# Introduction to HTML Forms

Forms are a part of everyday life. When we use a physical form in real life, we write down information and give it to someone to process. Think of the times you've had to fill out information for various applications like a job, or a bank account, or dropped off a completed suggestion card — each instance is a form!

Just like a physical form, an HTML <form>element is responsible for collecting information to send somewhere else. Every time we browse the internet we come into contact with many forms and we might not even realize it. There's a good chance that if you're typing into a text field or providing an input, the field that you're typing into is part of a <form>!

In this lesson, we'll go over the structure and syntax of a <form> and the many elements that populate it.

# How a Form Works

We can think of the internet as a network of computers which send and receive information. Computers need an HTTP request to know how to communicate. The HTTP request instructs the receiving computer how to handle the incoming information. More information can be found in our article about [HTTP requests](https://www.codecademy.com/articles/http-requests).

The <form> element is a great tool for collecting information, but then we need to send that information somewhere else for processing. We need to supply the <form> element with both the location of where the <form>'s information goes and what HTTP request to make. Take a look at the sample <form> below:

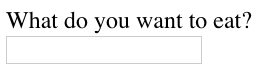
**Adding a Label**

In the previous exercise we created an <input>element but we didn't include anything to explain what the <input> is used for. For a user to properly identify an <input> we use the appropriately named <label> element.

The <label> element has an opening and closing tag and displays text that is written between the opening and closing tags. To associate a <label> and an <input>, the <input> needs an id attribute. We then assign the for attribute of the <label> element with the value of the id attribute of <input>, like so:

<form action="/example.html" method="POST"> <label for="meal">What do you want to eat?</label> <br> <input type="text" name="food" id="meal"> </form>

The code above renders:



Let's see the <label> element in action!

<form action="/example.html" method="POST">

<label for="meal">What do you want to eat?</label>

<br>

<input type="text" name="food" id="meal">

</form>

Look, now users know what the <input>element is for! Another benefit for using the <label> element is when this element is clicked, the corresponding <input> is highlighted/selected.

E

<form>

                <h1>Login to start creating a burger!</h1>

<!--Add your code below-->

<label for="username">Username</label>

<input type="text" name="username" id="username">

</form>

# Password Input

Think about all those times we have to put sensitive information, like a password or PIN, into a <form>. We wouldn't want our information to be seen by anyone peeking over our shoulder! Luckily, we have the type="password" attribute for <input>!

An <input type ="password"> element will replace input text with another character like an asterisk (\*) or a dot (•). The code below provides an example of how to create a password field:

<form>

<h1>Login to start creating a burger!</h1>

<label for="username">Username:</label>

<input type="text" name="username" id="username">

<br>

<label for="user-pw">Password:</label>

<input type="password" name="user-pw" id="user-pw">

</form>

After a user types into the field, it would look like:

password field in a form with 6 dots showing text added to the field

Even though the password field obscures the text of the password, when the form is submitted, the value of the text is sent. In other words, if "hunter2" is typed into the password field, "user-password=hunter2" is sent along with the other information on the form.

# Number Input

We've now gone over two type attributes for <input> related to text. But, we might want our users to type in a number — in which case we can set the type attribute to… (you guessed it)... "number"!

By setting type="number" for an <input> we can restrict what users type into the input field to just numbers (and a few special characters like -, +, and .). We can also provide a stepattribute which creates arrows inside the input field to increase or decrease by the value of the

step attribute. Below is the code needed to render an input field for numbers:

<form>

<label for="years"> Years of experience: </label>

<input id="years" name="years" type="number" step="1">

</form>

Which renders:

rendered number input field with arrows to the right hand side of the field

Now it's time to apply this knowledge.

# Range Input

Using an <input type="number"> is great if we want to allow users to type in any number of their choosing. But, if we wanted to limit what numbers our users could type we might consider using a different type value. Another option we could use is setting type to "range"which creates a slider.

To set the minimum and maximum values of the slider we assign values to the min and max attribute of the <input>. We could also control how smooth and fluid the slider works by assigning the step attribute a value. Smaller step values will make the slider more fluidly, whereas larger step values will make the slider move more noticeably. Take a look at the code to create a slider:

<form>

<label for="volume"> Volume Control</label>

<input id="volume" name="volume" type="range" min="0" max="100" step="1">

</form>

The code above renders:rendered slider for volume control

In the example above, every time the slider

moves by one, the value of the <input>'s value attribute changes.

<form>

<section class="cooked">

<label for="doneness">How do you want your patty cooked</label>

<br>

<span>Rare</span>

<!--Add your code below-->

<input type="range" name="doneness" id="doneness" min="0" max="5" step="0.5">

<span>Well-Done</span>

</section>

</form>

# Checkbox Input

So far the types of inputs we've allowed were all single choices. But, what if we presented multiple options to users and allow them to select any number of options? Sounds like we could use checkboxes! In a <form> we would use the <input> element and set type="checkbox". Examine the code used to create multiple checkboxes:

<form>

<p>Choose your pizza toppings:</p>

<label for="cheese">Extra cheese</label>

<input id="cheese" name="topping" type="checkbox" value="cheese">

<br>

<label for="pepperoni">Pepperoni</label>

<input id="pepperoni" name="topping" type="checkbox" value="pepperoni">

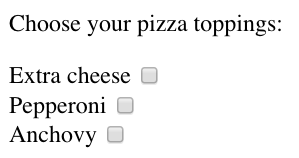
<br>

<label for="anchovy">Anchovy</label>

<input id="anchovy" name="topping" type="checkbox" value="anchovy">

</form>

Which renders:



Notice in the example provided:

there are assigned values to the valueattribute of the checkboxes. These values are not visible on the form itself, that's why it is important that we use an associated <label> to identify the checkbox.

* each <input> has the same value for the name attribute. Using the same name for each checkbox groups the <input>s together. However, each <input> has a unique id to pair with a <label>.

Alright, time to use checkboxes in our code!

# Radio Button Input

Checkboxes work well if we want to present users with multiple options and let them choose one or more of the options. However, there are cases where we want to present multiple options and only allow for one selection — like asking users if they agree or disagree with the terms and conditions. Let's look over the code used to create radio buttons:

<form>

<p>What is sum of 1 + 1?</p>

<input type="radio" id="two" name="answer" value="2">

<label for="two">2</label>

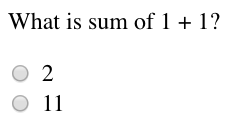
<br>

<input type="radio" id="eleven" name="answer" value="11">

<label for="eleven">11</label>

</form>

Which renders:



Notice from the code snippet, radio buttons (like checkboxes) do not display their value. We have an associated <label> to represent the value of the radio button. To group radio buttons together, we assign them the same name and only one radio button from that group can be selected.

# Dropdown list

Radio buttons are great if we want our users to pick one option out of a few visible options, but imagine if we have a whole list of options! This situation could quickly lead to a lot of radio buttons to keep track of.

An alternative solution is to use a dropdown list to allow our users to choose one option from an organized list. Here's the code to create a dropdown menu:

<form>

<label for="lunch">What's for lunch?</label>

<select id="lunch" name="lunch">

<option value="pizza">Pizza</option>

<option value="curry">Curry</option>

<option value="salad">Salad</option>

<option value="ramen">Ramen</option>

<option value="tacos">Tacos</option>

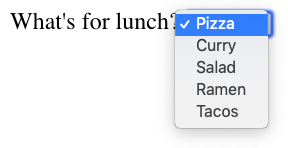
</select>

</form>

Which renders:

rendered dropdown list with the first option showing

And if we click on the field containing the first option, the list is revealed:



Notice in the code that we're using the element <select> to create the dropdown list. To populate the dropdown list, we add multiple <option>elements, each with a value attribute. By default, only one of these options can be selected.

The text rendered is the text included between the opening and closing <option> tags. However, it is the value of the value attribute that is used in <form> submission (notice the difference in the text and valuecapitalization). When the <form> is submitted, the information from this input field will be sent using the name of the <select> and the value of the chosen <option>. For instance, if a user selected Pizza from the dropdown list, the information would be sent as "lunch=pizza".

# Datalist Input

Even if we have an organized dropdown list, if the list has a lot of options, it could be tedious for users to scroll through the entire list to locate one option. That's where using the <datalist> element comes in handy.

The <datalist> is used with an <input type="text"> element. The <input> creates a text field that users can type into and filter options from the <datalist>. Let's go over a concrete example:

The <datalist> is used with an <input type="text"> element. The <input> creates a text field that users can type into and filter options from the <datalist>. Let's go over a concrete example:

<html>

<body>

<form>

<label for="city">Ideal city to visit?</label>

<input type="text" list="cities" id="city" name="city">

<datalist id="cities">

<option value="New York City"></option>

<option value="Tokyo"></option>

<option value="Barcelona"></option>

<option value="Mexico City"></option>

<option value="Melbourne"></option>

<option value="Other"></option>

</datalist>

</form>

</body>

</html>

Notice, in the code above, we have an <input> that has a list attribute. The <input> is associated to the <datalist> via the <input>'s list attribute and the id of the <datalist>.

From the code provided, the following form is rendered:

input field with a label 'Ideal city to visit?'

And when field is selected:

clicking on the input field reveals a dropdown 
list

While <select> and <datalist> share some similarities, there are some major differences. In the associated <input> element, users can type in the input field to search for a particular option. If none of the <option>s match, the user can still use what they typed in. When the form is submitted, the value of the <input>'s name and the value of the option selected, or what the user typed in, is sent as a pair.

Now it's time to make a <datalist> of our own!

# Textarea element

An <input> element with type="text" creates a single row input field for users to type in information. However, there are cases where users need to write in more information, like a blog post. In such cases, instead of using an <input>, we could use <textarea>.

The <textarea> element is used to create a bigger text field for users to write more text. We can add the attributes rows and cols to determine the amount of rows and columns for the <textarea>. Take a look:

<form>

<label for="blog">New Blog Post: </label>

<br>

<textarea id="blog" name="blog" rows="5" cols="30">

</textarea>

</form>

In the code above, an empty <textarea> that is 5 rows by 30 columns is rendered to the page like so:



If we wanted an even bigger text field, we could click and drag on the bottom right corner to expand it.

When we submit the form, the value of <textarea> is the text written inside the box. If we want to add a default value to text to <textarea> we would include it within the opening and closing tags like so:

<textarea>Adding default text</textarea>

This code will render a <textarea> that contains pre-filled text: "Adding default text".

But don't just take our word for it, let's test it out!

# Submit Form

Remember, the purpose of a form is to collect information that will be submitted. That's the role of the submit button — users click on it when they are finished with filling out information in the <form> and they're ready to send it off. Now that we've gone over how to create various input elements, let's now go over how to create a submit button!

To make a submit button in a <form>, we're going to use the reliable <input> element and set the type to "submit". For instance:

<form>

<input type="submit" value="Send">

</form>

Which renders:

rendered submit button

Notice in the code snippet that the valueassigned to the <input> shows up as text on the submit button. If there isn't a valueattribute, the default text, Submit shows up on the button.

# Introduction to HTML Form Validation

**Validation is the concept of checking user provided data against the required data.**

 server-side validation, this happens when data is sent to another machine (typically a server) for validation. An example of this type of validation is the usage of a login page. The form on the login page accepts username and password input, then sends the data to a server which checks that the pair matches up correctly.

we use client-side validation if we want to check the data on the browser (the client). This validation occurs before data is sent to the server.

# Requiring an Input

Sometimes we have fields in our <form>s which are not optional. we can add the required attribute to an <input> element. Take for example:

<form action="/example.html" method="POST">

<label for="allergies">Do you have any dietary restrictions?</label>

<br>

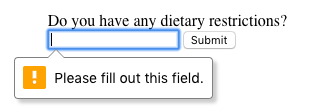
<input id="allergies" name="allergies" type="text" required>

<br>

<input type="submit" value="Submit">

</form>

This renders a text box, and if we try to submit the <form> without filling it out we get this message:



# Set a Minumum and Maximum

Another built-in validation we can use is to assign a minimum or maximum value for a number field, e.g. <input type="number"> and <input type="range">.

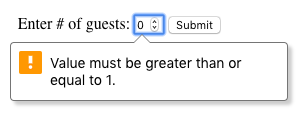
<form action="/example.html" method="POST">

<label for="guests">Enter # of guests:</label>

<input id="guests" name="guests" type="number" min="1" max="4">

<input type="submit" value="Submit">

</form>

If a user tries to submit an input that is less than 1 a warning will appear:

# Checking Text Length

There are certainly cases where we wouldn't want our users typing more than a certain number of characters (think about the message cap for Twitter).

To set a minimum number of characters for a text field, we add the minlength attribute and a value to set a minimum value.

form action="/example.html" method="POST">

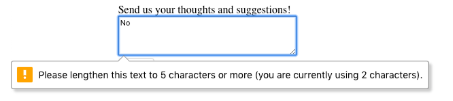
<label for="summary">Summarize your fillings in less than 250 characters</label>

<input id="summary" name="summary" type="text" minlength="5" maxlength="250" required>

<input type="submit" value="Submit">

</form>

If a user tries to submit the <form> with less than the set minimum, this message appears:



And if a user tries to type in more than the maximum allowed number of characters, they don't get a warning message, but they can't type it in!

Let's add this validation to our <form>.

# Matching a Pattern

For cases when we want user input to follow specific guidelines, we use the pattern attribute and assign it a regular expression, or regex. Regular expressions are a sequence of characters that make up a search pattern. If the input matches the regex, the form can be submitted.

Let's say we wanted to check for a valid credit card number (a 14 to 16 digit number). We could use the regex: [0-9]{14,16} which checks that the user provided only numbers and that they entered at least 14 digits and at most 16 digits. To add this to a form

<form action="/example.html" method="POST">

<label for="payment">Credit Card Number (no spaces):</label>

<br>

<input id="payment" name="payment" type="text" required pattern="[0-9]{14,16}">

<input type="submit" value="Submit">

</form>



https://www.codecademy.com/courses/learn-html/projects/form-a-story?action=resume\_content\_item