React State & Hooks

What is State?

State is a way to store data in a component that can change over time. When state changes, React **automatically re-renders** the component.

State is an **updatable structure** used to store data or information about a component. It can **change over time** because of user actions or system events.

Imagine a **traffic light** with 3 colors: **Red, Yellow, Green. The current color** is the **state. When** the timer runs out, the state changes → Red → Green → Yellow → Red ... and so on. Each change in **state** updates what drivers see.

In React terms:

- The **traffic light component** has a **state** variable called `currentLight`.
- That state can be `"Red"`, `"Yellow"`, or `"Green"`.
- When `currentLight` changes, React **re-renders** the UI to show the new light.

Key Points:

- State determines the **behavior** of a component and **how it renders**.
- Must be kept as **simple as possible**.
- Represents a component's **local data** (private to that component).
- Can only be **accessed or modified** inside the component itself.

In short: **State = memory inside a component that can change and update the UI.**

What are React Hooks?

Hooks let your simple function components “tap into” React’s advanced features without writing extra complex code.

A **hook** lets your function **connect (hook into)** Reacts built-in abilities, like:

- **State** → remembering values that can change (like a counter).
- **Lifecycle** → running code when a component first shows up or updates (like fetching data when a page loads).

Examples: Hook.

- A normal function component is like a **basic phone**.
- When you “plug in” a **hook** (like `useState`), your component gains a **new ability**:
    - `useState` → memory (like saving your game progress).
    - `useEffect` → automatic actions (like alarm clock going off when the time comes).
    - `useRef` → a shortcut to something (like speed-dial for your favorite contact).

Rules of Hooks

1. **Only call hooks at the top level**
    a. Do not call hooks inside loops, conditions, or nested functions.
    b. Ensures hooks are always called in the **same order** each render.
2. **Only call hooks from React functions**
    a. Hooks can only be used inside **React function components** or **custom hooks**.
    b. Not allowed in regular JavaScript functions.

Why Are Hooks Important?

- Make functional components **more powerful**.
- Allow you to **reuse logic** (instead of duplicating code).
- Simplify components → no need for long, complex class-based code.

Common React Hooks

- **`useState`** → add state (data that changes over time).
- **`useEffect`** → run side effects (fetch data, run code when component mounts/updates).
- **`useContext`** → share data without passing props manually.
- **`useRef`** → reference DOM elements or store values without causing re-renders.
- **`useReducer`** → manage more complex state logic.

ReactJS `useState` Hook

What is `useState`?

A React hook used to add state in components. Allows you to **declare state variables** inside functional components.

Syntax

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

- **`currentState`** → holds the current value.
- **`setState`** → function used to update the state.
- **`initialState`** → the starting value.

Think of it like:

- `currentState` = getter(gets current value)
- `setState` = setter(updates current Value)

◇ Example 1: Basic Counter

```
import React, { useState } from "react";

function Counter() {
  // Define a state variable named 'count' with initial value 0
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      {/* Update state when button clicked */}
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}

export default Counter;
```

How it works:

- count starts at **0**.
- When button is clicked → `setCount(count + 1)` runs.
- React re-renders the component → UI updates with the new value.

Key Takeaways

- **State = memory that can change over time.**

- **Hooks = special functions** that unlock React features in functional components.
- **`useState` = simplest and most common hook**, used for managing local component state.(get and update the state)

Event Handling in React

Handling Events

React uses **camelCase** event names: on`Click`, on`Change`, on`Submit`. Instead of strings, you pass a **function** as the event handler.

```
<button onClick={handleClick}>Click Me</button>
<button onClick={() => greetUser("Shanice")}>Say Hello</button>
```