**OBJECTIVES**

**1. Define Props**

**Props (short for "properties")** are read-only data passed from a parent component to a child component in React. They allow components to communicate and share data in a unidirectional flow (parent → child).

**Key Characteristics:**

* **Immutable:** Props cannot be modified by the child component.
* **Used for Configuration:** They help customize and configure components.
* **Can be Any Data Type:** Strings, numbers, arrays, objects, functions, etc.

**Example:**

jsx

// Parent Component

function App() {

return <Greeting name="Gopika" age={20} />;

}

// Child Component

function Greeting(props) {

return <h1>Hello, {props.name}! You are {props.age} years old.</h1>;

}

Here, name and age are props passed from App to Greeting.

**2. Explain Default Props**

**Default Props** are fallback values assigned to props if the parent component doesn't provide them. They ensure that a component works even when certain props are missing.

**How to Set Default Props:**

* Using defaultProps (class components & functional components).
* Using default parameters (functional components).

**Example:**

jsx

// Using defaultProps

function Greeting(props) {

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

}

Greeting.defaultProps = {

name: "Guest" // Default value if 'name' is not provided

};

// Using ES6 default parameters

function Greeting({ name = "Guest" }) {

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

}

If <Greeting /> is called without name, it will display "Hello, Guest!".

**3. Differences Between State and Props**

| **Feature** | **State** | **Props** |
| --- | --- | --- |
| **Mutability** | Mutable (can change using setState or useState) | Immutable (read-only) |
| **Ownership** | Managed within the component | Passed from parent to child |
| **Purpose** | Tracks internal component changes | Used for parent-child communication |
| **Initialization** | Defined inside the component (this.state or useState) | Passed as attributes (<Component prop="value" />) |
| **Updates** | Can be updated internally | Can only be updated by parent |
| **Example** | function Counter() {  const [count, setCount] = useState(0); // State (internal)  return <Display count={count} />;  //Passing state as a prop  } | function Display({ count }) {  // Props (external)  return <p>Count: {count}</p>;  } |

* count is state in Counter but becomes a **prop** in Display.

**4. Explain ReactDOM.render()**

ReactDOM.render() is a method used to render a React component into the DOM (usually in the root div of an HTML file).

**Syntax:**

jsx

ReactDOM.render(ReactElement, DOMContainer);

* **ReactElement:** The component or JSX to render.
* **DOMContainer:** The HTML element where the component will be mounted (e.g., document.getElementById('root')).

**Example:**

jsx

import React from 'react';

import ReactDOM from 'react-dom';

function App() {

return <h1>Hello, React!</h1>;

}

ReactDOM.render(<App />, document.getElementById('root'));

* This renders <App /> inside the <div id="root"></div> in HTML.

**Note:**

In **React 18+**, ReactDOM.render() is replaced with ReactDOM.createRoot() for better performance:

jsx

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<App />);