Hooks

- Hooks are one of the most powerful features in React.
- They were introduced in React 16.8 to let us use state and lifecycle methods inside functional components — without writing a class.

Before hooks:

- Functional components were stateless.
- We used class components for state & lifecycle.

After hooks:

- Functional components can now manage state, handle side effects, and reuse logic easily.

What is a Hook?

- A hook is a special function provided by React.
- It lets you "hook into" React features like state, lifecycle, context, refs, etc.
- Hooks must start with “use” (e.g., useState, useEffect).

Rules of Hooks:

- Only call hooks at the top level of your component (not inside loops/conditions).
- Only call hooks from:
  - React function components
  - Custom hooks

Commonly Used Built-in Hooks

- useState
- useEffect
- useRef
- useReducer
- useMemo
- useCallback
- useLayoutEffect
- useContex

useEffect Hook

- If useState is for managing state, then useEffect is for managing side effects in React.
- A side effect means any code that affects something outside React's render cycle.

Examples:

- Fetching data from an API
- Setting up event listeners (like resize, scroll)
- Working with timers (setInterval, setTimeout)
- Updating the browser's DOM or title manually

- Normally, React components should only return UI, but side effects let us interact with the outside world.
- useEffect is a React hook that lets us perform side effects in functional components.

```
useEffect(() => {
  // side effect code here
  return () => {
    // cleanup code (optional)
  };
}, [dependencies]);
```

Thank You