MODULE: 4 (List and Hooks)

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

- The lifecycle of components in React refers to the series of events that occur during the creation, updating, and destruction of components. Here's an explanation for Class Components and Functional Components with Hooks:
- ---
- Lifecycle in Class Components
- Class components have distinct lifecycle methods that can be grouped into three phases:
- 1. Mounting Phase (When a component is added to the DOM)
- constructor(): Initializes the component's state and binds event handlers.
- static getDerivedStateFromProps(): Updates state based on props before rendering. Rarely used.
- render(): Renders the component's UI.
- componentDidMount(): Executes after the component is mounted. Used for side effects like fetching data.
- 2. Updating Phase (When props or state change)
- static getDerivedStateFromProps(): Same as in mounting. Updates state based on new props.
- shouldComponentUpdate(): Determines whether to re-render the component (default is true).
- render(): Re-renders the component.
- getSnapshotBeforeUpdate(): Captures information from the DOM before changes (e.g., scroll position).
- componentDidUpdate(): Executes after the update is complete. Used for side effects based on state/props changes.
- 3. Unmounting Phase (When a component is removed from the DOM)
- componentWillUnmount(): Clean-up tasks like removing event listeners or canceling API calls.
- ---
- Lifecycle in Functional Components with Hooks
- Functional components don’t have lifecycle methods but achieve similar functionality using React Hooks.

- 1. Mounting Phase
- useEffect(): Runs after the component mounts. You can pass an empty dependency array ([]) to mimic componentDidMount.
- javascript
- useEffect(() => {
- // Side effects like API calls
- }, []);
- 2. Updating Phase
- useEffect(): Runs whenever specified dependencies change. Used to handle updates.
- javascript
- Copy code
- useEffect(() => {
- // Code runs when dependencies update
- }, [dependency1, dependency2]);
- 3. Unmounting Phase
- useEffect(): Return a cleanup function to mimic componentWillUnmount.
- javascript
- Copy code
- use Effect(() => {
- // Effect logic here
- return () => {
- // Cleanup logic here
- };
- }, []); // Runs when the component unmounts
- Other Hooks Supporting Lifecycle:
- use State(): Manages component state like this.state.
- use Memo() and useCallback(): Optimize re-renders like should Component Update.

- Key Differences:

Aspect	Class Components	Functional Components with Hooks
Syntax	Requires class declaration	Simpler function-based syntax
State Management	this. State and set State()	use State()
Lifecycle Methods	Separate methods for each phase	Single use Effect() for side effects
Readability	May become complex for large components	Cleaner and easier to manage
Performance	Requires optimization manually	Hooks like use Memo and use Callback simplify this

- ---
- Conclusion
- Class components are traditional but more verbose, while functional components with Hooks are modern, concise, and easier to understand. Hooks like use Effect simplify lifecycle management, making functional components the preferred choice in newer React applications.