**Reactjs**

React is a **JavaScript library** for building **fast, interactive, and reusable UI components**. It is maintained by **Meta (Facebook)** and widely used for developing modern web applications.

**Why Use React?**

 **Component-Based Architecture** – Helps in building reusable UI components, making development efficient.

 **Virtual DOM** – Improves performance by updating only the necessary parts of the actual DOM.

 **Fast & Scalable** – React efficiently handles large-scale applications with dynamic data.

 **Strong Community & Ecosystem** – Large community support with extensive third-party libraries.

 **Easy to Learn & Use** – Uses JSX (JavaScript XML) for a simple and declarative UI-building experience.

**Components, Props, and JSX in React**

**1.Components**

 The building blocks of a React application.

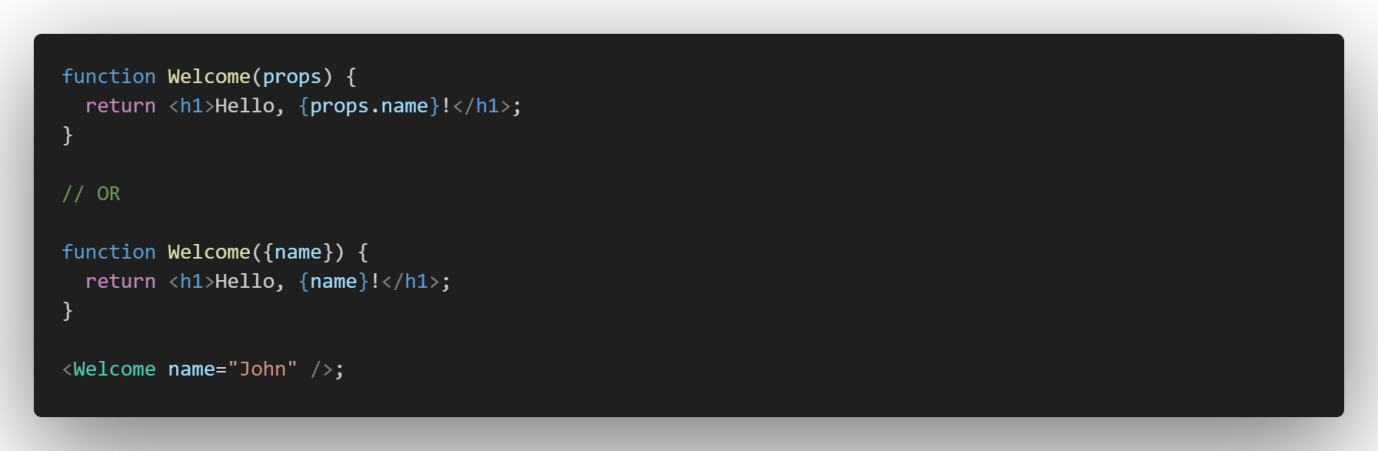
 Can be **functional** or **class-based**.

 Reusable and independent UI elements.



**2. Props (Properties)**

* Used to **pass data** from a parent component to a child component.
* **Immutable** (cannot be modified inside the child component).

****

**3. JSX (JavaScript XML)**

 A **syntax extension** of JavaScript used in React.

 Allows writing HTML inside JavaScript.

 Makes UI code more readable and easier to maintain.

****

**Hooks & State in React**

**Hooks**

Special functions that let you use React features inside functional components.

Common hooks:

* **useState** → Manages state.
* **useEffect** → Handles side effects.
* **useContext** → Accesses global state.

**State**

 State stores dynamic data and updates trigger re-renders.

 Managed using the **useState** hook.

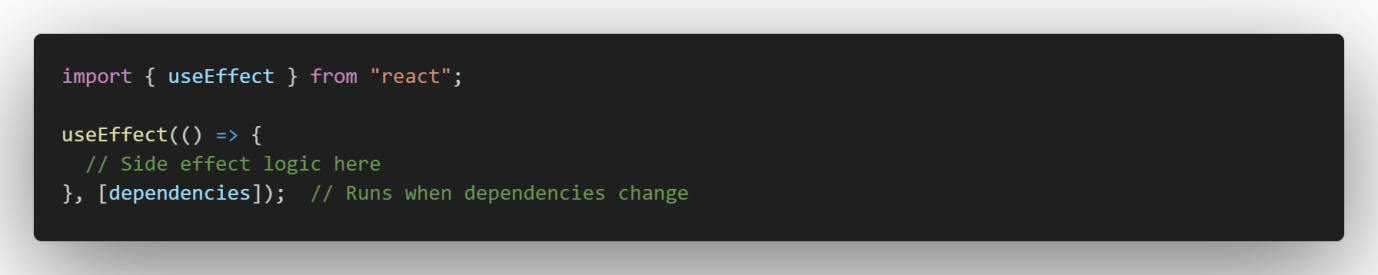


**useEffect Hook in React**

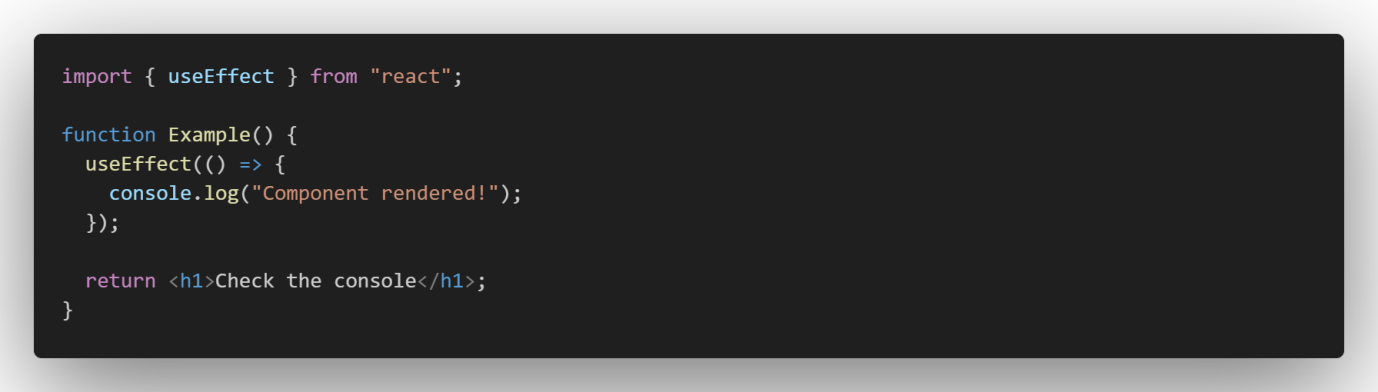
A React **hook** that performs **side effects** in functional components.

Used for:

* Fetching data (API calls)
* Subscriptions (e.g., WebSockets)
* DOM updates
* Cleanup operations



a) Run useEffect on Every Render (Runs after every render)



b) Run useEffect Only on Mount (Empty Dependency Array [])(Runs only once when the component mounts.)



c) Run useEffect When a Specific State Changes (Runs whenever count changes.)



d) Cleanup in useEffect (e.g., Unsubscribe Events) (Cleans up event listeners when the component unmounts.)



**useEffect helps manage side effects in React functional components efficiently!**

**useRef Hook in React**

 A React **hook** that creates a **mutable reference** to store values **without causing re-renders**.

 Common use cases:

* **Accessing DOM elements** (e.g., focus an input).
* **Storing previous values** without re-rendering.
* **Persisting values between renders** without triggering updates.

****

**a) Accessing DOM Elements** (The button clicks will focus on the input field.)

****

**b) Storing Previous State without Re-Rendering** (Stores the previous count value without causing re-renders.)



**c) Persisting Values across Re-Rendering** (value of a persist)



**useRef is useful for managing DOM elements, tracking values across renders, and handling performance optimizations in React!**

**Conditional Rendering & Rendering Lists in React**

**Conditional Rendering**

Conditional rendering in React allows rendering different UI elements based on conditions.

****

**Rendering Lists** (Displaying Array and Array of Objects**).**

****

****

**Handling Events in React**

React handles events similarly to vanilla JavaScript but uses JSX syntax and follows a few key differences.

**1.Adding Event Handlers**

 Events are written in camelCase (onClick, onChange).

 Functions are passed as event handlers instead of strings.

(The function handleClick runs when the button is clicked.)



**2. Passing Arguments to Event Handlers** (Using an arrow function inside onClick to pass arguments.)



**3. Handling Events in Forms** (Updates state whenever the input value changes.)





**4. Preventing Default Behavior** (Prevents form from refreshing the page on submission.)



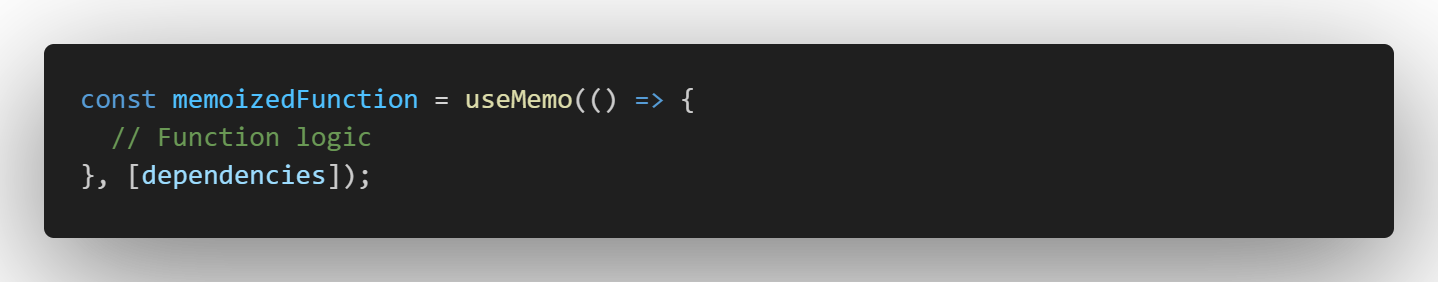
**5. Handling Events in Lists** (Dynamic Events). (Clicking an item displays its name in an alert.)



**React efficiently handles user interactions with event handlers, making UI interactive and dynamic!**

**useMemo - Memoizing Computed Values**

The useMemo hook **caches the result of an expensive computation** and only recalculates it when its dependencies change.





 **Without useMemo**, every render would recalculate the value unnecessarily.

 **With useMemo**, React only recalculates when numbers changes, **improving performance**.

**useCallback - Memoizing Functions**

The useCallback hook **caches a function reference** so that it's not recreated on every render, avoiding unnecessary re-renders of child components.

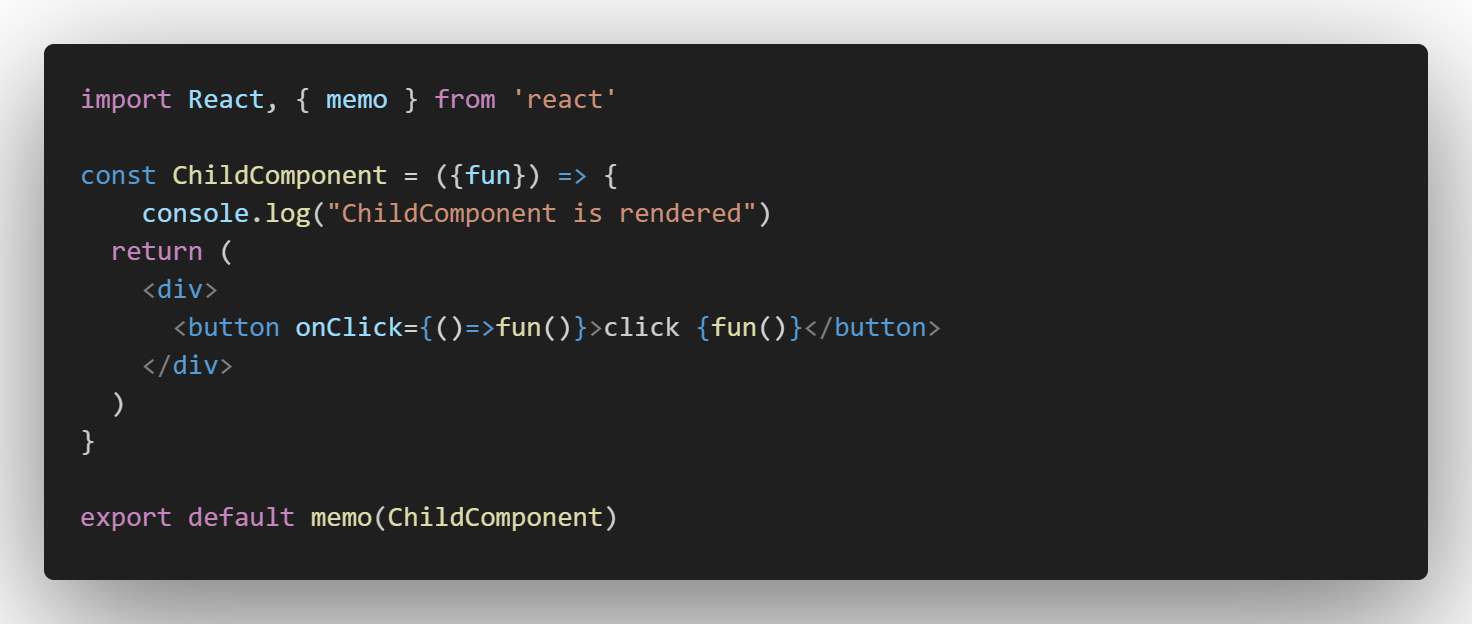
****

**Now see an example,**

Parent component

****

Child component

****

In the above code there are state and a function in parent component also we are updating state in parent component and in child component we use memo which basically stop rendering the component until the props are changed so as in parent component we are passing a function and if we do not use useCallback then every time when state is changed the function will recreate and this will change props in child due to which the child Component re-render every time when the count is changed but when we use useCallback then it memorise function so when the count is change the function will not recreate and does not affect props of child due to which the child Component will not re-render.

**When to Use?**

✅ Use useMemo when:

* You have **expensive calculations** that don’t need to run on every render.
* You want to **prevent re-computation** of derived values.

✅ Use useCallback when:

* You pass a **function as a prop** to a child component that uses React.memo() or memo().
* You want to **prevent function recreation** on every render.