

# 1) React- Components, State, Props

## Introduction to React.js

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

Ans: **React.js** is an open-source JavaScript library developed by **Facebook** (now Meta) for building **user interfaces (UIs)**, especially for **single-page applications (SPAs)**. It allows developers to create large web applications that can update and render efficiently in response to data changes without reloading the page.

React uses a **component-based architecture**, meaning UIs are broken into small, reusable pieces called **components**.

---

### Key Features of React.js:

- **JSX (JavaScript XML):** Lets you write HTML-like syntax in JavaScript.
- **Virtual DOM:** Improves performance by minimizing direct DOM manipulation.
- **Component-Based Architecture:** Encourages reusable and modular UI components.
- **Unidirectional Data Flow:** Data flows from parent to child, making state management predictable.
- **Hooks:** Enable the use of state and lifecycle methods in functional components.

---

### How is React Different from Other JS Frameworks/Libraries?

Feature/Aspect	React.js	Angular	Vue.js	jQuery
Type	Library (for UI)	Framework (full-fledged)	Framework (lightweight)	Library (for DOM manipulation)
Architecture	Component-based	Component-based + MVC	Component-based + MVVM	Procedural, not component-based
DOM	Virtual DOM	Real DOM + change detection	Virtual DOM	Real DOM
Data Binding	One-way	Two-way	Two-way	Manual
Learning Curve	Moderate	Steep (TypeScript, DI, etc.)	Easy	Very easy
Performance	High (thanks to Virtual DOM)	Good but heavier	Good	Slower for complex UIs
Use Case	UI rendering	Full-stack SPA	Progressive UI enhancement	DOM manipulation

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

Ans: **1. Virtual DOM (VDOM)**

◆ **What is the DOM?**

The **DOM (Document Object Model)** is the structured representation of your HTML page in memory. Changing the DOM directly (as in vanilla JavaScript or jQuery) can be slow, especially for large applications.

#### ◆ What is the Virtual DOM?

The **Virtual DOM** is a **lightweight copy of the real DOM** that React keeps in memory. Whenever the UI needs to change:

1. React creates a new virtual DOM.
2. It compares it with the previous version using a process called **diffing**.
3. It updates **only the parts of the real DOM that actually changed**, not the whole page.

#### ✓ Benefits:

- Improves performance
  - Minimizes direct DOM manipulation
  - Leads to smoother and faster user interfaces
- 

## ✿ 2. Component-Based Architecture

#### ◆ What is a Component?

A **component** in React is a **reusable, self-contained piece of UI**, like a button, form, or even an entire page. Components can be:

- **Functional components** (modern and preferred)
- **Class components** (older style)

#### ◆ Types of Components:

- **Presentational components:** Focus on how things look.
- **Container components:** Handle logic and state.

#### ✓ Benefits:

- **Reusability:** Create once, use many times.
- **Modularity:** Break complex UIs into smaller, manageable pieces.

- **Maintainability:** Easier to read, debug, and test.

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

Ans: **Advantages of React.js (Simple Version)**

1. **Fast Performance**

React uses **Virtual DOM**, which updates only the parts of the page that change — making apps fast.

2. **Reusable Components**

You can build small parts (components) and reuse them, saving time and effort.

3. **Easy to Learn**

Uses **JSX** (HTML + JavaScript) which is easy to write and understand.

4. **One-Way Data Flow**

Data flows in one direction, making apps easy to debug and manage.

5. **Big Community Support**

Lots of tutorials, tools, and help available online.

6. **Supports Mobile Apps**

With **React Native**, you can build mobile apps using the same skills.

7. **Easier to Test**

React code is simple and easy to test.

8. **Modern Features**

React offers **Hooks** (like `useState`) to manage state in a clean way.

## 2) JSX (JavaScript XML)

Que 1) What is JSX in React.js? Why is it used?

Ans: **What is JSX in React.js?**

**JSX (JavaScript XML)** is a **syntax extension** for JavaScript used in React.

It lets you **write HTML-like code inside JavaScript**.

### ✅ Example:

```
const element = <h1>Hello, JSX!</h1>;
```

This looks like HTML, but it's actually JavaScript!

---

### 🤖 Why is JSX used in React?

1. ✅ **Easier to Read and Write**

Looks like HTML, so it's familiar and simple.

2. ✅ **Faster Development**

You can build UI directly in JavaScript — no need to separate HTML and JS.

3. ✅ **Powerful Features**

You can use **JavaScript inside HTML** using `{}`.

Example:

```
const name = "John";
```

```
<p>Hello, {name}</p>
```

4. ✅ **Helps React Create UI**

JSX is converted to `React.createElement()` behind the scenes, which builds the UI.

Que 2) How is JSX different from regular JavaScript? Can you write JavaScript inside JSX?

Ans: How is JSX different from regular JavaScript?

**JSX** looks like HTML but works inside JavaScript.

It's **not regular JavaScript**, but a **syntax extension** that React uses.

---

### 🔍 Key Differences:

Feature	JSX	Regular JavaScript
HTML-like syntax	Yes (<h1>Hello</h1>)	No (document.createElement())
Used in UI building	Yes, in React	Not used directly for UI
Needs to be compiled	Yes (converted to JS by Babel)	No
Mix HTML + JS easily	Yes (<p>{name}</p>)	No

### 🤔 Can you write JavaScript inside JSX?

✅ Yes!

You can write JavaScript **inside curly braces {}** in JSX.

✅ Example:

```
const name = "Alice";
```

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

You can also use expressions like:

```
<p>{5 + 10}</p> // Outputs: 15
```

```
<p>{isLoggedIn ? "Welcome" : "Login"}</p>
```

### 🔙 In Short:

- JSX is not the same as regular JavaScript.
- You can write JavaScript **expressions** inside JSX using {}.

Que 3) Discuss the importance of using curly braces {} in JSX expressions.

Ans: Importance of Using Curly Braces {} in JSX

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

---

## ✅ Why are curly braces important?

### 1. Embed JavaScript Values

- You can insert variables, functions, or expressions.

```
const name = "John";
```

```
<h1>Hello, {name}</h1> // Output: Hello, John
```

### 2. Use Logic and Math

- Perform calculations or conditional rendering.

```
<p>{5 + 10}</p> // Output: 15
```

```
<p>{isLoggedIn ? "Welcome" : "Please login"}</p>
```

### 3. Call Functions

- You can run functions inside JSX.

```
<p>{getMessage()}</p>
```

### 4. Keep UI Dynamic

- Makes your components **interactive and data-driven**.

---

## ❌ Without {}:

JSX treats it as plain text, not JavaScript.

```
<p>name</p> // Shows "name", not the value
```

```
<p>{name}</p> // Shows the value of the variable
```

---

## 📄 Summary:

- Curly braces {} allow you to run JavaScript **inside JSX**.
- They help make the UI **dynamic, logical, and powerful**.

# 3) Components (Functional & Class Components)

Que 1) What are components in React? Explain the difference between functional components and class components.

Ans :  What Are Components in React?

**Components** are the **building blocks of a React application**.

They are **reusable pieces of UI**—like buttons, forms, or even entire pages.

You can think of them as **functions or classes** that return **JSX**.

---

## Types of Components

React has two main types of components:

---

### 1. Functional Components (Modern and Recommended)

- **Simple JavaScript functions**
- Use **hooks** (like `useState`, `useEffect`) for state and lifecycle
- Easier to write and understand

#### ◆ **Example:**

```
function Greeting(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

Or using arrow functions:

```
const Greeting = ({ name }) => <h1>Hello, {name}</h1>;
```

---

### 2. Class Components (Older Style)

- Use JavaScript **classes**



- Use `this.state` and `this.setState()` for managing state
- Use **lifecycle methods** like `componentDidMount`

◆ **Example:**

```
class Greeting extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
}
```

Que 2) How do you pass data to a component using props?

Ans: **How to Pass Data Using Props:**

### 1. Pass Props from Parent Component

You add an attribute to the child component tag and assign it a value:

```
// ParentComponent.jsx
```

```
import ChildComponent from './ChildComponent';
```

```
function ParentComponent() {
  return (
    <div>
      <ChildComponent name="Alice" age={25} />
    </div>
  );
}
```

### 2. Access Props in Child Component

You access the passed data via the props object or destructuring:

```
// ChildComponent.jsx
```

```
function ChildComponent(props) {  
  return (  
    <div>  
      <h1>Hello, {props.name}!</h1>  
      <p>You are {props.age} years old.</p>  
    </div>  
  );  
}
```

// OR using destructuring

```
function ChildComponent({ name, age }) {  
  return (  
    <div>  
      <h1>Hello, {name}!</h1>  
      <p>You are {age} years old.</p>  
    </div>  
  );  
}
```

#### **Notes:**

- Props are **read-only** — you **cannot** change them in the child component.
- Props allow **reusability** of components with different data.

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

Ans: Role of render() in Class Components

- The render() method **returns JSX** (or null) which React then converts to actual DOM elements.

- It is **called automatically** whenever the component is rendered or re-rendered (due to `setState()` or new props).
- 

### Syntax Example:

```
import React, { Component } from 'react';

class Greeting extends Component {

  render() {

    return (

      <div>

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

      </div>

    );

  }

}
```

### How It Works:

- When `<Greeting name="Alice" />` is used, React calls the `render()` method.
  - `this.props.name` gets the value "Alice".
  - JSX is returned and React displays: Hello, Alice!
- 

### Key Points:

- **Required** in every class component.
- Must **return a single parent element** (like `<div>`).
- **No side effects** should be done in `render()` (like API calls; use `componentDidMount()` for that).
- Always **pure** — the same input (props/state) gives the same output (UI).

## 4) Props and State

Que 1) What are props in React.js? How are props different from state?

Ans: **What are Props in React.js?**

**Props** (short for **properties**) are **read-only inputs** passed from a **parent component to a child component** in React. They allow you to configure or customize a component with data from outside.

---

### **Example of Props:**

```
function Welcome(props) {  
  return <h1>Hello, {props.name}!</h1>;  
}
```

// Usage:

```
<Welcome name="Alice" />
```

- name="Alice" is a prop.
  - props.name in the Welcome component is "Alice".
- 

### **Key Features of Props:**

- Passed **from parent to child**.
  - **Immutable** inside the child component.
  - Used to make components **dynamic and reusable**.
- 

### **Difference Between Props and State:**

Feature	Props	State
Definition	External data passed to a component	Internal data managed by the component
Mutability	<b>Immutable</b>	<b>Mutable</b> (via setState)
Ownership	Controlled by <b>parent</b> component	Controlled <b>within</b> the component
Usage	For <b>configuration</b>	For <b>dynamic behavior/data changes</b>
Accessibility	Available via this.props or props	Available via this.state or state
Purpose	Communication between components	Handling UI changes, user input, etc.

Que 2) Explain the concept of state in React and how it is used to manage component data.

Ans : **What is State in React?**

**State** in React is like a **memory** for a component.

It stores **information** that can **change** when the user interacts with your app.

For example:

If you have a counter, the number it shows is stored in the **state**.







### Why Do We Need State?

Because apps are **dynamic** — things change!

State helps React know **when something has changed** so it can update the screen automatically.



### Key Points about State:

Concept	Explanation
 <b>Changes Over Time</b>	State holds data that can change (like a user clicking a button).
 <b>Stored Inside Component</b>	State belongs to the component itself — it's not passed like props.
 <b>Used to Control UI</b>	When the state changes, the UI updates to match the new state.
 <b>You Should Not Change It Directly</b>	Always use special functions (setState or setCount) to change it.

---

### Real-Life Example:

Think of a **fan**.

- The **speed setting** is the **state**.
- When you press the button to increase speed, it changes the state.
- The fan reacts by spinning faster — just like React re-renders the screen!

---

### Summary:

- **State** is used to **store and manage data** inside a component.
- It helps make apps **interactive**.
- React watches the state — if it changes, the UI updates **automatically**.

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

Ans : In React **class components**, this.setState() is used to **update the component's state**.

You **can't change state directly** like this.state.count = 1 because React won't know that it needs to re-render the UI.

Instead, this.setState() tells React:

“Hey, I’ve updated something — please re-render the component with the new data.”

---

### **How Does this.setState() Work?**

1. You call this.setState() with the new state data.
2. React **merges** the new state with the existing state.
3. React automatically **re-renders** the component to update the UI.

### **Summary:**

Feature	Explanation
Used For	Updating state in class components
Automatically Re-renders	Yes, after calling this.setState()
Safe to Use	Yes, always use setState instead of direct assignment
Merges State	Only the part of the state you update gets changed; the rest stays the same