**Lists:**

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

In React, you can render a list of items using the map() function to iterate through an array and return JSX for each element in the list.

**Steps to render a list:**

1. Use the map() function to loop over the array.
2. Return the appropriate JSX for each item in the list.
3. Assign a **unique key** prop to every element for better rendering performance.

The **key** prop is important because it allows React to identify which elements have changed, been added, or removed.

* **Efficient Rendering:** Keys help React avoid unnecessary re-rendering by tracking specific elements and optimizing updates.
* **Avoid Bugs:** Without unique keys, React may incorrectly reorder or reuse elements, causing unintended behaviors in the UI.

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

Keys are special attributes used to identify each element uniquely when rendering lists in React. They play an important role in efficiently updating the UI.

* React uses keys to track which list items have **changed**, **added**, or **removed**.
* This ensures React only updates the necessary elements instead of re-rendering the entire list.

**What happens if keys are not unique?**

* React may **confuse elements**, causing incorrect ordering or outdated states.
* Items may get duplicated or omitted during updates.
* Performance is reduced as React defaults to re-rendering the entire list.

**Forms in React:**

**Question 1: How do you handle forms in React? Explain the concept of controlled components.**

Forms in React are usually handled using **controlled components**. In controlled components, the form input values are managed by the React state. The value of the form is updated whenever the state changes.

**Controlled Components:**

* In a controlled component, the **input value** is controlled by React state.
* Changes to the input are handled using the onChange event, which updates the state dynamically.

This approach ensures React has complete control over the form data, making it easier to validate and manipulate inputs.

**Question 2: What is the difference between controlled and uncontrolled components in React?**

**Controlled Components:**

* The input's value is managed by React state.
* The onChange handler updates the state whenever the input changes.
* React has complete control over the form's behavior and values.

**Uncontrolled Components:**

* The input value is managed by the DOM, not React state.
* React does not track or control the value of the input directly.
* To access the value, **refs** are used to interact with the DOM manually.

In summary, controlled components give React full control, while uncontrolled components rely on the DOM for state handling.

**Hooks:**

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

**React Hooks** allow functional components to use features like state and lifecycle methods, which were previously available only in class components.

* useState is used to manage local component state.
* useEffect is used to handle side effects like fetching data, DOM manipulation, or subscriptions.

**useState() Hook:**  
It allows you to declare state in functional components. It returns an array with two values: the **current state** and a function to **update** the state.

**useEffect() Hook:**  
It allows you to perform side effects in functional components. You pass a callback function as the first argument and a dependency array as the second argument. The effect runs only when dependencies change.

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

React hooks solved several challenges faced during React development:

1. **State and Side Effects in Functional Components:**  
   Previously, only class components could handle state and lifecycle methods. Hooks like useState and useEffect enabled functional components to handle these features seamlessly.
2. **Code Reusability:**  
   Sharing logic was difficult with class components. Custom hooks now allow logic to be reused without the need for HOCs or render props.
3. **Complexity of Class Components:**  
   Class components required this binding and had confusing lifecycle methods. Hooks simplify these issues.
4. **Lifecycle Management:**  
   useEffect consolidates lifecycle methods, making setup and cleanup more organized and reducing bugs.
5. **Global State Management:**  
   Hooks like useReducer and useContext simplify managing global state without heavy libraries like Redux.
6. **Scalability Issues:**  
   Hooks encourage modular and reusable code, making applications easier to scale.

**Why are hooks important?**

* Hooks unify functional and class components.
* They reduce code complexity and improve readability.
* Developers can now write cleaner, more maintainable code.

**Question 3: What is useReducer? How is it used in a React app?**

useReducer is a hook used for state management in functional components, particularly when state logic is complex or depends on the previous state.

* It takes two arguments: a **reducer function** and an **initial state**.
* It returns the **current state** and a **dispatch function** to trigger updates.

useReducer is ideal when you need to handle multiple state transitions in a predictable way, similar to Redux.

**Question 4: What is the purpose of useCallback and useMemo hooks?**

**useCallback Hook:**

* Used to memoize a function.
* It ensures the function reference does not change unless its dependencies change.
* Useful when passing functions as props to child components to avoid unnecessary re-renders.

**useMemo Hook:**

* Used to memoize an **expensive value**.
* It prevents re-execution of costly calculations unless the dependencies change.

**Question 5: What is the difference between useCallback and useMemo hooks?**

* **useCallback:** Memoizes a function to ensure it does not get recreated on every render.
* **useMemo:** Memoizes a value to prevent unnecessary recalculations.

**Question 6: What is useRef? How does it work in a React app?**

useRef is a hook that allows you to:

1. **Store mutable values** without triggering re-renders.
   * Useful for persisting values across renders.
2. **Access DOM elements directly.**
   * It provides a reference to a DOM element, enabling direct manipulation.

useRef acts like a container that holds a value or DOM reference, which remains consistent across renders.