React Components

What Are React Components?

React components are reusable, independent building blocks of a React application. They define the UI and logic in a modular way, making the application easier to manage. They are fundamental units of any React application.

A React component is a JavaScript function or class that:

- Accepts inputs (props)
- Returns React elements (JSX)
- Manages state and logic

Types of React Components

React has two main types of components:

Functional Components (Modern & Preferred)

- Simple JavaScript functions.
- Use React Hooks (useState, useEffect, etc.) for state and lifecycle.
- More concise and easier to test.

Example:

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

Class Components (Older, Less Used)

- Use ES6 classes and extend React.Component.
- Require this to access props and state.
- Use lifecycle methods (componentDidMount, componentDidUpdate, etc.).

Example:

```jsx
class Greeting extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
}
```

Note: From React 16, React recommends Functional Components with Hooks over Class Components

Component Hierarchy & Composition

React applications follow a tree-like structure where components nest within each other.

Example Hierarchy:

```
App (Root)
├── Header
├── Main
│   ├── Sidebar
│   ├── Content
├── Footer
```

Example Composition:

jsx

```jsx
function App() {
  return (
    <div>
      <Header />
      <Main />
      <Footer />
    </div>
  );
}
```

```jsx
function Main() {
  return (
    <div style={{ display: "flex" }}>
      <Sidebar />
      <Content />
    </div>
  );
}
```

Below is the sample example where u can create UI in functional components using JSX.

```jsx
// Content.js
import React from "react";

function Content() {
  return (
    <div style={{ flex: 1, padding: "20px" }}>
      <h1>Welcome to the Content Area</h1>
      <p>This is the main content section of the page.</p>
    </div>
  );
}

export default Content;
```

How It Works

- **Composition**: Main acts as a container for Sidebar and Content.
- **Reusability**: Sidebar and Content can be used in other places if needed.
- **Separation of Concerns**: Each component has its own responsibility.

Best Practices for React Components

- Use functional components over class components.
- Keep components small and reusable.
- Use props for passing data and state for managing local data.
- Prefer composition over inheritance.
- Use React Hooks (useState, useEffect) for state and lifecycle.
- Follow the Single Responsibility Principle (SRP).

Conclusion

React components enable modular, reusable, and efficient UI development. With functional components, hooks, and composition, building scalable React apps has become easier.