React Tutorial Basics

Here's a beginner-friendly **React tutorial** that will help you understand and build a basic React application step by step.

---

1. What is React?

React is a JavaScript library for building user interfaces. It allows developers to create reusable UI components and manage state efficiently.

---

2. Prerequisites

Before starting, you should know:

- Basic HTML, CSS, and JavaScript.
- ES6 features like `let`, `const`, arrow functions, and destructuring.

---

3. Setting Up the Environment

Option 1: Using Create React App

1. **Install Node.js**
   Download and install Node.js. It includes npm (Node Package Manager).
2. **Create a React App**
   Run the following commands in your terminal:

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

This sets up a React project and starts the development server at `http://localhost:3000`.

Option 2: Online Playground (Optional)

Use CodeSandbox or StackBlitz to write and test React code without local setup.

---

4. Understanding the Folder Structure

Key files:

- `src/index.js`: The entry point of the app.
- `src/App.js`: The main component of your app.
- `public/index.html`: Contains the root `<div>` where the app renders.

---

5. React Basics

A. JSX (JavaScript XML)

JSX allows you to write HTML-like syntax in JavaScript:

```jsx
const element = <h1>Hello, React!</h1>;
```

B. Components

- **Functional Components**: Simple and stateless.

```jsx
function Welcome() {
  return <h1>Welcome to React!</h1>;
}
export default Welcome;
```

- **Class Components**: Used for stateful components.

```jsx
import React, { Component } from 'react';

class Welcome extends Component {
  render() {
    return <h1>Welcome to React!</h1>;
  }
}
export default Welcome;
```

C. Rendering

Use `ReactDOM.render` to display components:

```jsx
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

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

---

6. Building a Simple React App

Step 1: Create a Functional Component

Edit `App.js`:

```jsx
function App() {
  return (
    <div>
      <h1>Hello, React!</h1>
      <p>Welcome to your first React app.</p>
    </div>
  );
}
```

```
export default App;
```

Step 2: Add Props

Pass data to components using props:

jsx

```jsx
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

function App() {
  return (
    <div>
      <Greeting name="John" />
      <Greeting name="Jane" />
    </div>
  );
}
```

Step 3: Add State with useState Hook

State allows you to manage dynamic data.

jsx

```jsx
import React, { useState } from 'react';

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

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}

function App() {
  return (
    <div>
```

```
      <Counter />
    </div>
  );
}
```

---

7. Key React Concepts

A. Handling Events

React uses camelCase for event names:

jsx

```jsx
function Button() {
  function handleClick() {
    alert('Button clicked!');
  }

  return <button onClick={handleClick}>Click Me</button>;
}
```

B. Conditional Rendering

Display elements conditionally:

jsx

```jsx
function App() {
  const isLoggedIn = true;
  return <h1>{isLoggedIn ? 'Welcome back!' : 'Please log in.'}</h1>;
}
```

C. Lists and Keys

Render a list using `map()`:

jsx

```jsx
function App() {
  const fruits = ['Apple', 'Banana', 'Cherry'];
  return (
```

```
    <ul>
      {fruits.map((fruit, index) => (
        <li key={index}>{fruit}</li>
      ))}
    </ul>
  );
}
```

---

8. Next Steps

1. Learn about **React Router** for navigation.
2. Understand **lifting state up** and **context API**.
3. Explore **React Developer Tools** for debugging.
4. Learn about **useEffect** for managing side effects.

If you have any specific questions or need examples, feel free to ask!

Here's a more detailed breakdown of these React concepts:

---

1. Props (Properties)

Props are used to pass data from a parent component to a child component.

Basic Usage

jsx

```jsx
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

function App() {
```

```
  return (
    <div>
      <Greeting name="Alice" />
      <Greeting name="Bob" />
    </div>
  );
}
```

Default Props

Set default values for props:

jsx

```jsx
function Greeting({ name = 'Guest' }) {
  return <h1>Hello, {name}!</h1>;
}
```

Props Destructuring

Instead of using `props.name`, destructure directly in the function signature:

jsx

```jsx
function Greeting({ name }) {
  return <h1>Hello, {name}!</h1>;
}
```

---

2. Events

React handles events in a similar way to DOM events, but uses camelCase syntax and JSX.

Basic Example

jsx

```jsx
function Button() {
  function handleClick() {
    alert('Button clicked!');
```

```
  }

  return <button onClick={handleClick}>Click Me</button>;
}
```

Passing Arguments to Event Handlers

Use an arrow function to pass arguments:

jsx

```jsx
function Button() {
  function handleClick(message) {
    alert(message);
  }

  return <button onClick={() => handleClick('Hello!')}>Click Me</button>;
}
```

---

3. State

State allows components to manage and react to dynamic data.

Using the useState Hook

jsx

```jsx
import React, { useState } from 'react';

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

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
```

```
  );
}
```

State in Class Components

jsx

```jsx
import React, { Component } from 'react';

class Counter extends Component {
  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>
    );
  }
}
```

---

4. Conditional Rendering

Conditionally display elements based on state or props.

Using Ternary Operator

jsx

```jsx
function Greeting({ isLoggedIn }) {
  return <h1>{isLoggedIn ? 'Welcome back!' : 'Please log in.'}</h1>;
}
```

Using if-else

```jsx
function Greeting({ isLoggedIn }) {
  if (isLoggedIn) {
    return <h1>Welcome back!</h1>;
  }
  return <h1>Please log in.</h1>;
}
```

---

5. Lists and Keys

Keys help React identify which items have changed.

Rendering a List

```jsx
function App() {
  const fruits = ['Apple', 'Banana', 'Cherry'];
  return (
    <ul>
      {fruits.map((fruit, index) => (
        <li key={index}>{fruit}</li>
      ))}
    </ul>
  );
}
```

---

6. React Router

React Router is used for navigation in a React app.

Installation

bash

```
npm install react-router-dom
```

Basic Setup

jsx

```jsx
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';

function Home() {
  return <h1>Home Page</h1>;
}

function About() {
  return <h1>About Page</h1>;
}

function App() {
  return (
    <Router>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </Router>
  );
}
```

---

7. Hooks

Hooks allow you to use state and lifecycle features in functional components.

useEffect Hook

Used for side effects (e.g., data fetching, subscriptions):

jsx

```jsx
import React, { useState, useEffect } from 'react';

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

  useEffect(() => {
    const timer = setInterval(() => {
      setCount((prev) => prev + 1);
    }, 1000);

    return () => clearInterval(timer); // Cleanup on unmount
  }, []);

  return <p>Timer: {count}</p>;
}
```

Custom Hooks

Reuse logic by creating custom hooks:

jsx

```jsx
import { useState, useEffect } from 'react';

function useCounter(initialValue) {
  const [count, setCount] = useState(initialValue);

  const increment = () => setCount((prev) => prev + 1);

  return [count, increment];
}

function Counter() {
  const [count, increment] = useCounter(0);

  return (
    <div>
```

```
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
}
```

---

8. Styling in React

Inline Styles

jsx

```jsx
function App() {
  const style = { color: 'blue', fontSize: '20px' };
  return <h1 style={style}>Hello, styled world!</h1>;
}
```

CSS Modules

css

```css
/* App.module.css */
.title {
  color: red;
}
```

jsx

```jsx
import styles from './App.module.css';

function App() {
  return <h1 className={styles.title}>Hello, styled world!</h1>;
}
```

Styled Components

Install the library:

bash

```
npm install styled-components
```

Usage:

jsx

```jsx
import styled from 'styled-components';

const Title = styled.h1`
  color: green;
  font-size: 24px;
`;

function App() {
  return <Title>Hello, styled world!</Title>;
}
```

This guide provides a deeper understanding of each concept. If you'd like examples or help with advanced topics, let me know!