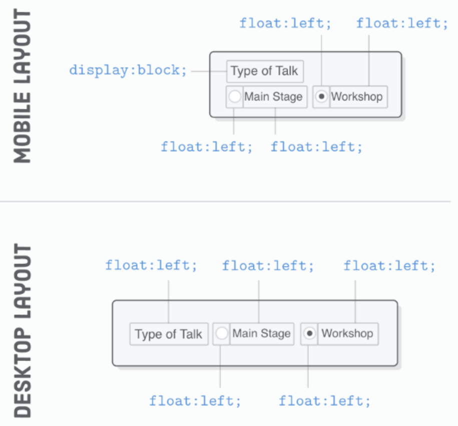
STYLING RADIO BUTTONS

We have a few things working against us with when it comes to styling radio buttons. First, there's simply more elements to worry about. Second, the `<fieldset>` and `<legend>` elements have rather ugly default styles, and there's not a whole lot of consistency in these defaults across browsers. Third, at the time of this writing, `<fieldset>` doesn't support flexbox.

But don't fret! This is a good example of floats being a useful fallback for legacy/troublesome elements. You'll notice that we didn't wrap the radio buttons in our existing `.form-row` class, opting instead for a new `.legacy-form-row` class. This is because it's going to be completely separate from our other elements, using floats instead of flexbox.



Start with the mobile and tablet styles by adding the following rules outside of our media query. We want to get rid of the default `<fieldset>` and `<legend>` styles, then float the radio buttons and labels so they appear in one line underneath the `<legend>`:

```
.legacy-form-row {
  border: none;
  margin-bottom: 40px;
}

.legacy-form-row legend {
  margin-bottom: 15px;
}

.legacy-form-row .radio-label {
  display: block;
  font-size: 14px;
  padding: 0 20px 0 10px;
}

.legacy-form-row input[type='radio'] {
  margin-top: 2px;
}

.legacy-form-row .radio-label,
.legacy-form-row input[type='radio'] {
  float: left;
}
```

For the desktop layout, we need to make the `<legend>` line up with the `<label>` elements in the previous section (hence the `width: 120px` line), and we need to float *everything* to the left so they appear on the same line. Update our media query to include the following:

```
@media only screen and (min-width: 700px) {
  /* ... */
  .legacy-form-row {
    margin-bottom: 10px;
  }
  .legacy-form-row legend {
    width: 120px;
    text-align: right;
    padding-right: 20px;
  }
  .legacy-form-row legend {
    float: left;
  }
}
```

As far as layouts go, this is a pretty good cross-browser solution. However, customizing the appearance of the actual button is another story. It's possible by taking advantage of the `checked` attribute, but it's a little bit complicated. We'll leave you to Google "custom radio button CSS" and explore that rabbit hole on your own.

SELECT ELEMENTS (DROPDOWN MENUS)

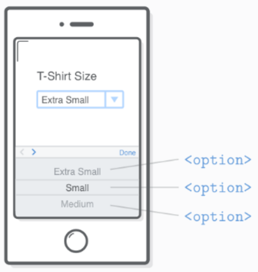
Dropdown menus offer an alternative to radio buttons, as they let the user select one out of many options. The `<select>` element represents the dropdown menu, and it contains a bunch of `<option>` elements that represent each item.

```
<div class='form-row'>
  <label for='t-shirt'>T-Shirt Size</label>
  <select id='t-shirt' name='t-shirt'>
    <option value='xs'>Extra Small</option>
    <option value='s'>Small</option>
    <option value='m'>Medium</option>
    <option value='l'>Large</option>
  </select>
</div>
```

Just like our radio button `<input/>` elements, we have `name` and `value` attributes that get passed to the backend server. But, instead of being defined on a single element, they're spread across the `<select>` and `<option>` elements.

STYLING SELECT ELEMENTS

And, also just like our radio buttons, `<select>` elements are notoriously hard to style. However, there's a reason for this. Dropdowns are a complex piece of interactivity, and their behavior changes significantly across devices. For instance, on an iPhone, clicking a `<select>` element brings up a native scrolling UI component that makes it much easier to navigate the menu.



It's usually a good idea to let the browser/device determine the best way to preset a `<select>` element, so we'll be keeping our CSS pretty simple. Unfortunately, even the simplest things are surprisingly hard. For instance, try changing the font size of our `<select>` element:

```
.form-row select {
  width: 100%;
  padding: 5px;
  font-size: 14px;            /* This won't work in Chrome or Safari */
}
```

This will work in Firefox, but not in Chrome or Safari! To sort of fix this, we can use a vendor-specific prefix for the `appearance` property:

```
.form-row select {
  width: 100%;
  padding: 5px;
  font-size: 14px;            /* This won't work in Chrome or Safari */
  -webkit-appearance: none;   /* This will make it work */
}
```

The `-webkit` prefix will *only* apply to Chrome and Safari (which are powered by the WebKit rendering engine), while Firefox will remain unaffected. This is effectively a hack, and even MDN says not to use this CSS property.

Style difficulties like this are a serious consideration when building a form. If you need custom styles, you may be better off using radio buttons or JavaScript UI widgets. Bootstrap Dropdowns and jQuery Selectmenu's are common JavaScript solutions for customizing select menus. In any case, at least you now understand the problem. You can read more about `<select>` issues here.

TEXTAREAS

The `<textarea>` element creates a multi-line text field designed to collect large amounts of text from the user. They're suitable for things like biographies, essays, and comments. Go ahead and add a `<textarea>` to our form, along with a little piece of instructional text:

```
<div class='form-row'>
  <label for='abstract'>Abstract</label>
  <textarea id='abstract' name='abstract'></textarea>
  <div class='instructions'>Describe your talk in 500 words or less</div>
</div>
```

Note that this isn't self-closing like the `<input/>` element, so you always need a closing `</textarea>` tag. If you want to add any default text, it needs to go *inside* the tags opposed to a `value` attribute.

STYLING TEXTAREAS

Fortunately, styling textareas is pretty straightforward. Add the following to our `styles.css` file (before the media query):

```
.form-row textarea {
  font-family: "Helvetica", "Arial", sans-serif;
  font-size: 14px;

  border: 1px solid #D6D9DC;
  border-radius: 3px;

  min-height: 200px;
  margin-bottom: 10px;
  padding: 7px;
  resize: none;
  resize: none;
}

.form-row .instructions {
  color: #999999;
  font-size: 14px;
  margin-bottom: 30px;
}
```

By default, many browsers let the user resize `<textarea>` elements to whatever dimensions they want. We disabled this here with the `resize` property.

We also need a little tweak in our desktop layout. The `.instructions` `<div>` needs to be underneath the `<textarea>`, so let's nudge it left by the width of the `<label>` column. Add the following rule to the end of our media query:

```
@media only screen and (min-width: 700px) {
  /* ... */
  .form-row .instructions {
    margin-left: 120px;
  }
}
```

CHECKBOXES

Checkboxes are sort of like radio buttons, but instead of selecting only one option, they let the user pick as many as they want. This simplifies things, since the browser doesn't need to know which checkboxes are part of the same group. In other words, we don't need a `<fieldset>` wrapper or shared `name` attributes. Add the following to the end of our form:

```
<div class='form-row'>
  <label class='checkbox-label' for='available'>
  <input id='available'
         name='available'
         type='checkbox'
         value='is-available'/>
  <span>I'm actually available the date of the talk</span>
  </label>
</div>
```

The way we used `<label>` here was a little different than previous sections. Instead of being a separate element, the `<label>` wraps its corresponding `<input/>` element. This is perfectly legal, and it'll make it easier to match our desired layout. It's still a best practice to use the `for` attribute.

STYLING CHECKBOXES

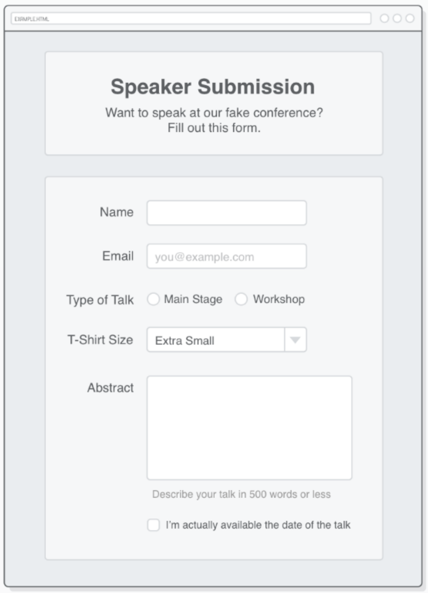
For the mobile layout, all we need to do is override the `margin-bottom` that we put on the rest the `<label>` elements. Add the following to `styles.css`, outside of the media query:

```
.form-row .checkbox-label {
  margin-bottom: 0;
}
```

And inside the media query, we have to take that 120-pixel label column into account:

```
@media only screen and (min-width: 700px) {
  /* ... */
  .form-row .checkbox-label {
    margin-left: 120px;
    width: auto;
  }
}
```

By wrapping both the checkbox and the label text, we're able to use a `width: auto` to make the entire form field be on a single line (remember that the `auto` width makes the box match the size of its contents).

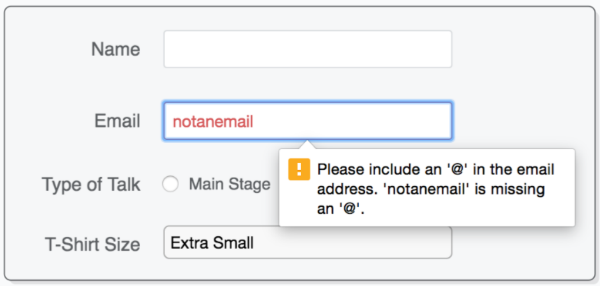


SUBMIT BUTTONS

Finally, let's finish off our form with a submit button. The `<button>` element represents a button that will submit its containing `<form>`:

```
<div class='form-row'>
  <button>Submit</button>
</div>
```

Clicking the button tells the browser to validate all of the `<input/>` elements in the form and submit it to the `action` URL if there aren't any validation problems. So, you should now be able to type in something that's not an email address into our email field, click the `<button>`, and see an error message.


Name
Email
notanemail
Please include an '@' in the email address. 'notanemail' is missing an '@'.
Type of Talk
Main Stage
T-Shirt Size
Extra Small

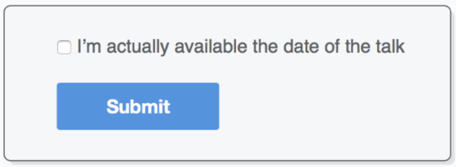
This also gives us a chance to see how the user's input gets sent to the server. First, enter some values into all the `<input/>` fields, making sure the email address validates correctly. Then, click the button and inspect the resulting URL in your browser. You should see something like this:

```
speaker-submission.html?full-name=Rick&email=rick%40internetingishard.com&talk-
```

Everything after the `?` represents the variables in our form. Each `<input/>`'s `name` attribute is followed by an equal sign, then its value, and each variable is separated by an `&` character. If we had a backend server, it'd be pretty easy for it to pull out all this information, query a database (or whatever), and let us know whether the form submission was successful or not.

STYLING BUTTONS

We had some experience styling buttons in the pseudo-classes section of the *CSS Selectors* chapter. Back then, we were applying these styles to an `<a>` element, but we can use the same techniques on a `<button>`.



Clean up that ugly default `<button>` styling by adding the following to our stylesheet:

```
.form-row button {
  font-size: 16px;
  font-weight: bold;

  color: #FFFFFF;
  background-color: #5995DA;

  border: none;
  border-radius: 3px;

  padding: 10px 40px;
  cursor: pointer;
}

.form-row button:hover {
  background-color: #76AEED;
}

.form-row button:active {
  background-color: #407FC7;
}
```

As with our checkbox, we need to take that `120px` label column into account, so include one more rule inside our media query:

```
@media only screen and (min-width: 700px) {
  /* ... */
  .form-row button {
    margin-left: 120px;
  }
}
```

SUMMARY

In this chapter, we introduced the most common HTML form elements. We now have all these tools for collecting input from our website visitors:

- `<input type='text'/>`
- `<input type='email'/>`
- `<input type='radio'/>`
- `<select>` and `<option>`
- `<textarea>`
- `<input type='checkbox'/>`
- `<button>`

You should be pretty comfortable with the HTML and CSS required to build beautiful forms, but actually making these forms functional requires some skills we don't have yet. That stuff is out of scope for this tutorial, but it might help to have some context. Generally speaking, there are two ways to process forms:

- Use the `action` attribute to send the form data to a backend URL, which then redirects to a success or error page. We got a little glimpse of this in the previous section, and it doesn't require any JavaScript.
- Use AJAX queries to submit the form without leaving the page. Success or error messages are displayed on the same page by manipulating the HTML with JavaScript.

Depending on how your organization is structured, form processing may not be part of your job role as a frontend web developer. If that's the case, you'll need to coordinate closely with a backend developer on your team to make sure the `<form>` submits the correct name-value pairs. Otherwise, it'll be up to you to make sure the frontend and backend of your forms fit neatly together.

Next, we have our final chapter in *HTML & CSS Is Hard*. We'll round out our frontend skills with a thorough discussion of web fonts and practical typographic principles that every web developer should know about.

NEXT CHAPTER >