Assignment of LHLAP

- Lists: -

1. How do you render a list of items in React? Why is it important to use keys when rendering lists?

How do you render a list of items: -

- **Using the map () Function:**
  - The map () function iterates over an array, and for each item, it returns a React element.
- **Assigning a Unique key Prop:**
  - Each item should have a unique key prop to help React efficiently manage updates to the DOM.

Why is it Important to Use Keys When Rendering Lists: -

- **Efficient Updates:**
  - React uses a **reconciliation algorithm** to determine which parts of the DOM need to be updated.
  - Keys help React identify which elements have changed, been added, or removed, improving performance.
- **Avoiding Unnecessary Re-renders:**
  - If React can uniquely identify an item in the list via its key, it avoids re-rendering elements that haven't changed.
- **Maintaining State:**
  - Keys help maintain the state of components across updates. For example, if an input field is part of a list, using a proper key ensures its value persists correctly.

2. What are keys in React, and what happens if you do not provide a unique key?

- **keys** are special string attributes used to uniquely identify elements in a list. They help React efficiently update and re-render components during state or prop changes.

- **Purpose: -**
  - Keys are used to track changes in a list of elements **(e.g., addition, deletion)** and enable React to determine which items have been modified.
- **What Happens if You Don't Provide a Unique Key?**

- If you don't provide a unique key, React will display a warning in the developer console: -
- Warning: Each child in a list should have a unique "key" prop.

- **Performance Issues**: -
  - React cannot efficiently update the DOM and may re-render unnecessary elements.
- **Loss of State**:
  - Components may lose their state after list updates if the key is not unique.

Hooks: -

1. **What are React Hooks? How do useState() and useEffect() hooks work in functional components?**
   - React hooks are special functions introduced in React V16.8.0 that allow you to hook into react features, such as state management and lifecycle methods in functional components. Before hooks, these features were only available in class components.
   - **How useState() works: -**
     - The useState hook allows to add state to functional components.
     - Const [state, setState] = useState(initialValue);
     - State – Holds the current state value.
     - SetState – A function to Update the state.
     - IntialValue – The initial value of the state.
   - **How useEffect Hook work: -**
     - The useEffect hook allows you to handle side effects in functional components.
   - **Common side effects include: -**
     - Fetching data from API.
     - Setting up subscriptions or timers.
     - Manipulating the DOM.
2. **What problems did hooks solve in react development? Why are hooks considered an important addition to react?**
   - **Problems Hooks solved in React: -**
     1. **Complex class components: -**
        - Managing state and lifecycle in class components often required complex and repetitive code.

2. **Code reusability: -**
   - Sharing logic between components was hard with classes. Hooks enable reusability through custom hooks.

3. **Long Lifecycle Methods: -**
   - Class components bundled unrelated logic in lifecycle methods. (e.g., componentDidMount). Hooks let you split logic into smaller, manageable pieces.
4. **This keyword issues: -**
   - Class component required binding methods to **this,** which could confuse beginners. Hooks eliminate this issue since functional components don't use this.

- **Why Hooks are important: -**
  1. **Simple code: -**
     - Functional components with hooks are easier to write and understand.
  2. **Improved reusability: -**
     - Hooks like custom hooks make it easy to share logic between components.
  3. **Modern development: -**
     - Hooks align with modern react pattern, making it easier to build and maintain application.
  4. **Lightweight Components: -**
     - Hooks remove the need for boilerplate code (like class declarations) making components smaller and faster.

3. What is useReducer? How we use in react app?

- UseReducer is a react hook used for managing complex state logic in functional components.
- **Syntax: -** const [state, Dispatch] = useReducer(reducer,initialvalue);
  - State – The Current State.
  - Dispatch – A function to trigger state updates.
  - Reducer – A function that determines how state updates happen.
  - InitialState – The initial state value.

4. What is the purpose of useCallback & useMemo hooks?

- **useCallback: -**
  - The usecallback hook is used to memorize functions. It ensures that a function reference remains the same between renders unless its dependencies change.
  - **Purpose of usecallback: -**
    - Prevent unnecessary re-creation of a function.
    - Useful when passing callbacks to child components.
  - **UseMemo: -**
    - The useMemo hooks is used to memorize expensive computations.
    - It ensures that a computed value is recalculated only when its dependencies change.

- **Purpose of useMemo: -**
    - Avoid re-computing heavy calculations during every render.

5. What is useRef? How to work in react app?

- useRef is a React hook used to create a mutable reference that persist across renders.
- It does not trigger a re-render when its changes.
- Syntax – const ref = useRef(initialvalue);

6. What's the difference between useCallback & useMemo Hooks?

- **UseCallback: -**
    - Memoizes Function to avoid re-creation.
    - Returns a memorized function.
    - When passing functions as props to child components.
    - Updates the function only when dependencies change.

- **UseMemo: -**
    - Memoizes computed values to avoid re-calculation.
    - Returns a memorized value.
    - For expensive calculation or derived state.
    - Recomputes the value only when dependencies change.