

So now that we already know what props are and how we use them in practice, let's quickly review them and even learn some important additional things about props. So as we just learned, we use props in React to pass data from parent components to child components. So essentially to pass information down the component tree. This means that essentially we use props to communicate between parent and child components. Therefore, props are an essential react tool to configure and also to customize components. So we can imagine props as settings that we can use to make a parent component control how its child component should look like and how it should work. So in that regard, props are just like arguments passed to regular JavaScript functions. Also, we can pass anything into JavaScript functions, right? And so the same is actually true for props. So we can pass any type of value as a prop. So we can pass single values, arrays, objects, functions and even other React components, which is a really powerful technique that we will explore a bit later. So those are the fundamentals of props in React. But now let's go dig a little bit deeper. But before we do that, we need to first take a step back. So at this point of the course, we have already learned about the components appearance and its logic. So by writing both JSX and JavaScript logic inside components. Now, I've also been saying since the beginning of the course that React renders a component based on its current data and that the UI will always be kept in sync with that data. Right. But now it's time to get a bit more specific about what that data actually is. So this data that React uses to render a component is made out of props and state. And actually there are even more types of data. But what matters for now are props and state. Now, state is basically internal component data that can be updated by the component's logic. So by the component itself, while props on the other hand, is data that is coming from the parent component. So from the outside basically. So it's the parent component who owns that data. And so therefore it cannot be modified by the child component. Instead, props can only be updated by the parent component itself. And this brings us to one of the few strict rules that React gives us, which is that props are immutable so they cannot be changed. They are read only. And if at any point you feel like you need to mutate props, actually what you need is state because state is for data that changes over time as we will learn soon. But why is that actually why are props immutable in React? Well, to start props are just an object. Therefore, if you change the props object in your component, you would also affect the parent component because that's just how objects work in JavaScript. So when you copy an object and mutate the copy, the original object will also be mutated. Now if you change an object that is located outside of the component function, that function has then created a so-called side effect. So in general, a side effect happens whenever you change some data that's located outside of the current function. React, however, is all about pure functions, so functions without side effects, at least when it's about a component's data. So components have to be pure in terms of their props and state because this allows react to optimize your application and it avoids some strange bugs that can appear when you manipulate external data. And in fact we can extend this idea of immutability to react development in general. So a component should never mutate any data that we write outside of its function scope like in this example here. And now to finish, it's important to understand that React uses a so called one way data flow. Now, what does that have to do with props? Well, in simple terms, one way data flow means that in React applications, data can only be passed from parent to child components, which happens by using props. So in other words, data can flow from parents to children, but never the opposite way. And therefore we have a one way data flow, so only from top to bottom of the component tree. Now, this may sound obvious to you, but other frameworks such as Angular actually employ a two way data flow. So if you know one of those frameworks already, this might be quite a change for you. But there is actually a reason or multiple reasons why React uses a one way data flow like this. The first is that it makes applications way more predictable and way easier to understand for developers because it is just a lot easier to understand where the data is coming from if it only flows in one direction. In a similar vein, it makes applications way easier to debug. Again, because we have way more control over the data and we understand exactly how that data flows around. And finally, two way data binding is usually less efficient, so it's less performant to implement. Okay, so that sounds great, but you might be wondering what if I actually wanted to pass some data, for example, some state up to a parent component? Well, there is actually a very clever way to do that. But as so often we will learn about that a bit later and actually in the next section to be specific. So it's not far away. There are just so many moving pieces in learning a whole library like React that of course you can't learn it all at once, but trust me, you will get there and then everything will fall nicely into place at the end.