In the last video, we saw how we could react to a user clicking on a button by using the onclick event. We set that equal to a function reference, handleClick, which runs the function when the click occurs, changing the beltColor value to orange. We can see this working when I click on the button and the color changes from black to orange. This is cool, but what if I want to allow the user to type into an input field and let them choose their own color? As they type, the color changes dynamically.

To do that, we first create an input of type text. Then, we need to react to an input event, similar to how we reacted to the click event. An input event occurs whenever a user types something into the input field. So, we can attach an event listener to this field, like oninput, and set that equal to a function, which we’ll call handleInput. By the way, we don’t invoke these functions immediately; we pass function references so that they are only invoked when the event occurs.

Next, let’s create that function. I’ll define const handleInput as an arrow function. Inside the function, we need to take whatever the value is of the input field and assign it to the beltColor. To get the value, we use the event parameter, which automatically gets passed when an event occurs. We access the target element of the event (in this case, the input field) and its value property, which holds the text typed by the user. So, we can say beltColor = e.target.value. Every time the user types something, this will update the beltColor with the new value, and it should reflect on the screen in real-time.

If we save and test this, we can see that the text changes dynamically as I type into the input field. If I type “yellow,” it updates the belt color to yellow. This is great!

However, right now, we only have one-way data binding. If the value of beltColor changes elsewhere, it won’t update the input field. For example, if I click the button and change the color to orange, the input field doesn’t reflect that change. To achieve two-way data binding, we need to update the value of the input field whenever beltColor changes. We can do that by setting the input’s value property equal to beltColor. This way, whenever the beltColor changes, the input field will update automatically.

After doing that, we see that the value of the input field matches the beltColor. Now, if I type “yellow,” it updates the belt color and vice versa. If I change the belt color to “orange,” the input field also reflects this change. This is two-way binding in action!

If you want a shorthand for two-way data binding, you can use bind in the input field. Instead of manually setting the value, we just use <input type="text" bind="beltColor">. This does both the one-way and two-way binding under the hood, so whenever the user types into the input, it updates beltColor, and if beltColor changes elsewhere, the input field reflects that change.

Finally, we can also bind data to other attributes or elements. For example, I could set the style attribute of the paragraph tag that outputs the beltColor. So, when beltColor changes, it also dynamically updates the color of the text. If the belt color is black, the text will be black. If it changes to red, the text will turn red, and so on.

In conclusion, you can bind data both one-way and two-way in a very flexible way in JavaScript. If you need two-way binding, using bind is a convenient shortcut. You can also bind data to other attributes, like style, to dynamically update the UI.