

Fundamentals of React

React is a **JavaScript library** used for building user interfaces, especially for single-page applications (SPAs). It allows you to create interactive UIs using a **component-based architecture**.

1. Components — Building Blocks of React

React is built around **components**, which are like small, reusable pieces of the UI.

- **Functional Components:** The modern and simpler way to write components.

jsx

```
function Welcome() { return <h1>Hello, World!</h1>; }
```

 Copy  Edit

- **Class Components:** Older way (but still in use), uses a class-based syntax.

jsx

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

 Copy  Edit

2. JSX — JavaScript + HTML

JSX is a syntax extension for JavaScript that looks like HTML but allows you to write HTML elements within JavaScript code.

- JSX must be compiled into regular JavaScript using **Babel**.

jsx

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

 Copy  Edit


Note: JSX allows for dynamic content with curly braces `{}` .

3. State — Managing Data in Components

State is used to store and manage **data** that changes over time. When state changes, React re-renders the component to reflect the changes in the UI.

- In **functional components**, you use the `useState` hook.

jsx

 Copy  Edit

```
import React, { useState } from 'react'; function Counter() { const [count, setCount] = useState(0); // State initialization return ( <div> <p>You clicked {count} times</p> <button onClick={() => setCount(count + 1)}>Click me</button> </div> ); }
```

Key Points:

- `useState` returns an array with the current state value and a function to update it.
- State is local to the component.

4. Props — Passing Data to Components

Props (short for "properties") allow you to pass data from parent to child components. Props are read-only and can't be modified by the child.

- Example of passing props to a component:

jsx

 Copy

 Edit

```
function Welcome(props) { return <h1>Hello, {props.name}!</h1>; } function App() { return <Welcome name="Alice" />; }
```

Props are immutable, which means that child components can only receive them, not change them.

5. Event Handling — Responding to User Actions

React allows you to handle user interactions, such as clicks, form submissions, and keyboard events.

- Example of handling a button click:

jsx

 Copy

 Edit

```
function ClickButton() { function handleClick() { alert('Button clicked!'); } return <button onClick={handleClick}>Click me</button>; }
```

Note: React uses camelCase for event names (`onClick` , `onChange` , etc.) instead of the lowercase HTML attributes (`onclick` , `onchange`).

6. Conditional Rendering — Showing Different UI Based on Conditions

You can use **conditional statements** to render different content based on certain conditions.

- Example using an `if` statement:

jsx

Copy

Edit

```
function UserGreeting() { const isLoggedIn = true; if (isLoggedIn) { return <h1>Welcome back, User!</h1>; } else { return <h1>Please sign in</h1>; } }
```

- You can also use **ternary operators** or **logical operators** for inline conditionals.

7. Lists and Keys — Rendering Multiple Elements

React makes it easy to render lists of elements using the `map()` method. Each element in a list must have a **unique key** to help React identify which items have changed.

- Example of rendering a list:

jsx

Copy

Edit

```
function ItemList() { const items = ['Apple', 'Banana', 'Cherry']; return ( <ul> {items.map((item, index) => ( <li key={index}>{item}</li> ))} </ul> ); }
```

Key is used to ensure optimal performance in React when updating or reordering lists.

8. Effect Hook (`useEffect`) — Performing Side Effects

The `useEffect` hook allows you to perform **side effects** in functional components. Side effects can be operations like data fetching, subscriptions, or manually changing the DOM.

- Example of fetching data:

jsx

Copy

Edit

```
import React, { useState, useEffect } from 'react'; function DataFetcher() { const [data, setData] = useState([]); useEffect(() => { fetch('https://api.example.com/data') .then(response => response.json()) .then(data => setData(data)); }, []); // Empty array means
```

```
effect runs only once after the initial render
return <ul>{data.map(item => <li key={item.id}>
{item.name}</li>)}</ul>; }
```

The empty array `[]` as the second argument ensures that the effect runs only once (similar to `componentDidMount` in class components).

Recap of React Fundamentals

Concept	What it Does
Components	Reusable building blocks for the UI
JSX	JavaScript with HTML-like syntax
State	Manages data that changes over time
Props	Passes data from parent to child components
Event Handling	Responds to user actions (click, submit, etc.)
Conditional Rendering	Renders different UI based on conditions
Lists & Keys	Renders arrays of elements with unique keys
<code>useEffect</code>	Performs side effects (data fetching, subscriptions)