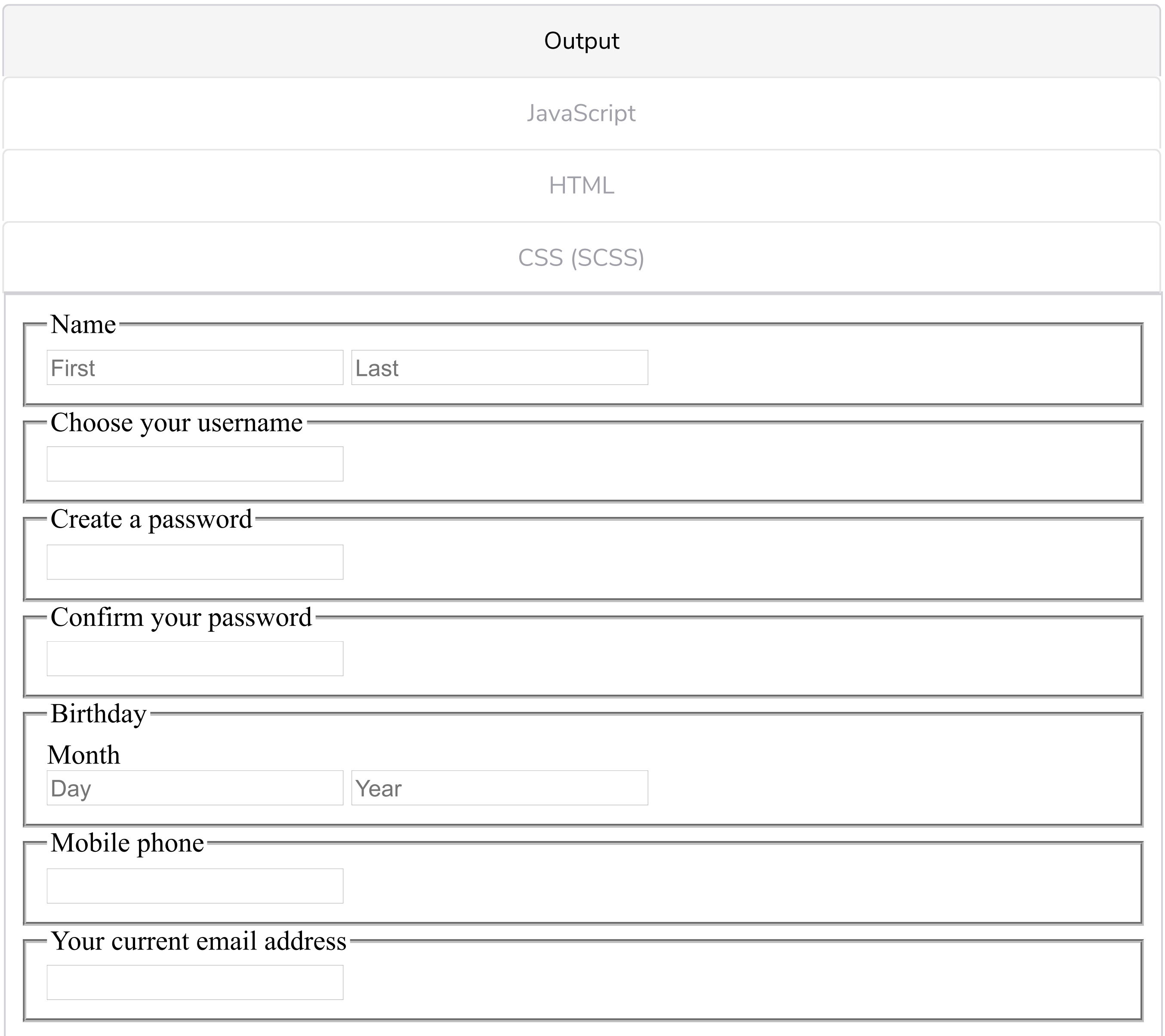
Unique Errors for Each Field

Back to our form! In this lesson, we'll learn how to distinguish each field instead of showing the same error for each one.

WE'LL COVER THE FOLLOWING

- Selecting the right validator

Returning to our component, we last left it in a state that was ready to accept validation rules. If you'll remember, we wrote some basic code to show an error on each input field upon blur, as if validation failed.



Name
First Last

Choose your username

Create a password

Confirm your password

Birthday
Month
Day Year

Mobile phone

Your current email address

Selecting the right validator

To correctly validate each field, of course, we have to differentiate which one is being blurred and apply the rules for that specific field.

Take a moment to inspect the code above. Without changing anything, how would you know which validation rule to apply in the `validate` function?

Q

How do you know which validation rule to apply?

COMPLETED 0%

1 of 1

To answer this question, we should consider what’s important to us.

As much as possible, you should keep the **view layer** separate from the **logic layer**, which is where you do computations (people also call this the business or controller layer). This is for **maintainability** of your code as it evolves. If you’re building something that lasts, you should expect adding and removing features, doing redesigns, and other sorts of changes. If every time you make changes to your HTML, you also have to go and make changes in your JavaScript code that encodes the business logic, it's extra work and is an unnecessary dependency. And dependencies expose more points for bugs to occur.

Given that, would you change your answer to the quiz? The order is subject to revision – maybe you'll want to collect additional data in the very near future or remove one due to some legal implications. The placeholder's purpose is entirely presentational and can easily change. The accessibility label should be human-readable and isn't suitable either for being the source of truth for this input. The class name is the most reasonable out of the choices since it doesn't change often, and we have to rely on it already in JavaScript to select elements. It's not the best, though. What if we want to wrap inputs around another element later, and we change the class name to reflect the new hierarchy? Instead, we can give elements of our own attributes for this purpose. These tags are just there for the logic layer consumption, so no matter what changes occur in the HTML (view layer), they stay the same.

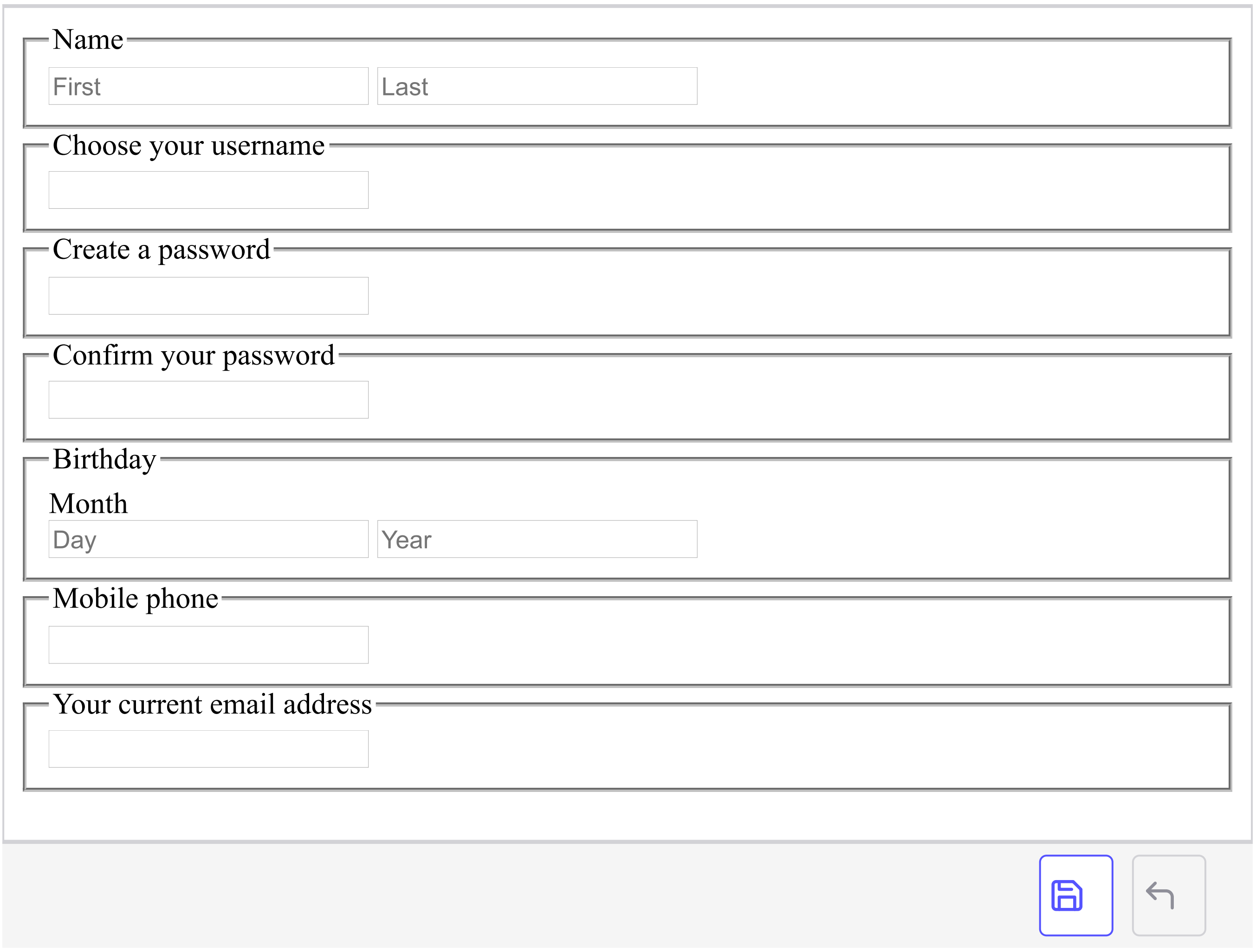
So let's add a `data-field` to each one. `data-` is just a prefix that the HTML specs decided is valid for adding custom properties. So instead of like `<div field='name'>`, we use `data-field=name`.

Output

JavaScript

HTML

CSS (SCSS)



Name

First | Last

Choose your username

Create a password

Confirm your password

Birthday

Month

Day | Year

Mobile phone

Your current email address

In JavaScript, we can access those custom attributes via `element.dataset`, and now we can start applying validation rules to individual fields!