

We have written some pieces of JS in this course already, but what actually is JS? And why is it such a big deal in React? Well, when we first talked about components, we talked about how a component contains its own data, logic and appearance. And that makes sense, right? Because if a component is a piece of the user interface, it means that we must be able to describe exactly what that component looks like. And so that's where JSX comes into play. So JSX is a declarative syntax that we use to describe what components look like and how they work based on their data and logic. So it's all about the components appearance. In practice, this means that each component must return one block of JSX, which react will then use to render the component on the UI. Now, looking at this code, this JSX looks a lot like HTML, right? But in fact JSX is an extension of JavaScript which allows us to combine parts of HTML, CSS and JavaScript all into one block of code. So basically we can write HTML and embed some pieces of JavaScript where necessary, for example, to reference some JavaScript variables and we can even reference other react components so that we can then combine nest and reuse multiple components. But now you might be thinking if React is a JavaScript framework, then how will it understand this HTML looking code? Well, remember that JSX is just an extension of JavaScript, which means that there is a simple way of converting JSX to JavaScript. This is done by a tool called Babel, which was automatically included in our application by Create React app. And the result of this conversion looks something like this code on the right where each element was converted to a react.createelement function call. And does this look familiar? Well, I hope it does because this is exactly what we returned from the app component in the pure React lecture. So in that lecture where we couldn't use JSX because we didn't have that Babel tool. But anyway, this conversion is necessary because browsers of course do not understand js. They only understand HTML. So behind the scenes, all the JS that we write is converted into many nested react.createelement function calls. And these function calls are what in the end create the HTML elements that we see on the screen. Now what this means is that we could actually use React without JS at all. So we could just manually write these create element functions instead of JS. But that doesn't look like a lot of fun, right? It also makes the code really hard to read and to understand. And so actually everyone just uses JS. All right. So now that we know what JS is all about, let's go back to that first paragraph where I say that JS is a declarative syntax. So what does it actually mean? That JS is declarative? Well, before we can understand what declarative means, we first have to review what imperative means. So when we try to build UIs using vanilla JavaScript, we will by default use an imperative approach. This means that we manually select elements, traverse the Dom and attach event handlers to elements. Then each time something happens in the app, like a click on a button, we give the browser a step by step instructions on how to mutate those Dom elements until we reach the desired updated UI. So in the imperative approach, we basically tell the browser exactly how to do things. However, doing this in a complex app is completely unfeasible for all the reasons that we have learned about before. And remember that that's the reason why frameworks like React exist in the first place, and it's why React chose to use a declarative approach to building user interfaces. So a declarative approach is to simply describe what the UI should look like at all times, always based on the current data that's in the component. And we will soon learn that this data is props and state. And so again, basically we use to describe the UI based on props and state. So the data that's currently in the component and all that happens without any Dom manipulation at all. So there are no query selectors, no Add event listeners, no classlist, no text content properties anywhere to be seen here because in fact React is basically a huge abstraction away from the Dom so that we developers never have to touch the Dom directly. Instead, we think of the UI as a reflection of the current data and let react automatically synchronize the UI with that data. So in essence, the difference between imperative and declarative is that in the declarative approach we use JSX to tell react what we want to see on the screen, but not how to achieve it step by step. React can figure that out on its own, basically, and this has many, many advantages, as we will see throughout the course.