1. What is React.js? How is it different from other JavaScript frameworks and libraries?
Ans.
React.js is a JavaScript library used for building fast, interactive, and component-based user interfaces, mainly for single-page applications (SPAs).
It was created by Facebook (Meta).

2. Explain the core principles of React such as the virtual DOM and componentbased architecture.
Ans
React is built on a few powerful principles that make it fast, scalable, and easy to maintain:

What is the Virtual DOM?

The **Virtual DOM** is a lightweight copy of the real DOM that React keeps in memory.

3. What are the advantages of using React.js in web development?
Ans.
React offers several benefits that make it one of the most widely used libraries for building modern web applications.

4. What is JSX in React.js? Why is it used?
Ans.
JSX (JavaScript XML) is a **syntax extension** for JavaScript that allows you to write **HTML-like code inside JavaScript**

Why is JSX used?

✅ 1. Makes UI Code Easy to Write and Understand

5. How is JSX different from regular JavaScript? Can you write JavaScript insideJSX?
Ans
JSX is different from regular JavaScript because it allows HTML-like syntax inside JavaScript, requires compilation, and

follows stricter rules. Yes, you can write JavaScript inside JSX using curly braces `{}` — but only expressions, not statements.

6. Discuss the importance of using curly braces {} in JSX expressions.
Ans
Curly braces in JSX are used to embed JavaScript expressions inside UI markup. They allow React to display dynamic values, run functions, make calculations, apply conditions, and render lists. Essentially, curly braces connect JavaScript logic with JSX, making components dynamic and interactive.

7. What are components in React? Explain the difference between functional components and class components. ?
Ans

Components in React are reusable UI building blocks. React has two types of components: functional and class components. Functional components are simple JavaScript functions and use Hooks for state and lifecycle operations. Class components use ES6 classes, rely on `this`, and use lifecycle methods. Modern React recommends using functional components due to cleaner syntax, better performance, and full Hooks support.

8. How do you pass data to a component using props?
Ans
You pass data to a component using props by adding attributes to the component in the parent and accessing them through the `props` object in the child component. Props are used for one-way data flow and are read-only.

9. What is the role of render() in class components?
Ans
In React class components, the `render()` method is responsible for returning the UI (JSX) that should be displayed on the screen.

10. What are props in React.js? How are props different from state?

Ans
Props (short for “properties”) are used to pass data from a parent component to a child component.
. State is internal, changeable data that belongs to the component and can be updated using setState/useState. Unlike props, state is used to handle dynamic changes in the UI.

11.Explain the concept of state in React and how it is used to manage componentdata.
Ans
State in React is an object that stores dynamic, changeable data for a component. When the state updates, React automatically re-renders the component to reflect the new data on the screen. State is managed inside the component (unlike props) and is commonly used for handling user interactions, form inputs, toggles, counters, and UI changes.

12. Why is this.setState() used in class components, and how does it work?
Ans
Why `setState` is asynchronous
👉 `setState` vs direct state update examples
👉 How React batches multiple state updates

13.How are events handled in React compared to vanilla JavaScript? Explain the concept of synthetic events.
Ans
“React uses Synthetic Events—React’s wrapper around browser events—to provide cross-browser compatibility and performance optimization. Unlike vanilla JS, React uses CamelCase event names (onClick), passes functions instead of strings, and handles events using an internal event delegation system for better performance.”

14. What are some common event handlers in React.js? Provide examples of onClick, onChange, and onSubmit.
Ans
React provides several event handlers that are similar to DOM events but written in camelCase and accept functions instead of strings.

15.Why do you need to bind event handlers in class components?
Ans.
In React class components, event handlers must be bound to the class instance so that the keyword `this` correctly refers to the component.

16.What is conditional rendering in React? How can you conditionally render elements in a React component?
Ans

Conditional rendering means showing or hiding UI elements based on certain conditions, just like conditions in JavaScript (`if`, `else`, `&&`, `?:`, ternary operators, etc.).

React updates the UI dynamically depending on state, props, or logic.

Example:
✔ Show a login button if the user is not logged in
✔ Show a logout button if the user is logged in

17.Explain how if-else, ternary operators, and && (logical AND) are used in JSXfor conditional rendering.
Ans

In JSX, conditional rendering can be done using:

1. **`if-else` → Assign JSX to a variable outside JSX, then render it.**

2. **Ternary operator (`condition ? true : false`) → Inline conditional rendering.**

3. **Logical AND (`&&`) → Render JSX only if a condition is true.**

These methods let React dynamically display UI based on state, props, or other conditions."

18.How do you render a list of items in React? Why is it important to use keyswhen rendering lists?
Ans
“In React, lists are rendered using the `map()` method to generate JSX for each element. Keys are used to give each list element a unique identifier so that React can efficiently detect changes, additions, or deletions, improving performance and avoiding unnecessary re-renders.”

19.What are keys in React, and what happens if you do not provide a unique key?
Ans
“Keys in React are unique identifiers used when rendering lists to help React track which items have changed, been added, or removed. Without unique keys, React may re-render the entire list inefficiently and can cause bugs, especially when the list order changes or elements have internal state

20. How do you handle forms in React? Explain the concept of controlled components
Ans
In React, forms are usually handled using controlled components, where the form inputs’ values are tied to React state. Each change in the input updates the state via `onChange`, and the state value controls the input. This approach allows predictable, dynamic, and easily validated forms. Uncontrolled components can also be used but are less common.”

21.: What is the difference between controlled and uncontrolled components in React?
Ans
“Controlled components are form elements whose values are controlled by React state, allowing easy validation and predictable behavior. Uncontrolled components manage their own state internally in the DOM, and React accesses their values using refs. Controlled components are preferred in modern React for dynamic and complex forms.”

22. What are lifecycle methods in React class components? Describe the phases of a component's lifecycle
Ans
"Lifecycle methods in React class components are special methods called at different stages of a component's life: mounting, updating, and unmounting. They allow developers to initialize state, perform side effects like API calls, optimize performance, and clean up resources

23. Explain the purpose of componentDidMount(), componentDidUpdate(),and componentWillUnmount()
Ans
`componentDidMount()` runs once after a component is mounted for initialization and API calls. `componentDidUpdate()` runs after state or props change, useful to react to updates. `componentWillUnmount()` runs just before a component is removed, used for cleanup tasks like clearing timers, unsubscribing, or removing event listeners."

24. 1: What are React hooks? How do useState() and useEffect() hooks work in functional components?
React Hooks are special JavaScript functions that let you "hook into" React features like state and lifecycle methods from functional components.[1] They allow functional components to manage state and side effects, something previously reserved for class components.[2]

- **useState():**
  - **Function: Gives functional components the ability to manage local state.[3]**
  - **Mechanism: It returns an array containing the current state value and a setter function to update that value.[4] Calling the setter function triggers a re-render of the component with the new state.[5]**
- **useEffect():**

- Function: Performs side effects (like data fetching, subscriptions, or manually changing the DOM) after a component renders.[6]
- Mechanism: It runs the side-effect logic after every render by default.[7] You control when it runs using an optional dependency array ([]):
  - An empty array means the effect runs only once after the initial render and again on cleanup/unmount.
  - An array with values means the effect runs whenever those values change.

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

Hooks are a crucial addition because they solved three main architectural problems:

1. Reusing State Logic: Before hooks, reusing stateful logic required complex patterns like Higher-Order Components (HOCs) or Render Props, often leading to deeply nested code ("wrapper hell").[8] Hooks allow for easy, flat reuse via Custom Hooks.[9]
2. Complex Class Components: Logic related to a single feature (e.g., setting up a subscription and cleaning it up) was split across different lifecycle methods (componentDidMount, componentWillUnmount). Hooks, particularly useEffect(), allow grouping related logic together.
3. The Confusion of this: Hooks eliminate the need for class components, constructors, and the often-confusing context binding of the this keyword in JavaScript.[10]

Question 3: What is useReducer? How we use in a React app?

- What is it?

    - **useReducer is an alternative to useState for managing more complex state logic, especially when state transitions depend on the previous state or involve multiple related values.[11]**
- **How to use it?**
    - **You initialize it with a reducer function (which takes state and an action and returns the new state) and an initial state.**
    - **It returns the current state and a dispatch function.**
    - **You update the state by calling the dispatch function and passing it an action object. The action tells the reducer *what* happened, and the reducer determines *how* the state should change.[12]**

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

Both are performance optimization tools that use memoization (caching) to prevent unnecessary work in child components and expensive re-calculations.[13]

- **useCallback:**
    - **Purpose: Memoizes a function (a callback).[14] It returns the same function instance across renders unless one of its dependencies changes.**
    - **Benefit: Prevents unnecessary re-renders in optimized child components that receive the function as a prop, as the function's reference remains stable.**
- **useMemo:**
    - **Purpose: Memoizes a value (the result of a function call).[15] It returns the cached value unless one of its dependencies changes, only recalculating when necessary.**
    - **Benefit: Avoids re-running expensive calculations or operations on every render.**

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

The difference is simple: what they return.

Hook	What it Returns	Primary Goal
useCallback	The memoized function itself.	To optimize referential equality for functions passed as props.
useMemo	The memoized value calculated by the function.	To optimize expensive calculations by skipping re-execution.

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

- **What is it?**
  - **useRef returns a mutable ref object with a single property: .current.**[16]
  - **The value of .current persists throughout the component's lifetime, and crucially, changing it does not trigger a re-render.**
- **How to use it?**
  - **1. Direct DOM Access: It is primarily used to get a direct reference to a DOM element (e.g., to focus an input or measure an element's height) by passing the ref object to the element's ref attribute.**[17]
  - **2. Storing Mutable Data: It can store any mutable value (like a timer ID, a subscription, or a counter) that needs to persist between renders but whose changes shouldn't cause the component to update.**

1: What is React Router? How does it handle routing in single-page applications?

React Router is the standard library used for declarative routing in React applications.

It is essential for Single-Page Applications (SPAs) because SPAs only load a single HTML page (`index.html`). Traditional routing (where the browser requests a new HTML file from the server for every link click) doesn't happen.

How it Handles Routing:

1. **URL Sync: React Router uses the browser's History API to change the URL shown in the address bar without triggering a full page reload.**
2. **Component Mapping: It looks at the current URL and, based on its configuration, determines which React component should be rendered.**
3. **Conditional Rendering: Instead of loading a new page, React Router simply conditionally renders the correct component and hides the components that don't match the current URL path. This makes navigation instant, providing a seamless user experience.**

Question 2: Explain the difference between `BrowserRouter`, `Route`, `Link`, and `Switch` components in React Router.

These are the foundational components of React Router:

- **`BrowserRouter` (or Router):**
  - **Purpose: This is the main container for your entire routing setup. It uses the HTML5 History API (`pushState`, `replaceState`, etc.) to keep your UI in sync with the URL.**
  - **Usage: It must wrap all other router components (`Route`, `Link`, etc.) in the application.**
- **`Route`:**

    - **Purpose: It defines the mapping between a specific URL path and the component that should be rendered when that path is active.**
    - **Usage: You specify the path using the `path` prop and the component using the `element` (or `component`) prop.**
- **`Link`:**
    - **Purpose: Used to create navigation links within the application.**
    - **Usage: It's a declarative, accessibility-friendly replacement for the standard HTML anchor tag (`<a>`). Critically, clicking a `Link` prevents the default browser action (a full page reload) and uses the Router's internal mechanism to update the URL and conditionally render the new component.**
- **`Switch` (Less common in modern React Router, but historically important):**
    - **Purpose: Used to group multiple `<Route>` components together.**
    - **Mechanism (Historically): It ensures that only the first matching `<Route>` component is rendered. Without it, multiple routes that partially match the URL could render simultaneously.**
    - **(Note for Modern React Router v6): The `<Switch>` component has been replaced by the `<Routes>` component, which handles this single-match behavior by default.**

What is Redux, and why is it used in React applications? Explain the core concepts of actions, reducers, and the store.

Redux is a predictable state container for JavaScript applications.

Why it's used in React:

In complex React applications, state often needs to be shared across many components that are widely separated in the component tree (e.g., a user's login status or a shopping cart). This often leads to prop drilling (passing props down through many

layers of unrelated components). Redux solves this by providing a single, central place for the entire application state, making state changes predictable, traceable, and easier to debug.

Core Concepts of Redux

1. Store:
    - This is the single source of truth for the application's state. It holds the entire state tree.
    - The Store is immutable; you can't change the state directly. The only way to update it is by dispatching an action.
2. Action:
    - A plain JavaScript object that describes what happened. It is the only way to send data from your application to the Store.
    - It must have a type property (e.g., 'ADD_TODO', 'USER_LOGOUT').
3. Reducer:
    - A pure function that takes the current state and an action and returns the new state.
    - It is the only thing that actually changes the state in the Store. It must never mutate the original state; it must return a brand-new state object.
    - Conceptually:
      $$(currentState, action) \to newState$$

⚛️ Recoil: Atomic State Management

Question 2: How does Recoil simplify state management in React compared to Redux?

Recoil is a state management library built specifically for React by Facebook (Meta). It simplifies state management by adopting a more React-centric approach based on *atoms* and *selectors*.

Recoil offers key simplifications over Redux:

- 1. Boilerplate Reduction: Redux requires significant boilerplate (actions, action creators, reducers, thunks, etc.)

even for small state changes. Recoil allows you to define a piece of state with just an atom, which is much faster to set up.

- **2. Distributed State: Unlike Redux's single global Store, Recoil manages state via atoms. An atom is a single unit of state that any component can subscribe to. This makes the state distributed, more flexible, and easier to scale locally without impacting unrelated parts of the app.**
- **3. Familiar React API: Recoil uses a hook-based API (e.g., useRecoilState, useSetRecoilState) that feels much more natural and integrated with modern React functional components, making the learning curve shallower than Redux's connect-based pattern.**
- **4. Concurrent Mode Ready: Recoil was designed to be compatible with React's modern features like Concurrent Mode, ensuring better performance and non-blocking rendering.**