

Okay. So we have just seen the power of state by using the use state function. But now let's get a better understanding of how exactly state works in React. And let's start from a fundamental react principle that we have already discussed earlier. So remember how we learned that in React? We do not manipulate the Dom directly when we want to update a component's view, Right? So React is declarative, not imperative. And so we never touch the Dom in our code. But if that's the case, then this leads us to the question of how do we update the component on the screen whenever some data changes or whenever we need to respond to some event like a click. Now, we already know that the answer to this question is state, but here we are trying to derive it from first principles. So anyway, to answer that question, we need to understand another fundamental react principle, which is the fact that React updates a component view by re rendering that entire component whenever the underlying data changes. Now, as soon as we will reach the section about how React works behind the scenes, we will learn exactly what actually happens inside React when a component renders. But for now, just know that re rendering basically means that react calls the component function again so each time the component is rendered. So conceptually, we can imagine this as react removing the entire view and replacing it with a new one each time a re render needs to happen. But again, we will learn exactly what happens later. Now react preserves the component state throughout Rerenders. And so even though a component can be rendered and re rendered time and time again, the state will not be reset unless the component disappears from the UI entirely, which is what we call unmounting. Now, speaking of state, it is when state is updated that a component is automatically re rendered. So let's imagine that there is an event handler in the view, for example, on a button that the user can click. So the moment that button is clicked, we can update a piece of state in our component using the set function coming from the use state hook. So just like we did in the last lecture, right? Then when React sees that the state has been changed, it will automatically rerender the component which will result in an updated view for this component. Or as a more real example, we can look at the simple advice app that we built right in the first lecture of the course. So in that application, each time we click the Get advice button, a new piece of advice is fetched from the API. Then when that data arrives, we store the data in the advice state variable. So we update the advice state. So let's imagine that the new advice is quality beats, quantity. React will notice this state change and rerender the component so it will remove the old one and display a new updated component view on the screen. And with this I hope that the mechanics of state in React are now really clear to you. So the conclusion of all this is that as React developers, whenever we want to update a component view, we update its state. And so React will then react to that update and do its thing. And in fact, this whole mechanism is so fundamental to react that it's actually the reason why react is called react in the first place. So on a high level, moving from the component level to the application level now, React reacts to state changes by Rerendering. The user interface. That's the main thing that it does. And therefore it was decided to call this library react. And with this, we have come full circle from the first lecture about why frameworks exist. There we learned that frameworks exist to keep UI in sync with data and so now we have learned a bit better. How react does that?