**React I**

**1. What is React?**

React is a JavaScript library developed by Facebook for building user interfaces, especially single-page applications (SPAs).

**2. What is the Role of React in Software Development?**

Role in Software Development:  
- Helps build interactive, dynamic, and fast UIs.  
- Promotes component-based architecture for modular and reusable code.  
- Efficiently updates and renders using virtual DOM.

**2. What are the Key Features of React?**

- Component-Based Architecture  
- Virtual DOM  
- JSX (JavaScript XML)  
- Unidirectional Data Flow  
- Reusable Components  
- Strong Community Support

**3. What is DOM? What is the Difference Between HTML and DOM?**

DOM (Document Object Model) is a programming interface for HTML and XML documents.  
  
HTML vs DOM:  
- HTML: Markup language used to create structure (Static)  
- DOM: In-memory representation of HTML (Dynamic, manipulated by JS)

**4. What is Virtual DOM? Difference Between DOM and Virtual DOM?**

Virtual DOM is a lightweight copy of the real DOM kept in memory.  
React uses VDOM to efficiently compare and update changes.  
  
DOM vs Virtual DOM:  
- DOM: Slower updates, causes reflows/repaints.  
- Virtual DOM: Faster, updates only the changed parts.

**5. What are React Components? What are the Main Elements of It?**

React components are the building blocks of a React app.  
  
Types:  
- Functional Components (with hooks)  
- Class Components  
  
Main Elements:  
- Props  
- State  
- Lifecycle Methods / Hooks  
- Render method / JSX

**6. What is the Difference Between React and ReactDOM?**

React vs ReactDOM:  
- React: Core library for component logic.  
- ReactDOM: Handles rendering to the browser DOM.

**7. What is SPA (Single Page Application)?**

SPA loads a single HTML page and dynamically updates content without refreshing.  
Benefits:  
- Faster navigation  
- Seamless UX  
- Less server load

**8. What are the 5 Advantages of React?**

1. Fast Rendering with Virtual DOM  
2. Reusable Components  
3. Strong Ecosystem  
4. Declarative Syntax with JSX  
5. Great for SPAs

**9. What are the Disadvantages of React?**

1. JSX Learning Curve  
2. Fast Pace of Change  
3. Requires Build Tools  
4. SEO Challenges  
5. Heavy Reliance on JavaScript

**10. What is the Role of JSX in React?**

JSX allows writing HTML inside JavaScript.  
Simplifies UI creation.  
  
Example:  
const element = <h1>Hello, world!</h1>;

**11. What is the Difference Between Declarative and Imperative?**

Declarative vs Imperative:  
  
Declarative:  
- Focuses on what to do  
- React is declarative  
  
Imperative:  
- Focuses on how to do it  
- Vanilla JS or jQuery is imperative  
  
Example (React - Declarative):  
<button onClick={handleClick}>Click Me</button>  
  
Example (Imperative):  
const btn = document.createElement('button');  
btn.innerText = 'Click Me';  
btn.addEventListener('click', handleClick);  
document.body.appendChild(btn);

**React II**

**What is Arrow Function Expression in JSX?**

Arrow function in JSX is a shorthand way to define functions.  
Example: const MyComponent = () => <h1>Hello</h1>;  
It's often used for defining functional components or inline functions.

**How to Setup React first project?**

1. Install Node.js  
2. Run `npx create-react-app my-app`  
3. Navigate: `cd my-app`  
4. Run: `npm start`  
This starts the development server and opens the React app in your browser.

**What are the Main Files in a React project?**

- public/index.html: HTML template with a root div  
- src/index.js or main.jsx: Entry point of React  
- src/App.js: Root component  
- package.json: Project metadata and dependencies

**Whether React is a Framework or a Library? What is the difference?**

React is a library focused on building UI.

**How React provides Reusability and Composition?**

- React allows creating reusable components (e.g., Button, Card)  
- Components can be composed inside each other to build complex UIs  
- Encourages DRY (Don't Repeat Yourself) principles

**What are props in JSX?**

Props (short for properties) are used to pass data from parent to child components.  
They are read-only and help make components dynamic.  
Example:  
<Greeting name='Neel' /> — here, 'name' is a prop passed to Greeting.

**Files & Folder**

**What is NPM?**

NPM (Node Package Manager) is the default package manager for Node.js. It allows you to:  
- Install libraries/packages (like React, Axios, Express, etc.)  
- Manage dependencies for a project using package.json  
- Run scripts (like npm start, npm run build, etc.)

**What is the role of the node\_modules folder?**

The node\_modules folder:  
- Stores all the dependencies and sub-dependencies listed in package.json  
- Gets created when you run npm install  
- Should not be manually edited  
- Is usually ignored in version control (.gitignore)

**What is the role of the public folder in React?**

The public folder:  
- Contains static files like index.html, images, icons, etc.  
- Files in this folder are served as-is without any processing  
- Accessible using relative paths like /logo.png  
- Should not contain JavaScript or React components

**What is the role of the src folder in React?**

The src (source) folder:  
- Contains all the React components, CSS, JS/TS files, and logic  
- This is where you write your actual app code  
- Gets compiled and bundled by the build tools like Vite or Webpack

**What is the role of index.html page in React?**

The public/index.html file:  
- Acts as the single HTML page in a Single Page Application (SPA)  
- Contains a <div id='root'></div> element  
- React uses this as a root mount point to render the app using JavaScript

**What is the role of App.js file in React?**

The App.js file:  
- Defines the main root component of your application  
- This component is usually rendered inside index.js or main.jsx  
- You structure your UI and logic here or import other components into it

**What is the role of function and return inside App.js?**

In App.js:  
- The function defines the component  
- The return provides JSX output which is rendered on the page  
- Example:  
 function App() {  
 return <h1>Hello React</h1>;  
 }

**Can we have a function and return inside App.js?**

Yes, this is the standard way to define functional components in React.

**What is the role of export default inside App.js?**

export default App; makes the App component available for import in other files.  
- Example usage: import App from './App';

**Does the file name and the component name must be same in React?**

No, it's not required, but it's a best practice:  
- Good: App.js contains function App()  
- Bad: App.js contains function MyComponent()  
This improves readability and consistency.

**JSX**

**1. What is the role of JSX in React?**

JSX (JavaScript XML) is a syntax extension for JavaScript used in React to describe UI elements. It allows writing HTML-like code inside JavaScript, which gets compiled into React.createElement calls.

**2. What are the 5 Advantages of JSX?**

1. Readable Syntax: JSX looks like HTML.  
2. Better Error Messages: Babel provides helpful errors.  
3. Boosts Productivity: Combines JS and UI logic.  
4. Powerful Tooling: Works well with IDEs.  
5. Component-Friendly: Easier props passing and nesting.

**3. What is Babel?**

Babel is a JavaScript transpiler that converts modern JavaScript (including JSX) into browser-compatible JavaScript. It transforms JSX into React.createElement calls and allows use of ES6+ features.

**4. What is the role of Fragment in JSX?**

Fragments let you return multiple elements from a component without adding extra nodes to the DOM.  
Syntax:  
<Fragment>...</Fragment> or shorthand <>...</>

**5. What is the Spread Operator in JSX?**

The spread operator (...) allows passing all props from an object into a component. Example:  
<Component {...props} />

**6. What are the types of Conditional Rendering in JSX?**

1. if-else statement  
2. Ternary Operator  
3. Logical AND (&&)  
4. IIFE (Immediately Invoked Function Expression)

**7. How do you iterate over a list in JSX? What is map() method?**

The map() method is used to iterate over arrays and render UI elements. Example:  
{items.map((item, index) => (<li key={index}>{item}</li>))}

**8. Can a browser read a JSX File?**

No, browsers cannot directly read JSX. JSX must be transpiled (usually by Babel) into regular JavaScript.

**9. What is Transpiler? What is the difference between Compiler and Transpiler?**

Transpiler: Converts code from one version to another of the same language (e.g., JSX/ES6 to ES5).  
Compiler: Converts code from one language to another (e.g., TypeScript to JS).  
Difference: Compiler outputs a different language; transpiler outputs the same language.

**10. Is it possible to use JSX without React?**

Yes, but it's impractical. JSX is mainly for React. To use it without React, you must provide your own function to handle JSX (e.g., a custom createElement).

**Routing**

**1.What is Routing and Router in React?**

Routing in React is the process of navigating between different components (pages) based on the URL.

Think of it like how websites change the content when you go to /home, /about, etc., without reloading the page.

Router (from react-router-dom package) is the component that enables routing. It keeps the UI in sync with the browser URL.

Most common Router: BrowserRouter (used for web apps using HTML5 history API)

**2.How to implement Routing in React?**

Routing in React is the process of navigating between different components (pages)

Install React Router:

Wrap your App with BrowserRouter:

Use <Routes> and <Route> in your App:

**3.What are the roles of <Routes> & <Route> component in React Routing?**

<Routes>: It acts as a container for all the route definitions. It looks through all <Route> inside and renders the first match.

<Route>: It maps a path to a component. When the URL matches the path, the component in element is rendered.

**4.What are the Route Parameters in React Routing?**

Route parameters allow you to pass dynamic values via the URL.

**5.What is the role of Switch component in React Routing?**

<Switch> was used in React Router v5 to render the first matching route.

It has been replaced by <Routes> in React Router v6.

So, do not use Switch in v6.

**6.What is the role of exact prop in React Routing?**

Used in React Router v5 to prevent partial matching.

Without exact, / would match every route that starts with /, like /about.

In React Router v6, exact is no longer needed, because matching is exact by default.

**Hooks useState/UseEffect**

1. **What are React Hooks?**  
   React Hooks are functions introduced in React 16.8 that let you use state and other React features (like lifecycle methods) inside functional components.  
   Before Hooks, only class components could use state and lifecycle methods. Hooks bring this power to functional components.
2. **What are the Top React Hooks?**useState() – for adding state in functional components  
   useEffect() – for side effects (API calls, timers, DOM updates)  
   useContext() – for accessing context values (like global state)  
   useRef() – for referencing DOM elements or persisting values across renders  
   useReducer() – for complex state management (alternative to useState)  
   useMemo() – for performance optimization (memoizing expensive values)  
   useCallback() – for memoizing functions
3. **What are State.stateless,Stateful and State Management terms?  
   State**State is a data container that determines how a component behaves or renders.  
   It’s mutable (can change over time) and triggers re-renders when updated.  
   **Stateless Components**Do not hold state  
   Pure functions that take props and return JSX  
   **Stateful Components**Hold and manage internal state  
   Can be either class components or functional components with hooks  
   **Stateful Components**  
   Strategy or tools to manage and share state across components.  
   Can be:  
   Local state (using useState)  
   Global state (using useContext, Redux, Zustand, etc.)
4. **What is the role of useState() hook and how it works?**useState is a Hook that lets you add state to a functional component.const [count, setCount] = useState(0)  
   count is the current state value.  
   setCount is the function to update it.  
   On update, React re-renders the component with the new state.
5. **what is Dependancy Array in useEffect() hook?**useEffect() lets you perform side effects in your components. These can include:  
   API calls  
   Subscribing/unsubscribing  
   Setting up timers  
   Direct DOM manipulation
6. **what is the role of useEffect(). How it works and what is its use?**The second argument to useEffect() is called the dependency array. It tells React when to re-run the effect:

useEffect(() => {

// runs when 'count' changes

}, [count]);

If you leave out the array, the effect runs on every render.

If you pass an empty array ([]), it runs only once (like componentDidMount).

If you list variables, the effect re-runs only when any of those variables change.

**7.What does an Empty Dependency Array ([]) mean?**

useEffect(() => {

console.log('Runs only once on mount');

}, []);

It means:

The effect runs once after the first render (component mounts).

It will not run again, even if the component re-renders later.

Used for:

Fetching data on component load

Setting up subscriptions or event listeners

**Hooks useContext/useReducer**

1. **What is the role of useContext() hook?**The useContext() hook in React allows a functional component to access the values stored in a context created by createContext(), without having to pass props manually at every level (known as prop drilling).   
   const value = useContext(MyContext)
2. **what is createcontext() method ? What are Provider & consumet Properties?**createContext() is used to create a Context object. It provides two main components:  
   Provider: Supplies the context value.  
   Consumer: Reads the context value (though useContext is preferred instead of Consumer in functional components).  
   const MyContext = React.createContext()
3. **When to use useContext() hook instead of props in real application?**You should use the useContext() hook when you need to share data across multiple components without prop drilling — especially when that data is needed deeply in the component tree.
4. **what are the similarities between useSate() and useReducer() hook?**Both useState and useReducer are React hooks used for managing state in functional components, and they share several core similarities:Purpose: Manage Local State  
   State Persistence  
   Trigger Component Re-render   
    Functional Updates Supported  
    Can Store Any Type of Data
5. **what is useReducer() hook ?**The useReducer() hook is a React hook used for managing complex state logic in functional components. It is an alternative to useState(), especially useful when:  
   State logic is dependent on previous state  
   The component state is an object or nested structure  
   There are multiple sub-values or actions that update the state in different ways
6. **when to use useState()?**Depends on how complex your component's state and logic are.   
   **Simple State**You only need to manage primitive values like number, string, boolean.  
   **Independent State Variables**Each state value is separate (e.g. isLoading, count, inputValue)  
   **Quick Prototyping**useState is easier and cleaner for small components.
7. **when to use useReducer()?**Depends on how complex your component's state and logic are.   
   **Complex State Logic**You have multiple related state variables or need switch-like decision making.  
   **Dependent on Previous State**Next state relies on previous state values.  
   **Many Actions**You have different ways to update state (e.g., add, remove, reset).  
   **Better State Organization**Helpful for grouping logic in one reducer function.  
   **Reusable Logic or Redux Migration**Good when the logic might grow or move into Redux later.
8. **what are the difference between useState and useReducer() Hook?  
   State Complexity**Best for Simple, single state values (string, number, boolean)  
   Best for Complex or grouped state (objects, arrays)  
   Example:-  
   const [count, setCount] = useState(0)   
   const [state, dispatch] = useReducer(reducer, { count: 0 })  
   **State Update Logic**Logic Direct and simple update   
   Logic Centralized and condition-based (via reducer function)  
   Example:-  
   setCount(count + 1)   
   dispatch({ type: 'INCREMENT' })  
   **Readability & Structure**Readability Easier for beginners and small components  
   Readability Better for organizing complex update logic  
   **Multiple Related States**  
   Handling Requires multiple useState() calls  
   Handling Can group all in a single state object  
   Example:-  
   useState() for name, email, age   
   Single reducer managing { name, email, age }  
   **Dispatching Actions**Update Method setState(newValue) Update dispatch({ type: 'ACTION', payload })Example:-  
   setUser("Neel")  
   dispatch({ type: 'SET\_USER', payload: "Neel" })
9. **What are dispatch and reducer function in usereducer Hook?**dispatch(action): Sends an action object to the reducer function.  
   reducer(state, action): A pure function that takes the current state and action, and returns the new state.
10. **What is the purpose of passing initial state as an object in UseReduce?**

Passing an object allows you to:

Group related pieces of state (e.g., { count: 0, loading: false })

Manage complex state transitions easily.

Avoid multiple useState calls.

Make code more scalable and closer to Redux-style logic.

**Components LifeCycle Method**

**What are component life cycle phase?**

React class components go through three main phases in their lifecycle:

**Mounting** – When the component is being created and inserted into the DOM.

**Updating** – When the component is being re-rendered due to changes in props or state.

**Unmounting** – When the component is removed from the DOM.

**What are Component life cycle method?**

Lifecycle methods are special class methods that get called at specific points in a component's life:

**Mounting Phase:**

constructor()

static getDerivedStateFromProps()

render()

componentDidMount()

**Updating Phase:**

static getDerivedStateFromProps()

shouldComponentUpdate()

render()

getSnapshotBeforeUpdate()

componentDidUpdate()

**what are Constructor in class components ? when to use them?**

constructor() is a special method used in class components:

Called once when the component is being created.

Used to initialize state and bind methods.

**Use it only if you need to:**

Set the initial state.

Bind methods (though arrow functions are preferred now).

**what is the role of super keyword in constructor?**

The super(props) call is required to access this.props inside the constructor.

It calls the constructor of the parent class (React.Component).

Without super(), this won't work in the constructor.

render() {

return <h1>Hello, {this.props.name}</h1>;

}

**what is the role of render() method in component life cycle?**

The render() method is mandatory in class components.

It returns JSX that defines what to show on the screen.

Called every time the component is rendered (initial + re-renders).

**how the state can be maintainer in a class component?**

The state is maintained using this.state object.

Use this.setState() to update the state and trigger a re-render.

**What is the role of componentDidMount() method in component life cycle?**

Called once after the component is mounted (rendered to the DOM).

Commonly used for:

Fetching data from an API.

Setting up subscriptions or timers.

Interacting with the DOM.

**Components- Function/Class**

**1.What are the Types of React components?**

1. Functional Components

2. Class Components

* Use ES6 class syntax and extend React.Component.
* Before Hooks, class components were the only way to handle state and lifecycle methods.
* Use this.state, this.props, and lifecycle methods like componentDidMount().

**2.What are Functional components?**

Functional components are JavaScript functions that return JSX (JavaScript XML). Functional components are now the default choice in React unless you need something very specific that class components offer (which is rare with Hooks).

Benefits

* Cleaner and shorter code
* Easy to write and read
* No this binding issues

**3.How do you pass data between functional component in React?**

Passing data between components is essential for building dynamic UIs. In functional components, data is typically passed using props and callback functions.

**4.What are Class Components In React?**

Class Components are one of the two main types of components in React (the other being functional components). They are built using ES6 classes and extend React.Component.

**5.What is Prop drilling in React?**

Prop Drilling is a term used in React when you pass data (props) from a parent component down to deeply nested child components, even if intermediate components don’t need that data themselves.

**6.Why to Avoid Prop Drilling ?**

* Makes code hard to read and maintain
* Intermediate components become unnecessarily complex
* Difficult to refactor or reuse components
* Increases tight coupling between components

**7.In how many ways can avoid Prop Drilling?**

Two Ways:- React Context API – Best for medium complexity

State Management Tools – e.g., Redux, Zustand, Recoil

**8.How to pass data between class component in React?**

In React, data is always passed in a unidirectional flow — from parent to child via props, and from child to parent using callback functions.

**9.what is the role of this keyword in class components?**

In React class components, the this keyword refers to the current instance of the component. It’s used to access:

* this.state – component's state
* this.props – props passed to the component
* this.methodName() – class methods
* this.setState() – to update the state

**10.What are the 5 difference between Functional components and class components?**

1. Syntax

* Functional components are written using plain JavaScript functions or arrow functions.
* Class components are written using ES6 class syntax and extend React.Component.

2. State Management

* Functional components use Hooks like useState and useReducer to handle state.
* Class components manage state using this.state and update it with this.setState().

3. Lifecycle Methods

* In functional components, lifecycle features are handled using Hooks such as useEffect, useLayoutEffect, etc.
* In class components, you use lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount.

4. Use of this Keyword

* Functional components do not use the this keyword.
* Class components require this to access state, props, and methods.

5. Code Simplicity and Readability

* Functional components are simpler, more concise, and easier to read and test.
* Class components tend to be more verbose and involve more boilerplate code.

**11.What are controlled and uncontrolled components in React, particularly in the context of form handling?**

In React, when you're working with forms like inputs, checkboxes, or textareas, the component can be either controlled or uncontrolled, depending on how the data is managed.

**Controlled & Uncontrolled Components**

**What are the Difference bewtween controlled and Uncontrolled Components?**

**Data Source**

React state (useState or this.state)

DOM (ref and internal state)

**Input Control**

Controlled by React

Controlled by the DOM

**Accessing Value**

Via state and onChange

Via ref.current.value

**Ideal For**

Dynamic, real-time validation

Simple forms, minimal logic

**What are characteristics of controlled components?**

Form inputs are bound to React state.

The source of truth is React's state.

UI reflects the state, and any change is handled via an onChange handler.

**what are the advantages of using controlled components in React forms?**

Single source of truth – Form data is in React state.

Easier validation – Validate input in real-time inside onChange.

Predictable UI – You can force reset, pre-fill, or disable fields programmatically.

Better testing/debugging – Values are observable and manageable via state.

**In what scenarios might using Uncontrolled components be advantageous?**

Uncontrolled components are better when:

You don’t need to validate or manipulate input data frequently.

You want faster performance (no re-renders on every keystroke).

You are integrating with non-React libraries or raw form elements.

Simpler input fields like file uploads (using ref to access file)