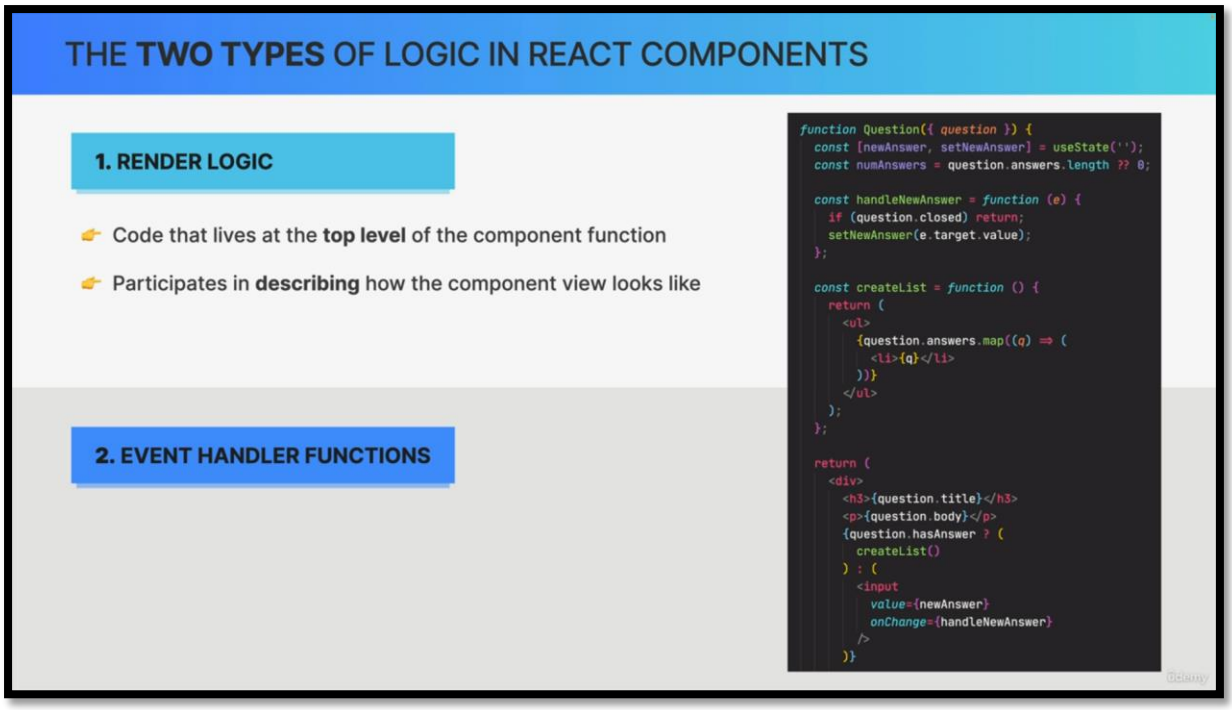
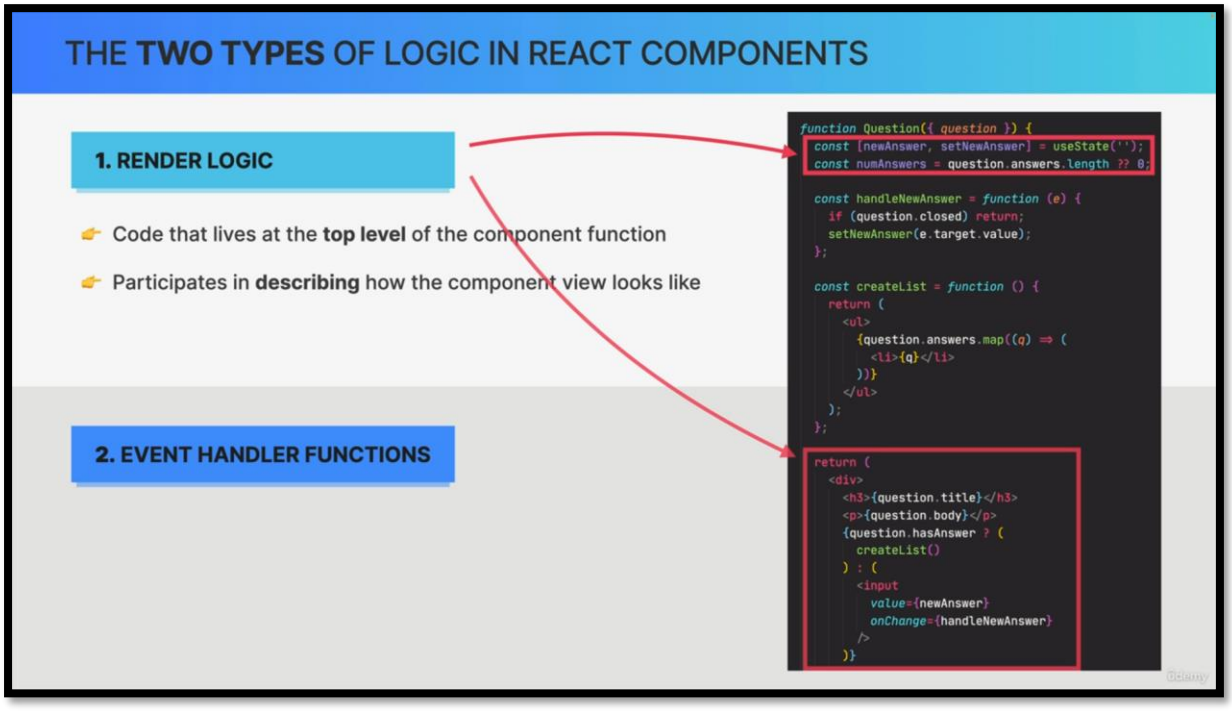
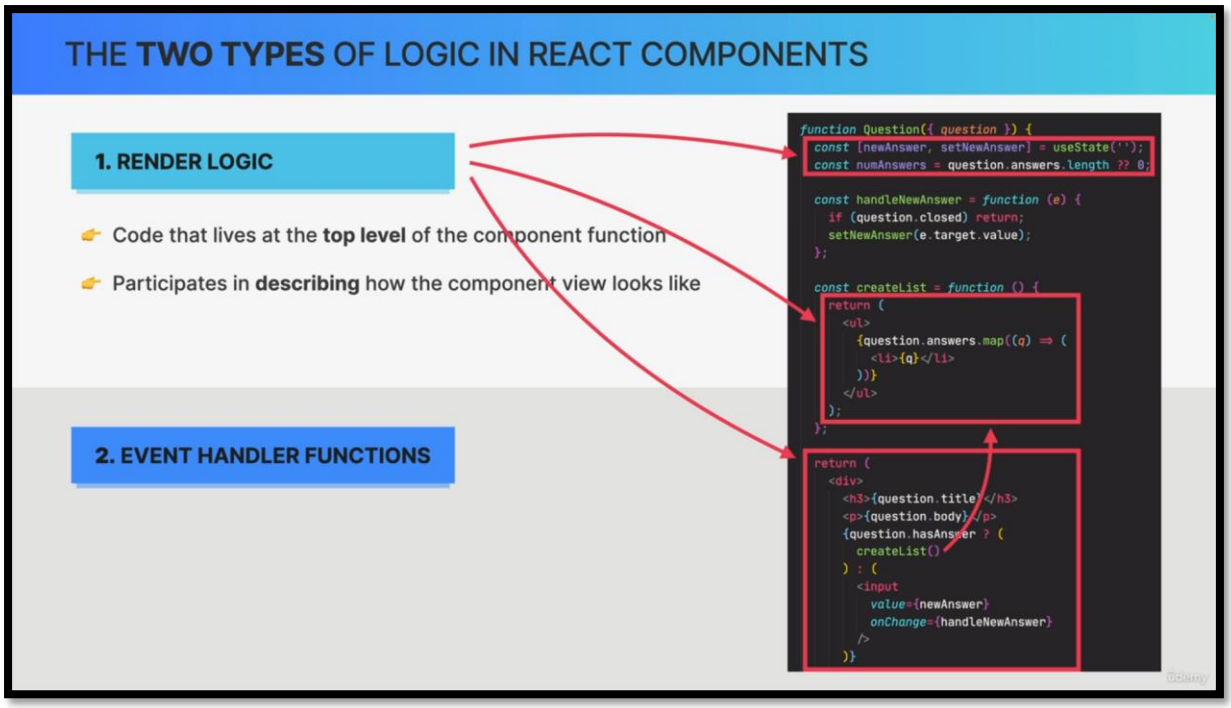


Render logic is basically all the code that lifts at the top level of your component functions and that participates in describing how the view of a certain component instance should look like.

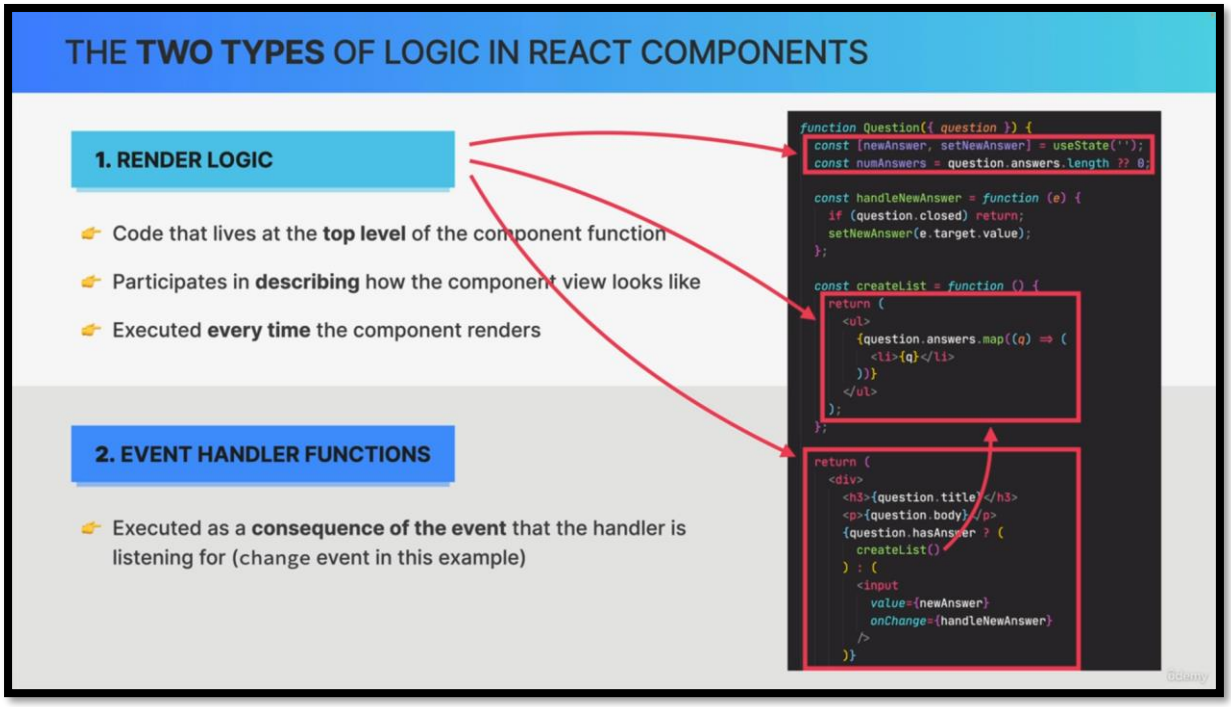


In this code example, there is a lot of render logic. So, we have these two lines of code at a top level and then also the return block where our component returns it's JSX. So, these describe exactly how the component will be displayed on the screen.

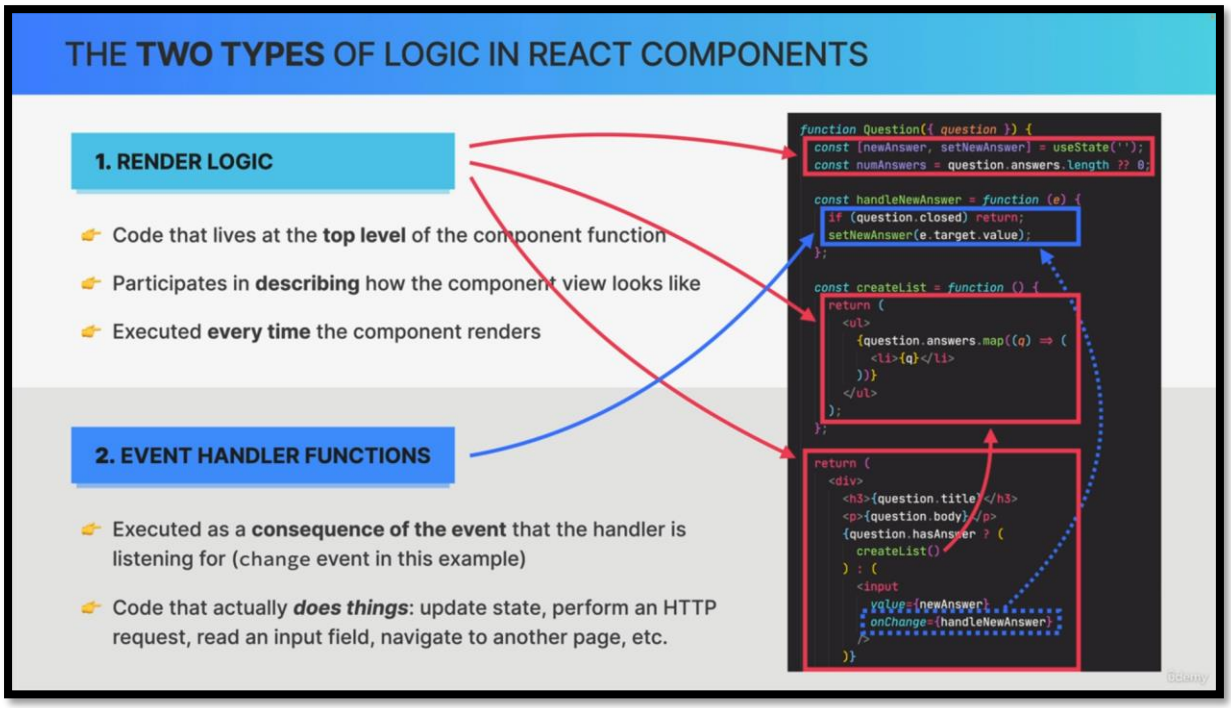


However, if we look closely, we can identify yet another piece of render logic here, even though this code is actually inside a function. So, as you can see in the return block, the code there is actually calling this createList function. Therefore, that logic also participates in describing the component view. So, it's also render logic.

So basically, render logic is all the code that is executed as soon as the component is rendered. So, each time that the function is called.



Event handler functions are basically pieces of code that are executed as a consequence of the event that the handler is listening to.



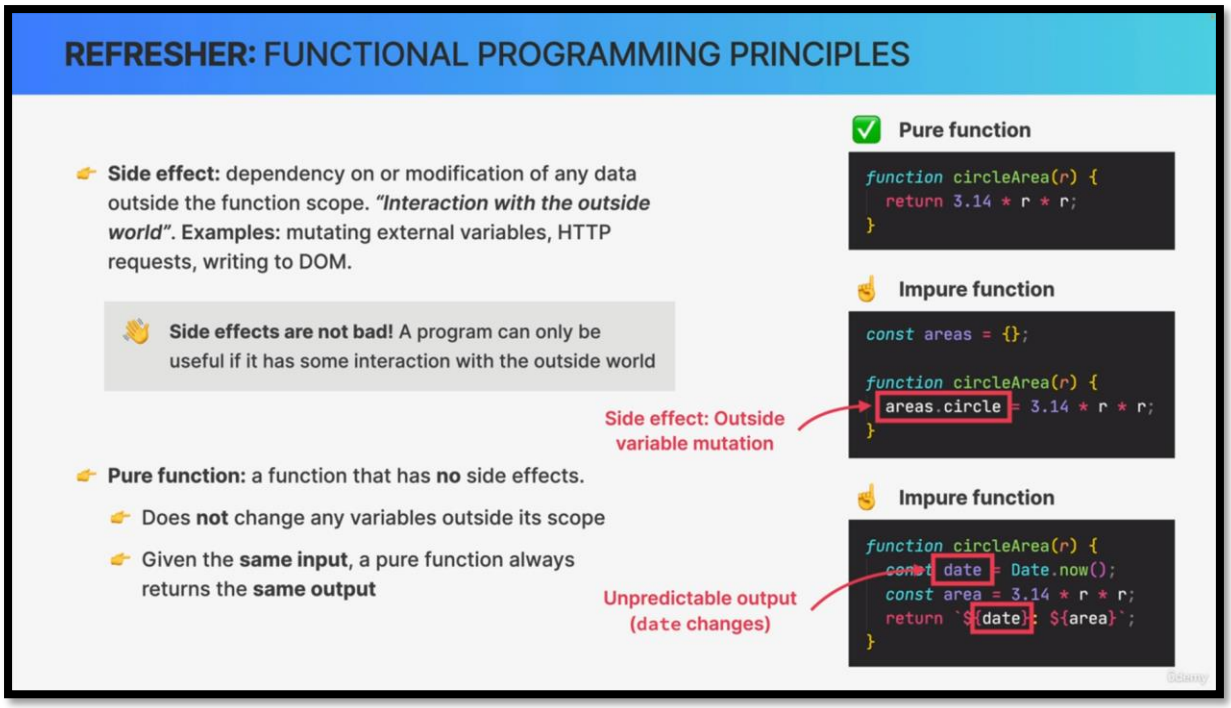
So, in our example, we have this line of code that essentially registered handle new answer for the change event and therefore handle new answer is our event handle function.

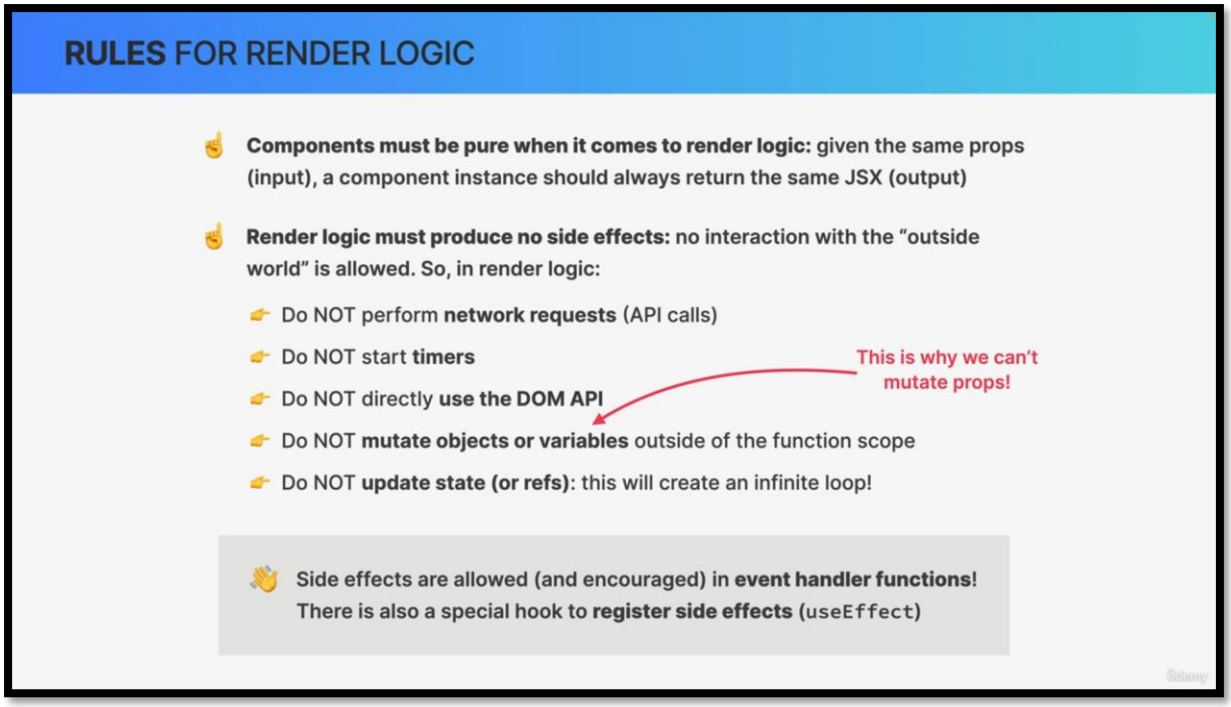
So, while render logic is code that renders the component event handlers contain code that actually does things. So, basically code that makes things happen in the application.

So, event handlers contain things like state updates, HTTP requests, reading input fields, page navigation and many more.

So, all things that basically change and manipulate the application in some way.

Now why is this all so important?
Well, it's important because React requires that components are pure when it comes to render logic in order for everything to work as expected.





There's just one big rule which is that components must be pure functions when it comes to render logic. This means that if we give a certain component instance the same props, so the same input, then the component should always return the exact same output in the form of JSX.

In practice, this means that render logic is not allowed to produce any side effects. So, in other words, the logic that runs at the top level and is responsible for rendering the component should have no interaction with the outside world.

This means that render logic is not allowed to perform network requests to create timers or to directly work with the DOM API.

Render logic must also not mutate objects or variables that are outside the scope of the component function. This is actually the reason why we cannot mutate props which is one of the hard rules of React.

You know why that rule exists. It's because doing so would be a side effect and side effects are not allowed.

Finally, we really cannot update state or refs in render logic. Updating state in render logic would actually create an infinite loop, which is why we can never do that.

State updates are technically not side effects but it's still important for them to be on this list.

Now, there are other side effects that are technically not allowed as well, but that we create all the time like using console.log or creating random numbers. So, these are clearly interactions with the outside world

But now you might be wondering if all this stuff is not allowed, then how will I ever be able to make an API call to fetch some data?

Well, keep in mind that these side effects are only forbidden inside render logic. This means that you have other options for running your side effects.

Event handler functions are not render logic and therefore, side effects are allowed and actually encouraged to be used inside these functions.

If we need to create a side effect as soon as the component function is first executed, we can register that side effect using a special hook called useEffect.