**Module – 4 Lists and Hooks**

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

=> The Class and Function component lifecycle consists of three phases:

* Mounting lifecycle method:- that is inserting elements into the DOM.
* Updating lifecycle method :- which involves methods for updating components in the DOM.
* Unmounting lifecycle method :- that is removing a component from the DOM.

**Class Component:-**

1. componentDidMount:- This is called after a component has been inserted into the DOM. It's a great place to perform initial setup tasks, like fetching data from an API or setting up event listeners.

#### 2. componentDidUpdate:- This is called after a component has re-rendered due to changes in its state or props. It's a great place to handle side effects or perform additional actions based on those changes.

#### 3. componentWillUnmount:- This is called just before a component is removed from the DOM. It's a crucial place to perform cleanup tasks, such as clearing timers, unsubscribing from events.

#### 4. shouldComponentUpdate:- We use this lifecycle method to control whether a component should re-render when its state or props change. It is particularly useful for optimizing performance by preventing unnecessary renders.

#### Constructor() The constructor is where the initial state and the values are set in a React component. This method is called before the component is mounted.

#### Render() This is the only required method in React components. It is the most widely used one as well. This method is used in both mounting and updating phases.

import React, { Component } from 'react'

    class Lifecycle extends React.Component{

    constructor(props){

    super(props);

        this.state = { hello : "world"}

    }

    componentDidMount(){

        console.log("componentDidMount")

    }

    changeState(){

        this.setState({hello : "react"})

    }

    render(){

    return (

        <div>

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

            <h2>

                <a onClick={this.changeState.bind(this)}> Click here</a>

            </h2>

        </div>

        )

    }

    shouldComponentUpdate(nextProps,nextstate){

    console.log("shouldComponentUpdate")

    return true

    }

    componentDidUpdate(){

    console.log("componentDidUpdate")

    }

    }

    export default Lifecycle;

#### 

 import { render } from "@testing-library/react";

    import React, {Component} from "react";

    class Classcomponent extends React.Component{

    constructor(){

    super()

    this.state = {

    count : 0

    }

    this.increase = this.increase.bind(this)

    }

    increase()

    {

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

    }

    render (){

    return (

    <div>

    <h3>Counter app using class component</h3>

    <h2> {this.state.count}</h2>

    <button onClick={this.increase}>Add</button>

    </div>

    )

    }

    }

    export default Classcomponent;

#### 

**Functional Component:-**

\***useState hook:-** useState lets you add state to functional components. It returns an array with the current state value and a function to update it.

\* **useEffect hook:-** It is used for side effects in functional components, similar to componentDidMount and componentDidUpdate. It runs after rendering and can be controlled by specifying dependencies.

import React , {useState} from 'react'

    const FunctionlComponent = () => {

    const [ count , setCount] = useState(0)

    const Increase = () => {

    setCount(count + 1)

    }

    return(

    <>

    <div>

    <h3>COUNTER APP USING FUNCTIONAL COMPONENT</h3>

    <h3>{count}</h3>

    <button onClick={Increase}> Add </button>

    </div>

    </>

    )

    }

    export default FunctionlComponent

