HOOKS

- React **Hooks are functions** that allow **functional components** to manage state and side effects, as well as access other React features, without the need for class components.
- It was introduced in React 16.8 and provide a simpler and more efficient way to manage component logic.
- Ex.- useState, useEffect, useMemo, useCallback etc.

Rules of Hooks :

- **Top Level Only** – Don't use hooks inside loops, conditions, or nested functions.
- **Only in React Functions** – Use hooks only in functional components or custom hooks.
- **Same Order Always** – Call hooks in the same order every render.

Features of React Hooks:

1.**State Management in Functions** →UseState lets functional components manage state.

2.Side Effects Handling →UseEffect handles tasks like API calls, timers, or event listeners.

3.Code Reusability → Custom hooks allow logic reuse without repeating code.

4.No Need for Classes→Hooks remove the need for class components—code is shorter and cleaner.

5.Better Separation of Concerns→UI logic is grouped by feature, not lifecycle methods (like in class components).

6.Improved Readability→ Functional components with hooks are easier to understand and maintain.

7.Access to Context→useContext allows easy use of global data (like user info, theme).

8.Performance Optimization→useMemo and useCallback help reduce unnecessary renders.

9.DOM References→useRef gives access to DOM elements or persistent values.

10.Complex State Logic→useReducer is great for managing advanced state logic (like Redux style).

Difference Between Hooks and Class Components

Feature	Class Components	React Hooks
State Management	this.state and lifecycle methods	useState and useEffect
Code Structure	Spread across methods, can be complex	Smaller, focused functions
Reusability	Difficult to reuse logic	Easy to create and reuse custom hooks
Learning Curve	Familiar to OOP developers	Requires different mindset than classes
Error Boundaries	Supported	Not currently supported
Third-party Libraries	Some libraries rely on them	May not all be compatible yet

ReactJS State

- The **state** refers to an JavaScript object that holds data or information about a component's current situation.
- **state** in React as a **container like a JavaScript variable**, but with some key differences.
- This information can change over time, typically as a result of user actions or **data fetching**, and when **state changes,** React **re-renders** the **component** to reflect the **updated UI.**
- Whenever state changes, React re-renders the component to reflect the updated data. This enables you to build dynamic UIs that respond to user interactions.
- **"State** is a built-in React object used to **store dynamic data** in a component.
  It determines how that component **behaves and renders**."

Syntax

```
const [state, setState] = useState(initialState);
```

In this syntax

- **state:** The current state value.
- **setState:** A function that is used to update the state.
- **initialState:** The initial value that the state will hold when the component is first rendered.

Why We Need State:

- **Dynamic UI** – It allows components to respond to **user input**, **API calls**, etc.
- **Re-rendering** – When state changes, React **automatically re-renders** the component.
- **Component Memory** – It stores data that's **local to a component** (like form inputs, toggles, counters).
- **Interactivity** – Needed to make the app **interactive** (e.g., show/hide elements, update lists, etc.)

Difference Between **Variables** and **State** in React

Feature	JavaScript Variable (let, const)	React State (useState)
Definition	Standard data holder	React hook for managing dynamic data
Reactivity	Not reactive (UI doesn't update)	Reactive (UI updates on change)
Defined	Defined using let, const	Defined using let, const
UI Updates	Manual DOM update needed	Auto re-render by React
Scope	Function or block level	Component-level
Update Method	Direct assignment (x = 10)	setState function (setX(10))

- **JavaScript variable** = regular data holder
- **React state** = special variable that **reacts** to changes and **updates UI**