Module 4 (List And Hooks)

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

Answer 1 : To render a list of items in React, you typically use the .map() method to iterate through an array of data and create a corresponding React element for each item, assigning a unique "key" prop to each element to help React efficiently identify which items have changed, been added, or removed when the data updates; using keys is crucial for optimal performance and to avoid potential bugs in your application when manipulating list items.

Key points about using keys in React lists:

**Efficient updates:**

When data changes, React uses the "key" prop to compare elements in the old and new lists, allowing it to update only the necessary items instead of re-rendering the entire list, improving performance.

**Unique identifier:**

Each key should be a unique identifier within the list, often coming from a data source like an ID field.

**Potential issues without keys:**

If you don't use keys, React might incorrectly identify which items have changed, leading to unexpected behavior and potential rendering issues, especially when re-ordering list items.

Question 2: What is key sin React, and what happens if you do not provide a unique key?

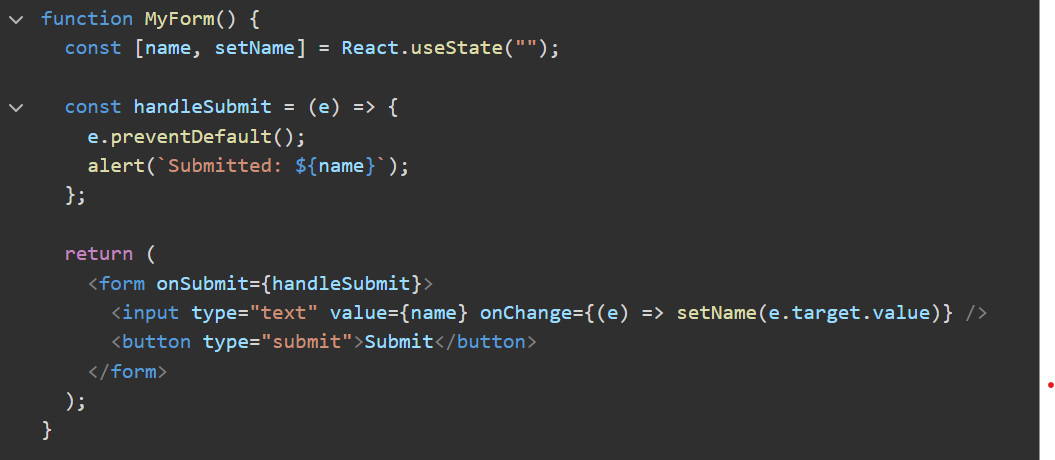
Answer 2: React uses the unique key value of each list item to keep track of it. If you neglect to add them, React uses the index value of each item as a default. Although this sounds like a logical solution, the React docs explain that this can lead to performance issues and confusing bugs.

(Forms in React)

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

Answer 1 : To handle forms in React, you typically use ****controlled components**** where the form elements' values are managed by React state.

* **Handling Forms in React:**
* 1️ ****Use State to Control Input Values****
* Store form values in React state using useState.
* 2️ ****Update State on Input Change****
* Use onChange to update state whenever the user types.
* 3️ ****Prevent Default Form Submission****
* Use e.preventDefault() to stop page reload on submit.
* 4️ ****Submit Handler for Processing Data****
* Handle form submission inside an onSubmit function.
* ****Controlled Components****:
* Definition:
* A form element (input,textarea,select) whose value is controlled by React state.
* Why Use:
* Single Source of Truth – React manages imput values.
* Validation & Formatting – Modify values before submitting.
* Better Control – Easily reset or update inputs dynamically.
* Example:



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

Answer 2: Difference **between controlled and uncontrolled components:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Controlled Component** | **Uncontrolled Component** |
| Value Storage | Stored in React state | Stored in DOM (default behavior) |
| Updates | Controlled by usestate and onChange | Directly modified by the user |
| Access | State variable (value prop) | useRef (ref.current.value) |
| Best For | Form validation, real-time updates | Simple forms, uncontrolled inputs |
| Example Input | <input value={state} onChange={handler} /> | <input ref={input Ref /> |

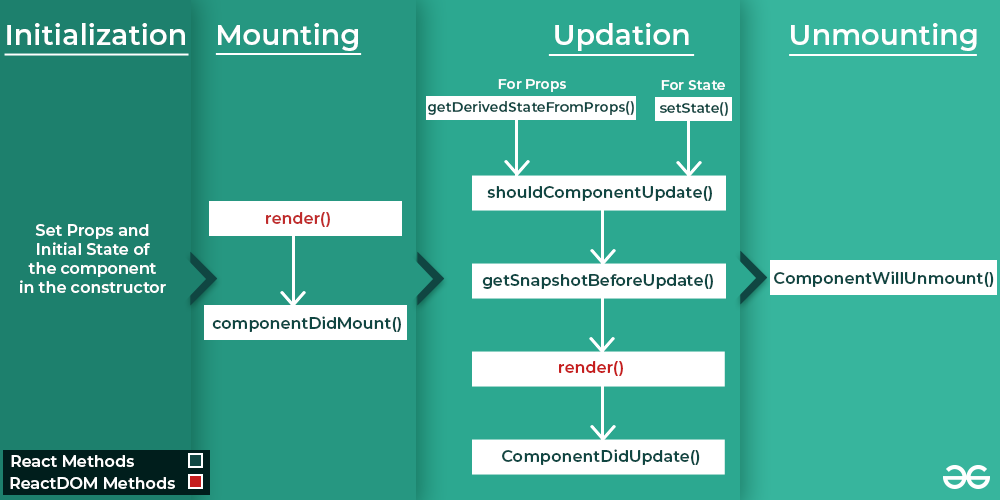
* **Which One to Use?**
* Use Controlled Components for forms with validation, dynamic behavior, or conditional rendering.
* Use Uncontrolled Components for simple forms where you only need the value on submit (e.g., file uploads).
* Controlled = React manages state
* Uncontrolled = Browser manages state

Lifecycle Methods(Class Components)

Question 1: What are lifecycle methods in React class components? Describe the phases of a component's lifecycle?

Answer 2: In React, a component’s lifecycle consists of three main phases:

* **Mounting** (Component is created and added to the DOM)
* **Updating** (Component is re-rendered due to changes in state or props)
* **Unmounting** (Component is removed from the DOM)



* **Initialization:** (**When the component is created)**
* These methods are called in the Set Props and Initial State of the component in the constructor:
* constructor()
* **Mounting:** **(When the component is added to the DOM)**
* These methods are called in the following order when an instance of a component is being inserted into the DOM:
* [**render()**](https://legacy.reactjs.org/docs/react-component.html#render)
* [componentDidMount()](https://legacy.reactjs.org/docs/react-component.html#componentdidmount)
* **Updating**: **(When the component’s state or props change, causing it to re-render)**
* An update can be caused by changes to props or state. These methods are called in the following order when a component is being re-rendered:
* [static getDerivedStateFromProps()](https://legacy.reactjs.org/docs/react-component.html#static-getderivedstatefromprops)
* [shouldComponentUpdate()](https://legacy.reactjs.org/docs/react-component.html#shouldcomponentupdate)
* [getSnapshotBeforeUpdate()](https://legacy.reactjs.org/docs/react-component.html#getsnapshotbeforeupdate)
* [**render()**](https://legacy.reactjs.org/docs/react-component.html#render)
* [componentDidUpdate()](https://legacy.reactjs.org/docs/react-component.html#componentdidupdate)
* **Unmounting:** **(When the component is removed)**
* This method is called when a component is being removed from the DOM:
* [componentWillUnmount()](https://legacy.reactjs.org/docs/react-component.html#componentwillunmount)

Question 2: Explain the purpose of componentDidMount(), componentDidUpdate(), and componentWillUnmount().

Answer 2: ****Explanation of Lifecycle Methods:****

****1. componentDidMount()****

* ****When it runs****: After the component is added to the DOM (rendered on the screen).
* ****Purpose****: To set up anything your component needs after it’s visible.
* ****Uses****:
  + Fetching data from an API.
  + Starting a timer.
  + Subscribing to events.

****2. componentDidUpdate()****

* ****When it runs****: After the component updates (re-renders) because of changes in its props or state.
* ****Purpose****: To perform actions based on updates.
* ****Uses****:
  + Making an API call when props/state change.
  + Updating the DOM after changes.
  + Reacting to user interactions.

****3. componentWillUnmount()****

* ****When it runs****: Right before the component is removed from the DOM.
* ****Purpose****: To clean up resources and prevent memory leaks.
* ****Uses****:
  + Clearing timers.
  + Unsubscribing from events.
  + Canceling network requests.

These methods are key for managing side effects in class components!

Hooks(useState, useEffect, use Reducer, useMemo, useRef, useCallback)

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

Answer 2:

* React ****hooks**** are tools that let functional components do more. They let you use features like ****state**** (data that changes) and ****lifecycle methods**** (actions at different times) without needing class components.
* ****Hooks Key Overview:****
* **useState**: For state management.
* **useEffect**: For handling side effects (e.g., data fetching, subscriptions).
* Others: use Context, use Reducer, useRef, etc.

### **useState()**

* Adds and manages data (state) in a component.
* The component updates when the data changes.
* Example: Track button clicks or toggle themes.

### **useEffect()**

* Handles tasks outside the component (e.g., fetching data or starting a timer).
* Runs on render or when specified values change.
* Can clean up tasks when the component is removed.
* ****In short****
* **useState()**: Manage data.
* **useEffect()**: Handle side tasks.

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

Answer 2:

* ****Problems Hooks Solved in React Development:****
* **State management in functional components**: Before hooks, only class components could have state. Hooks like useState allow functional components to manage state.
* **Reusability of logic**: Hooks like useEffect let you reuse logic between components without changing their structure.
* **Cleaner code**: They remove the need for lifecycle methods in class components, simplifying code.
* **Easier to understand**: Hooks make functional components more powerful, reducing complexity compared to class components.
* ****Why Hooks Are Important****:
* ****Simplify Development****: Hooks make functional components as powerful as class components, but with less complexity.
* ****Cleaner Code****: They reduce boilerplate code, making components easier to read and maintain.
* ****Better Reusability****: Custom hooks allow logic to be reused across components.
* ****Improved Side Effect Management****: useEffect() helps keep side effects clean and easy to manage in one place.

In short, hooks make React development simpler, more efficient, and easier to maintain!

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

Answer 4:

**Purpose of useCallback & useMemo Hooks:**

* useCallback:
* useCallback memorizes a function, so it doesn’t get recreated on every render.
* Useful when passing function as props to child components (prevents unnecessary re-renders).
* useMemo:
* useMemo memorizes a computed value, so React doesn’t recalculate it on every render.
* Useful for expensive calculations, filtering, or sorting large lists.

**Difference:**

|  |  |  |
| --- | --- | --- |
| **Hook** | **Optimizes** | **Use Case** |
| useCallback | Functions | Prevents function recreation |
| useMemo | Values/Calculations | Avoids unnecessary recalculations |

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

**Difference between the useCallback & useMemo:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **useCallback** | **useMemo** |
| What it does? | Memorizes a function | Memorizes a computed value |
| Returns | A function | A value(computed result) |
| Use case | Prevents unnecessary function recreation | Avoids expensive recalculations |
| Common usage | Passing stable functions to child components | Optimizing heavy calculations |

**Simple Explanation:**

* useCallback:
* Used when you want to memorize a function and avoid re-creating it on every render.
* useMemo:
* Used when you want to memorize a value so that React doesn’t recompute it unnecessarily.

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

Answer 6: useRef is a Hook provided by React, and it is commonly used for accessing and interacting with DOM elements. It can also be used for preserving values between renders without causing re-renders. In the code above, we create a my Ref using the useRef hook and assign it to the input element using the ref prop.