# **Fundamental Concepts of React: JSX, Components, State, and Props**

React is a JavaScript library developed by Facebook for building user interfaces. It has revolutionised the way web applications are built, offering a component-based approach that allows for the creation of reusable UI components. This report delves into the fundamental concepts of React, including JSX, components, state, and props, with code examples to illustrate how they contribute to building a React application.

## **1. JSX (JavaScript XML)**

### **Concept**

JSX is a syntax extension for JavaScript that allows developers to write HTML-like code within JavaScript. It makes it easier to create and visualize the structure of the UI. JSX is not mandatory in React, but it is highly recommended because of its readability and the way it integrates HTML and JavaScript seamlessly.

### **Code Example**

function Welcome(props) {

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

}

### **Explanation**

In the above example, the Welcome function is a React component that returns a JSX element. The JSX syntax looks similar to HTML but is transformed into JavaScript objects by tools like Babel. This transformation allows React to efficiently update and render the UI.

## **2. Components**

### **Concept**

Components are the building blocks of a React application. They encapsulate a part of the user interface and its related logic into a self-contained unit. Components can be functional or class-based, and they can be composed together to build complex UIs.

### **Code Example**

#### **Functional Component**

function Greeting() {

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

}

#### **Class Component**

class Greeting extends React.Component {

render() {

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

}

}

### **Explanation**

Functional components are simpler and are typically used for presenting UI elements without managing state. Class components, on the other hand, are more powerful and can manage local state and lifecycle methods. React has been moving towards functional components with hooks to manage state and side effects, making functional components capable of almost everything that class components can do.

## **3. State**

### **Concept**

State is an object that holds some information that may change over the lifetime of the component. State is managed within the component and can be updated using the setState method in class components or the useState hook in functional components.

### **Code Example**

#### **Using Class Component**

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>

);

}

}

#### **Using Functional Component with Hooks**

function Counter() {

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

return (

<div>

<p>Count: {count}</p>

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

</div>

);

}

### **Explanation**

State allows React components to respond to user input, network responses, and other events. The useState hook in functional components is a modern way to manage state, providing a simpler and more readable syntax compared to class components.

## **4. Props**

### **Concept**

Props (short for properties) are read-only attributes that are passed from a parent component to a child component. Props allow data to flow from the top of the application to the bottom, following the unidirectional data flow principle of React.

### **Code Example**

function Welcome(props) {

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

}

function App() {

return <Welcome name="Alice" />;

}

### **Explanation**

In the above example, the Welcome component receives the name prop from its parent component, App. Props make components reusable by allowing them to be customized with different inputs. Unlike state, props cannot be modified by the component that receives them.

## **Building a React Application**

### **Combining Concepts**

To illustrate how JSX, components, state, and props come together in a React application, let's build a simple counter application.

### **Complete Example**

import React from 'react';

import ReactDOM from 'react-dom';

function DisplayCount(props) {

return <h1>Count: {props.count}</h1>;

}

function Counter() {

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

return (

<div>

<DisplayCount count={count} />

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

</div>

);

}

function App() {

return (

<div>

<h1>My Counter App</h1>

<Counter />

</div>

);

}

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

### **Explanation**

1. **JSX**: Used to define the structure of the UI within JavaScript.
2. **Components**: DisplayCount, Counter, and App are functional components.
3. **State**: Managed in the Counter component using the useState hook.
4. **Props**: Passed from the Counter component to the DisplayCount component to display the current count.