Module-18) React - Components, State, Props

1. Introduction to React.js

THEORY EXERCISE

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

Ans: React.js is a JavaScript library developed by Facebook for building user interfaces (UIs), particularly for single-page applications (SPAs). It allows developers to build reusable UI components and manage the state of an application efficiently.

- React uses a declarative programming style, meaning you describe what the UI should look like for a given state, and React updates the actual DOM to match it using a technique called the Virtual DOM, which makes updates fast and efficient.
- **Key Features of React.js:**
  - **Component-based architecture:** Break UI into reusable components.
  - **Virtual DOM:** Efficient rendering by only updating parts of the actual DOM.
  - **JSX (JavaScript XML):** Syntax extension to write HTML-like code inside JavaScript.
  - **One-way data binding:** Data flows in one direction, making it easier to debug.
  - **Hooks:** Functions that let you use state and lifecycle features in functional components.

❖ **How React.js is Different from Other JavaScript Frameworks/Libraries:**

Feature	React.js	Angular (Framework)	Vue.js (Framework)	jQuery (Library)
Type	Library (UI-focused)	Full-featured MVC Framework	Progressive Framework	DOM Manipulation Library

Developed by	Facebook	Google	Evan You (ex-Google)	John Resig
Architecture	Component-based	Component-based + MVC	Component-based	Imperative
DOM	Virtual DOM	Real DOM (with optimizations)	Virtual DOM	Real DOM
Data Binding	One-way binding	Two-way & one-way	Two-way & one-way	Manual
Learning Curve	Moderate	Steep (TypeScript, RxJS, etc.)	Easy to Moderate	Very Easy

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

Ans: React is a popular JavaScript library developed by Facebook for building user interfaces. Its strength lies in a few core principles, which make it powerful, efficient, and developer-friendly. Two of the most important principles are:

1) Virtual DOM (Document Object Model)

- **What is it?**
  - The Virtual DOM is a lightweight, in-memory representation of the actual DOM elements in the browser.
- **How it works:**
  - When the state or data in a React app changes, React creates a new Virtual DOM.
  - It then compares the new Virtual DOM with the previous one using a diffing algorithm.
  - Only the parts of the actual DOM that changed are updated — this process is called reconciliation.
- **Why it matters:**

- Updating the real DOM is slow; the Virtual DOM improves performance by minimizing direct DOM manipulation.
- It ensures faster and more efficient UI updates.

2) Component-Based Architecture

- **What is it?**
  - In React, the UI is built using components, which are independent, reusable pieces of code that represent parts of the user interface.
- **Types of components:**
  - **Functional components:** These are JavaScript functions that return JSX (UI).
  - **Class components** (older): JavaScript classes that extend React.Component.
- **Benefits:**
  - **Reusability:** Components can be reused across different parts of the application.
  - **Separation of concerns**: Each component manages its own logic and UI.
  - **Maintainability:** Easier to manage, test, and update small pieces of the UI.

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

Ans: There are many advantages of React.js in web development.

- **Component-Based Architecture**
  - Breaks UI into reusable, self-contained components.
  - Promotes modularity, making code easier to read, maintain, and test.
- **Virtual DOM for Better Performance**
  - Uses a **virtual DOM** to update only the changed parts of the UI.
  - Reduces expensive DOM operations, resulting in **faster performance** and **smooth user experience**.
- **Reusability and Maintainability**

- Components can be reused across multiple parts of the app or even across projects.
- Makes large-scale application development more **organized** and **efficient**.

➢ **Fast Rendering with One-Way Data Binding**

- React uses **unidirectional (one-way) data flow**, which makes data tracking more predictable.
- Easier to debug because data flows in a single direction.

➢ **Strong Ecosystem and Community Support**

- Large and active developer community.
- Huge number of **third-party libraries, tools, and extensions** available.

➢ **JSX – Simpler Syntax**

- Uses JSX, which allows writing HTML-like syntax in JavaScript.
- Makes it easier to **visualize and design** component structure.

➢ **Support for Modern Development Tools**

- Works seamlessly with tools like Redux, React Router, Webpack, and Babel.
- Has great support for **unit testing and integration testing** tools (like Jest, Enzyme).

➢ **SEO-Friendly (with SSR)**

- Can be rendered on the server side using Next.js or similar frameworks.
- Helps improve **SEO** performance for web apps.

➢ **Easy to Learn and Use**

- Simple to understand for developers familiar with JavaScript.
- Clean component-based design makes it **beginner-friendly**.

➢ **Backed by Facebook**

- Maintained and constantly improved by Meta (Facebook)
- Long-term support and regular updates.

2. JSX (JavaScript XML)

THEORY EXERCISE

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

Ans: 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 within JavaScript.

- **Example: const element = <h1>Hello, world!</h1>;**

❖ **Why is JSX used in React?**

✓ JSX is used in React for the following reasons:

1. **Improves Readability and Structure**
   - ✓ JSX makes it easy to visualize the component structure.
   - ✓ It closely resembles HTML, making the code more familiar and readable.
2. **Combines Markup and Logic**
   - ✓ Allows writing **UI structure and logic in the same file**.
   - ✓ Helps in building reusable components with both behavior and layout.
3. **More Powerful than HTML**
   - ✓ JSX allows you to embed JavaScript expressions using {}.
4. **Tooling and Compile-Time Checks**
   - ✓ Tools like Babel check JSX syntax and catch errors early.
   - ✓ JSX helps with **auto-completion, linting, and formatting** in modern editors.
5. **Virtual DOM Efficiency**
   - ✓ JSX is compiled into React.createElement() calls, which are optimized for the virtual DOM to efficiently update the UI.

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

Ans: <u>JSX Different from Regular Javascript</u>

Feature	JSX	Regular JavaScript
Syntax	HTML-like syntax in JavaScript	Pure JavaScript syntax

Compilation	Needs to be compiled by Babel to JS	Runs directly in browsers
Purpose	Used to describe UI in React	Used to write logic, functions, variables etc.
Tags/Elements	Looks like HTML (e.g., , <h1>)	Uses JS methods and objects
Output	Translates to React.createElement()	Outputs standard JS objects, functions etc.

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

- **<u>Can You Write JavaScript Inside JSX?</u>**
  - **Yes, you can!** You can embed JavaScript expressions inside JSX using curly braces {}.

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

Ans: In JSX, curly braces {} are used to embed JavaScript expressions inside HTML-like markup. They allow you to dynamically insert values, call functions, and apply logic within JSX.

- **<u>Why are curly braces important?</u>**
  - **Dynamic Content Rendering**
    - ✓ Curly braces let you insert dynamic values into your UI.
  - **Embed JavaScript Expressions**
    - ✓ Only JavaScript expressions (not full statements) can be written inside {}.
  - **Use Functions or Logic Inside JSX**
    - ✓ You can call functions or methods inside curly braces to generate content.
  - **Loop Through Data**
    - ✓ Useful for rendering lists with methods like .map().
  - **Conditional Rendering**
    - ✓ Use expressions with conditionals to render elements selectively.

3. Components (Functional & Class Components)

THEORY EXERCISE

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

Ans: Components are the building blocks of any React application. They are independent, reusable pieces of UI (User Interface) that can be combined to create complex user interfaces. Essentially, a component is a JavaScript function or class that optionally accepts inputs (called "props") and returns a React element that describes what should appear on the screen.

Difference Between Functional components and Class componets

Feature	Functional Components	Class Components
State Management	It can use Hooks like useState, useReducer	It uses this.state and this.setState()
Lifecycle Methods	It uses useEffect Hook for lifecycle methods	It uses traditional lifecycle methods like componentDidMount, componentWillUnmount
Rendering	Returns JSX directly inside the function	Uses a render() method to return JSX
Performance	Faster and more lightweight due to simpler structure	Slightly heavier due to the overhead of class instances
Hooks	Can use React hooks (useState, useEffect, etc.)	Cannot use hooks; relies on lifecycle methods and state
This Keyword	Does not use this keyword	Uses this to access props and state
Code Complexity	Less boilerplate code, easier to write and understand	More boilerplate code, especially for state and methods
Event Handling	Simple and direct event handling	Requires method binding for event handling

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

Ans: To pass data to a component using props in a component-based framework (like React, Vue, or Angular), you generally follow these steps:

1. **Define the data in the parent component**
   - This is where the data originates. It could be state, a variable, or even data fetched from an API.
2. **Pass the data as an attribute to the child component**
   - When you render the child component within the parent, you add attributes to the child component's tag. These attributes become the "props" that the child component receives.
3. **Access the data in the child component**
   - The child component receives these props as an argument (often an object) in its function or class constructor.

- **Example: 1. Parent Component**

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

function ParentComponent() {
  const userName = "Karan Sharma";
  const userAge = 25;
  const hobbies = ["Reading", "Playing", "Coding"];

  return (
    <div>
      <h1>Parent Component</h1>
      {/* Passing data to ChildComponent using props */}
      <ChildComponent
        name={userName}
        age={userAge}
        hobbies={hobbies}
        isLoggedIn={true}
      />
    </div>
  );
```

```
}

export default ParentComponent;
```

2. Child Component

```
import React from 'react';
function ChildComponent(props) {
// props is the argument that holds all the passed data
return ( <div>
        <h2>Child Component</h2>
        <p>Name: {props.name}</p>
        <p>Age: {props.age}</p>
        <p>Hobbies:</p>
<ul> {props.hobbies.map((hobby, index) => (
    <li key={index}>{hobby}</li>
    ))}
</ul>
<p>User is logged in: {props.isLoggedIn ? "Yes" : "No"}</p>
</div>
);
} export default ChildComponent;
```

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

Ans: Render in React JS is a fundamental part of class components. It is used to display the component on the UI returned as HTML or JSX components. The ReactDOM.render() function takes two arguments, HTML code and an HTML element.

- In React class components, the render() method is a crucial and mandatory lifecycle method. Its primary role is to return the JSX (JavaScript XML) that describes the UI you want to display on the screen.

➢ Purpose of render() method

- React renders HTML to the web page by using a function called render().
- The purpose of the function is to display the specified HTML code inside the specified HTML element.
- In the render() method, we can read props and state and return our JSX code to the root component of our app.
- In the render() method, we cannot change the state, and we cannot cause side effects(such as making an HTTP request to the webserver).

➢ Example: 1. Classcomponent .js

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

class Classcompo extends Component {
  render() {
    return (
      <div>
        <h1>Hello this class component</h1>
        <h2>First Program to Class Component</h2>
      </div>
    )
  }
}

export default Classcompo;
```

➢ 2. App .js

```
import React from "react";
```

```
import Classcompo from "./Compoenet/Classcompo";
function App(){
        return(
       <div>
                    <h1>This is App </h1>
                    < Classcompo  />
</div>
)
}
  export default App;
```

4. Props and State

THEORY EXERCISE

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

Ans: Props are known as properties it can be used to pass data from one component to another. Props cannot be modified, read-only, and Immutable.

❖ How are props different from state

PROPS	STATE
The Data is passed from one component to another.	The Data is passed within the component only.
It is Immutable (cannot be modified).	It is Mutable (can be modified).
Props can be used with state and functional components.	The state can be used only with the state components/class component (Before 16.0).
Props are read-only.	The state is both read and write.

Props are known as properties it can be used to pass data from one component to another. Props cannot be modified, read-only, and Immutable.	The state represents parts of an Application that can change. Each component can have its State. The state is Mutable and It is local to the component only.

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

Ans: In React, the state refers to an object that holds information about a component's current situation. This information can change over time, typically as a result of user actions or data fetching, and when state changes, React re-renders the component to reflect the updated UI.

- Whenever state changes, React re-renders the component to reflect the updated data. This enables you to build dynamic UIs that respond to user interactions.
- **Key Concepts of State**
    1. **Local and Private**
        - ✓ State is local to the component it's defined in. It's encapsulated within that component and not directly accessible to other components, unless explicitly passed down via props. Each instance of a component has its own independent state.
    2. **Reactivity and Re-rendering**
        - ✓ When state changes, React automatically triggers a re-render of the component to reflect the updated state in the UI. React ensures that only the parts of the UI that depend on the state are re-rendered, improving performance.
    3. **Mutable Within the Component**
        - ✓ When state changes, React automatically triggers a re-render of the component to reflect the updated state in the UI. React ensures that only the parts of the UI that depend on the state are re-rendered, improving performance.
    4. **Initial Value**
        - ✓ State needs an initial value when the component first mounts.

❖ **Syntax:**

- **const [state, setState] = useState(initialState);**
  - ✓ **state:** The current state value.
  - ✓ **setState:** A function that is used to update the state.
  - ✓ **initialState:** The initial value that the state will hold when the component is first rendered.

❖ How State is Used to Manage Component Data

- ➢ React provides two primary ways to manage state, depending on whether you're using depending on whether you're using **Functional Components** (with Hooks) or **Class Components.**

1. Functional Components (with the useState Hook)

- ➢ The useState Hook is the most common and recommended way to manage state in functional components.
- ➢ **Syntax: import React, { useState } from 'react';**

```
function MyComponent() {
const [stateVariable, setStateVariable] = useState(initialValue);
// ... rest of your component logic
}
```

  - ✓ useState is a Hook (a special function that lets you "hook into" React features in functional components).
  - ✓ It returns an array with two elements
    - **stateVariable:** The current value of the state.
    - **setStateVariable:** A function that you use to update the state variable.
  - ✓ **initialValue:** The value you want the state to have when the component first renders.
- ➢ **Example : import React, { useState } from 'react';**

```
function FuncState() {
    // state define and method
  const [name, setname] = useState("uttam");
  const [count, setcount] = useState(0);
console.log(name);
```

```
return(
  <div>
    <h1>This is State in Function <h1>
    <h1>hello name :- {name} </h1>
   <button onClick={() => setname("Pintu")}>Change Name</button>
  <button onClick={() => setname("Manish")}>Change Name</button>
 <h1>Hello count :- {count}</h1
 <button onClick={() => setcount(count + 1)}>Increment</button>
 <button onClick={() => setcount(count + 2)}>Incmrent by 2</button>
<button onClick={() => setcount(count - 1)}>Decrement</button>
<button onClick={() => setcount(0)}> Zero</button>
</div>
      )
    }
    export default FuncState
```

- **How it manages data:**
  - ✓ **The count variable holds the current value of our counter.**
  - ✓ **When the user clicks "Increment" or "Decrement," we call setcount().**
  - ✓ **Calling setcount() tells React that the count state has changed.**
  - ✓ **React then automatically re-renders the FuncState component, and the count value**

2. Class Components (using this.state and this.setState())

- This is the traditional way to manage state in class components. While still valid, it's less common in new React development.
- **Syntax:**

```
import React, { Component } from 'react';
class MyClassComponent extends Component {
  constructor() {
    super();
```

```
this.state = {     // Define initial state properties here
someData: 'initial value',
anotherValue: 0  }; }

// Method to update state

updateState = () => {

this.setState({  someData: 'new value',

anotherValue: this.state.anotherValue + 1

});

};

render() {
return (
// Access state using this.state.propertyName
<p>{this.state.someData}</p>
);
}
}
```

- Example:

```
import React, { Component } from 'react'
class ClassState extends Component {
    constructor(){
        super();
        this.state = {
        name : “Rahul Kumar”,
        count : 0,
}
}
Incrementby2 = () =>{
        this.setState({
                count : this.state + 2
})
}
```

```
render() {
        console.log(this.state)
 return (
        <div>
        <h1>hello this class in State</h1>
        // use state this.state.objectname
        <h1>Hello name :- {this.state.name}</h1>
<button onClick={() => this.setState({ name: "Mohan" })}>Change name</button>
<button onClick={() => this.setState({ name: "Karan" })}>Chnage name2</button>.

        <h1>Hello count :- {this.state.count}</h1>

<button onClick={() => this.setState({ count: this.state.count + 1 })}>increment</button>
<button onClick={() => this.setState({ count: this.state.count - 1 })}>decrment</button>
<button onClick={() => this.setState({ count: 0 })}>Zero</button>
<button onClick={this.incrementby2}>Incmrent by 2</button>
        </div>
    ) }
    }
```

- **How it manages data:**
  - ✓ **The name property in this.state holds the current name.**
  - ✓ **The constructor is where you define the initial state.**
  - ✓ **When Incrementby2 is called (e.g., by clicking the button), this.setState() is used to update the count property.**
  - ✓ **this.setState() signals to React that the component's state has changed, prompting a re-render.**

- ✓ **The render() method then uses the updated this.state.isOn value to determine the button text and status message.**

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

Ans: this.setState() is a fundamental method used in React class components for managing and updating their internal state. It's crucial because it's the only correct way to trigger a re-render of the component and its children when the state changes.

❖ **Why this.setState() is Used in Class Components**

➢ **Immutability of State**

- ✓ In React, the state object should be treated as immutable. You should *never* directly modify this.state like this.state.count = 5;. Doing so will not trigger a re-render, and your UI will not reflect the changes. setState() provides a controlled and predictable way to update the state.

➢ **Triggering Re-renders**

- ✓ When this.setState() is called, React knows that the component's state has potentially changed. This signals React to:
  - Re-run the component's render() method.
  - Compare the new virtual DOM tree with the previous one.
  - Efficiently update the actual DOM to reflect only the necessary changes (a process called "reconciliation").

➢ **Batching Updates**

- ✓ React often batches multiple setState() calls together for performance optimization. If you call setState() multiple times within a single event handler, React might consolidate them into a single update to avoid unnecessary re-renders.

➢ **Asynchronous Nature**

- ✓ setState() is asynchronous. This means that React might not update the state immediately after you call setState(). If you need to perform actions *after* the state has definitely updated, you should use the optional callback function of setState().

❖ **How this.setState() Works**

1. **You call this.setState() with the desired state changes (either as an object or a function).**
2. **React queues the update.**
3. **React batches multiple updates for efficiency.**
4. **React calculates the new state by shallowly merging your update with the current state (or by calling your updater function).**
5. **React triggers a re-render of the component.**
6. **The render() method is called, creating a new virtual DOM tree.**
7. **React's reconciliation algorithm compares the new virtual DOM with the previous one.**
8. **React updates the actual DOM only where necessary.**
9. **If a callback function was provided, it's executed after the re-render is complete.**