# Props

What props are, sending data from parent to child, using multiple props, destructuring, default values, and how props differ from regular function parameters.

**Beginner** 10 min

Last updated: August 18, 2026
Read online: https://www.learnforgex.com/en/topics/props

---

# Props

In Components, we used the `Button` component three times, but all three were identical. What if we want to show a different label or a different color each time? That's what **props** are for. This lesson covers how to send data into a component from outside, with simple examples.

## What Are Props?

Props (properties) are how you send data into a component from outside -- much like giving an HTML tag an attribute (remember "Attributes and className" from the JSX lesson), except here the value reaches the component function as a parameter.

## Sending Data from Parent to Child

When you use a component (that is, "render" it), you can give it values like attributes -- these become that component's props:

```
// Props (properties) are a way to pass data INTO a component from the
// outside -- just like giving an HTML tag an attribute, except here the
// value reaches your component function as a parameter.
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

function App() {
  return <Greeting name="Emma" />;
}

console.log(App());
// Result: <h1>Hello, Emma!</h1>
```

Here, `App` is the component that uses `Greeting` (the "parent"), and `Greeting` is the component being used (the "child"). By writing `name="Ayşe"`, we send `Greeting` a prop named `name`.

## Using Multiple Props

You can send a component as many props as you want:

```
// You can pass as many props as you like to a component -- each one
// reaches the function as a separate field of the props object.
function UserCard(props) {
  return (
    <div>
      <h2>{props.name}</h2>
      <p>Age: {props.age}</p>
      <p>City: {props.city}</p>
    </div>
  );
}

function App() {
  return <UserCard name="Emma" age={28} city="Boston" />;
}

console.log(App());
```

Each prop reaches the component as its own field on the `props` object -- `props.name`, `props.age`, `props.city`, and so on.

## Reading Props with Destructuring

Instead of writing `props.name` everywhere, most React code uses destructuring:

```
// Instead of writing "props.name", "props.age" everywhere, most React code
// uses destructuring -- inside the function parameter, you pull the fields
// you need out directly as variables. Both examples do the SAME thing,
// the second one is just shorter and the more commonly used style.

// 1) WITHOUT destructuring:
function UserCardLong(props) {
  return <p>{props.name} -- {props.age} years old</p>;
}

// 2) WITH destructuring (the commonly used style):
function UserCard({ name, age }) {
  return <p>{name} -- {age} years old</p>;
}

console.log(UserCardLong({ name: "Emma", age: 28 }));
console.log(UserCard({ name: "Emma", age: 28 }));
```

Both versions do exactly the same thing -- destructuring just removes the repeated `props.` and makes the code a bit shorter. This is the more common style, and we'll keep using it in later lessons.

## Default Values (Default Props)

What happens if a prop is never sent? You can define a default value right in the destructuring:

```
// What should happen if a prop is never passed at all? You can define a
// "default value" during destructuring -- exactly the same idea as default
// values for regular JavaScript function parameters.
function Greeting({ name = "Guest" }) {
  return <h1>Hello, {name}!</h1>;
}

console.log(Greeting({ name: "Emma" })); // Hello, Emma!
console.log(Greeting({}));               // Hello, Guest! (default was used)
```

`name = "Guest"` is exactly the same idea as default values for regular JavaScript function parameters -- if `name` isn't sent, `Guest` is used.

## Props vs. Regular Function Parameters

Props really are nothing more than a regular function parameter -- there's no special mechanism from React here. The difference is just in how you use it: you'd call a normal function as `Greeting({ name: "Ayşe" })`, while you "call" a component inside JSX as `<Greeting name="Ayşe" />`. There's also an important rule: props are **read-only** -- a component should never change a prop it receives. If you need to change data over time, that's what "state" is for -- we'll cover that in the State & Events category.

## Summary and Glossary

Props are how you send data into a component from outside. A parent component gives a child values like attributes; the child reads them via `props` (or directly as variables, with destructuring). If a prop isn't sent, a default value can be defined. Props are read-only.

**Glossary**

**Props** — Read-only data sent into a component from outside.

**Parent Component** — A component that uses (renders) another component.

**Child Component** — A component used by a parent component.

**Destructuring** — Syntax for pulling an object's fields directly into variables (like `{ name, age }`).

**Default Prop** — A predefined value used when a prop isn't sent.