# Lists,Hooks,Localstorage

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

**Ans.** In React, we can render a list of items by using the map() method. Loop through the array of items. And then Return JSX for each item.

Keys are a special attribute used to identify elements in a list. React uses keys to keep track of which items have changed, been added, or removed, ensuring efficient updates. So its important for Efficient Rendering,Preserving State,Avoiding Warnings.

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

**Ans.** Keys are a special attribute in React that uniquely identifies elements in a list.

If we don’t provide unique keys then we face **Performance Issues,** **Incorrect Component Updates and**  **Console Warning.**

Hooks (useState, useEffect, useReducer, useMemo, useRef, useCallback)

**• Question 1:** What are React hooks? How do useState()and useEffect()hooks work in functional components?

**Ans.** React Hooks are special functions introduced in React 16.8 that allow developers to use state and other React features in functional components without converting them to class components.

**How useState() Works:**

The **useState hook** allows you to add state to a functional component.

**Syntax:**

const [state, setState] = useState(initialValue);

**What Are React Hooks?**

**React Hooks** are special functions introduced in React 16.8 that allow developers to use state and other React features in functional components without converting them to class components.

**Commonly Used Hooks:**

1. **useState:** Manages state in a functional component.
2. **useEffect:** Manages side effects such as data fetching, subscriptions, or DOM updates.
3. Other hooks: useContext, useReducer, useRef, etc.

**How useState() Works**

The **useState hook** allows you to add state to a functional component.

**Syntax:**

jsx

Copy code

const [state, setState] = useState(initialValue);

* **state:** Current state value.
* **setState:** Function to update the state.
* **initialValue:** Initial value of the state.

**Example:**

jsx

Copy code

import React, { useState } from 'react';

function Counter() {

const [count, setCount] = useState(0); // Initialize state to 0

return (

<div>

<p>Count: {count}</p>

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

}

export default Counter;

**How useEffect() Works:**

The **useEffect hook** allows you to handle side effects like data fetching, setting up subscriptions, or manually changing the DOM.

**Syntax:**

useEffect(() => {

// Side effect logic

return () => {

// Cleanup (optional)

};

}, [dependencies]);

**• Question 2:** What problems did hooks solve in React development? Why are hooks considered an important addition to React?

**Ans.** Before the introduction of React Hooks, managing state and lifecycle logic in functional components was challenging.

Hooks like useState and useEffect allow state management and side effects directly in functional components, making the code cleaner and more modular.

Hooks enable logic reuse through **custom hooks** (e.g., useFetch, useAuth), making it easy to share stateful logic without altering component hierarchies.

Hooks eliminate the need for classes entirely, allowing developers to use simple functional components without worrying about this.

Hooks allow splitting stateful logic into smaller, manageable pieces using multiple useEffect or custom hooks.

Hooks help separate logic from UI by abstracting stateful operations into custom hooks.

So for Simplified Code, Improved Reusability, Enhanced Developer Experience, and to Lower More Class-Component Limitations, and for better performance Hooks considered an important addition to React.

**Question 3:** What is useReducer ? How we use in react app?

**Ans.** The useReducer hook is a React hook that allows us to manage complex state logic in a functional component. It is an alternative to the useState hook.

**How to use:**

import React, { useReducer } from 'react';

// Reducer function

function reducer(state, action) {

  switch (action.type) {

    case 'increment':

      return { count: state.count + 1 };

    case 'decrement':

      return { count: state.count - 1 };

    case 'reset':

      return { count: 0 };

    default:

      throw new Error('Unknown action');

  }

}

function Counter() {

  const [state, dispatch] = useReducer(reducer, { count: 0 }); // Initial state is { count: 0 }

  return (

    <div>

      <p>Count: {state.count}</p>

      <button onClick={() => dispatch({ type: 'increment' })}>Increment</button>

      <button onClick={() => dispatch({ type: 'decrement' })}>Decrement</button>

      <button onClick={() => dispatch({ type: 'reset' })}>Reset</button>

    </div>

  );

}

export default Counter;

**Question 4:** What is the purpose of useCallback & useMemo Hooks?

**Ans.**

**purpose of useCallback:** Avoid unnecessary re-creation of functions on every render and Prevent child components from re-rendering unnecessarily when passing callback functions as props.

**purpose of useMemo:** Optimize expensive calculations by caching their results and Prevent unnecessary recalculations during renders.

**Question 5:** What’s the Difference between the useCallback & useMemo Hooks?

**Ans.** The key difference between useCallback and useMemo is:

useCallback:

* Optimizes functions.
* Prevents unnecessary re-creation of the same function instance.

useMemo:

* Optimizes values.
* Prevents unnecessary recomputation of expensive derived data.

**Question 6 :** What is useRef ? How to work in react app?

**Ans.** The useRef hook in React is used to create a mutable reference that persists across renders. It does not cause a component to re-render when its value changes.

It can be used for:

1. Accessing DOM elements directly (e.g., to focus an input field).
2. Storing mutable values (e.g., timers, previous state values, etc.) that don’t need to trigger re-renders.

# Context API

**Question 1:** What is the Context API in React? How is it used to manage global state across multiple components?

**Ans.** The Context API in React is a tool for managing global state that can be shared across multiple components without passing props through every level of the component tree. It helps eliminate the prop drilling problem, where props need to be passed through intermediate components unnecessarily.

**How is it Used to Manage Global State?**

1. **Create Context:** Use React.createContext() to create a context object.
2. **Provide Context:** Wrap the component tree with a Provider and pass the state or data as a value prop.
3. **Consume Context:** Use the useContext hook or Consumer to access the context's value in any component.

**Question 2:** Explain how createContext()and useContext()are used in React forsharing state.

**Ans.**

**createContext():-**

* createContext() is a function provided by React to create a context object.
* This object allows you to define global data that can be accessed by multiple components without passing props through every level.
* It provides two components:
  + Provider: Supplies the data to the component tree.
  + Consumer: (Optional) Accesses the shared data (less common now due to useContext).

**useContext():-**

* useContext() is a hook that allows components to consume the value provided by the context's Provider directly.
* It eliminates the need for the Consumer component and simplifies accessing context data.