Introduction to React

React is a popular JavaScript library for building user interfaces, developed and maintained by Facebook. It allows developers to create large web applications that can update and render efficiently in response to data changes.

**Definition:** React is a declarative, efficient, and flexible JavaScript library for building user interfaces.

**Key Features:**

* Declarative: React makes it painless to create interactive UIs.
* Component-Based: Build encapsulated components that manage their state and compose them to make complex UIs.
* Learn Once, Write Anywhere: Develop new features in React without rewriting existing code.

**Example with Description:**

import React from 'react';  
import ReactDOM from 'react-dom';  
  
// Functional component rendering a simple message  
function App() {  
 return <h1>Hello, React!</h1>; // This returns an HTML-like structure called JSX  
}  
  
ReactDOM.render(<App />, document.getElementById('root')); // Renders the App component to the

In the above example, the App component renders a "Hello, React!" message, and the ReactDOM.render function mounts it to the root element in the DOM.

**Environment Setup**

To start with React development:

1. **Install Node.js**: Download and install Node.js from <https://nodejs.org/>. Node.js is required to run the React development environment.
2. **Install a Text Editor**: VS Code is recommended as it provides extensive support for JavaScript and React.
3. **Create a New React App**: Use the following command:

npx create-react-app my-app  
cd my-app  
npm start

This sets up a boilerplate React project.

1. Your React application will run at http://localhost:3000.

**Example:** After running the above commands, the default React app will display "Edit src/App.js and save to reload."

**Components in React**

**Definition:** Components are the building blocks of a React application. They are independent, reusable pieces of UI.

**Functional Components**

A functional component is a JavaScript function that accepts props as an argument and returns a React element.

**Example with Description:**

function Welcome() {  
 return <h1>Welcome to React!</h1>; // JSX returns a simple heading  
}

In this example, Welcome is a functional component that outputs a heading.

**Class Components**

A class component requires extending the React.Component class and implementing the render() method.

**Example with Description:**

import React, { Component } from 'react';  
  
class Welcome extends Component {  
 render() {  
 return <h1>Welcome to React!</h1>; // Render method returns JSX  
 }  
}

In this example, the Welcome class component provides the same functionality but with a different syntax.

**Props and States**

**Props**

**Definition:** Props (short for properties) are read-only inputs passed to components to customize their behavior or appearance.

**Example with Description:**

function Greeting(props) {  
 return <h1>Hello, {props.name}!</h1>; // Props are used to display dynamic data  
}  
  
function App() {  
 return <Greeting name="Alice" />; // Passing "Alice" as a prop to Greeting  
}

In this example, the Greeting component dynamically displays the name passed via props.

**States**

**Definition:** State is a built-in object used to store dynamic data and control component behavior.

**Example with Description:**

import React, { useState } from 'react';  
  
function Counter() {  
 const [count, setCount] = useState(0); // Initial state is set to 0  
  
 return (  
 <div>  
 <p>Count: {count}</p> // Displays the current count  
 <button onClick={() => setCount(count + 1)}>Increment</button> // Updates state on button click  
 </div>  
 );  
}

In this example, the useState hook manages the count state, which updates when the button is clicked.

**React Events**

**Definition:** React events are triggered by user actions such as clicks, form submissions, or keyboard inputs. React uses camelCase for event handlers.

**Example with Description:**

function ButtonClick() {  
 function handleClick() {  
 alert('Button clicked!'); // Alert triggered on button click  
 }  
  
 return <button onClick={handleClick}>Click Me</button>; // Attaching event handler  
}

In this example, the handleClick function is triggered when the button is clicked.

**React Forms**

**Definition:** Forms in React are used to collect user input. Controlled components are commonly used, where form data is managed by the component's state.

**Example with Description:**

import React, { useState } from 'react';  
  
function FormExample() {  
 const [name, setName] = useState(''); // State to track form input  
  
 function handleSubmit(event) {  
 event.preventDefault(); // Prevents page refresh  
 alert(`Name: ${name}`); // Displays the entered name  
 }  
  
 return (  
 <form onSubmit={handleSubmit}>  
 <input  
 type="text"  
 value={name} // Binds input value to state  
 onChange={(e) => setName(e.target.value)} // Updates state on input change  
 />  
 <button type="submit">Submit</button>  
 </form>  
 );  
}

In this example, the form collects the user's name and displays it in an alert on submission.

**React Router**

**Definition:** React Router is a library for routing in React applications. It enables navigation between views while maintaining the application state.

**Example with Description:**

import React from 'react';  
import { BrowserRouter as Router, Route, Link } from 'react-router-dom';  
  
function Home() {  
 return <h2>Home</h2>; // Displays Home content  
}  
  
function About() {  
 return <h2>About</h2>; // Displays About content  
}  
  
function App() {  
 return (  
 <Router>  
 <nav>  
 <Link to="/">Home</Link>  
 <Link to="/about">About</Link>  
 </nav>  
 <Route exact path="/" component={Home} />  
 <Route path="/about" component={About} />  
 </Router>  
 );  
}

In this example, clicking on the "Home" or "About" links changes the view without reloading the page.

**React CSS**

**Definition:** CSS in React is used to style components. You can use traditional CSS files, inline styles, or libraries like styled-components.

**Example with Description:**

function StyledComponent() {  
 const style = { color: 'blue', fontSize: '20px' }; // Inline styling  
  
 return <p style={style}>This is a styled component!</p>; // Applying inline styles  
}

In this example, the text is styled with blue color and a font size of 20px using inline styles.

**More React Features**

**React Lists**

**Definition:** Lists are used to display multiple similar elements dynamically.

**Example with Description:**

function NumberList() {  
 const numbers = [1, 2, 3, 4, 5];  
 return (  
 <ul>  
 {numbers.map((number) => (  
 <li key={number}>{number}</li> // Key ensures unique identification of each list item  
 ))}  
 </ul>  
 );  
}

In this example, the numbers array is dynamically rendered as a list of items.

**React Refs**

**Definition:** Refs provide a way to access DOM nodes or React elements directly.

**Example with Description:**

import React, { useRef } from 'react';  
  
function FocusInput() {  
 const inputRef = useRef(null); // Ref to access the input element  
  
 function handleFocus() {  
 inputRef.current.focus(); // Focuses the input element  
 }  
  
 return (  
 <div>  
 <input ref={inputRef} type="text" />  
 <button onClick={handleFocus}>Focus Input</button>  
 </div>  
 );  
}

In this example, the inputRef ref is used to programmatically focus the input element.

**React Table**

**Definition:** Tables in React are used to display tabular data dynamically.

**Example with Description:**

function DataTable() {  
 const data = [  
 { id: 1, name: 'Alice', age: 25 },  
 { id: 2, name: 'Bob', age: 30 },  
 ];  
  
 return (  
 <table>  
 <thead>  
 <tr>  
 <th>ID</th>  
 <th>Name</th>  
 <th>Age</th>  
 </tr>  
 </thead>  
 <tbody>  
 {data.map((item) => (  
 <tr key={item.id}>  
 <td>{item.id}</td>  
 <td>{item.name}</td>  
 <td>{item.age}</td>  
 </tr>  
 ))}  
 </tbody>  
 </table>  
 );  
}

In this example, the data array is rendered as rows in a table, with columns for ID, name, and age.

**React Hooks**

**useEffect**

**Definition:** useEffect is a hook that lets you perform side effects in functional components, such as data fetching, subscriptions, or manual DOM updates.

**Example with Description:**

import React, { useState, useEffect } from 'react';  
  
function DataFetcher() {  
 const [data, setData] = useState(null);  
  
 useEffect(() => {  
 fetch('https://api.example.com/data')  
 .then((response) => response.json())  
 .then((data) => setData(data)); // Updates state with fetched data  
 }, []); // Empty dependency array ensures effect runs only once  
  
 return <div>{data ? JSON.stringify(data) : 'Loading...'}</div>;  
}

**useReducer**

**Definition:** useReducer is an alternative to useState for managing more complex state logic.

**Example with Description:**

import React, { useReducer } from 'react';  
  
function reducer(state, action) {  
 switch (action.type) {  
 case 'increment':  
 return { count: state.count + 1 };  
 case 'decrement':  
 return { count: state.count - 1 };  
 default:  
 throw new Error();  
 }  
}  
  
function Counter() {  
 const [state, dispatch] = useReducer(reducer, { count: 0 });  
  
 return (  
 <div>  
 <p>Count: {state.count}</p>  
 <button onClick={() => dispatch({ type: 'increment' })}>+</button>  
 <button onClick={() => dispatch({ type: 'decrement' })}>-</button>  
 </div>  
 );  
}

**useMemo**

**Definition:** useMemo memoizes a value to optimize performance for expensive computations.

**Example with Description:**

import React, { useState, useMemo } from 'react';  
  
function ExpensiveCalculation({ num }) {  
 const calculate = (n) => {  
 let result = 0;  
 for (let i = 0; i < 1e6; i++) {  
 result += n;  
 }  
 return result;  
 };  
  
 const result = useMemo(() => calculate(num), [num]);  
  
 return <div>Result: {result}</div>;  
}

**Formik**

**Definition:** Formik simplifies form handling in React applications.

**Example with Description:**

import React from 'react';  
import { useFormik } from 'formik';  
  
function SignupForm() {  
 const formik = useFormik({  
 initialValues: { email: '', password: '' },  
 onSubmit: (values) => {  
 alert(JSON.stringify(values, null, 2));  
 },  
 });  
  
 return (  
 <form onSubmit={formik.handleSubmit}>  
 <label>Email:</label>  
 <input  
 type="email"  
 name="email"  
 onChange={formik.handleChange}  
 value={formik.values.email}  
 />  
  
 <label>Password:</label>  
 <input  
 type="password"  
 name="password"  
 onChange={formik.handleChange}  
 value={formik.values.password}  
 />  
  
 <button type="submit">Submit</button>  
 </form>  
 );  
}