = "Dishank" })

    }

    Decrement = () => {

        this.setState({ count: this.state.count - 1 })

        this.setState({ name: this.state.name = "Dishu" })

    }

    render() {

        return (<>

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

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

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

            <button onClick={this.Decrement}>Decrement</button>

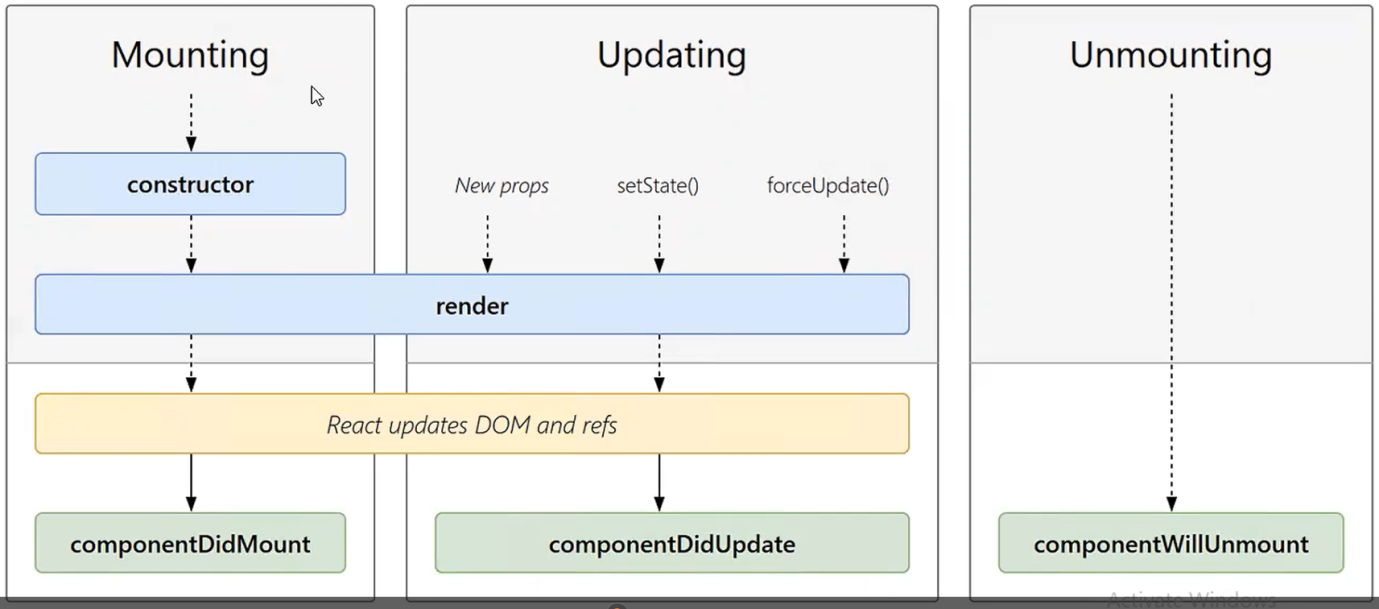
        </>);

    }

}

export default Classcompostate;

* **Class component Life cycle**

****

* **Mounting**
* First Phrase is mounting when component run successfully without any error that is called the mounting.
* Inside mounting constructor, render, Dom and componentDidMount are there without error.
* If error is occurred component is not mount.
* Mount call only one time.
* **Updating**
* In Updating when we update the component data then Updating is occur.
* Inside Updating we are able to add New props, update set state and create external function also.
* Then it goes to render after render Dom is update and componentDidUpdate are also there.
* When component not update if some error occur then componentDidnotUpdate occur.
* **Unmounting**
* Here we move from one component to another or close the component then componentwillUnmount occur.
* **Conditional Rendering Statement**
* **When we have to give condition in render then we use below condition statement.**
* First we define constructor and state.

constructor(){

        super()

        this.state ={

            isLogin:true

        }

    }

* If- Else condition
* Inside render when we have multiple condition check then we use if else

   render() {

        // 1. if else condition

      if(this.state.isLogin){

            return <h1>Welcome Dishank</h1>;

        }

        else{

            return <h1>Welcome Dishu</h1>;

        }

}

* Memory allocation method using If- Else
* Create a variable and use If-Else.

 render() {

let msg;

if(this.state.isLogin){

msg = <h1>Welcome Dishank</h1>

}

else{

msg = <h1>Welcome Dishu</h1>

}

return msg;

}

* Ternary Operator
* If we have only two conditions then use ternary operator.

return this.state.isLogin ?

<h1>Welcome Dishank</h1> :

<h1>Welcome Dishu</h1>

* Double end (&&) condition statement
* When we have only one condition to check then we use Double end (&&) condition statement

return this.state.isLogin && <h1>Welcome Dishank</h1>

* **How to get the data from Form input, select and textarea**
* There are two ways to get the data

1. Uncontrolled Component
2. Controlled Component
3. **Uncontrolled Component**

* Uncontrolled Component is handle by JavaScript or DOM,

import React, { Component } from "react";

class ClassUncontrolledcompo extends Component {

    constructor() {

        super()

        this.inputref = React.createRef()

    }

    submitData = (event) => { // here we receive the event

        //return false;

        console.log(this.inputref.current.value);

        event.preventDefault(); //Prevent default is used to change the default value for ex true to false.

    }

    render() {

        return (

            <>

                <form onSubmit={(event) => this.submitData(event)}>

                    <input type="text" ref={this.inputref} /> {/\* we have assign inputref variable in ref={this.inputref}\*/ }

                    <input type="submit" value="submitData" />

                </form>

            </>

            );

    }

}

export default ClassUncontrolledcompo;

* First we create the form in we add onSubmit method here we pass the event then we have add callback from this.submitData method after declaring the submitData method we receive event.
* We use event.preventDefault for stop refreshing the form while press submit button.
* Here we create reference using createRef method and store in the variable.
* We have assign the reference to input tag ref=this.inputref
* After assign we have print the data in submitData() method using console.log(this.inputref.current.value).

1. **Controlled Component**

* Controlled Component is handle by the React and we have use state to get the data.

import React, { Component } from "react";

class ClassControlledcompo extends Component {

    constructor() {

        super()

        this.state = {

            name: ""

        }

    }

    submitData = (event) => { // here we receive the event

        //return false;

        console.log(this.state.name);

        console.log(this.state.age);

        event.preventDefault(); //Prevent default is used to change the default value for ex true to false.

    }

    render() {

        return (

            <>

                <form onSubmit={(event) => this.submitData(event)}>

                    <input type="text" onChange={(event) => this.setState({ name: event.target.value })} value={this.state.name} />

                    <input type="text" onChange={(event) => this.setState({ age: event.target.value })} value={this.state.age} />

                    <input type="submit" value="submitData" />

                </form>

            </>

        );

    }

}

export default ClassControlledcompo;

* First declare state in constructor.

constructor() {

        super()

        this.state = {

            name: ""

        }

    }

* Then same process for stop refreshing that we have use in Uncontrolled component.
* Then assign the value in input tag value ={this.state.name}.
* We have define onChange method in Input tag and set the state for, get the change value while we press submitData button the below syntax is used for setState.

**onChange={(event) => this.setState({ name: event.target.value })}**

* After assign and setState we have print the data in submitData() method using console.log(this.state.name).
* **Class List and Keys**

import React,{Component} from "react";

class Classlistandkeys extends Component {

    constructor(){

        super();

        this.state={

            items : [1,2,3,4,5,6]

        }

    }

    add(){

        console.log("add");

        this.setState({

            items : [8,1,2,3,"dishank",4,5,6,7]

        })

    }

    render() {

        // const items = [1,2,3,4,5,6];

        const Listitems = this.state.items.map((item)=><li key={item.toString()}>{item}</li>) // when we want to add update and delete element from state normally it rerender everything but whn we pass the key as a string only add update remove index can change. we pass key as a string so the key same as a new element value.

        return (

        <>

            <h3>Classlistandkeys</h3>

            <ul>

                {Listitems}

            </ul>

            <button onClick={()=>this.add()}>Add</button>

            <div style={{margin: "250px"}}>

            </div>

            {/\* <ul>

               <li>{items[0]}</li>

               <li>{items[1]}</li>

               <li>{items[2]}</li>

               <li>{items[3]}</li>

               <li>{items[4]}</li>

               <li>{items[5]}</li>

            </ul> \*/}

        </>

        );

    }

}

export default Classlistandkeys;

* Basically List and keys are used in navbar.
* Mostly we use unordered list in list and keys.
* First we define a state inside state we create list and pass as a object.
* Then if we want to print array value as a new array then we use map.
* Map is use to destructor the array and returns the new array. Map works as a loop it accepts three argument value, index and array but here we required only value so we pass item as a callback function.
* Then print using {item} one by one array value is print.
* But when we want to update, add, remove the array element we use setState but on problem occur.
* When we add, update delete the array element from any middle or first index all the elements of an array are re rendering then backend or api has imagine that all data has added again so, the keys comes into the picture.
* Keys basically we pass as a string using <li>keys={item.toString()} after add, update, delete the element only that element is render not all elements are render so, api or backed developers are known that this particular element as affected.
* Keys value as same as the element value because we pass keys as a string and pass the callback as a item.toString().
* If suppose some of the element data are same then the keys have also same but the index value is difference. So, we know that particular index element data is change.
* Important note that original state that we define in constructor it doesn’t change but the setState array we define has change.
* **State Lifting**
* Suppose when we want to pass the data from child component to parent component then we use State lifting.
* Create a function in parent component and load the function in child component.
* Pass the data using props this is the props example for pass a data from child to parent.

import React from "react";

import { Component } from "react";

import Othercompo from "./Othercompo";

class Statelifting extends Component {

    Clickstatelift = (name) =>{

        console.log("Clickstatelift" + name);

    }

    render() {

        return (<>

            <h1>Statelifting</h1>

            <Othercompo name="dishank" Clickstatelift={this.Clickstatelift} />

        </>);

    }

}

export default Statelifting;

import React from "react";

import { Component } from "react";

class Othercompo extends Component {

    render() {

        return (<>

         <h3>Othercompo</h3>

        <p>{this.props.name}</p>

        <button onClick={()=> this.props.Clickstatelift(“Dishank”)}>Click Me</button>

        </>);

    }

}

export default Othercompo;

* Here the state is not mention so this is not state lifting.

import React from "react";

import { Component } from "react";

import Othercompo from "./Othercompo";

class Statelifting extends Component {

    constructor(props) {

        super(props);

        this.state = props

    }

    Clickstatelift = (name) => {

        console.log("Clickstatelift" + name);

        this.setState({

            name: name

        })

    }

    render() {

        return (<>

            <h1>Statelifting</h1>

            <Othercompo name="dishank" Clickstatelift={this.Clickstatelift} />

        </>);

    }

}

export default Statelifting;

* Here we create a state and pass props as an argument we give the data as a props in child component where we create a button and we have pass the data as an argument

import React from "react";

import { Component } from "react";

class Othercompo extends Component {

    render() {

        return (<>

         <h3>Othercompo</h3>

        <p>{this.props.name}</p>

        <button onClick={()=> this.props.Clickstatelift("Dishank")}>Click Me</button>

        </>);

    }

}

export default Othercompo;

* We receive the data as a state. If we don’t bind the function in parent component then it gives an error because we declare a state and state is always returns object so we bind the function as an arrow function.

    constructor(props) {

        super(props);

        this.state = props

    }

    Clickstatelift = (name) => {

        console.log("Clickstatelift" + name);

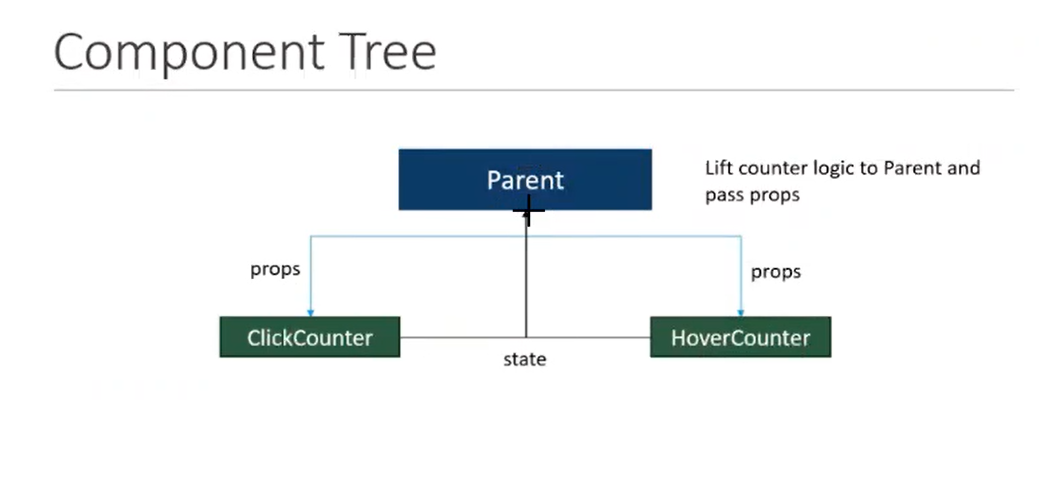
        this.setState({

            name: name

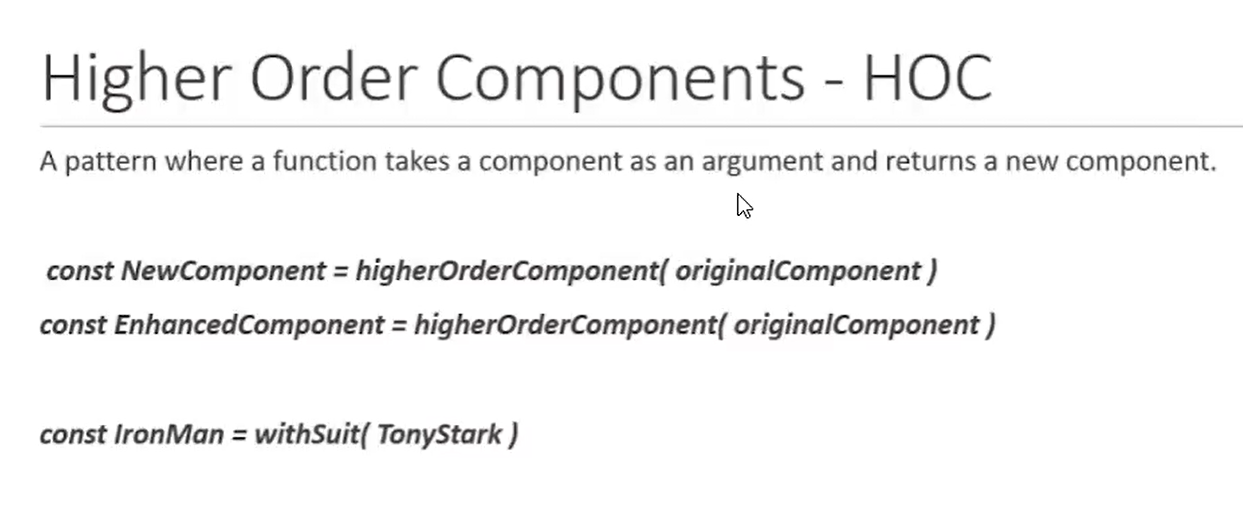
        })

    }

* After all of these we have receive the data from parent to child this is called as state lifting.
* **Higher Order Component**
* If we have same functionality of different components and we have reuse this functionality then HOC comes into the picture.



* In above diagram we have ClickCounter and HoverCounter two components have same functionality.
* We gave parent component as a state and we pass the data from parent to two components using props but directly it is not possible so, we use HOC.



* Hoc means – A pattern where a function takes a component as an argument and returns a new Component it is called HOC.
* Here IronMan is a function withSuit is component and TonyStark is an argument it always return a new Component.
* **MainHOC.jsx**

import React from "react";

import { Component } from "react";

const Updatecompo = Originalcompo => {

    class Newcompo extends Component {

        constructor() {

            super();

            this.state = {

                count: 0

            }

        }

        increment = () => {

            this.setState({

                count: this.state.count + 1

            })

        }

        render() {

            return (<>

                <Originalcompo name="Dishank Shah" increment={this.increment} count={this.state.count}/>

            </>);

        }

    }

    return Newcompo;

}

export default Updatecompo

* Here Updatecompo is my function. Originalcompo is my argument.
* It return Newcompo(Class component) We provide in return as an argument as an component here our argument is Originalcompo so, in return Originalcompo as a component return.
* Originalcompo will return Newcompo (Class component).
* Now we have passing the data from originalcompo (component).
* We provide all state data in MainHOC,jsx
* Then we pass the current count value only using this line count={this.state.count} in Originalcompo (Component).
* If we want to update the count value then also we pass increment function using increment={this,increment()}in Originalcompo (Component).
* **Clickcounter.jsx**

import React from "react";

import { Component } from "react";

import Updatecompo from "./MainHOC";

class Clickcounter extends Component {

    render() {

        const { count, increment } = this.props

        return (<>

            <h1>Clickcounter</h1>

            <p>{this.props.name}</p>

            <button onClick={increment}>Clickcounter-{count}</button>

        </>);

    }

}

export default Updatecompo(Clickcounter);

* **Hovercounter.jsx**

import React from "react";

import { Component } from "react";

import Updatecompo from "./MainHOC";

class Hovercounter extends Component {

    render() {

        const {count,increment} = this.props

        return (<>

            <h1>Hovercounter</h1>

            <button onMouseOver={increment}>Hovercounter-{count}</button>

        </>);

    }

}

export default Updatecompo(Hovercounter);

* Here both child components have same functionalities and we receive data has a props.
* Here we receive the name but we can’t do without passing argument with Updatecompo now, click counter is my argument and we receive the data from originalcompo with these line export default Updatecompo(Clickcounter) or Updatecompo(Hovercompo).
* We also receive increment function as a props in both child components using two below lines:
* Here Updatecompo as a main function and we pass different arguments.

const {count,increment} = this.props

<button onClick={increment}>Clickcounter-{count}</button>

<button onMouseOver={increment}>Clickcounter-{count}</button>

* **Hoc.jsx**

import React from "react";

import { Component } from "react";

import Clickcounter from "./Clickcounter";

import Hovercounter from "./Hovercounter";

class Hoc extends Component {

    render() {

        return (

        <>

            <h1>HOC</h1>

            <Clickcounter/>

            <Hovercounter/>

            </>

     );

    }

}

export default Hoc;

* **Composition and Inheritance**
* Inheritance is same as props that we have already pass the data from parent to child component using props.

import React from "react";

import { Component } from "react";

import Childcompoforclass from "./Childcompoforclass";

class CompositionandInheritance extends Component {

    render() {

const no = 45;

        return (<>

            <h2>CompositionandInheritance</h2>

            <div>

                <Childcompoforclass data="Dishank" no={no}>

                </Childcompoforclass>

            </div>

        </>

        );

    }

}

export default CompositionandInheritance;

* Here we declare no in parent component and load using no= {no}.
* Here we load child component data in parent component.

import React, { Component } from "react";

class Childcompoforclass extends Component {

    render() {

        return (<>

            <h1>Child compo</h1>

            <h4>{this.props.data}</h4>

<h6>{this.props.no}</h6>

            {/\* <h4>{this.props.email}</h4> \*/}

        </>);

    }

}

export default Childcompoforclass;

* Pass the data from parent component to child component using props it is also known as Inheritance because inheritance means child access all parent properties.
* Composition means we have multiple tags in one component(inside child component), when we want to access multiple tags that loaded inside child component then we use composition,

    render() {

        return (<>

            <h2>CompositionandInheritance</h2>

            <div>

                <Childcompoforclass data="Dishank">

                    <input type="text" name="" id="name" className="name" />

                    <button type="submit">Submit</button>

                </Childcompoforclass>

            </div>

        </>

        );

    }

* Here we load input type text and button.

import React, { Component } from "react";

class Childcompoforclass extends Component {

    render() {

        return (<>

            <h1>Child compo</h1>

            <h4>{this.props.data}</h4>

            <h4>{this.props.email}</h4>

            <div>

                {this.props.children}

            </div>

        </>);

    }

}

export default Childcompoforclass;

* If we want to access multiple tags then we write **{this.props.children}** it will access all tags that inside opening and closing childcompoforclass.
* **Function Component**
* Functional component is just a simple JavaScript function; it accepts the data in the form of props and returns the react element.
* Whereas the class component will be created using the class keyword, and it extends the React. Component to make the class as a react component.
* Without writing a class it provide all class functionality it is called function component.
* It starts with function and function component name same as function name is called function component.
* It provides some advance features like Hooks, API.

Ex of Function Component

import React from "react";

const Functioncompointro = () => {

    return (<>

<h2>FunctionCompoIntro</h2>

    </>);

}

export default Functioncompointro;

* **Configuring Routes in Function Component**
* **Functioncomporoutes.jsx**

const Functioncomporoutes = () => {

    return (<>

        <Routes>

            <Route path="/" element={<Functioncompomenu />}>

                <Route path="Functioncompointro" element={<Functioncompointro />} />

                <Route path="Functionprops" element={<Functionprops />} />

            </Route>

        </Routes>

    </>);

}

* **MainRoutes.jsx**

const Functioncomporoutes = React.lazy(()=> import("./FunctionCompo/Functioncomporoutes"));

{

        path:"/example",

        element:<><Header/><Example/></>,

        children:[

            {

                path:"FunctionCompo/\*",

                element:<Suspense fallback={<div>Loading...</div>}><Functioncomporoutes/></Suspense>

            }

        ]

}

* **Example.jsx**

                <div className="col-6">

                    <h3>

                        <Link to="FunctionCompo">Function Component</Link>

                    </h3>

                </div>

                    <div className="my-3">

                        <div className="col-md-12">

                            <Outlet></Outlet>

                        </div>

                    </div>

* Here if we don’t write outlet then menu will not show.
* **Functioncompomenu.jsx**

import React from "react";

import { Link, Outlet } from "react-router-dom";

const Functioncompomenu = () => {

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <ul>

                    <li><Link to="Functioncompointro">Functioncompointro</Link></li>

                </ul>

                <ul>

                    <li><Link to="Functionprops">Functionprops</Link></li>

                </ul>

            </div>

        </div>

        <Outlet></Outlet>

    </>);

}

export default Functioncompomenu;

* Here if we don’t write an outlet then inside menu content will not show.
* **Props in Function Component**
* If we want to pass the data from parent to child component we use props.

import React from "react";

import Functionchildcompo from "./Functionchildcompo";

const Functionprops = () => {

    const data1 = 50

    const data2 = 50

    return (<>

        <div className="container">

            <div className="row">

                <div className="col-6 offset-6">

                    <h2>Functionprops</h2>

                    <Functionchildcompo name="Dishank" ans={data1 + data2} data1={data1} data2={data2} />

                </div>

            </div>

        </div>

    </>);

}

export default Functionprops;

* Here we load child component in parent component and load the data into child component.

import React from "react";

const Functionchildcompo = (props) => {

    return (<>

        <h2>Functionchildcompo</h2>

        <p>{props.name}</p>

        <p>The first value {props.data1}</p>

        <p>The second value {props.data2}</p>

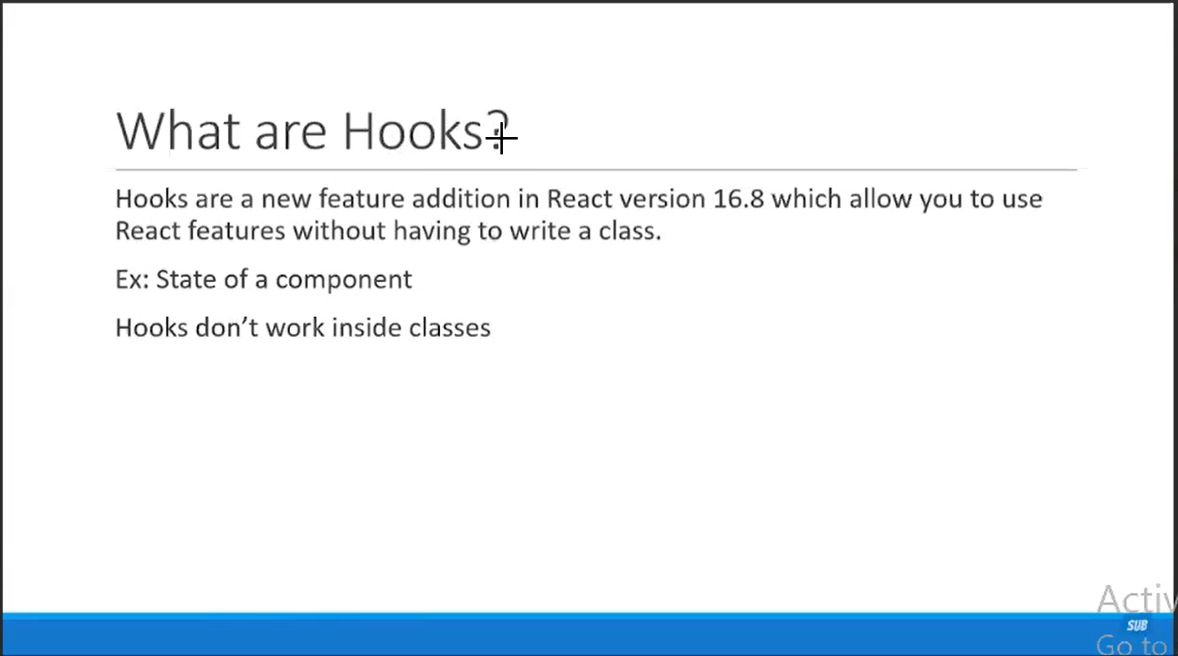
        <p>The sum is {props.ans}</p>

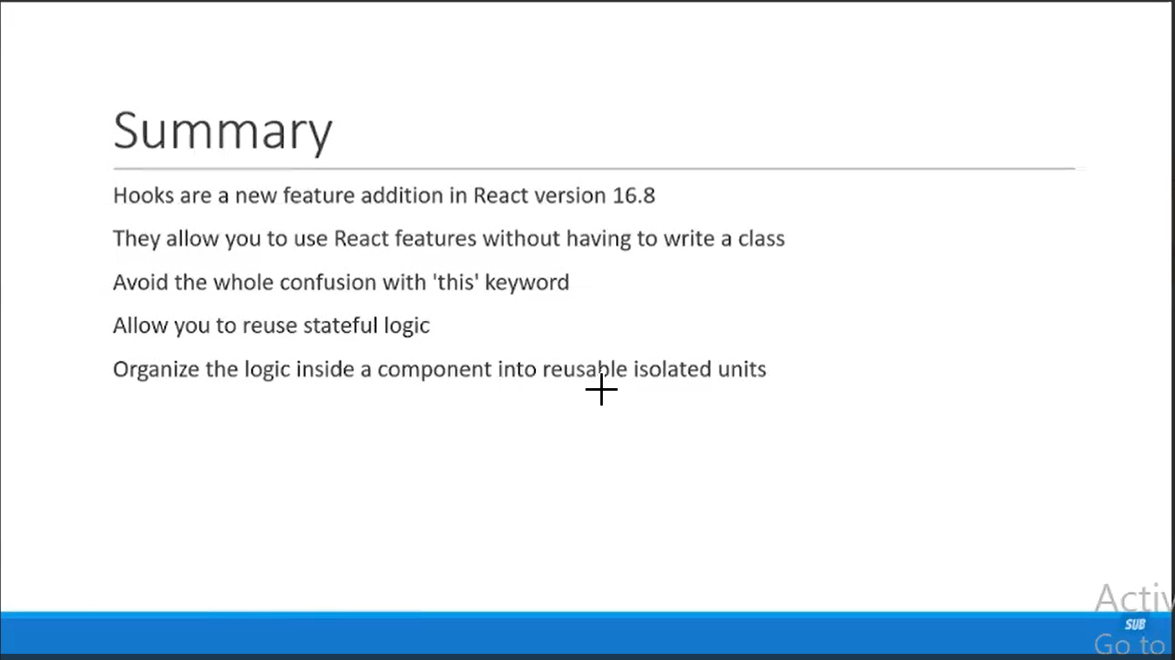
    </>);

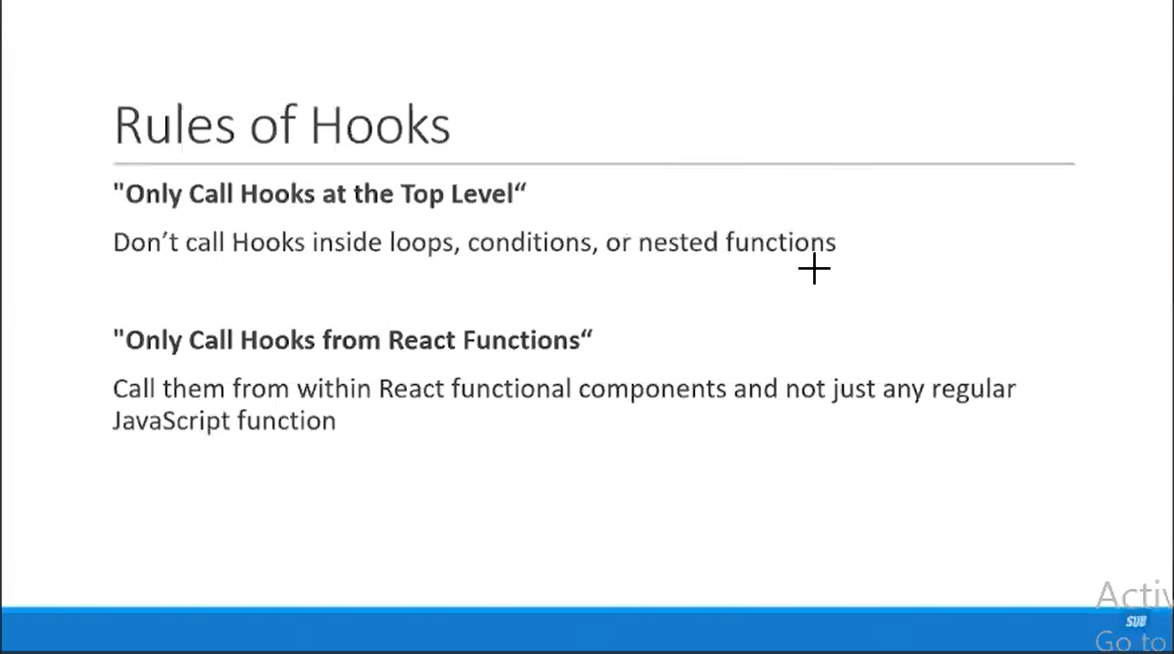
}

export default Functionchildcompo;

* Here we receive the data as a props parameter and after receiving the data we print the data onto particular tag.
* **Hooks**







* Whenever we saw use it means it is hook.
* **List of Hooks**

1. useState Hook
2. useEffect Hook
3. useRef Hook
4. useMemo Hook
5. useContext Hook
6. useCallback Hook
7. useReducer Hook

* **UseState**
* useState is React Hook that allows you to add state to a function component. It returns an array with two values: the current state and a function to update it. The Hook takes an initial state value as an argument and returns an update state value whenever the setter function is called.
* Here setter function are onClick, onChange, onfocus, onload, onchange.

import React from "react";

import { useState } from "react";

const Usestate = () => {

    const [count, setCount] = useState(0)

    const [item, setItem] = useState(1)

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <h2>Usestate</h2>

                <p>{count}</p>

                <p>{item}</p>

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

                <button onClick={() => setItem(item \* 2)}>Update</button>

            </div>

        </div>

    </>);

}

export default Usestate;

* Here count means current state(value), setCount means function for update the state and useState(0) here 0 is an initial state value as an argument.
* **UseEffect**
* When we want to display some content on component render time then we use UseEffect.
* Generally we use UseEffect when we want to saw the data to user when component is render.
* Basically useEffect works asynchronously.

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

const Useeffect = () => {

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

    const [text, setText] = useState("");

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

    // useEffect(() => {

    //     console.log("useEffect")

    // }, [])//dependency

    useEffect(() => {

        if (!data) {

            setText("Code Start...")

        }

        else {

            setText("Code END...")

        }

        console.log("useEffect");

    },[])

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <h3>{count}</h3>

                <h4>{text}</h4>

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

            </div>

        </div>

    </>);

}

export default Useeffect;

* In useEffect we pass [] means only the render time the useEffect data will be show [] this is called dependency.
* But when we pass the dependency on button click event [count] then only on button click event useEffect functionality will show.
* **Pass the data using props in Useeffect**

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

import Useeffectchild from "./Useeffectchild";

const Useeffect = () => {

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

    const [item,setItem] = useState(0);

    const [text, setText] = useState("");

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

    useEffect(() => {

        if (!data) {

            setText("Code Start...")

        }

        else {

            setText("Code END...")

        }

        console.log("useEffect");

    }, [count])

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <h3>{count}</h3>

                <h4>{text}</h4>

                <Useeffectchild item={item}/>

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

                <button onClick={() => setItem(item + 5)}>Item Update</button>

            </div>

        </div>

    </>);

}

export default Useeffect;

* Here I have load the Usereffectchild component and pass the item data.
* In count button I have gave the dependency on same file so every time when the count button is click the data is show.
* **Useeffectchild.jsx**

import React from "react";

import { useEffect } from "react";

const Useeffectchild = (props) => {

  useEffect(() => {

        alert("User effect child update")

    }, [props.item])//dependency

return ( <>

        <h4>{props.item}</h4>

    </> );

}

export default Useeffectchild;

* Here I receive the data using props.
* I give the dependency of item so when click on item button useeffect data will show.
* **UseLayoutEffect**
* useLayoutEffect is a version of [useEffect](https://react.dev/reference/react/useEffect) that fires before the browser repaints the screen.
* The [useLayoutEffect](https://www.geeksforgeeks.org/reactjs-uselayouteffect-hook/) works similarly to useEffect but rather working asynchronously like useEffect hook, it fires synchronously after all DOM loading is done loading. This is useful for synchronously re-rendering the DOM and also to read the layout from the DOM. But to prevent blocking the page loading, we should always use useEffect hook.
* The useLayoutEffect hook works in the same phase as componentDidMount and componentDidUpdate methods. We should only use useLayoutEffect if useEffect isn’t outputting the expected result.
* It works synchronously.
* It works when we click on component before render the page.
* We generally use in admin panel when we want to show data before render of the page.

import React, { useEffect } from "react";

import { useLayoutEffect, useState } from "react";

const UseLayoutEffect = () => {

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

    const [text, setText] = useState("");

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

    useEffect(() => {

        console.log("useEffect")

    }, [count]) //dependency - if we want to print the data on some particular event is trigg

    useLayoutEffect(() => {

        console.log("useLayouteffect")

        if (data) {

            setText("useLayouteffect admin data");

        }

        else {

            setText("End of useLayouteffect");

        }

    }, [text])

    return (

        <>

            <div className="row">

                <div className="col-6 offset-6">

                    <h2>UseLayouteffect</h2>

                    <h3>{count}</h3>

                    <h3>{text}</h3>

                    <h3>{data}</h3>

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

                </div>

            </div>

        </>);

}

export default UseLayoutEffect;

* Here first print useLayoutEffect because first load useLayouteffect.
* Then print useEffect because component render at that time it print.
* Here I pass two dependencies first dependency that I have pass in useEffect when we click on count button then it will print.
* Second dependency when if condition is called useLayoutEffect dependency again prints.
* **UseMemo**
* Basically when we create a function in our component each and every time our component is render or particular button is called then the function data is show.
* Use Memo use when the memorized value continuously show then we gave dependency and stop this memorized value to show each and every time.
* Use memo always return memorized value.

import React, { useMemo, useState } from "react";

const Usememo = () => {

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

    const [item, setItem] = useState(0);

    const [text, setText] = useState("Sachin");

    const Usememohook = useMemo(function multicount() {

        console.log("multicountDemo");

        return count \* 2

    }, [count])

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <h2>Usememo</h2>

                <h2>{count}</h2>

                <h2>{item}</h2>

                <h2>{text}</h2>

                <h2>{Usememohook}</h2>

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

                <button onClick={() => setItem(item + 5)}>Update Item</button>

                <button onClick={() => setText("Virat")}>Update Text</button>

            </div>

        </div>

    </>);

}

export default Usememo;

* Here I have pass count dependency so only count memorized value data display when count button is called.
* **UseCallback**
* When we create a child component and load in parent. In child component we create a function then if we provide any functionality in parent component the child component is also render every time.
* So we use memo to stop these rendering.
* **Callbackchild.jsx**

import React from "react";

import { memo } from "react";

const Callbackchild = (props) => {

   console.log("Callbackchild");

   return ( <>

        <h2>Callbackchild</h2>

        <button onClick={props.add}>Addtion</button>

    </> );

}

// function Callbackchild() {

//     console.log("Callbackchild");

// }

export default memo(Callbackchild);

* But when we have pass the function that is created in parent component and pass into child component then the same problem is occurred the child component function data is show that is not required.
* The solution is useCallback.
* useCallback always return memorized function.

import React, { useCallback, useState } from "react";

import Callbackchild from "./Callbackchild";

const Usecallback = () => {

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

    const [item, setItem] = useState(0);

    const add = useCallback(function addtion() {

        var a = 10;

        var b = 20;

        var c = a + b;

        console.log("addtion of two numbers ", c);

    }, [count])

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <h3>Usecallback</h3>

                <h3>{count}</h3>

                <h3>{item}</h3>

                <Callbackchild add={add} />

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

                <button onClick={() => setItem(item + 5)}>Update Item</button>

            </div>

        </div>

    </>);

}

export default Usecallback;

* Here I pass the count dependency when we click on the count child component function is display data.
* **Use Context Concept**
* Use context basically declare with three things:

1. createContext(): It create a function to create a context.
2. Provider: It is used to pass the data to other component.
3. Consumer: It is used to receive the data to component.

* **Usecontext.jsx**

import React from 'react';

import CompoA from './CompoA';

import { createContext } from 'react';

const Fname = createContext();

const Usecontext = () => {

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <h1>Usecontext</h1>

                <Fname.Provider value="dishank">

                    <CompoA />

                </Fname.Provider>

            </div>

        </div>

    </>);

}

export default Usecontext;

export {Fname};

* First we create a context using createContext() and store it in the variable.
* If we want to pass the data then we use Provider and give the value to the Provider.
* Here I load CompoA that is child of Usecontext.
* Then mention whichever component we want to pass the data in between.
* Then export the variable using export {variable name}.
* **CompoA.jsx**

import React from 'react';

import CompoB from './CompoB';

const CompoA = () => {

    return ( <>

        <CompoB/>

        {/\* <h2>CompoA</h2> \*/}

    </> );

}

export default CompoA;

* Here I load the CompoB that is child of CompoA.
* **CompoB.jsx**

import React from 'react';

import CompoC from './CompoC';

const CompoB = () => {

    return ( <>

        <CompoC/>

    </> );

}

export default CompoB;

* Here I load the CompoC component that is child of CompoB.
* **CompoC.jsx**

import React from "react";

import { Fname } from "./08Usecontext";

const CompoC = () => {

    return (<>

        <Fname.Consumer>

            {

                (data) => {

                    return <h2>Hello {data}</h2>

                }

            }

        </Fname.Consumer>

    </>);

}

export default CompoC;

* Import the variable then I use Consumer to receive the data.
* Consumer default use JSX and callback function then return the value then we have pass.
* **UseContext**
* UseContext is a hook that is used to pass the data from nested component (parent child to child or vice versa).
* The process is above as same but expect consumer I use useContext hook.
* **Usecontextmain.jsx**

import React, { createContext } from 'react';

import One from './One';

const Fname = createContext();

const Usecontextmain = () => {

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <h2>Usecontextmain</h2>

                <Fname.Provider value="Dishank Shah">

                    <One />

                </Fname.Provider>

            </div>

        </div>

    </>);

}

export default Usecontextmain;

export {Fname};

* **One.jsx**

import React from 'react';

import Two from './Two';

const One = () => {

    return ( <>

        <Two/>

    </> );

}

export default One;

* **Two.jsx**

import React from 'react';

import Three from './Three';

const Two = () => {

    return ( <>

        <Three/>

    </> );

}

export default Two;

* **Three.jsx**

import React from 'react';

import { Fname } from './09Usecontextmain';

import { useContext } from 'react';

const Three = () => {

  const data = useContext(Fname)

  return (<>

        <h2>Hello {data}</h2>

    </>);

}

export default Three;

* Instead of use consumer we define useContext hook create a variable here import and receive the data from Fname and just print the variable name using any of html tag.
* **Usereducer**
* Usereducer is used to create a reducer function inside reducer function we create multiple functions using state and action we give.
* When we want to short your code and create multiple if conditions inside reducer function then we use reducer.

import React, { useReducer } from 'react';

const initialState = 0

const reducer = (state, action) => {

    console.log(state, action);

    if (action.type === "INCREMENT") {

        return state + 1;

    }

    if (action.type === "DECREMENT") {

        return state - 1;

    }

    return state;

}

const Usereducer = () => {

    // const [count, setCount] = useState(1);

    // const increment = () => {

    //     setCount(count + 1)

    // }

    // const decrement = () => {

    //     setCount(count - 1)

    // }

    const [state, dispatch] = useReducer(reducer, initialState)

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <h3>{state}</h3>

                <button onClick={() => dispatch({ type: "INCREMENT" })}>increment</button>

                <button onClick={() => dispatch({ type: "DECREMENT" })}>decrement</button>

            </div>

        </div>

    </>);

}

export default Usereducer;

* First we create a useReducer hook. Inside reducer hook we pass two arguments first is reducer it is a function, and second is initialState that we define top of the component.
* Here we pass state and dispatch. State is takes intialState value and dispatch it takes a object that is define in button click event what action is perform in condition.
* In reducer function we pass two arguments first is state and second is action.
* We receive state value and action that we want to perform.
* In action we write the condition action type that we have created in dispatch object type. If condition is satisfy then particular data is shown.