Introduction to React.js

Question 1: What is React.js? How is it different from other JavaScript frameworks and libraries?

Answer 1:

1. **React.js** is a JavaScript library for building user interfaces.
2. It uses a component-based structure for reusable code.
3. React updates the UI efficiently using a Virtual DOM.
4. JSX lets you write HTML-like code in JavaScript.

**Different from other JavaScript frameworks and libraries:**

* **React is a library**, not a full framework, meaning it only handles the user interface, while other tools handle the rest.
* It uses **components** to break the UI into smaller, reusable pieces of code.
* React updates the UI **efficiently** with a Virtual DOM, making it faster than regular JavaScript.
* It’s **flexible** and can be combined with other libraries or frameworks.

Question 2: Explain the core principles of React such as the virtual DOM and component based architecture.

Answer 2:

**Virtual DOM:**

* The Virtual DOM is a lightweight copy of the actual DOM which make DOM manipulation easier.

**Component based architecture:**

* Components are like function that return HTML element.

Question 3: What are the advantages of using React.js in web development?

Answer 3:

**The advantages of using React.js in web development:**

1. **Fast Performance**: React uses a Virtual DOM, which makes updates and rendering faster.
2. **Reusable Components**: You can create components that can be reused, saving time and effort.
3. **Easy to Learn**: React’s syntax, especially with JSX, is simple and intuitive for developers.
4. **Large Community**: React has a big community, so you get lots of support and resources.
5. **Scalability**: React makes it easier to scale up your application as it grows.
6. **Cross-Platform**: React Native lets you build mobile apps for iOS and Android using the same concepts.

JSX (JavaScript XML)

Question 1: What is JSX in React.js? Why is it used?

Answer 1:

* JSX means JavaScript XML (Extensible Markup Language)
* JSX stands for JavaScript XML.
* JSX allows us to Write HTML in React.
* JSX makes it easier to write and add HTML in React.
* JSX code is copied into JavaScript by React.

Question 2: How is JSX different from regular JavaScript? Can you write JavaScript inside JSX?

Answer 2:

* "JavaScript needs to be written inside the <script> tag in an HTML file, but JSX can't be written directly in HTML files. JSX is written in React, and React converts it into JavaScript. JSX code is written in a JavaScript file."
* "Yes, you can write JavaScript inside JSX. To do this, you use curly braces {} around JavaScript expressions, like {name} or {2 + 2}, which inserts dynamic values into JSX."

Question 3: Discuss the importance of using curly braces {} in JSX expressions.

Answer 3:

* In JSX, {} are used to include JavaScript expressions.
* They allow you to display variables, functions, or calculations.
* Example: <p>Hello, {name}! </p> will show dynamic content.
* Without {}, JSX treats content as plain text.
* Curly braces help mix logic and UI together.

Components (Functional & Class Components)

Question 1: What are components in React? Explain the difference between functional components and class components.

Answer 1:

* Components are independent and reusable bite of code.
* Components are reusable pieces of code that define a specific part of User Interface.

**Difference between Functional Components and Class Components:**

|  |  |
| --- | --- |
| **Function Components** | **Class Components** |
| Defined using a regular JavaScript Function. | Defined using ES6 Class. |
| Managed using the useState hook. | Managed using this.state. |
| Uses useEffect for lifecycle behavior. | Uses lifecycle methods like componentDidMount. |
| Can use hooks like useState, useEffect. | Not applicable. |

Question 2: How do you pass data to a component using props?

Answer 2:

* In the parent component, pass data as attributes to the child component:
  + <ChildComponent data="some data" />
* In the child component, access the data through props:
  + function ChildComponent(props) {
    - return <div>{props.data}</div>;

}

* The value of data is passed and used in the child component.

Question 3: What is the role of render() in class components?

Answer 3:

* The render() method in class components returns the JSX to be displayed on the screen.
* It's required in every class component and defines what the component should look like.

Props and State

Question 1: What are props in React.js? How are props different from state?

Answer 1:

* Props are inputs passed from a parent component to a child component.
* They allowing it to receive data and behave dynamically.
* They are read-only and cannot be modified by the child component.

**Props Different from State:**

* Props come from the parent component while state is managed inside the component.
* Props are read-only, but state can change over time.
* Props are used for passing data While state is used for handling dynamic changes.

Question 2: Explain the concept of state in React and how it is used to manage componentdata.

Answer 2:

* State in React is an object that holds data specific to a component. It allows components to store and manage information that can change over time. When the state updates, React re-renders the component to reflect the changes.

Question 3: Why is this.setState() used in class components, and how does it work?

Answer 3:

* this.setState() is used in class components to update the state. When called, it changes the component’s state and triggers a re-render, so the component updates with the new state. It ensures the UI stays in sync with the state changes.
* this.setState() updates the current state. When you call setState(), React merges the new state with the old state and triggers a re-render of the component to reflect the changes.

Handling Events in React

Question 1: How are events handled in React compared to vanilla JavaScript? Explain the concept of synthetic events.

Answer 1:

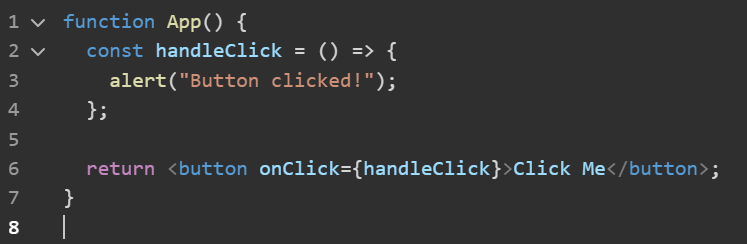
Question 2: What are some common event handlers in React.js? Provide examples of onClick, onChange, and onSubmit.

Answer 2:

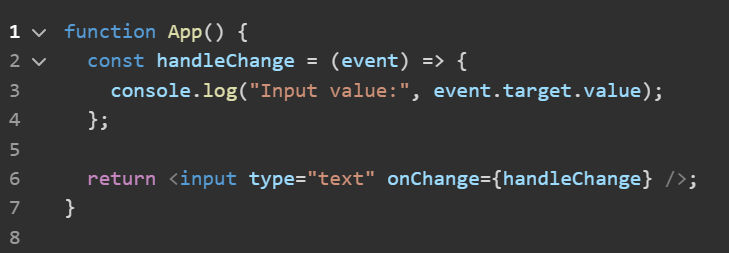
* **Common Event Handlers in React.js:**
* React provides various event handlers to handle user interactions. Some of the most common ones includes:

1. onClick -Happens when a user clicls on something(like a button).
2. onChange – Happens when the value in an input box changes.
3. onSubmit – Happens when a user submits a form.

* **Examples:**
* onClick(Button Click)



* onChange(Input Change)



* onSubmit(Form Submit)



Question 3: Why do you need to bind event handlers in class components?

Answer 3:

Conditional Rendering

Question 1: What is conditional rendering in React? How can you conditionally render elements in a React component?

Answer 1:

* **Conditional Rendering in React:**
* Conditinal rendering in React means showing or hiding elements based on certain conditions.

Forms in React

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

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**

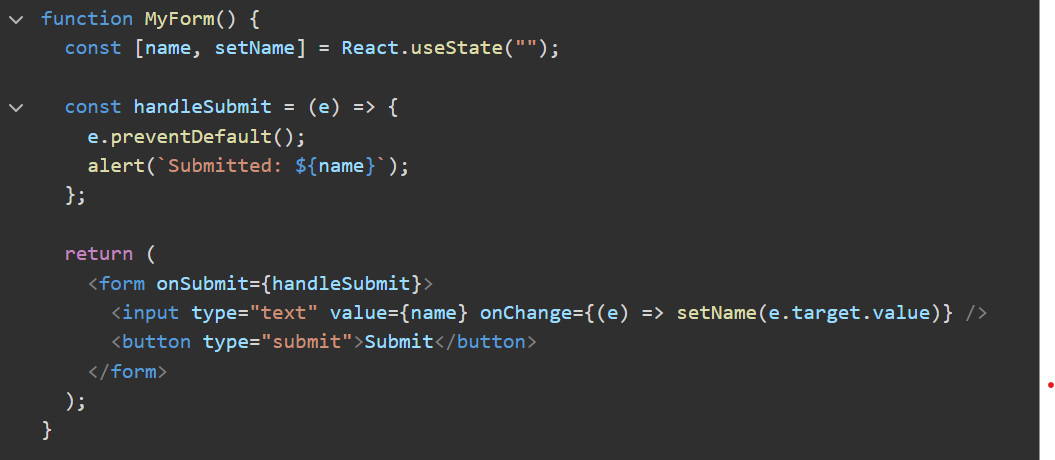
* Use e.preventDefault() to stop page reload on submit.

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 submittimg.
* 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={inputRef /> |

* **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.f., file uploads).
* Controlled = React manages state
* Uncontrolled = Browser manages state

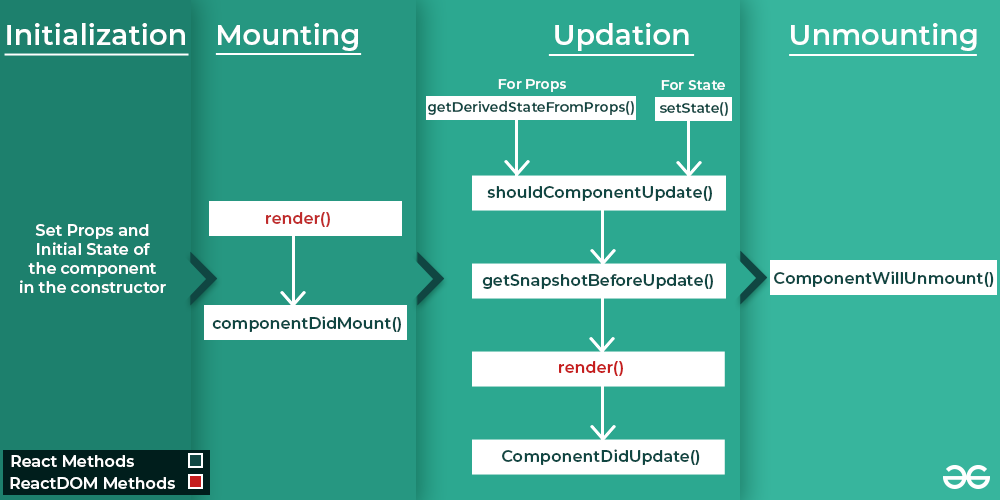
Lifecycle Methods (Class Components)

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

Answer 1:

* In React class components, **lifecycle methods** are special functions that let you run code at different stages of a component’s life. A component has three main phases in its lifecycle:

1. Mounting
2. Updatio
3. Unmounting



* **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)

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

Answer 1:

* 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: useContext, useReducer, 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 componens(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? | Memoizes a function | Memoizes 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:

* It can be used to store a mutable value that does not cause a re-render when updated.
* It can be used to access a DOM element directly.

**Work in react app:-**

1️ **Accessing DOM Elements** – Like focusing an input box directly without using state.  
2️ **Storing Previous Value** – Like tracking the previous value of a counter without re-rendering.  
3️ **Does Not Trigger Re-render** – Unlike useState, updating a useRef value does not cause the component to re-render.

**Or**

1️ **DOM Element ko Access Karna** – Jaise ek input box ko directly focus karna, bina state use kiye.  
2️ **Previous Value Store Karna** – Jaise ek counter ki purani value track karna, bina re-render kiye.  
3️ **Re-render Trigger Nahi Karta** – Agar aap useState use karein toh har update pe component re-render hota hai, par useRef me aisa nahi hota.