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

**React.js** is an open-source JavaScript **library** developed by **Facebook** for building **user interfaces**, especially for **single-page applications (SPAs)**. It allows developers to create large web applications that can change data, without reloading the page.

**Key Differences from Other Frameworks/Libraries:**

| **Feature** | **React.js** | **Angular** | **Vue.js** |
| --- | --- | --- | --- |
| Type | Library | Framework | Framework |
| DOM | Virtual DOM | Real DOM (uses dirty checking) | Virtual DOM |
| Data Binding | One-way | Two-way | Two-way |
| Learning Curve | Moderate | Steep | Easy to moderate |
| Control | More flexible; developer controls the architecture | Opinionated; provides full structure | Lightweight; somewhere in between |

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

**✅ Virtual DOM (VDOM):**

* React creates a **lightweight copy** of the actual DOM called the **Virtual DOM**.
* When the state changes, React updates the VDOM first and compares it to the previous version (called **diffing**).
* It then updates **only the changed parts** in the real DOM, improving performance.

**✅ Component-Based Architecture:**

* In React, UI is broken down into **reusable components** (e.g., Header, Button, Form).
* Each component is **self-contained** and manages its **own state** and rendering.
* Components can be **nested**, making UI easier to manage and scale.

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

**🔹 1. Performance**

* Uses **virtual DOM** for efficient updates and rendering.

**🔹 2. Reusable Components**

* Encourages building **modular, maintainable** code with reusable components.

**🔹 3. One-way Data Binding**

* Ensures **predictable data flow**, making debugging and maintenance easier.

**🔹 4. Strong Ecosystem**

* Supports tools like **Redux**, **React Router**, **Next.js**, and a large community with many libraries.

**🔹 5. SEO-Friendly**

* With server-side rendering (e.g., using **Next.js**), React can improve SEO compared to other SPA libraries.

**🔹 6. Backed by Facebook**

* Ongoing support, frequent updates, and robust documentation.

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

**JSX (JavaScript XML)** is a **syntax extension** for JavaScript used in **React** to describe what the UI should look like.

It allows you to write **HTML-like code** directly inside JavaScript, which is then **transpiled** to React.createElement() calls by tools like **Babel**.

**✅ Why JSX is used:**

* Makes the code **more readable and declarative**.
* Keeps **HTML and JavaScript logic in the same file** (component).
* Simplifies the creation of UI by mixing layout and logic together.

**Example:**

jsx

CopyEdit

const element = <h1>Hello, React!</h1>;

This is transformed into:

js

CopyEdit

const element = React.createElement('h1', null, 'Hello, React!');

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

**✅ Differences from Regular JavaScript:**

* JSX looks like **HTML**, but it is **not valid JavaScript** until it's transpiled.
* You can't use control structures like if or for directly inside JSX (you use them outside or in helper functions).

**✅ Can you write JavaScript inside JSX?**

Yes, **JavaScript expressions** can be embedded inside JSX using **curly braces {}**.

**Example:**

jsx

CopyEdit

const name = "Himani";

const element = <h1>Hello, {name}</h1>;

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

In JSX, **curly braces {}** are used to **embed JavaScript expressions** inside HTML-like markup.

**✅ Why curly braces are important:**

* They tell the JSX parser to **evaluate the JavaScript code** inside.
* Without them, React treats the content as a **string or plain text**, not code.

**What you can include inside {}:**

* Variables: {username}
* Expressions: {a + b}
* Function calls: {formatDate(date)}
* Ternary operators: {isLoggedIn ? 'Logout' : 'Login'}

**❌ Invalid usage (won't work):**

jsx

CopyEdit

<h1>Hello, name</h1> // React renders "name" as plain text

**✅ Correct usage:**

jsx

CopyEdit

<h1>Hello, {name}</h1> // React renders "Hello, Himani"

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

**✅ What are components in React?**

* **Components** are the **building blocks** of a React application.
* They allow you to **break the UI into reusable, independent pieces**.
* Each component returns JSX to define the part of the UI it should render.

**✅ Types of Components in React:**

| **Feature** | **Functional Component** | **Class Component** |
| --- | --- | --- |
| Syntax | JavaScript **function** | JavaScript **class** (extends React.Component) |
| State | Uses useState, useEffect (from Hooks) | Uses this.state and this.setState() |
| Lifecycle Methods | With Hooks (e.g., useEffect) | Has built-in methods like componentDidMount() |
| Simpler Syntax | ✅ Yes | ❌ No |
| Performance | Lightweight and preferred in modern React | Slightly heavier |

**✅ Example of Functional Component:**

jsx

CopyEdit

function Welcome(props) {

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

}

**✅ Example of Class Component:**

jsx

CopyEdit

class Welcome extends React.Component {

render() {

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

}

}

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

**✅ What are props?**

* **Props (short for properties)** are used to **pass data from parent to child** components.
* Props are **read-only** in the child component.

**✅ Passing props:**

jsx

CopyEdit

<Welcome name="Himani" />

**✅ Receiving props (in functional component):**

jsx

CopyEdit

function Welcome(props) {

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

}

**✅ Receiving props (in class component):**

jsx

CopyEdit

class Welcome extends React.Component {

render() {

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

}

}

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

**✅ Role of render():**

* In class components, the **render()** method is **required**.
* It **returns the JSX** (UI) that should be displayed on the screen.
* It is called **automatically** whenever the component’s **state or props change**.

**✅ Example:**

jsx

CopyEdit

class Greeting extends React.Component {

render() {

return <h2>Welcome, {this.props.name}!</h2>;

}

}

🔹 Functional components do **not** need a render() method — they directly return JSX.

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

**✅ Props (Properties):**

* **Props** are used to **pass data from a parent component to a child component**.
* They are **read-only** and **cannot be modified** by the child.
* Think of props as **function parameters**.

**Example:**

jsx

CopyEdit

<Welcome name="Himani" />

jsx

CopyEdit

function Welcome(props) {

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

}

**✅ Difference Between Props and State:**

| **Feature** | **Props** | **State** |
| --- | --- | --- |
| Definition | Data passed from parent to child | Internal data of a component |
| Mutability | Immutable (read-only) | Mutable (can be changed) |
| Ownership | Controlled by the parent component | Controlled by the component itself |
| Usage | For communication between components | For dynamic data and user interaction |
| Access | props.propertyName | this.state.propertyName (class) or [state, setState] (function) |

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

**✅ What is State?**

* **State** is a built-in object used to store **dynamic data** that affects the component's behavior and rendering.
* It allows components to be **interactive** and **reactive** to user inputs or events.

**✅ Usage in Functional Component (with Hooks):**

jsx

CopyEdit

import React, { useState } from 'react';

function Counter() {

const [count, setCount] = useState(0); // state declaration

return (

<div>

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

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

}

**✅ Usage in Class Component:**

jsx

CopyEdit

class Counter extends React.Component {

constructor() {

super();

this.state = { count: 0 }; // state initialization

}

render() {

return (

<div>

<p>Count: {this.state.count}</p>

<button onClick={() => this.setState({ count: this.state.count + 1 })}>Increment</button>

</div>

);

}

}

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

**✅ Why use this.setState()?**

* In class components, you **can’t update state directly** like this.state.count = 1 because it won’t re-render the component.
* You must use this.setState() to **update the state and re-render the UI**.

**✅ How this.setState() works:**

* Merges the new state with the existing state.
* Triggers a **re-render** of the component with the updated state.

**✅ Example:**

jsx

CopyEdit

this.setState({ count: this.state.count + 1 });

**✅ Important Points:**

* setState() is **asynchronous** – updates may not appear immediately.
* You can pass a **callback** or use the **previous state** to ensure correct updates:

jsx

CopyEdit

this.setState((prevState) => ({

count: prevState.count + 1

}));

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

**✅ Event Handling in Vanilla JavaScript:**

In plain JavaScript, you typically attach event listeners like this:

html

CopyEdit

<button onclick="doSomething()">Click Me</button>

or in JavaScript:

js

CopyEdit

document.getElementById('btn').addEventListener('click', doSomething);

**✅ Event Handling in React:**

In React:

* Events are handled as **camelCase** (e.g., onClick, not onclick).
* You pass a **function reference**, not a string.

Example:

jsx

CopyEdit

<button onClick={handleClick}>Click Me</button>

**✅ What is a Synthetic Event?**

* React uses a **SyntheticEvent** system that wraps the browser's native event.
* It's a **cross-browser wrapper** to ensure consistency across all browsers.
* It behaves just like the native event but works identically in all environments.

jsx

CopyEdit

function handleClick(event) {

console.log(event); // SyntheticEvent

}

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

**✅ 1. onClick Example:**

jsx

CopyEdit

function App() {

const handleClick = () => {

alert("Button clicked!");

};

return <button onClick={handleClick}>Click Me</button>;

}

**✅ 2. onChange Example:**

jsx

CopyEdit

function InputBox() {

const handleChange = (e) => {

console.log(e.target.value);

};

return <input type="text" onChange={handleChange} />;

}

**✅ 3. onSubmit Example:**

jsx

CopyEdit

function MyForm() {

const handleSubmit = (e) => {

e.preventDefault(); // Prevent page reload

alert("Form submitted!");

};

return (

<form onSubmit={handleSubmit}>

<input type="text" />

<button type="submit">Submit</button>

</form>

);

}

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

**✅ Why Binding is Needed:**

In JavaScript classes, methods **don’t automatically bind** to the class instance. So this may be undefined or refer to the wrong context inside event handlers.

jsx

CopyEdit

class App extends React.Component {

constructor() {

super();

this.state = { count: 0 };

this.handleClick = this.handleClick.bind(this); // Binding in constructor

}

handleClick() {

this.setState({ count: this.state.count + 1 });

}

render() {

return <button onClick={this.handleClick}>Click Me</button>;

}

}

**✅ Alternative Binding Methods:**

1. **Arrow function in class field (modern way):**

jsx

CopyEdit

handleClick = () => {

this.setState({ count: this.state.count + 1 });

};

1. **Arrow function inline (not recommended for performance-critical apps):**

jsx

CopyEdit

<button onClick={() => this.handleClick()}>Click Me</button>

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

**✅ What is Conditional Rendering?**

Conditional rendering in React means displaying different UI **based on a condition**, such as user input, app state, or props.

Just like in JavaScript, you can use **if-else**, **ternary operators**, or **logical AND (&&)** to show or hide elements.

**✅ Example (based on user login state):**

jsx

CopyEdit

function Greeting(props) {

const isLoggedIn = props.isLoggedIn;

if (isLoggedIn) {

return <h1>Welcome back!</h1>;

} else {

return <h1>Please log in.</h1>;

}

}

**Question 2: Explain how if-else, ternary operators, and && (logical AND) are used in JSX for conditional rendering.**

**✅ 1. Using if-else (outside JSX):**

You **can’t use if-else directly in JSX**, but you can use it **before the return** statement:

jsx

CopyEdit

function Message(props) {

let message;

if (props.isMorning) {

message = <h2>Good Morning</h2>;

} else {

message = <h2>Good Evening</h2>;

}

return <div>{message}</div>;

}

**✅ 2. Using Ternary Operator (inline inside JSX):**

jsx

CopyEdit

function Status(props) {

return (

<div>

{props.isOnline ? <p>User is Online</p> : <p>User is Offline</p>}

</div>

);

}

* Syntax: condition ? trueCase : falseCase

**✅ 3. Using Logical AND (&&) Operator:**

Only renders the element **if the condition is true**.

jsx

CopyEdit

function Alert(props) {

return (

<div>

{props.hasError && <p style={{ color: 'red' }}>Something went wrong!</p>}

</div>

);

}

* If props.hasError is true, the message is shown.
* If false, nothing is rendered.

**✅ Summary of When to Use What:**

| **Technique** | **Use Case** |
| --- | --- |
| if-else | Complex logic (used before return statement) |
| ternary ? : | Simple true/false conditions inside JSX |
| condition && | Render something only when the condition is true |