**React - Components, State, Props**

**THEORY EXERCISE**

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

**Answer:**

**Components** in React are reusable, independent pieces of code that return React elements (UI elements). They let you split the UI into smaller pieces and work on them independently.

**Types of components:**

**Functional Components** – Written as JavaScript functions. They are simple, readable, and mostly used with React Hooks.

**Class Components** – Written as ES6 classes. They used to manage state and lifecycle methods before React Hooks were introduced.

| **Feature** | **Functional Component** | **Class Component** |
| --- | --- | --- |
| Syntax | Function | Class |
| State Management | Uses Hooks (useState, etc.) | Uses this.state |
| Lifecycle Methods | Handled using Hooks (useEffect) | Uses lifecycle methods (componentDidMount, etc.) |
| Simplicity | More concise and easier to test | More boilerplate code |
| this keyword | Not used | Required (this.state, this.props) |

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

**Answer:**

In React, **props** (short for properties) are used to pass data from a parent component to a child component.

**Example:**

function Greeting(props) {

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

}

// Usage

<Greeting name="Amit" />

Here, name="Amit" is the prop passed to the Greeting component.

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

**Answer:**

In **class components**, the render() method is required and responsible for returning the JSX (React elements) to be displayed on the UI.

Example:

class WelcomeMessage extends React.Component {

  render() {

    return <h1>Welcome to React!</h1>;

  }

}

render() is called every time the component needs to update or re-render.

**LAB EXERCISE**

**Task 1: Create a functional component Greeting that accepts a name as a prop and displays "Hello, [name]!".**

import React from 'react';

function Greeting(props) {

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

}

<Greeting name="Amit" /> // Output: Hello, Amit

**Task 2: Create a class component WelcomeMessage that displays "Welcome to React!" and a render() method.**

import React, { Component } from 'react';

class WelcomeMessage extends Component {

  render() {

    return <h1>Welcome to React!</h1>;

  }

}

<WelcomeMessage /> // Output: Welcome to React!

**THEORY EXERCISE**

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

**Answer:**

**Props (short for “properties”)** are used in React to pass data **from parent to child components**. Props are **read-only**, which means they cannot be modified by the receiving component.

**State** is a built-in React object used to hold **data that can change over time** within a component. Unlike props, state is **mutable**.

| **Feature** | **Props** | **State** |
| --- | --- | --- |
| Origin | Passed by parent | Maintained inside component |
| Mutability | Immutable (read-only) | Mutable (can change) |
| Usage | For configuration/input | For local data and interactivity |
| Controlled by | Parent component | Component itself |

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

**Answer:**

**State** in React is a JavaScript object that stores **dynamic data** related to a component. It allows a component to **react to user input, server responses**, or any other event that affects the UI.

In **functional components**, you use the useState hook:

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

In **class components**, you use this.state and this.setState():

this.state = { count: 0 };

this.setState({ count: this.state.count + 1 });

State is re-evaluated whenever it changes, causing the component to **re-render** with the new data.

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

**Answer:**

this.setState() is used in **class components** to update the component’s state and trigger a **re-render** of the UI. It ensures the component updates **reactively** when data changes.

**How it works:**

* It merges the updated data with the existing state.
* It schedules a re-render.
* It may batch updates for performance.

Example:

this.setState({ count: this.state.count + 1 });

**LAB EXERCISE**

Task 1 : Create a React component UserCard that accepts name, age, and location as props and displays them in a card format.

import React from 'react';

function UserCard(props) {

  const { name, age, location } = props;

  return (

    <div style={{

      border: '1px solid #ccc',

      padding: '10px',

      borderRadius: '10px',

      width: '200px'

    }}>

      <h2>{name}</h2>

      <p>Age: {age}</p>

      <p>Location: {location}</p>

    </div>

  );

}

Usage:

<UserCard name="Amit" age={23} location="Kota" />

Task 2: Create a Counter component with a button that increments a count value using React state. Display the current count on the screen.

Using Functional Component with useState:

  import React, { useState } from 'react';

function Counter() {

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

  return (

    <div>

      <h2>Count: {count}</h2>

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

    </div>

  );

}

export default Counter;

OR Using Class Component:

import React, { Component } from 'react';

class Counter extends Component {

  constructor() {

    super();

    this.state = {

      count: 0

    };

  }

  increment = () => {

    this.setState({ count: this.state.count + 1 });

  }

  render() {

    return (

      <div>

        <h2>Count: {this.state.count}</h2>

        <button onClick={this.increment}>Increment</button>

      </div>

    );

  }

}

export default Counter;