### ****1.Explain the Need and Benefits of Component Lifecycle****

**ANS:**

#### ****What is the Component Lifecycle?****

The **component lifecycle** refers to the **phases** a React component goes through:

* When it is **created (mounted)**
* When it is **updated (due to props/state changes)**
* When it is **destroyed (unmounted)**

React provides **lifecycle methods** (also called hooks in functional components) that allow developers to **run code at specific points** during these phases.

#### ****Why is Lifecycle Important?****

**Need:**

* To perform actions like **fetching data**, **setting timers**, **cleaning up resources**, or **logging** when the component changes.

**Benefits:**

* **Precise control** over what happens when and where.
* **Efficient resource management** (e.g., cleanup on unmount).
* **Performance optimization** (e.g., update only when needed).
* **State synchronization** with external systems or APIs.
* **Debugging and monitoring** component behavior easily.

### ****Identify Various Lifecycle Hook Methods****

ANS:

React provides different lifecycle methods depending on **Class** or **Function** components.

#### ➤ A. ****Class Component Lifecycle Methods****

| **Phase** | **Method** | **Purpose** |
| --- | --- | --- |
| Mounting | constructor() | Initialize state, bind methods |
|  | static getDerivedStateFromProps() | Update state from props |
|  | render() | Return JSX to render UI |
|  | componentDidMount() | Run after the component is mounted (e.g., API calls) |
| Updating | shouldComponentUpdate() | Optimize re-rendering |
|  | getSnapshotBeforeUpdate() | Capture state/info before update |
|  | componentDidUpdate() | Run after updates |
| Unmounting | componentWillUnmount() | Cleanup tasks (timers, listeners, etc.) |
| Error Handling | componentDidCatch() | Handle errors in child components |

#### ➤ B. ****Functional Component Lifecycle (via Hooks)****

| **Hook** | **Equivalent To** | **Use Case** |
| --- | --- | --- |
| useEffect(() => {}, []) | componentDidMount() | Run once after mount |
| useEffect(() => {}) | componentDidUpdate() + DidMount | Run on every render |
| useEffect(() => { return () => {} }, []) | componentWillUnmount() | Cleanup on unmount |
| useLayoutEffect() | Like useEffect, but runs **before** paint | UI measurement/DOM changes |

### ****List the Sequence of Steps in Rendering a Component****

ANS:

Here is the **complete lifecycle sequence** for a **class component**:

#### ****Mounting Phase**** (Component is being created)

#### constructor()

#### static getDerivedStateFromProps()

#### render()

#### componentDidMount()

**Updating Phase** (State or props changed)

* static getDerivedStateFromProps()
* shouldComponentUpdate()
* render()
* getSnapshotBeforeUpdate()
* componentDidUpdate()

**Unmounting Phase**

* componentWillUnmount()

**Error Handling Phase**

* componentDidCatch()

**CODE:**

**Post.js**

*// src/Post.js*

**class** Post {

**constructor**(id, title, body) {

    this.id = id;

    this.title = title;

    this.body = body;

  }

}

export default Post;

**Posts.js**

*// src/Posts.js*

import React from "react";

import Post from "./Post";

**class** Posts **extends** React.Component {

**constructor**(props) {

    super(props);

    this.state = {

      posts: [],

    };

  }

  loadPosts() {

    fetch("https://jsonplaceholder.typicode.com/posts")

      .then((response) **=>** response.json())

      .then((data) **=>** {

**const** postObjects = data.map(

          (item) **=>** new Post(item.id, item.title, item.body)

        );

        this.setState({ posts: postObjects });

      })

      .catch((error) **=>** {

        console.error("Error loading posts:", error);

        alert("Error loading posts");

      });

  }

  componentDidMount() {

    this.loadPosts();

  }

  componentDidCatch(error, info) {

    alert("An error occurred in Posts component.");

    console.error("Caught error:", error, info);

  }

  render() {

    return (

      <div>

        <h1>Blog Posts</h1>

        {this.state.posts.map((post) **=>** (

          <div key={post.id}>

            <h2>{post.title}</h2>

            <p>{post.body}</p>

            <hr />

          </div>

        ))}

      </div>

    );

  }

}

export default Posts;

**App.js**

*// src/App.js*

import React from "react";

import Posts from "./Components/Posts";

**function** App() {

  return (

    <div className="App">

      <Posts />

    </div>

  );

}

export default App;

**Output**

