✓ What Is React?

React is a **JavaScript library** for building **user interfaces (UIs)** — basically, it's used to create websites that change without reloading the page.

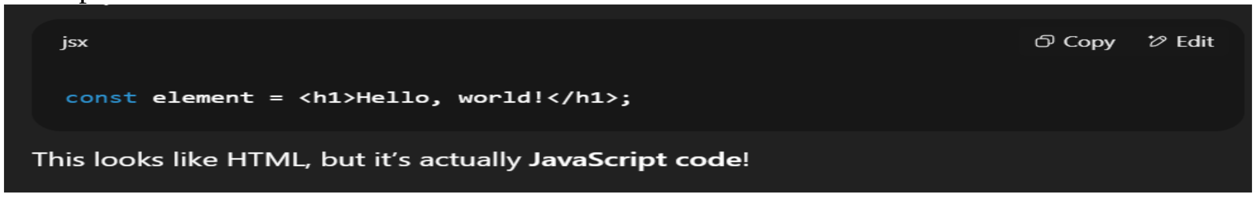
Example: Facebook, Instagram, Netflix use React. When you like a post or see new comments **without the page refreshing**, that’s React at work!

➢ React Basics We’ll Cover First

1. What is React and how to start
2. Creating a React App
3. JSX (React's HTML + JS syntax)
4. Components (like building blocks)
5. Props (like passing data into blocks)
6. State (for changing data inside the component)
7. Event Handling (like button clicks)
8. Styling in React (inline, class-based)
9. Hooks (useState, useEffect – start small)

➢ What is JSX?

JSX stands for **JavaScript XML**. It lets you write **HTML inside JavaScript**.
Example:



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

This looks like HTML, but it’s actually **JavaScript code!**

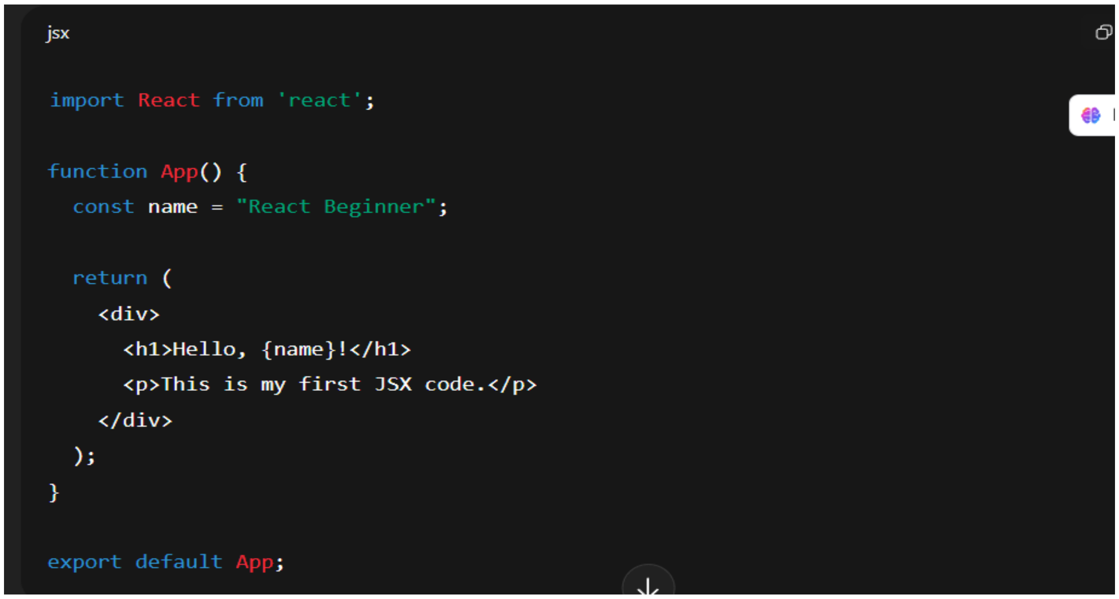
➢ Why Use JSX?

- It’s easier to create UI elements
- Looks like HTML, so it's familiar
- React uses JSX to describe what should appear on the screen

➢ PRACTICAL: Let's Use JSX in `App.js`

◆ Step-by-Step:

1. Open your React project (`my-first-app`)
2. Go to: `src/App.js`
3. Replace everything with this:

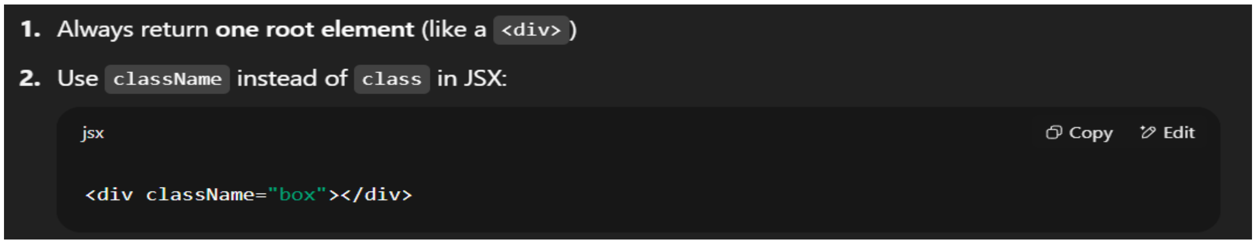


jsx

```jsx
import React from 'react';

function App() {
  const name = "React Beginner";

  return (
    <div>
      <h1>Hello, {name}!</h1>
      <p>This is my first JSX code.</p>
    </div>
  );
}

export default App;
```

Explanation:

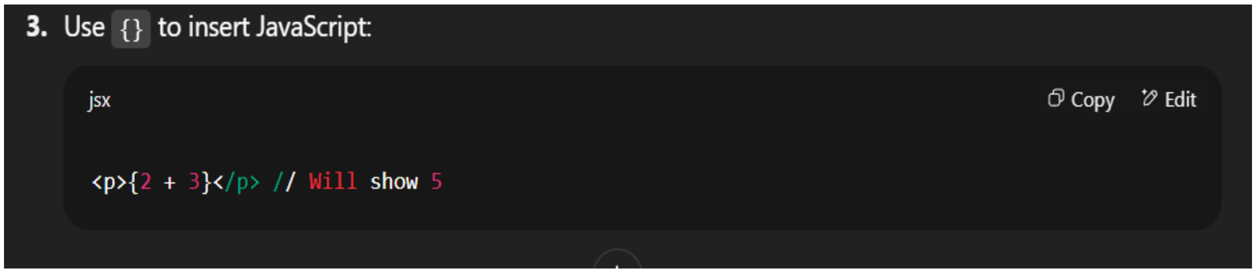
- `const name = "React Beginner";` → A JS variable
- `{name}` → Used inside JSX to display value
- Everything inside `<div>` is JSX (like HTML)



1. Always return **one root element** (like a `<div>`)
2. Use `className` instead of `class` in JSX:

jsx

```jsx
<div className="box"></div>
```



3. Use `{}` to insert JavaScript:

jsx

```jsx
<p>{2 + 3}</p> // Will show 5
```

➢ What is a Component?

A **component** is like a **mini app** or **reusable block** of code.

Just like in LEGO — each block is small, but together they build something big.

➢ Why Use Components?

- Reuse code (e.g., Button, Card, Navbar)
- Organize your UI
- Make large apps manageable

➢ Types of Components

1. **Functional Component** ✅ (this is what we use now)
2. Class Component (older, not needed for now)

We'll stick with **Functional Components**, using simple functions.

🎯 Step 4: Props — Passing Data to Components

➢ What Are Props?

Think of **props** as:

🎁 "Properties" or "Packages of Data" you send into a component.

Example:

If `App` is a parent, and `WelcomeMessage` is a child, `App` can send a message to `WelcomeMessage` using props — like a parent giving a child a note.

➢ Real-Life Analogy

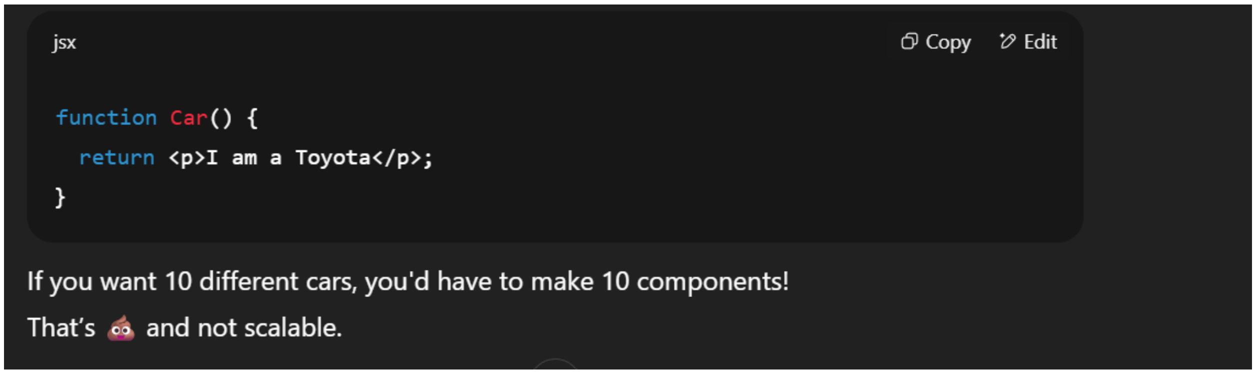
Imagine you're a restaurant waiter:

- You (the parent component) give a **menu item** (prop) to the **chef** (child component)
- The chef uses that item (prop) to make the dish (output)
-

✅ What is the Need for Props?

🚗 Imagine This:

You build a car component called `<Car />`.
Without props, every car looks the same:



jsx

```jsx
function Car() {
  return <p>I am a Toyota</p>;
}
```

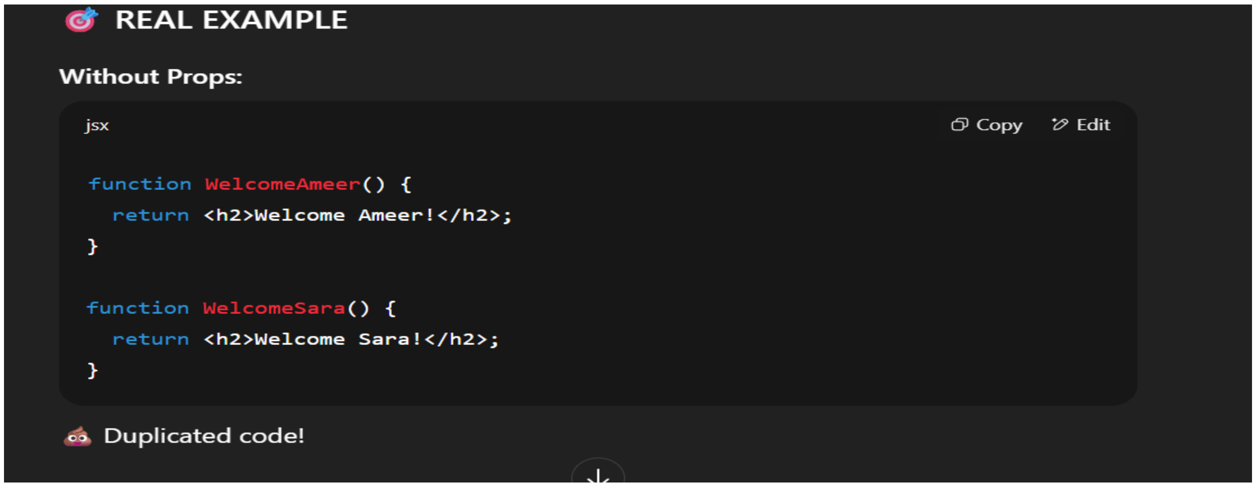
If you want 10 different cars, you'd have to make 10 components!
That's 💩 and not scalable.

PROBLEM Without Props:

- Repeating same component many times with different content = code duplication ❌
- Can't reuse components for different data ❌
- Not dynamic ❌

✅ Props SOLVE THIS!

- Props let us create **one reusable component**, and give it **custom content**.

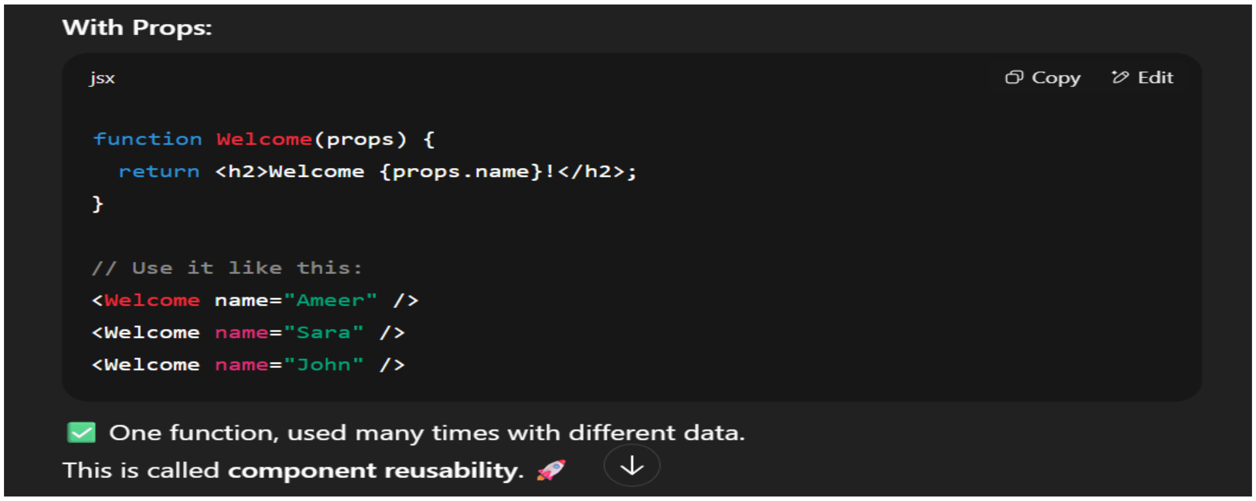


🎯 REAL EXAMPLE

Without Props:

```jsx
function WelcomeAmeer() {
  return <h2>Welcome Ameer!</h2>;
}

function WelcomeSara() {
  return <h2>Welcome Sara!</h2>;
}
```

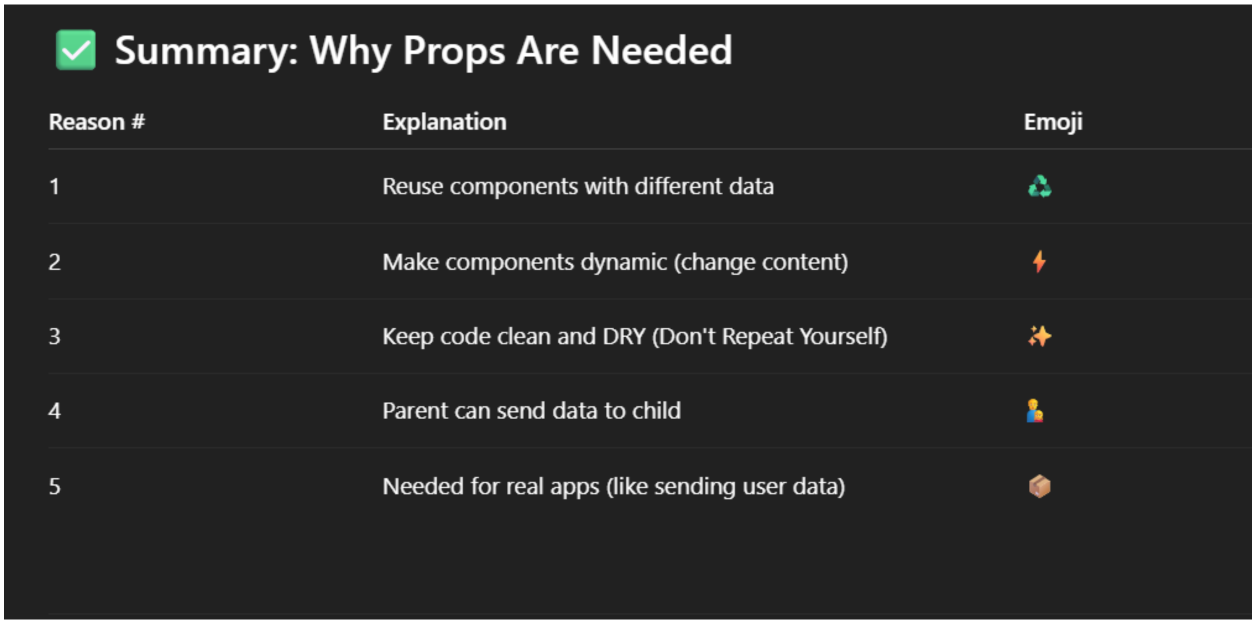
💩 Duplicated code!



With Props:

```jsx
function Welcome(props) {
  return <h2>Welcome {props.name}!</h2>;
}

// Use it like this:
<Welcome name="Ameer" />
<Welcome name="Sara" />
<Welcome name="John" />
```

✅ One function, used many times with different data.
This is called **component reusability**. 🚀



✅ Summary: Why Props Are Needed

Reason #	Explanation	Emoji
1	Reuse components with different data	♻️
2	Make components dynamic (change content)	⚡
3	Keep code clean and DRY (Don't Repeat Yourself)	✨
4	Parent can send data to child	👨‍👦
5	Needed for real apps (like sending user data)	📦

✅ Step 5: `useState` — React State

➢ What is `useState`?

React is great at building **dynamic** UIs.
But to make something **change**, like:

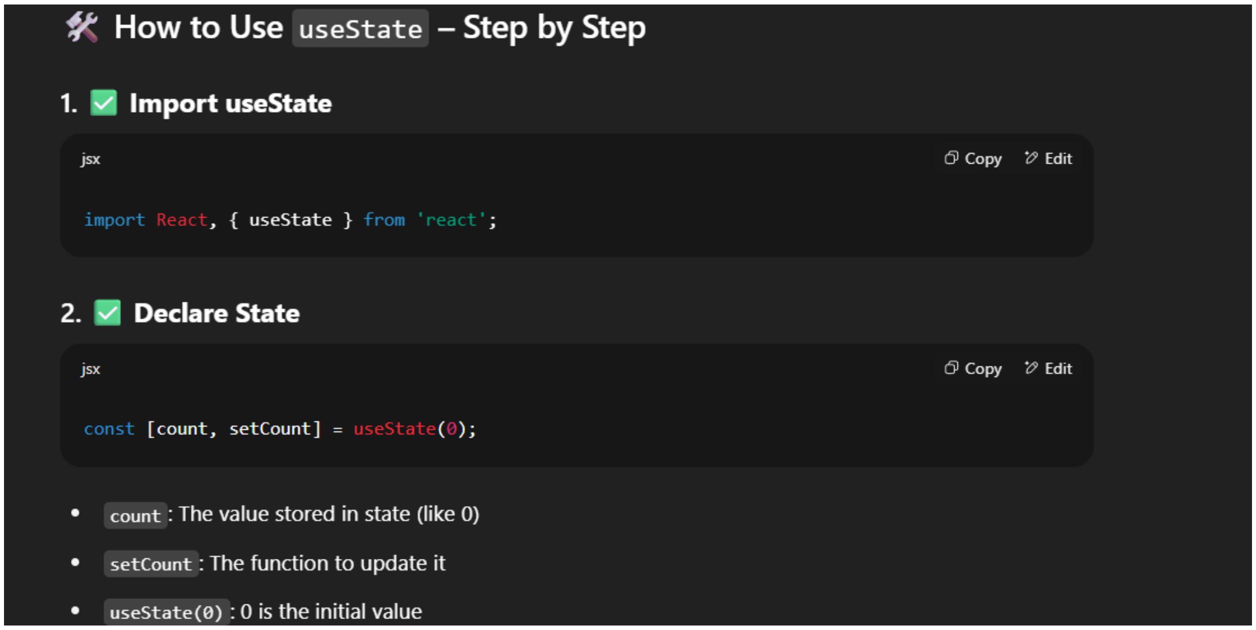
- A button that increases a counter □
- A form input that stores text 📝
- A toggle that shows/hides something 👁□

You need a way to **store and update values** inside a component.

That's what **`useState`** is for:
☞ It lets your component **remember** things, and **react** when they change.

Simple Way to Understand:

Imagine `useState` as a box 🎁 with a value inside.
You can open it (read it), and change it — and when you do, React **re-renders** the component to show the new value.



🛠 How to Use `useState` – Step by Step

1. ✅ **Import useState**

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

2. ✅ **Declare State**

```jsx
const [count, setCount] = useState(0);
```

- `count`: The value stored in state (like 0)
- `setCount`: The function to update it
- `useState(0)`: 0 is the initial value