

We learned before that state is the most important concept in React. Therefore, managing state is the most important aspect when it comes to thinking in React. And so let's not talk about the fundamentals of state management in React. So as you already know, we can use the use state function to create multiple pieces of state in order to track data that changes over the lifecycle of an application. Here we have many examples of state in the Udemy application that you are familiar with and if you want, you can pause the video and quickly analyze this. Now all this looks like a bit of a mess, right? Like, how do we know that we even need all of these pieces of state and how do we know where exactly to place them inside the code? Well, that's where state management comes into play. Now, state management can be defined in different ways by different people. But I like to think of state management as deciding when we need to create new pieces of state. What types of state we need, where to place each piece of state inside our code base, and also how all that data should flow through the app. And to summarize all this, I like to use the analogy that state management is basically giving each piece of state a home within our code base. Now, up until this point in our small apps, we never had to worry about state management at all. We simply placed each state in a component that needed it, and that's it. But as an application grows, the need to find the right home for each piece of state starts to become really important. No matter if that home is the component where we first need that state, some parent component or even global state. And speaking of global state, let's actually analyze the difference between the big two types of state that exist in React global, state and local state. Now, this will only become more important to us later on. But let's still start learning about this inside this thinking about state section. So in React, each piece of state is either local, state or global state. So local. State is state that is only needed in one component or in a few different components like child or sibling components. We simply create a piece of local state using the use state function inside a certain component, and that piece of state is then only accessible to that exact component and maybe to its child components if we pass the state using props. Now going back to our Udemy app, an example of local state might be the input text in the search bar. So probably only that component needs to know about this data and therefore this is local, state, so state local to that search bar component. Now about global state. This is a state that many different components in the app might need access to. Therefore, when we define state as being global, that piece of state will become accessible to every single component in the entire app. It's shared between all components and therefore we can also call this shared state. In practice we can define global state using React Context API or also an external global state management library like Redux that you might have heard of. Now in this Udemy app, one piece of global state is the shopping cart. So that piece of data is used all over the place here. So all these components need access to the shopping cart state and therefore it makes sense that this is global state. Now, this distinction between local and global state will matter more in large applications. So in the app we're building right now, we won't have any truly global state and we're actually going to keep using just local, state doing parts one and two of this course. And in fact, one important guideline in state management is to always start with local, state and only move to global state if you really, truly need it. And we will learn all about this in part three and four of the course. But for now, let's take a look at how to decide when we actually need state and where we should place it. So this light will basically be a flowchart that will help you take those decisions. So again, about when to create state and where to place it. So it all starts with you realizing that you need to store some data. Now, when this happens, the first question to ask is, will this data change at some point in time? And if the answer is no, then all you need is a regular variable. So probably a const variable. However, if the data does need to change in the future, the next question is it is possible to compute or to calculate this new data from an existing piece of state or props? If that's the case, then you should derive the state. So basically calculated based on an already existing state or prop. And this is a pretty important concept. So there is a separate lecture on derived state later in this section. However, most of the time you cannot derive state. And so in that case, you need to ask yourself whether updating the state should rerender the component. Now, we have already learned before that updating state always rerenders a component, but there is actually something called a ref which persists data over time like regular state but does not rerender a component. So it's basically a special type of state that we will look at later. Now, most of the time you actually do want state to rerender the component. And so what you do is to create a new piece of state using the use state function and you then place that new piece of state into the component that you are currently building. And so that's the always start with local state guideline that we talked about in the previous slide. And with this we have completed the decision process of when to create state. So again, most of the time you will just create a new piece of state using the use state hook. But there are also all these other cases. And so it's important that you are aware of when to create each of them. But anyway, let's now focus on where to place each new piece of state. So if the state variable that we just created is only used by the current component, then simply leave it in that component and you're done. So that's the end of the process right there. However, the state variable might also be necessary for a child component, and in that case, simply pass the state down into the child component by using props. So easy enough, right? Now, if the state variable is also necessary for one or a few sibling components, or even for a parent component of your current component, it's time to move that state to the first common parent component. And in React, this is what we call lifting state up. And this is another one of those super important topics which we will actually start using in practice in the next video. Now, finally, the state variable might be needed in even more than just a few siblings, so it might be necessary all over the place in the component tree. And what does that sound like to you? Well, that's right. That sounds just like global State. But since we won't need global state for some time, we will complete this diagram once we reach the global state management lectures. All right. So I hope that this diagram will be useful once you start building your own small apps or even throughout the rest of the course. Now, I know that this might look super confusing and like a lot of work for now, but it will become really intuitive over time. So at some point you will just automatically and intuitively know when to create a piece of state. You will know when to derive state from existing state and you will intuitively know when to lift state up. But this flowchart can still be quite helpful in the beginning.