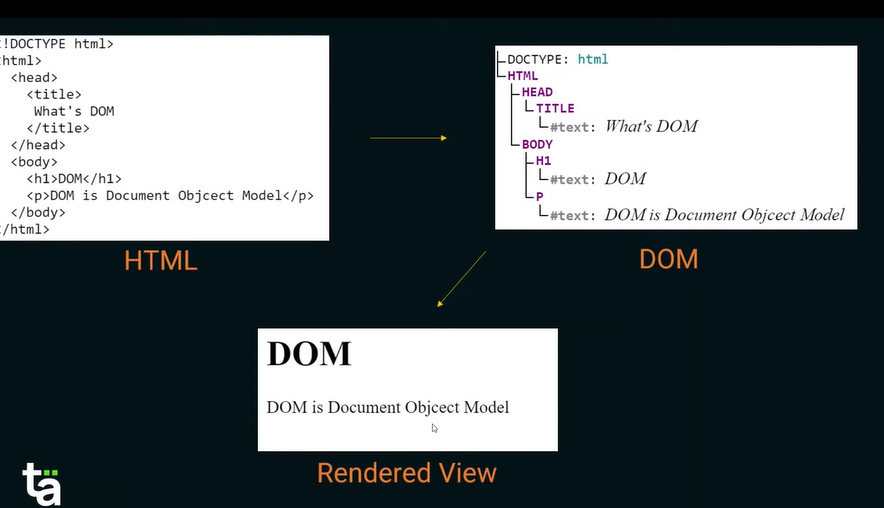
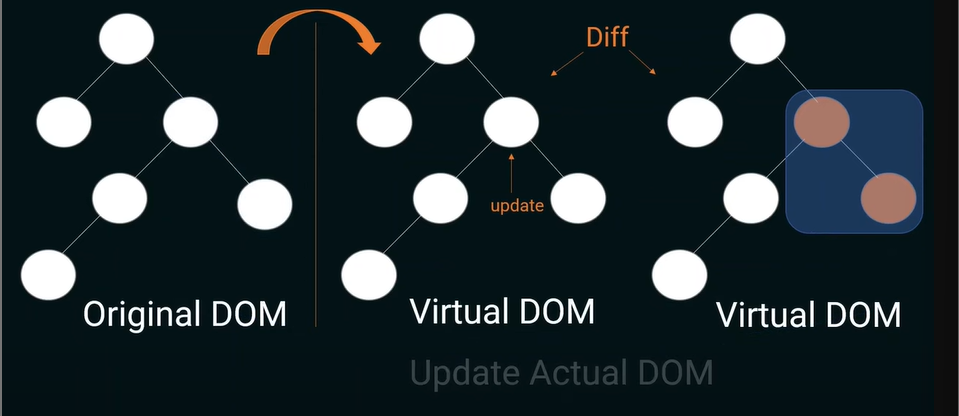
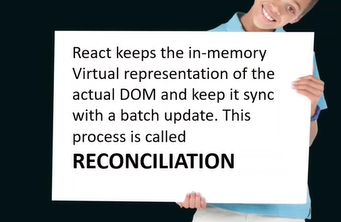
***React Knowledge***

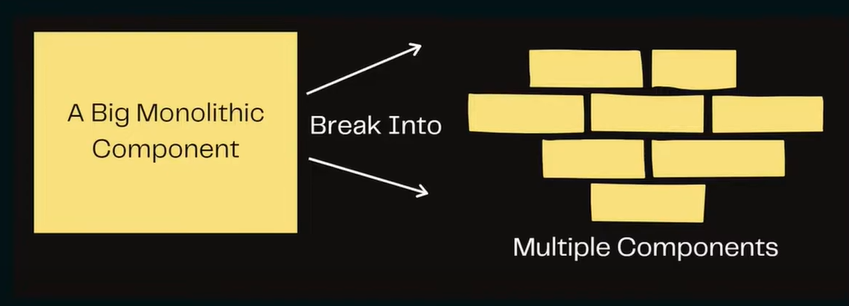
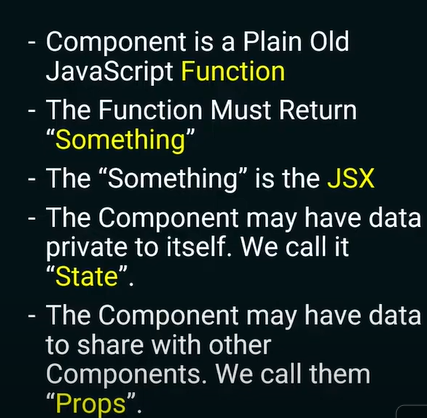
***ReactJS Virtual DOM - What are Virtual DOM, Reconciliation, Diffing, and Batch Update in React?***

* DOM – Document Object Model – HTML page with different html tags are static , we need ability to make it dynamic – DOM API is an API provided y JS to make changes in an HTML document
*  Tree Kind of Structure is used as an input to make changes in html
* DOM Manipulation – Consist of three things – Quering the DOM(find the DOM) , Updating DOM , Re-Render the DOM(Costly Operation) || If DOM update too frequently then our page can be little slower and some libraries come into place for this.
* React Never ever update the original DOM tree directly , react creates 1 object same as original DOM tree and that copy is known as virtual dom tree (Virtual DOM is nothing but in-memory copy of original DOM that react updates directly then updating orginal dom , because virtual dom is in memory and in memory operation is easy) and that virtual dom is easier to sync with original DOM.
* In React , Component gets mounted – operation gets started – component has JSX structure – that JSX is nothing but HTML like structure so DOM is created , So for that original DOM react creates Virtual DOM , each node represent element , if any node (element chamges) , react creates another Virtual DOM then react compares and update original DOM based on that.
* 
* 
* React performs a diffing algorithm to find different what change dom tree has.
*  That’s why we need key \*

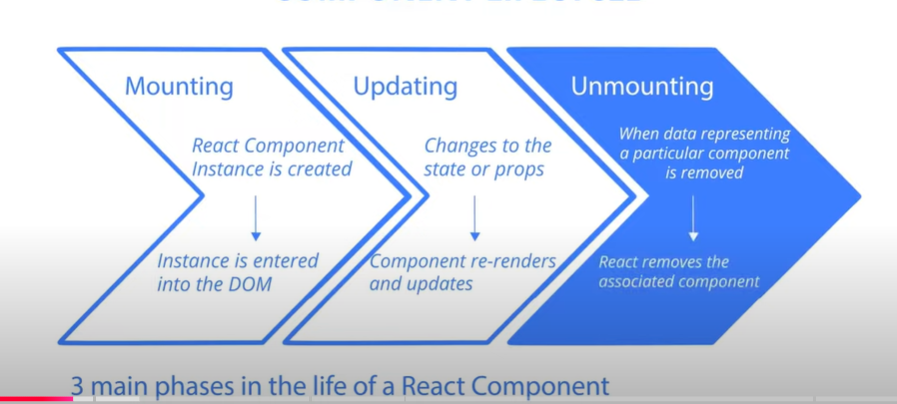
***Deep Understanding of JSX***

* JSX is a combination of Javascript and html JavaScript XML (Xtensible Markup language).JSX is a object. , JSX is something which browser understand.
* JSX is just a syntactic sugar in background react uses document.createelement to create that element
* Instead of JSX, we can simply return React.createElement(“h1”,null,”JSX is Awesome”) , this will become more complex as element grows
* Every JavaScript is we need to use in JSX we need to use expression (curly braces ex-{topic})
* Null, undefined, Boolean – are not printed on browser if we want to print it we need to convert it to string (String(true))
* Loops don’t work on jsx , use map instead as it transform data and return transformed data.
* React uses virtual dom – it only changes, updates a place which change in state and data , react only updates that. That’s why we use keys
* Everything evaluates as value is used in JSX

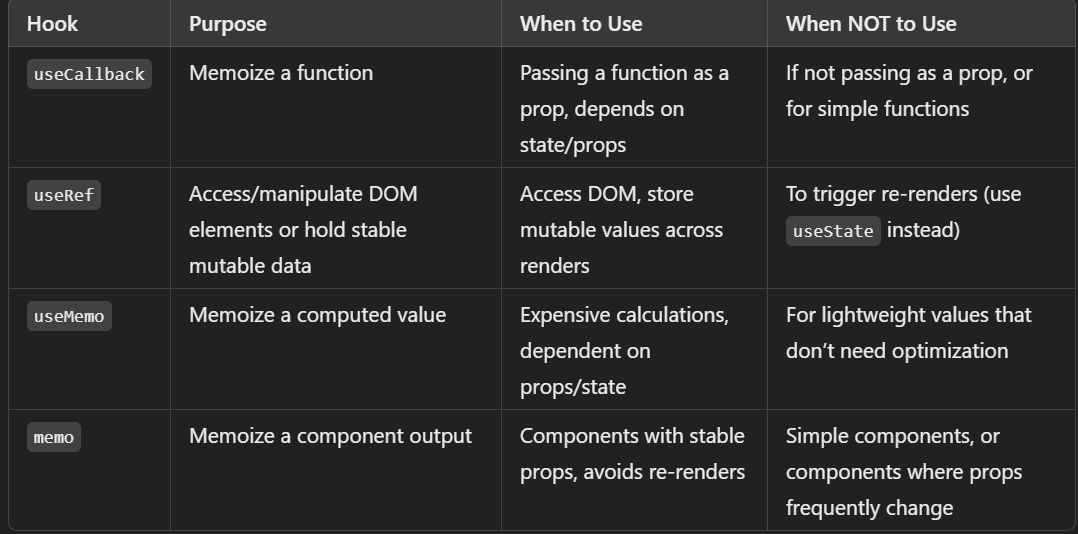
***Understanding React Components***

* 

***React Component Lifecycle*** –

* There are three main component in react lifecycle - mounting , updating , unmounting
* 
* In Class components –
  + Go through this video to learn for class component - <https://www.youtube.com/watch?v=zvM_FUVcB-0&t=1s>
* In functional Component –
  + Mounting updating unmounting with useeffect example

***Custom Hooks*** –

* First Understanding – Useful hooks
  + useCallback Hook –
    - useCallback returns a memoized version of a callback function, which only changes if the dependencies change.
    - It’s useful for passing callback functions as props to child components, especially when those callbacks depend on state or props.
    - **When to use useCallback:**
      * When you pass functions down to child components to avoid re-creating functions on every render, which can help prevent unnecessary re-renders in the child components
      * When you have a function that relies on certain state or props and want to ensure it only updates when those dependencies change.
    - **When NOT to use useCallback:**
      * If the function isn’t being passed as a prop to a child component or doesn’t cause expensive calculations, using useCallback might add unnecessary complexity without any performance gain.
      * If the function is simple, lightweight, or doesn’t need to be stable across renders, it’s better to declare it directly in the component.
    - **useCallback and Function Reference**
    - When you define a function directly within a component, React creates a new function reference every time the component re-renders, even if the logic inside the function hasn’t changed. This can be problematic, especially when that function is passed down as a prop to child components. Each new reference might cause child components to re-render unnecessarily, impacting performance.
    - useCallback helps optimize this by **preserving the same function reference across re-renders**—as long as the dependencies do not change. It only updates the function reference when one or more of its dependencies change, ensuring stability in function identity.
  + useRef Hook –
    - **What it does:**
      * useRef returns a mutable object that persists across renders. It’s often used to directly access or manipulate DOM elements, but it can also be used as an instance variable to store any mutable data.
      * Changing the current property of a ref doesn’t trigger a re-render.
    - **When to use useRef:**
      * When you need to directly access or manipulate a DOM element (e.g., focusing an input).
      * To store data that doesn’t need to trigger a re-render when it changes, such as a timer ID, previous state, or some cached data.
      * For holding stable values across renders, especially in event listeners or handlers.
  + Use Memo Hook –
    - **What it does:**
      * useMemo returns a memoized value, meaning it only recalculates the value if the dependencies change.
      * It’s used for optimizing expensive calculations to avoid re-running them on every render.
    - **When to use useMemo:**
      * When you have a heavy or computationally expensive function that doesn’t need to be recalculated every time the component renders.
      * When the value depends on certain dependencies, and you only want it to re-evaluate when those dependencies change.
  + memo –
    - **What it does:**
      * React.memo is a higher-order component that memoizes the rendered output of a functional component.
      * It prevents re-rendering of the component if its props haven’t changed, which can optimize performance by avoiding unnecessary updates.
      * memo is a **higher-order component (HOC)** that memoizes an entire **React component**.
      * It helps prevent re-rendering of the **whole component** if its **props haven’t changed**.
    - **When to use memo:**
      * When you have a child component that receives props from a parent and re-renders often without needing to, especially if the component has complex or expensive UI.
      * If the component doesn’t depend on many props or doesn’t frequently re-render due to prop changes, memo can reduce unnecessary re-renders.
* 
* List of custom hooks are there…

***React HOC pattern*** –

* HOC – Higher order components
* It receives a component, applies certain logic and then return that component with those additional logic
* HOC uses - we can use HOC when we want to apply same logic to multiple components
* Forward Ref – where , how to use – If parent passes ref to a child , then ref Kw cannot be passed , so we need to use forward ref to access ref from parent (for focus and all) , child component cannot get value of ref [we have to enclose child component into forward ref]
* 