Module 18 :- React js Theory

1) Components (Functional & Class Components) :-

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

ANS. In React, components are the building blocks of a React application. They are independent, reusable pieces of UI that define how a portion of the interface should appear and behave.

- **The difference between functional components and class components.**

Features	Functional Components	Class Components
Syntax	Functional components are written as a JavaScript function.	Class components are written as a JavaScript class.
State and Lifecycle Methods	Functional components do not have a state or lifecycle methods.	Class components have a state and can implement lifecycle methods like componentDidMount and componentDidUpdate.
Performance	Faster as they do not have state and lifecycle, react needs to do less work to render these components.	Slower as they have state and lifecycle, react needs to do comparatively more work to render these components.
Code Length	Functional components tend to be shorter and more concise	Class components require the boilerplate code, such as a constructor method and the use of “this” to access props and state.
Usage of “this	Functional components do not use “this” at all, which makes them easier to understand for beginners.	Class components use the “this” keyword is used to refer to the current instance of the component which can be confusing for new developers.

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

ANS. In React, props (short for "properties") are used to pass data from a parent component to a child component. Here's a step-by-step explanation:

1. Define the prop: In the parent component, define the data you want to pass to the child component.
2. Pass the prop: Add an attribute to the child component's JSX element with the same name as the prop. Assign the defined data to this attribute.
3. Access the prop: In the child component, access the passed prop using the props object or destructuring.

- **Example :-**

```
// ParentComponent
function ParentComponent() {
  const name = "John";
  return (
    <div>
      <ChildComponent name={name} />
    </div>
  );
}

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

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

ANS. :- In React class components, the render() method is responsible for describing what the UI should look like. It is a required method in every class component and returns JSX (JavaScript XML), which is then rendered to the DOM by React.

- **How's it Works :-**
- When a class component is instantiated, React calls the `render()` method to determine what to render.
- The `render()` method returns the JSX elements, which are then rendered to the DOM.
- When the component's state or props change, React calls `render()` again to determine if the UI needs to be updated.

➢ render() should be a pure function, meaning it should not have any side effects, such as setting state or making API calls.
➢ render() should not be used to handle user interactions or events; instead, use event handlers.

- **Example :-**

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

2) Props and State :-

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

ANS. :- In React, "props" is short for "properties". Props are inputs passed from a parent component to a child component. They are immutable, meaning they cannot be changed by the child component.

- **Difference between Props and 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.**

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

ANS. In React, state refers to the data that changes within a component over time. It's a way to store and manage data that affects the component's behavior and rendering.

How State is Used:

1. **Initialization**: State is initialized in the component's constructor or using the useState hook.
2. **Updating**: State is updated using the setState method or the useState hook's update function.
3. **Rendering**: The component re-renders when the state changes, reflecting the new data.

Benefits of State:

1. **Dynamic UI**: State enables you to create dynamic user interfaces that respond to user interactions and changing data.
2. **Component autonomy**: State allows components to manage their own data, making them more self-contained and reusable.

```
}
```

- **Example using Class component in state**

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

- **Example Functional Component with Hooks:-**

```
import { useState } from 'react';
function Counter() {
  const [count, setCount] = useState(0);
return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

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. It's a crucial method that helps manage the component's dynamic data.

1. **Passing an object**: You pass an object with the updated state values to this.setState().
2. **Merging with existing state**: React merges the new state object with the existing state, updating the relevant properties.
3. **Re-rendering**: After the state is updated, React re-renders the component with the new state.

- **Example** :-

```
class Counter extends React.Component {
 constructor(props) {
  super(props);
  this.state = { count: 0 };
 }

 increment = () => {
  this.setState({ count: this.state.count + 1 });
 };
render() {
  return (
   <div>
    <p>Count: {this.state.count}</p>
    <button onClick={this.increment}>Increment</button>
   </div>
  );
 }
}
```