**4-12-24**

|  |  |  |  |
| --- | --- | --- | --- |
| **Angular(2009)** | **React (2011)** | **Angular 2 (2014-15)** | **Vue (2015-16)** |
| By google | By facebook (JORDAN WALKE) | By google | Evan you |
| framework | library | framework | framework |
| MVC architecture | Component based architecture | Component based architecture | Component based architecture |
| Support - js | Support – js/type script | Support – type script | Support – js/type script |

**NPM-**

* Node package manager
* Worlds largest package manager registery.
* Package installer.
* Other package managers – yarn, chocolaty, bun

**Working of react app -**

CreateRoot(document.getElementById(“root”).render(<app/>)

* In index.html div is there we are fetching that div in main.jxs file by using – ‘document.getElementById(“root”).’
* Here ‘root’ is id for the div.
* By doing this we are creating a root for react application.

**Working of render(<app/>) -**

* it will render the all component of react means takes all components and execute and return the **jxs** back.
* All the returned jxs are added or appended into div tag of index html file.

**Component –**

* It is small elements of react where we write and execute the logic.
* Here we can write java script code as well as jsx code.
* After writing all code in component we have to export the component by which we can import that particular component in other jsx file.
* The syntax for exporting - ‘**export default Component\_name**;’

**Eg - export default App;**

* At the time of importing the exported component we have to write following line of code

The syntax for importing - **‘import Component\_name from 'file\_path;’**

**Eg - import App from './App';**

**Use of java script in component –**

* We can write java script code in react component.
* We can write js code outside of the component but it will not work sometimes.
* function App() {
* var a = 10;
* return (
* <div>Number is {a}</div>
* );
* }

**Jsx Expression –**

* If we want to write the java script code in jsx file then we have to use jsx expression.
* Syntax: {js code}
* <div>Number is {a}</div>

**Import/Export**:

1-**Named import:**

* If want to import the named export component the we have to specify component name in {}.

Eg –

export let Component\_name = () => {

    return (

        <div>

        </div>

    );

}

import {Component\_name} from ' Component\_path';

2- **Default import:**

* If want to import the default export component the we have to specify component name only.

Eg-

export let Component\_name = () => {

    return (

        <div>

        </div>

    );

}

export default Component\_name;

import Component\_name from ' Component\_path';

**JSX:**

* JSX stands for Java Script and XML.
* JSX is a HTML looks like structure.
* For writing JSX we need to follow some rules.

**JSX Rules:**

1. You must return the jsx, if you are not returning jsx then you will not get output.
2. We cannot return more than one jsx element at a time only one jsx element is allowed to be return, if we want to return more than one element then we have to wrap those elements into one single parent element.

//It will work as we are returning only one element

    return (

        <div>

            JsxRule2 - {rule2}

        </div>

    )

    // It will not work beacuse we are returning more than one element

    // return (

    //     <div>

    //         JsxRule2 - {rule2}

    //     </div>

    //     <div>

    //     JsxRule2 - {rule2}

    //     </div>

    // )

1. While writing jsx code each and every jsx element must be close.

// It will work beacuase we are closing the jsx elements

    return (

        <div>

            <br />

            <hr />

            <img src="" alt="" />

            <input type="text" />

        </div>

    )

    // It will not work beacuase we are not closing the jsx elements

    // return (

    //     <div>

    //          <br>

    //          <hr>

    //         <img>

    //          <input>

    //     </div>

    // )

1. We cannot use attributes like class and for while writing the jsx. Instead of that we have to use className and htmlFor.

<p className=''>JsxRule4 - {rule4}</p>

<label htmlFor="userName"></label>

1. **React Fragments-**

* React Fragments are simply a pair of empty angular braces.
* By using react fragment we avoid the creation of extra nodes.
* As we are not creating extra nodes it will increase the performance.
* We can write react fragment in three ways –

1. By using empty angular braces

return (

        <>

            JsxRule5 - {rule5}

        </>

    )

1. By using “React.Fragment”

//2nd way - As React.Fragment

    // return(

    //     <React.Fragment>

    //        JsxRule5 - {rule5}

    //     </React.Fragment>

    // )

1. By Using Fragment but this we need to import Fragment from React.

  //3rd way - As Fragment only but this we need to import Fragment from React

    // return(

    //     <Fragment>

    //        JsxRule5 - {rule5}

    //     </Fragment>

    // )

**Note -** We use the other way because there is one attribute called “key={value}” which only write in Fragment or React.Fragment it is not allowed to write that key in empty fragment.

// return(

    //     <Fragment key = {1}>

    //        JsxRule5 - {rule5}

    //     </Fragment>

    // )

1. We can write any java script in component directly by using jsx expression.

**JSX Expression Rules:**

1. We cannot write console statement directly inside jsx expression because it will not display on UI.
2. We cannot write conditional statement in jsx expression expect ternary operator and short circuit operator.

**Short circuit && operator: -**

If 1st statement is true then only second statement will be checked.

Ex: {mode == 'dark' && <p>☀️</p>}

**Short circuit || operator: -**

If 1st statement is true then second statement will not be checked.

Ex: {mode == 'dark' || mode == 'light'}

function JsxExpRule2() {

    const expRule2 = "We cannot write conditional statement in jsx expression expect ternary operator and short circuite operator."

    let mode = "dark";

    let api = {}

    let un = "abc"

    let pass = "123"

    let ph = '1234567890'

    return (

        <div>

            Jsx Expression Rule 2 - {expRule2}

            {/\* This is not allowed \*/}

            {/\* {if(mode == "dark"){

                return (

                    <p>☀️</p>

                )

            }else{

                return(

                    <p>🌑</p>

                )

            }} \*/}

            {/\* we have to use ternary operator and short circuit\*/}

            {(mode == "dark") ? <p>☀️</p> : <p>🌑</p>}

            {mode == 'dark' && <p>☀️</p>}

            {/\* eg : for short circuit && oparator:- if 1st statement is true thenonly second statement will be checked. \*/}

            { // Array.isArray(api) && api.map(val => {

                //     return val;

                // })}

            {(un == "abc" && pass == "123" && "Welcome user")}

</div>

    )

}

export default JsxExpRule2

**7-12-24**

1. We cannot use all looping statements inside jsx expression. We can use only the higher order functions, but all hof are also not useful, only those hof are useful that can return the jsx (map, reduce, filter).

import React, { Fragment } from 'react'

function JsxExpRule3() {

    let nums = [1, 214, 34, 14, 14]

    return (

        <div>

            <p>Numbers are {nums.map((val, i) => {

                //instead of creating new elements we go with fragments

                return <Fragment key={i}>{val + " "}</Fragment>

            })}

</p>

<p>Larger numbers are {nums.filter((val, i) => {

                //instead of creating new elements we go with fragments

                return val > 10 && <Fragment key={i}>{val + " "}</Fragment>

            })}

</p>

        </div>

    )

}

export default JsxExpRule3

1. We can return only one element without taking parentheses but the returned element must be t the same line of return keyword, if we want to return more than one element then it is mandatory to use parenthesis.
2. We can call the component by writing it in paired tag, and component name should be in upper case, we can’t write html tags in upper case as component name.

**6-12-24**

**NPM:** (Node package manager)

* It will compulsory download all the packages and libraries in system means we are downloading packages and then we can use them.
* npm create react

**NPX:** (Node package executer)

* It will download only the necessary packages and remaining packages we can access without downloading them (virtually).
* NPX do not come with Vite.
* Command to create project: npx create-react-app project\_name (lower case only)
* To run the app: npm start
* Eg: If we want to use face book in mobile we have 2 ways:

1. Downloading face book app which is similar with NPM means we are downloading and then only using it
2. Downloading the browser (necessary package) here in this example we are not downloading the face book but instead we are downloading the necessary package (browser and executing/ using face book on that package.

* Folder structure:
  + PeojectName

**9-12-24**

**render (): -**

Render method is used to display jsx message on the UI and during rendering multiple inbuild process will happen like transplilation, defing reconciliation and everything.

**Types of components: -**

1. **Class Based Components:**
   * CBC are the components that are created using the classes.
   * CBC is one of the traditional ways of creating components in the react.
   * It is important to understand difference between CBC and FBC.

| **Aspect** | | **Class-Based Components** | **Function-Based Components** |
| --- | --- | --- | --- |
| **Use** | | We will use classes for creating the components | We will use functions for defining the component. |
| **Extends Component** | | CBC extends Component | Does not extends the Component |
| **Super call** | | Has constructor in that we have to write **super()** call statement. | FBC does not have constructor and **super()** call statement |
| **render** | | We must use **render()** method inside that we will have return statement | FBC does not require **render()** method it will return the jsx to the caller. |
| **State** | | CBC is **stateful** component means that it has inbuild state and to manage the state we have inbuild **setState()** method. | FBC is **stateless** component means that it does not have its own state and **setState()** method. If we want to have state in FBC we have to use some extra hooks ex- **useState()** hook. |
| **Use Hook** | | We **cannot use hooks** inside CBC. | We **can use hooks** inside FBC. |
| **Life cycle methods** | | We have inbuild life cycle methods in CBC.  ex- constructor(), render(), getDerivedStateFromProps(), componentDidMount() etc. | FBC does not have any inbuild methods for managing life cycle, so to achieve life cycle methods in FBC we have to use hook called as **useEffectHook**(). |
| **Reusability** | Harder to share logic between components. | | Easier to reuse logic with custom hooks. |
| **Performance** | | Slightly less optimized due to this context. | More efficient; no need for this. |

import React, { Component } from 'react'

export default class ClassBasedComponentEx1 extends Component {

    //cbc is stateful component

    // eg 1

    // constructor() {

    //     super();

    //     this.state = {

    //         count: 0

    //     }

    // }

    // eg 2

    // let count = 0; in class we can't use var, let , const keyword for declaring the variables

    count = 0;

    constructor() {

        super();

        this.state = {

            count: 0

        }

    }

    render() {

        return (

            <div>

                ClassBasedComponentEx1

                <h2>

                    Count: {this.state.count}

                </h2>

                <button onClick={() => {

                    this.setState(() => {

                        // this.setState({ count: this.state.count++ }) it will not work

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

                    })

                }}>Add</button>

            </div>

        )

    }

}

**10-12-24**

**Hot Module Replacement: -**

* + HMR is one of the most useful features offered by react, it allows all kind of modules to be updated at runtime without a need for a full refresh.
  + HMR is not intended for use in production, means it should be only used in development.

**Hooks: -**

* + Hooks are nothing but the inbuild methods or functions
  + Hooks names start with use.
  + Ex - useState(), useEffect(), useContext(), useDispatch(), useReduser(), useCallback(), useMemo() etc.
  + We can create any function and we will name it useFunctioName().
  + Such functions can be treated as **custom hooks**.

**setState(val)** Hook: -

* + It is a hook used to make the function-based components stateful.
  + useState() returns the array in which we have two values 1st is value and another is state function.
  + We are distructuring the array into two variables state and setState for further use.

let UseStateEx1 = () => {

    let [state, setState] = useState(0);

    // useState returns the array in which we have two values 1st is value and another is state.

    // We are destructuring the array into two variables state and setState for further use.

    return (

        <div>

            UseStateEx1

            <h2>Count: {state}</h2>

            <button onClick={() => { setState(state + 1) }}>Click</button>

        </div>

    )

}

export default UseStateEx1

**Features of react: -**

* + It follows **Component based architecture**.
  + It is **declarative** in nature
  + **Declarative**: Everything is declared we just have to use it.

Ex - We don’t need to write any DOM methods because everything is already declared with jsx.

Ex - We can call react as declarative because react provides many inbuild hooks for different functionalities we just have to use them.

* + **Imperative**: Many of the things are not already declared we have to explicitly or manually explain them.
  + **Write once use anywhere:** We can create components for such functionalities that are repetitive and we can use such components n number of times.

EX – We can create button component once and we will use it where ever we want any number of times.

* + It uses DOM
  + Uses redux for globally state management

**11-12-24**

**UseState:**

**14-12-2024**

**React Questions:**

1. What is crops stealing? How to avoid that?
2. What is HOC, Context API, Redux?

**Java script Questions:**

1. Functions: arrow, hoc, cbf, generaton function and how this keyword works in all those functions
2. Object and objects methods
3. Array and methods
4. Closures
5. destructuring
6. Promises
7. Async vs await
8. Call, apply, bind
9. Hoisting

Props:

* + Used to pass the data from one component to another component.
  + Only number and string are allowed in react component.
  + Boolean, object, bigint, we cannot use directly in react component. we can display array but it is not useful because we cannot access the array element.
  + We have another type as **default props**.
  + We have to destructure the passed props object and assign the default value to that specific key.

const PropsEx1 = () => {

    let car = "BMW"

    return (

        <div>

            Props example 1

            <PropsEx1Child gift={car} />

        </div>

    )

}

export default PropsEx1

const PropsEx1Child = ({gift = 'SWIFT'}) => {

    console.log(gift);

    return (

        <div>

        </div>

    )

}

export default PropsEx1Child

* + When we write the jsx in between component opening and closing tag then it will be treated as props and passed to the child component and the key for props object is children.

return (

        <div>

           <PropsEx2Child> <h2>Heading</h2> </PropsEx2Child>

        </div>

    )

const PropsEx2Child = (props) => {

    // console.log(props.children);

    return (

        <div>

            {props.children}

        </div>

    )

}

export default PropsEx2Child

**Creating elements by using react:**

Ex: React.creactElement(‘ElementName’, Parameter, Child react element)

let myDIv = React.createElement('div', null, React.createElement('section', { title: "This is section" }, React.createElement('p', null, "Paragraph")))

**Props dealing:**

* It is the process in which we are passing the props from one react component to another react component.
* Sending the data from parent component to nth child component.
* Props dealing is not recommended.