Module 9 - Introduction to React.js

<u>THEORY EXERCISE</u>

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

React.js is a **JavaScript library** developed by **Facebook** for building **user interfaces**, especially **single-page applications (SPAs)**. It focuses on creating **reusable UI components** that update efficiently with dynamic data.

▢ Differences from other frameworks/libraries:

- **Library vs. Framework**: React is a **library**, not a full framework like Angular. It handles only the **view layer** (UI) in the MVC (Model-View-Controller) architecture.
- **Virtual DOM**: React uses a **virtual DOM** for faster rendering, while many frameworks manipulate the actual DOM directly.
- **Component-Based**: Everything in React is a component, encouraging modularity and reusability.
- **One-Way Data Binding**: React uses **unidirectional data flow**, which simplifies debugging, unlike Angular's two-way data binding.

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

a. **Virtual DOM**:
   - The **virtual DOM** is an in-memory representation of the real DOM.
   - When changes occur, React updates the virtual DOM first, compares it with the previous version using a process called **diffing**, and then updates only the changed parts in the real DOM.
   - This leads to **faster performance** and **efficient updates**.

b. **Component-Based Architecture**:
   - React apps are built using **independent, reusable components**.
   - Each component manages its own **state** and **logic** and can be combined to build complex UIs.
   - Encourages **code reusability**, **maintainability**, and **scalability**.

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

▢ Here are key advantages:

1. Performance: Thanks to the virtual DOM, React updates only what's necessary, improving speed.
2. Reusable Components: Write once, use anywhere—reduces duplication and improves maintainability.
3. Strong Ecosystem: Large community, rich libraries, and tools (like Redux, React Router, etc.).

4. SEO-Friendly: React can be rendered on the server (using Next.js), improving SEO for SPAs.
5. Easy to Learn: Uses plain JavaScript with JSX, making it accessible for developers with basic JS knowledge.
6. Unidirectional Data Flow: Makes apps more predictable and easier to debug.
7. Active Community: Supported by Facebook and a massive developer community, ensuring regular updates and support.

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 to **describe what the UI should look like**. It allows developers to **write HTML-like code inside JavaScript**, making it easier to create and visualize the structure of UI components.

Why is JSX used?

- **Improved readability**: Looks like HTML, making it intuitive and easy to read.
- **Better developer experience**: Helps developers visualize the UI within the component logic.
- **Powerful and flexible**: Allows embedding dynamic JavaScript expressions in the markup.
- **Tooling support**: JSX is supported by most code editors with syntax highlighting and autocompletion.

JSX is **not required** to use React, but it is widely adopted for convenience.

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

Differences from regular JavaScript:

- JSX looks like HTML but is not valid JavaScript—it must be **transpiled by tools like Babel** into React.createElement() calls.
- JSX has **stricter syntax**:
  - All tags must be closed (e.g., <img />, <input />)
  - You must use className instead of class, htmlFor instead of for, etc.

Can you write JavaScript inside JSX?

```
const name = "Alice"

return <h1>Hello, {name}!</h1>
```

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

Curly braces {} are used in JSX to **embed JavaScript expressions** inside HTML-like code. They act as a bridge between **JavaScript logic** and **markup**.

Why are {} important?

- Allow dynamic content: You can insert variables, function calls, or expressions.
- Enable conditional rendering: Use ternary operators or short-circuit logic.
- Help manage lists: Use .map() inside braces to render lists dynamically.

Example:-

```
// Displaying a variable

<h2>{username}</h2>

// Inline calculation

<p>{5 + 10}</p>

// Conditional rendering

{isLoggedIn ? <p>Welcome back!</p> : <p>Please log in.</p>}.
```

Components (Functional & Class Components)

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

Components are the **building blocks** of a React application. Each component is a **self-contained piece of UI** that can have its own structure, logic, and styling.

Types of Components:

Functional Components:

- ○ Simple JavaScript functions.
- ○ Can use **React hooks** (useState, useEffect, etc.) to manage state and lifecycle.
- ○ Preferred in modern React due to simplicity and better performance.

Ex:-

```
function Welcome(props) {

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

Class Components:

- ES6 classes that extend React.Component.
- Use the render() method to return JSX.
- State and lifecycle methods are managed using this.state, componentDidMount(), etc.

EX:-

```
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?

Props (short for properties) are used to **pass data** from a **parent component to a child component**.

Props are **read-only**: Child components cannot modify them.

Think of props as **function arguments** for components.

```
function Greeting(props) {

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

}

// Using the component and passing props

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

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

In **class components**, the render() method is **required** and is used to **describe what should be displayed** on the screen.

Key Points:

- It returns the **JSX (UI structure)** that React will render.
- It is called automatically by React when:

- The component is first rendered.
- The component's state or props change.

Example:

```
class Hello extends React.Component {

  render() {

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

  }

}
```

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** passed from a **parent component to a child component**. They allow components to be **reusable** and **dynamic** based on the data they receive.

Example of props:-

```
function Welcome(props) {

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

}

// Usage:

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

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

Feature	Props	State
Mutability	**Immutable (read-only)**	**Mutable (can change over time)**
Ownership	Passed from **parent**	Managed **inside the component**
Usage	For **configuration/input**	For **internal data management**
Example	<Greeting name="John" />	this.setState({ name: "John" })

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

State is a built-in React object used to store **dynamic data** that can **change over time** and affect what the component renders.

Key points:

- State is **local to the component**.
- When state changes, React automatically **re-renders** the component.
- Used for tracking user input, toggles, loading status, etc.

Example (Functional Component with useState):

```
import { useState } from "react";

function Counter() {

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

  return (

    <div>

      <p>{count}</p>

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

    </div>
```

```
 );
}
```

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

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

Why it's used:

- **Direct state mutation (this.state = ...) is not allowed**.
- this.setState() ensures:
  - React knows the state has changed.
  - The component is **re-rendered**.
  - Updates are **merged**, not replaced.

```
class Counter extends React.Component {
 constructor() {
  super();
  this.state = { count: 0 };
 }
 increment = () => {
  this.setState({ count: this.state.count + 1 });
 };
 render() {
  return (
   <div>
    <p>{this.state.count}</p>
    <button onClick={this.increment}>Increment</button>
   </div>
  );
 }
}
```