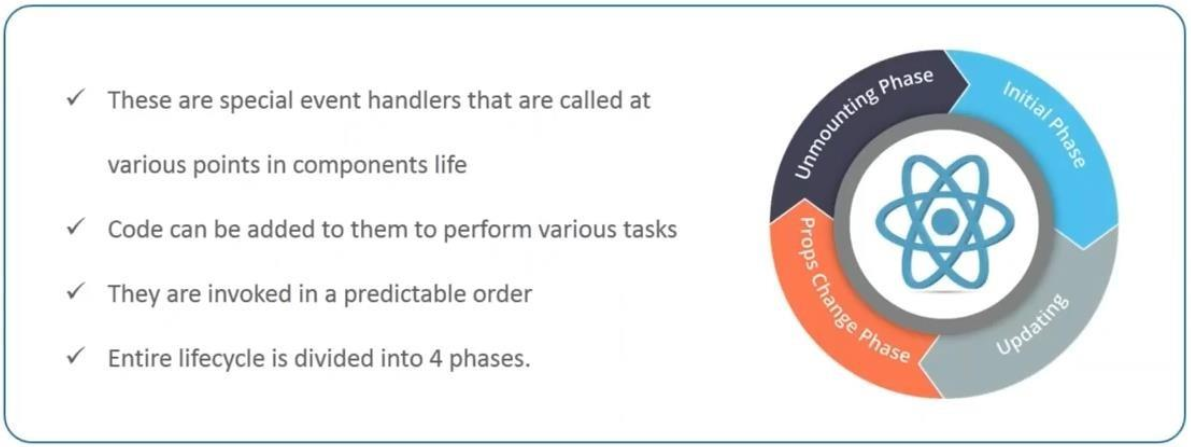
**React: List and Hooks**

Explain Life cycle in Class Component and functional component with Hooks

**Class Component Life cycle**:

* **Initialization**: This is the state where the component is constructed with the given props and default state.
* **Mounting**: Mounting is the state of rendering the JSX and returned by the render method itself. -> **componentWillMount()**
* **Updating:** Updating is the state when the state of a component is updated and the application is repaired. -> **render()**
* **Unmounting:** As the name suggest Unmounting is the final step of the component lifecycle where the component is removed from the page. -> **componentWillUnmount()**



Example code:

import React, { Component } from 'react'

class Class\_state extends Component {

  constructor(props) {

    super(props);

    this.state = {

      brand: "Ford",

      model: "Mustang",

      color: "red",

      year: 1964

    };

  }

  changeColor = () => {

    this.setState({color: "blue"});

  }

  render() {

    return (

      <div>

        <h1>My {this.state.brand}</h1>

        <p>

          It is a {this.state.color}  {this.state.model}

          from {this.state.year}.

        </p>

        <button

          type="button"

          onClick={this.changeColor}

        >Change color</button>

      </div>

    );

  }

}

export default Class\_state

**Functional Component Life cycle**:

* **Initialization:** Initialize state. -> **useState(initialState)**
* **Mounting:** -> **useEffect(() => {}, [])** similar to **useEffect(()=> {}, [])** – Runs after every render, including the first render. If you pass empty array as the second argument, it only runs once.(Equivalent to **componentDidMount**)
* **Updating:**
* useState(): Update state.
* useEffect(() => {}): Runs after every render, including update. You can conditionally run effect by passing dependencies as the second argument to useEffect. If the Dependencies change, the effect runs again.
* **Unmounting**: **useEffect(() => { return () => {} }, []) –**
* The function returned from the effect cleanup runs when the component is unmounted. This is equivalent to **componentWillUnmount.**

**Example Code:**

import React,{useState} from 'react';

function Func\_state() {

    const [color, setcolor] = useState("Red");

    const blue=()=> {

      setcolor("blue");

    }

    return (

    <>

        <h1>My Favourite Color is {color}</h1>

        <button onClick={blue}>Blue</button>

        <button onClick={()=> setcolor("Green")}>Green</button>

        <button onClick={()=> setcolor("Yellow")}>Yellow</button>

        <button onClick={()=> setcolor("Orange")}>Orange</button>

    </>

  )

}

export default Func\_state;

* **Functional component with hooks simplify the life cycle methods by allowing developers to organize code based on what it does rather than when it’s executed. However the fundamental concepts of initialization, Updating and unmounting remains the same.**