



Components

In React, components are the building blocks of user interfaces. They allow developers to break down complex UIs into smaller, reusable pieces of code.

Types of Components

1. Functional Components:

- These are JavaScript functions that return React elements (JSX).
- They are simpler and recommended for most use cases.
- Example:

javascript

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

- Alternatively, you can use arrow functions:
Javascript

```
const Welcome = (props) => <h1>Hello, {props.name}</h1>;
```

2. Class Components:

- These are ES6 classes that extend `React.Component`.
- They include a `render()` method that returns JSX.
- Example:

javascript

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

- Class components are less commonly used now due to the introduction of React Hooks.



Key Concepts

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

- A syntax extension that allows writing HTML-like code within JavaScript.

- Example: javascript

```
return <div><h1>Welcome to React</h1></div>;
```

2. **Props (Properties):**

- Used to pass data from a parent component to a child component.
- Props are immutable and read-only.

- Example: javascript

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

3. **State:**

- Used to manage data that changes over time within a component.
- Functional components use the `useState` Hook for state management, while class components use `this.state`.

4. **Fragments:**

- Allow grouping multiple elements without adding extra DOM nodes.
- Example: javascript

```
return (  
  <>  
    <h1>Title</h1>  
    <p>Subtitle</p>  
  </>  
);
```

Common Use Cases

- Buttons, menus, forms, and other UI elements can be created as components.
- Components can be nested and reused across the application.



By using components effectively, developers can create modular, maintainable, and scalable React applications