**Introduction to JSX**

**What is JSX?** JSX (JavaScript XML) is a syntax extension for JavaScript that allows you to write HTML-like code in your JavaScript files. It's used to create React elements, which are then rendered to the DOM.

**JSX Syntax and Usage** JSX syntax is similar to HTML, but with some key differences. You can use JSX to create elements, assign props, and embed expressions.

**Example**

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

**Embedding Expressions in JSX**

**Embedding JavaScript Expressions** You can embed JavaScript expressions in JSX using curly braces **{}**.

**Example**

const name = 'John';

const element = <h1>Hello, {name}!</h1>;

**Embedding Conditional Statements** You can use conditional statements like **if and ternary operators**to conditionally render elements.

**Example**

const isAdmin = true;

const element = <h1>{isAdmin? 'Admin' : 'User'}</h1>;

**JSX vs. HTML**

**Similarities**JSX and HTML share many similarities, such as using tags to define elements and attributes to add properties.

**Differences** JSX is more flexible and powerful than HTML, allowing you to embed JavaScript expressions and use React features like components and state.

**Components**

**What is a Component?** A component is a reusable piece of code that represents a UI element.

**Types of Components** There are two main types of components: Function components and Class components.

**Function vs. Class Components**

**Function Components** Function components are pure functions that take in props and return JSX. They are simpler and more concise than Class components.

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

**Class Components** Class components are classes that extend the **React.Component** class. They have more features than Function components, such as state and lifecycle methods.

**Example**

// components/Hello.js

import React, { Component } from 'react';

class Hello extends Component {

render() {

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

}

}

export default Hello;

**Creating and Exporting Components**

**Creating Components** You can create components using the **function or class** keyword.

**Exporting Components** You can export components using the **export**keyword, making them available for use in other files.

**Function based export:**

// components/Hello.jsx

function Hello(props) {

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

}

**export** default Hello;

**Class based export:**

// components/Hello.js

import React, { Component } from 'react';

class Hello extends Component {

render() {

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

}

}

**export default Hello;**

**Creating and Exporting a Component with Multiple Exports**

// components/Hello.js

function Hello(props) {

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

}

function Goodbye(props) {

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

}

**export { Hello, Goodbye };**

**Creating and Exporting a Component with a Default Export and Named Exports**

// components/Hello.js

function Hello(props) {

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

}

function Goodbye(props) {

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

}

**export { Goodbye };**

**export default Hello;**

**Props and PropTypes**

**What are Props?** Props ("properties") are read-only values passed from a parent component to a child component.

**PropTypes**PropTypes are a way to validate the types of props passed to a component.

**Example**

import PropTypes from 'prop-types';

function Hello(props) {

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

}

Hello.propTypes = {

name: PropTypes.string.isRequired

};

**State and setState**

**What is State?** State is an object that stores data that can change over time.

**setState setState** is a method used to update the state of a component.

**Example**

class Counter extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

handleClick = () => {

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

}

render() {

return <p>Count: {this.state.count}</p>;

}

}

**Handling Events**

**What are Events?** Events are actions triggered by user interactions, such as clicks or key presses.

**Handling Events** You can handle events by attaching event listeners to elements and calling event handler functions.

**Example**

function handleClick() {

console.log('Button clicked!');

}

return <button onClick={handleClick}>Click me!</button>;

**React Hooks**

**Introduction to Hooks** Hooks are a way to use state and other React features in Function components.

**useState**  is a Hook that allows you to add state to Function components.

**Example**

import { useState } from 'react';

function Counter() {

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

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

}

**useEffect**  is a Hook that allows you to run side effects, such as making API requests or setting timers.

**Example**

import { useState, useEffect } from 'react';

function FetchData() {

const [data, setData] = useState([]);

useEffect(() => {

fetch('https://api.example.com/data')

.then(response => response.json())

.then(data => setData(data));

}, []);

return <ul>{data.map(item => <li>{item}</li>)}</ul>;

}

**useContext** is a Hook that allows you to access context (shared state) in a component. Here's an example:

import { useContext } from 'react';

import { ThemeContext } from './theme-context';

function Button() {

const theme = useContext(ThemeContext);

return <button style={{ backgroundColor: theme.background, color: theme.text }}>Click me!</button>;

}

**Custom Hooks** are reusable functions that use React Hooks to manage state and side effects. Here's an example:

import { useState, useEffect } from 'react';

const useFetch = (url) => {

const [data, setData] = useState([]);

const [error, setError] = useState(null);

const [loading, setLoading] = useState(false);

useEffect(() => {

setLoading(true);

fetch(url)

.then(response => response.json())

.then(data => setData(data))

.catch(error => setError(error))

.finally(() => setLoading(false));

}, [url]);

return { data, error, loading };

};