📘 React.js – Week 1, Day 5

Topic: State (useState Hook) – Managing Dynamic Data

1️⃣ Why Do We Need Hooks?

- In React, **functional components** cannot have state or lifecycle methods without hooks.
- Hooks like `useState` allow functional components to **remember values**, handle **dynamic data**, and react to **user interactions**.
- Hooks make functional components as powerful as class components without the complexity.

2️⃣ What is State?

- State is a **special object** that stores **dynamic data** in a component.
- Unlike props, state is **local** to the component and can be **changed inside it**.
- Updating state triggers a **re-render** to reflect the new data in the UI.

3️⃣ Introduction to useState Hook

- `useState` is a **React Hook** to add state to functional components.
- Syntax:

```
const [stateVariable, setStateFunction] = useState(initialValue);
```

- `stateVariable` → current value of the state
- `setStateFunction` → function to update the state
- `initialValue` → default value when the component mounts

4️⃣ Basic Example: Counter

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

function Counter() {
  const [count, setCount] = useState(0); // declare state

  const increment = () => setCount(count + 1); // update state

  return (
    <div style={{ border: '1px solid blue', padding: '10px', borderRadius:
```

```
'8px' }}>
      <h2>Counter: {count}</h2>
      <button onClick={increment}>Increment</button>
    </div>
  );
}

export default Counter;
```

🔗Explanation: - `count` stores the current number. - `setCount` updates `count` and triggers re-render. - Clicking the button changes the state dynamically.

5️⃣Example: Toggle Button

```
function ToggleButton() {
  const [isOn, setIsOn] = useState(false);

  const toggle = () => setIsOn(!isOn);

  return (
    <button onClick={toggle}>{isOn ? 'ON 🔗' : 'OFF 🔩'}</button>
  );
}
```

🔗Explanation: - `isOn` is boolean state. - `toggle` flips its value using `setIsOn`. - UI updates automatically based on state.

6️⃣State with Objects & Arrays

Object Example:

```
const [user, setUser] = useState({ name: 'Rehan', age: 21 });
setUser({ ...user, age: 22 }); // update age only
```

Array Example:

```
const [items, setItems] = useState([1,2,3]);
setItems([...items, 4]); // add new item
```

7️⃣ Key Points to Remember

- State updates are **asynchronous**.
- Always use the **setter function** (`setState`) to update state.
- You can have **multiple useState hooks** in one component.
- Never **mutate state directly**, always create a new object/array.

8️⃣ Day 5 Exercise (15–20 min)

Goal: Practice dynamic data using useState.

1. Create a new file `StateDemo.js`
2. Create a component `StateDemo` with:
3. Counter with **Increment** and **Decrement** buttons
4. Toggle button (ON / OFF)
5. Input box that updates a `name` state and displays the typed name below
6. Optional: list of items that you can add dynamically
7. Import `StateDemo` in `App.js` and render it
8. Verify that **state updates dynamically** and UI changes accordingly

🔗 By the end of this task, you should understand: - How `useState` works for numbers, booleans, strings, and arrays - How to update state properly - How state changes trigger re-render