Module 9 - Introduction to React.js

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** (now Meta) used for building **user interfaces**, especially **single-page applications (SPAs)**.
It focuses mainly on the **view layer (UI)** of an application and allows developers to create **reusable UI components**.

Key Features of React.js:

1. **Component-Based Architecture** – The UI is divided into small, reusable components that make code more modular and easier to maintain.
2. **Virtual DOM** – React uses a virtual representation of the DOM, updating only the parts that change instead of re-rendering the entire page. This improves performance.
3. **Declarative Syntax** – Developers describe *what* the UI should look like, and React efficiently updates the DOM to match that state.

4. **One-Way Data Binding** – Data flows in a single direction (from parent to child), making it easier to debug and maintain.
5. **JSX (JavaScript XML)** – React allows writing HTML-like syntax directly in JavaScript, which makes UI code more readable and expressive.

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

=>React is built on a few **core principles** that make it powerful and efficient for building modern user interfaces.
The two most important among them are the **Virtual DOM** and **Component-Based Architecture**.

---

1. Virtual DOM (Document Object Model)

- The **Virtual DOM** is a **lightweight copy** of the actual browser DOM.
- When the state or data in a React app changes, React doesn't update the **real DOM** immediately (which is slow).
- Instead, it updates the **Virtual DOM first**, compares it with the previous version (a process called **diffing**), and then updates **only the changed elements** in the real DOM.

- This process is called **reconciliation** and it makes UI updates **much faster and more efficient**.

2. Component-Based Architecture

- React applications are built using **components**, which are **independent, reusable pieces** of UI.
- Each component can manage its own **state**, **logic**, and **styling**, making the app modular and easy to maintain.
- Components can be **functional** or **class-based**, and they can be **nested** to form complex interfaces.

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

=>React.js offers several advantages that make it one of the most popular choices for building modern, dynamic web applications.

---

1. Component-Based Architecture

- React allows developers to build **reusable components**, which means you can use the same UI pieces in multiple places.
- This improves **code reusability**, **readability**, and **maintenance**.

---

2. Virtual DOM for Better Performance

- React uses a **Virtual DOM** to efficiently update only the parts of the page that change, instead of re-rendering the entire page.
- This results in **faster performance** and a **smooth user experience**.

---

3. Declarative UI

- Developers describe what the UI should look like, and React automatically updates it when the data changes.
- This makes the code **more predictable** and **easier to debug**.

---

4. One-Way Data Binding

- React follows **unidirectional data flow**, meaning data moves from parent to child components.
- This simplifies debugging and makes applications **more stable**.

---

5. Strong Community Support

- Backed by **Meta (Facebook)** and a large open-source community.
- Rich ecosystem of libraries, tutorials, and developer tools.

---

6. JSX (JavaScript XML)

- JSX allows you to write **HTML-like syntax** within JavaScript, making the code more **intuitive** and **readable**.

---

7. Cross-Platform Development

- Using frameworks like **React Native**, developers can build **mobile applications** using the same React principles and components.

---

8. SEO-Friendly

- React supports **server-side rendering (SSR)** using frameworks like **Next.js**, which helps search engines index content more effectively.

---

9. Easy Integration

- React can be easily integrated with **existing projects**, **other libraries**, or **frameworks**, making it flexible to use.

JSX (JavaScript XML)

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

=>**JSX (JavaScript XML)** is a **syntax extension** for JavaScript used in **React.js**.
It allows developers to **write HTML-like code directly inside JavaScript**, which makes it easier to describe what the UI should look like.

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

=>**JSX (JavaScript XML)** is different from regular JavaScript because it allows you to **write HTML-like syntax directly within JavaScript code**.
While **regular JavaScript** uses functions like document.createElement() to create UI elements, **JSX** provides a more **declarative and readable** way to describe the UI.

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

=>n **JSX**, **curly braces {}** are used to **embed JavaScript expressions** inside HTML-like code.

They act as a **bridge** between **JavaScript logic** and the **JSX template**, allowing you to dynamically display values, perform calculations, or call functions within your UI.

Importance of Curly Braces {} in JSX:

1. 🧠 **Embed Dynamic Data:**
   You can insert variables or expressions directly inside JSX.
2. const name = "Neer";
3. <h1>Hello, {name}!</h1>
4. ⚡ **Use JavaScript Logic:**
   You can perform operations, call functions, or evaluate conditions.
5. Sum: {5 + 10}
6. Uppercase: {name.toUpperCase()}
7. 🔄 **Dynamic Rendering:**
   Curly braces allow React to re-render values automatically when the data (state or props) changes.
8. Current count: {count}
9. 🧩 **Works Only for Expressions, Not Statements:**
   You can use expressions (like x + y or func()) but **not**

statements (like if or for).

To use conditions, you typically write expressions such as:

{isLoggedIn ? Welcome! : Please login. }

Components (Functional & Class Components)

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

=>In **React**, **components** are the **building blocks** of a user interface.
Each component represents a **part of the UI** (like a button, form, or header) and can be **reused** across the application.
They help in breaking the UI into **smaller, manageable, and reusable pieces**.

Types of Components in React:

React mainly provides **two types of components**:

1. **Functional Components**
2. **Class Components**

---

1. Functional Components:

- These are **simple JavaScript functions** that return JSX.
- They are **stateless** (before React Hooks) but can now handle **state and lifecycle** using **Hooks** like useState, useEffect, etc.
- Easier to write, read, and test.

2. Class Components:

- These are **ES6 classes** that extend React.Component.
- They use a **render()** method to return JSX.
- Can manage **state** and **lifecycle methods** directly (like componentDidMount()).

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

=>In **React**, **props (short for "properties")** are used to **pass data** from a **parent component** to a **child component**.
They make components **dynamic and reusable** by allowing you to send different values each time.

How Props Work:

1. **Pass data** from the parent component using attributes.
2. **Receive data** in the child component using props.

Key Points about Props:

- Props are **read-only** (cannot be modified inside the child).
- Data flows **one way** — from parent to child (unidirectional).
- Used to make components **reusable and customizable**.

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

=>In **React class components**, the **render()** method is a **mandatory** function that tells **React what to display on the screen**.
It returns **JSX** (or null) which defines the **UI structure** of that component.

Key Points about render():

1. 🧩 **Defines the UI:**
   The render() method returns the **JSX** that represents how the component should look.
2. 🔁 **Automatic Re-rendering:**
   Whenever the **state** or **props** of a component change, React **automatically calls `render()`** to update the UI.

3. ⚙️ **Must Return a Single Root Element:**
   The JSX returned by `render()` must be wrapped in **one parent element** (like a `<div>` or `<>`).
4. 🚫 **No Side Effects:**
   The `render()` method should be **pure**, meaning it should not modify the state or interact with the DOM directly.

Props and State

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

=>**Props** (short for **properties**) are **read-only inputs** that are used to **pass data from one component to another**, usually from a **parent component to a child component**.
They allow components to be **reusable** and **dynamic** by providing them with customizable values.

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

=>**Concept of State in React**

State in React is a **built-in object** that allows a component to **store, track, and manage data** that may **change over time**.
It determines **how a component behaves and what it renders** on the screen.

In simple terms:

State = The component's memory.

Whenever the **state changes**, React **automatically re-renders** the component to show the updated UI — this is what makes React apps **interactive and dynamic**.

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

=>In **React class components**, we use the **this.setState()** method to **update the component's state**.

You **should never modify this.state directly** (like this.state.count = 1), because React **won't know** that the state has changed — meaning the component **won't re-render**.

✅ **this.setState()** safely updates the state **and** tells React to **re-render** the component with the new data.

How this.setState() Works

1. **Creates a new state object**
    - It merges the **updated values** with the existing state.
2. **Triggers a re-render**
    - After the update, React automatically re-renders the component to reflect the new state in the UI.
3. **Works asynchronously**
    - setState() may **batch multiple updates** together for better performance — so you shouldn't rely on the old state immediately after calling it.