Module 9 - Introduction to React.js - Components,State,Props

THEORY EXERCISE

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

Ans: React.js is an open-source JavaScript **library** (not a full framework) developed by **Facebook (now Meta)**, primarily used for building **user interfaces (UIs)** — especially for **single-page applications** where you need fast, interactive UI updates.

❖ **How React.js Differs from Other JavaScript Frameworks/Libraries**

Feature	React.js	Angular	Vue.js
Type	Library	Full-fledged framework	Progressive framework
Developer	Meta (Facebook)	Google	Evan You (ex-Google)
Architecture	Component-based	Component-based + MVC	Component-based + MVVM
DOM Manipulation	Virtual DOM	Real DOM (with change detection)	Virtual DOM
Data Binding	One-way (unidirectional)	Two-way binding	Two-way binding (optional)
Learning Curve	Moderate	Steep	Gentle
Flexibility	High (you choose tools)	Lower (opinionated)	Moderate
Routing & State Mgmt	External libraries (e.g., React Router, Redux)	Built-in	Vue Router, Vuex (optional)
Use in Industry	Widely used (Instagram, FB, Netflix)	Widely used (Google Apps)	Popular for smaller projects

Module 9 - Introduction to React.js - Components,State,Props

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

Ans :

React applications are made up of components - small, reusable building blocks that define how a part of the user interface should look and behave.

1. Component-Based Architecture

React applications are built using **components**, which are reusable, self-contained pieces of UI. Each component can have its own state, logic, and rendering behavior.

- **Functional Components**: Simple JavaScript functions that return JSX.
- **Class Components**: ES6 classes that extend `React.Component` (used less frequently now).
- **Props**: Short for "properties", these are inputs passed to components to make them dynamic and reusable.
- **State**: Internal data storage for a component that can change over time and trigger UI updates.

Example:

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

2. Virtual DOM

The **Virtual DOM** is a lightweight, in-memory representation of the real DOM.

- React keeps a virtual copy of the DOM in memory.
- When the state of a component changes, React creates a new Virtual DOM tree.
- It then **diffs** this new tree with the previous one to detect what has changed.
- Only the changed elements are updated in the real DOM (a process known as **reconciliation**).

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

Ans:

the key advantages of using React.js in web development:

1. Component-Based Architecture

- Builds UIs using **independent, reusable components**.
- Makes development more organized and scalable.
- Encourages code reusability and easier testing.

Module 9 - Introduction to React.js - Components,State,Props

2. Fast Rendering with Virtual DOM

- React uses a **Virtual DOM** to efficiently update only the parts of the real DOM that changed.
- Improves performance, especially for dynamic and interactive applications.

3. Declarative UI

- You describe **what** the UI should look like based on the app's state.
- React takes care of updating the DOM when the state changes.
- Leads to cleaner and more predictable code.

4. Unidirectional Data Flow

- Data flows in a single direction (parent to child).
- Makes it easier to debug and understand how data is being used.

5. JSX (JavaScript + XML)

- Allows you to write HTML-like syntax directly in JavaScript.
- Improves readability and combines logic and UI in one place.

6. Strong Community & Ecosystem

- Large, active developer community.
- Tons of libraries, tools, and tutorials available.
- Supported by major companies like **Meta (Facebook)**.

7. Rich Developer Tools

- **React Developer Tools** extension helps inspect components, props, and state.
- Makes debugging and performance tuning easier.

8. SEO-Friendly (with SSR using Next.js)

- Supports **server-side rendering** with frameworks like **Next.js**.
- Helps improve **SEO and initial page load speed**.

9. Cross-Platform Development (React Native)

- React knowledge can be reused to build **mobile apps** with **React Native**.
- Saves development time by sharing logic between platforms.

10. Strong Backing and Ongoing Support

- Maintained by **Meta (Facebook)** with regular updates and improvements.

Module 9 - Introduction to React.js - Components,State,Props

- Trusted by major companies like Netflix, Airbnb, and Instagram.

JSX (JavaScript XML)

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

Ans:

JSX (JavaScript XML) is a **syntax extension** for JavaScript used in **React** that lets you write **HTML-like code inside JavaScript**.

Why is JSX Used in React?

JSX is used because it makes writing and understanding **React components** easier and more intuitive. Here's why:

1. Improves Readability

- JSX looks like HTML, making the structure of the UI easy to understand.
- Keeps UI layout and logic in the same file.

Example:

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

This is more readable than the equivalent JavaScript:

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

2. Combines Markup and Logic

- JSX lets you embed JavaScript expressions directly within HTML using { }.

Example:

```
const name = "John";
const greeting = <h1>Hello, {name}!</h1>;
```

3. Helps Prevent Injection Attacks

- JSX is compiled to JavaScript and automatically escapes values to prevent **XSS (Cross-Site Scripting)** attacks.

4. Tooling Support

- JSX works well with modern development tools like **Babel** and **Webpack**, which convert JSX into regular JavaScript before running it in the browser.

5. Developer Experience

- Improves productivity with better **autocomplete**, **linting**, and **error messages** in editors like VS Code.

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

Ans:

JSX (JavaScript XML) is a **syntax extension for JavaScript**, commonly used with React to describe what the UI should look like. It looks similar to HTML, but it's actually syntactic sugar for React.createElement() calls.

Differences between JSX and Regular JavaScript:

Feature	JSX	Regular JavaScript
Syntax	Looks like HTML	Pure JS syntax
Purpose	Describes UI elements in React	General-purpose programming
Usage	Used within React components to return UI	Used anywhere in your code
Compiled to	`React.createElement()`	Direct execution by the JS engine

❖ Can You Write JavaScript Inside JSX?

Example:

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

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

- `{name}` is a JavaScript expression embedded inside JSX.

❖ Can also use functions, expressions, or even conditionals:

```
const user = { firstName: "Alice", lastName: "Smith" };


const element = (
 <h1>
  Hello, {user.firstName + ' ' + user.lastName}!
 </h1>
);
```

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

Ans :

1. Embedding JavaScript in JSX
2. Dynamic Content Rendering
3. Using Expressions (Not Statements)
4. Rendering Lists
5. Conditional Rendering
6. Binding Attributes Dynamically

Module 9 - Introduction to React.js - Components,State,Props

Components (Functional & Class Components)

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

Ans:

Components are the **building blocks of a React application**. They let you split the UI into **independent, reusable pieces**, each handling its own logic and rendering.

Think of components like **custom HTML elements**:

```
<App />
<Navbar />
<UserProfile />
```

Each component:

- Returns **JSX** (UI structure)
- Can hold **data** (via props or state)
- Can respond to **user events**

Module 9 - Introduction to React.js - Components,State,Props

❖ Two Main Types of Components in React:

Feature	Functional Component	Class Component
Syntax	JavaScript function	ES6 class
State	Uses `useState()` (React Hooks)	Uses `this.state`
Lifecycle methods	Uses `useEffect()`	Has built-in lifecycle methods like `componentDidMount()`
Simpler and lighter	Yes	More complex
Recommended in modern React	Yes	Less common now

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

Ans:

1. In the Parent Component:

- **Embed the child component:** Include the child component within the parent component's JSX.
- **Pass props as attributes:** Add attributes to the child component's tag, where the attribute name will be the prop name, and the value will be the data you want to

pass. If the value is a JavaScript expression (like a variable or an object), enclose it in curly braces `{}`.

2. In the Child Component:

- **Receive props as a parameter:** In a functional component, the `props` object is received as an argument to the function. In a class component, props are accessible via `this.props`.
- **Access prop values:** Use dot notation (`props.propName` or `this.props.propName`) to access the individual values passed as props. You can also use object destructuring for cleaner code in functional components.

Explanation:

- **Props (Properties):** Props are a mechanism for passing data from a parent component to its child components. They are read-only and immutable within the child component.
- **Data Types:** You can pass various data types as props, including strings, numbers, booleans, arrays, objects, and even functions.

- **Unidirectional Data Flow:** Data flow in React is primarily unidirectional, meaning data flows from parent to child components via props. While you can pass functions as props to enable child components to trigger actions in the parent, the data itself originates from the parent.

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

Ans:

The `render()` method in React class components is a required method that defines the component's user interface (UI). Its primary role is to return JSX (JavaScript XML), which describes the structure and content of the UI to be displayed on the screen.

Here's a breakdown of its key roles:

- **UI Definition:** The `render()` method is where you specify what the component should look like. It contains the JSX that React will use to construct the virtual DOM and eventually update the real DOM.

- **Responsiveness to State and Props Changes:** The `render()` method is automatically invoked by React

whenever the component's `state` or `props` change. This ensures that the UI is always up-to-date and reflects the latest data.

- **Pure Function (Ideally):** The `render()` method should ideally be a pure function. This means it should not modify the component's state, interact directly with the browser's DOM, or perform side effects. Its sole responsibility is to return the JSX based on the current `state` and `props`.
- **Foundation for Reconciliation:** The JSX returned by `render()` is converted into React elements, which are plain JavaScript objects representing the intended UI. React then uses these elements to perform its reconciliation process, efficiently updating the actual DOM to match the desired UI.

Props and State

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

Ans:

In React.js, props (short for properties) are a mechanism for passing data from a parent component to its child components. They are read-only and immutable within the child component, meaning a child component cannot

directly modify the props it receives. Props are essential for configuring and customizing child components, allowing for data flow down the component tree.

State, on the other hand, is an object that holds data specific to a component and can be changed over time. State is managed internally within a component and is mutable, meaning the component can update its own state using the `setState` method in class components or the `useState` hook in functional components. Changes in state typically trigger a re-render of the component and its children.

Here's a summary of the key differences:

Props:

- **Purpose:** Pass data from parent to child components.
- **Mutability:** Immutable (read-only) within the receiving component.
- **Ownership:** Owned by the parent component.
- **Flow:** Unidirectional (parent to child).

State:

- **Purpose:** Manage internal data within a component that can change over time.

- **Mutability:** Mutable (can be updated by the component itself).
- **Ownership:** Owned and managed by the component itself.
- **Flow:** Internal to the component.

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

Ans:

state refers to a plain JavaScript object that holds data specific to a component, which can change over time. Unlike props, which are passed down from parent components and are immutable within the child, state is internal to a component and can be modified by that component itself. When a component's state changes, React automatically re-renders the component and its children to reflect the updated data in the UI.

How State Manages Component Data:

- **Declaration:** In functional components, state is declared using the `useState` Hook. This Hook returns

an array containing the current state value and a function to update it.

- **Initialization:** The `useState` Hook or the `this.state` object in class components is initialized with an initial value or object, setting the starting point for the component's data.
- **Updating State:**
  - **Functional Components:** The setter function returned by `useState` is used to update the state. For example, `setCount(count + 1)` would increment the `count` state variable.
  - **Class Components:** The `this.setState()` method is used to update the state. It merges the new state with the current state. For example, `this.setState({ count: this.state.count + 1 })`.
- **Re-rendering:** When `setCount` or `this.setState()` is called, React detects the state change and triggers a re-render of the component and its descendants that depend on that state. This ensures the UI stays synchronized with the underlying data.
- Purpose of State:

- **Dynamic UI:** State enables components to have dynamic and interactive UIs that respond to user actions, data fetching, or other events.
- **Component Memory:** It acts as a component's internal memory, allowing it to "remember" information between renders.
- **Encapsulation:** State encapsulates data within a component, making it self-contained and managing its own internal logic.

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

Ans:

`this.setState()` is used in React class components to update the component's internal state and trigger a re-render of the component and its children. It is the primary mechanism for managing dynamic data within a class component that can change over time and influence the UI.

How `this.setState()` works:

- **Enqueues Changes:** When `this.setState()` is called, it enqueues a change to the component's state object. It does

not immediately update `this.state`. Instead, React batches multiple `setState` calls for performance optimization.

- **Triggers Reconciliation:** After the state update is scheduled, React initiates its reconciliation process. This involves comparing the new state with the previous state and determining what changes need to be made to the virtual DOM.
- **Virtual DOM Update:** React creates a new virtual DOM tree representing the component with the updated state. It then efficiently compares this new tree with the previous one to identify the minimal set of changes required to update the real DOM.
- **Real DOM Update:** Only the necessary changes are applied to the actual browser DOM, leading to an efficient and optimized UI update. This avoids directly manipulating the DOM, which can be slow and error-prone.
- **Re-rendering:** As a result of the state change and reconciliation, the component's `render()` method is called again, reflecting the new state in the user interface.

Key characteristics of `this.setState()`:

- **Asynchronous:** `this.setState()` is asynchronous. You should not rely on `this.state` being immediately updated

after calling `setState()`. If you need to perform actions after the state has definitely updated, use the optional callback function as the second argument to `setState()`.

- **Merges State:** `this.setState()` merges the provided object into the current state. It does not replace the entire state object.
- **Function as Argument:** You can pass a function to `this.setState()` instead of an object. This function receives the previous state and props as arguments, which is useful when the new state depends on the previous state.