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

**Ans :**

import React from 'react';

function Item() {

const items = ['apple', 'banana', 'cherry'];

return (

<ul>

{items. Map((item, index) => (

<li key={index}>{item}</li>

))}

</ul>

);

}

export default Item;

**🡪Why is it important to use keys when rendering lists**

**Efficient Updates**

React uses a **virtual DOM** to compare the current state of the UI with the updated state. Keys enable React to quickly determine which items:

* Have changed.
* Have been added.
* Have been removed.

**Maintaining Component State**

When components are rendered in a list, they may have internal states (e.g., input fields, animations). Keys ensure that React correctly associates each component instance with its corresponding data, preserving its state even if the list order changes.

**Readability and Debugging**

Using keys makes the codebase easier to maintain and debug, as each element in a list has a clearly defined identity.

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

**Ans :**  keys is special attribute that helps React identify which items have changed, are added, or are removed from a list. Keys are used when rendering arrays of elements, and they should be unique among sibling elements.

**If you don't provide a unique key, React will:**

1. Use the index as a key: React will use the index of the element in the array as a key. This can lead to problems when the list is updated, as the index of elements can change.

2. Re-render the entire list: Without unique keys, React may re-render the entire list, which can lead to performance issues and unexpected behaviour.

3. Lose state: Without unique keys, React may lose the state of components, especially when the list is updated.

**Q-3 : What are React hooks? How do useState() and useEffect() hooks work in functional components?**

**Ans :** React Hooks are tools that allow you to use state and other React features without writing class components. They're designed to simplify your code and make it easier to share logic across components

**How Word usestat():**

1. We import use State from React.

2. We call use State with an initial value (in this case, 0).

3. use State returns an array with two values: count (the current state) and setCount (a function to update the state).

4. We use count to display the current value.

5. We use set Count to update the state when the button is clicked

Example:

import React, { use State } from 'react';

function Counter() {

const [count, set Count] = use State(0);

return (

<div>

<h1>Count: {count}</h1>

<button on Click={() => set Count(count + 1)}>Increment</button>

<button on Click={() => set Count(count - 1)}>Decrement</button>

<button on Click={() => set Count(0)}>Reset</button>

</div>

);

}

export default Counter;

**How word use Effect() Hook:**

1. The first argument is a function that contains the code to run on mount or update.

2. The second argument is an array of dependencies. When any of these dependencies change, the effect function will re-run.

Use Effect (()>{

Console.log(“Use effect Hook called.”)

},[Dependency])

**Q-4 What problems did hooks solve in React development? Why are hooks considered an important addition to React?**

**Ans:-**  Hooks solved several problems in React development, making them a crucial addition to the library.

Avoiding class complexity Managing state and life-cycle methods in class components can sometimes lead to complex and hard-to-understand code. Hooks provide a more straightforward way to manage these aspects without the need for classes

Important in react :-

Hooks are considered a significant addition to React because they allow developers to manage state and side effects within functional components, leading to cleaner, more readable, and reusable code, making it much easier to build complex React applications compared they provide a way to use features like state and lifecycle methods within function components, simplifying the development process and promoting better code organization.

**Q-5 : What is the purpose of use Callback & use Memo Hooks?**

**Ans :-**

**use** Callback

**Purpose**: Memorize a function to prevent unnecessary re-renders.

**Use** **case**: When you want to pass a function as a prop to a child component and prevent the child component from re-rendering unnecessarily.

**use Memo**

**Purpose**: Memorize a value to prevent unnecessary re-computations.

**Use** **case**: When you want to compute a value only when the dependencies change, and not on every render.

**use** **Callback** memorizes a function.

**use** **Memo** memorize a value.

**Q-6 What’s the Difference between the use Callback & use Memo Hooks?**

**Ans :-** use Callback caches functions, while use Memo caches values.

**use Callback**

Memorize a function

Returns a function

Prevents unnecessary re-renders by caching the function

**use Memo**

Memorize a value

Returns a value

Prevents unnecessary computations by caching the value

**Q-7 : What is use Ref ? How to work in react app?**

**Ans :-** use Ref is a hook in React that allows you to create a reference to a DOM node or a value that persists across renders. It's a way to "remember" a value or a node between renders.

**How Work in React :-**

const Useref = () => {

    let dm=useRef(null)

    function click(){

        dm.current.value=1000

        dm.current.focus()

        dm.current.style.color="red"

        dm.current.style.border="20px solid red"

    }

  return (

    <div>

      <input type="text" name="" id="" ref={dm} />

      <button on Click={click}>Click Me</button>

    </div>

  )

}