

Learn to Code  
HTML & CSS

## Lesson 10

# Building Forms

**SPONSOR** Auth0 — What would you do with 33% less development time?**In this Lesson**

10

**HTML**

- [Initializing a Form](#)
- [Text Fields & Textareas](#)
- [Multiple Choice Inputs & Menus](#)
- [Form Buttons](#)
- [Other Inputs](#)
- [Organizing Form Elements](#)
- [Form & Input Attributes](#)

**SHARE**

Forms are an essential part of the Internet, as they provide a way for websites to capture information from users and to process requests, and they offer controls for nearly every imaginable use of an application. Through controls or fields, forms can request a small amount of information—often a search query or a username and password—or a large amount of information—perhaps shipping and billing information or an entire job application.

We need to know how to build forms in order to acquire user input. In this lesson we'll discuss how to use HTML to mark up a form, which elements to use to capture different types of data, and how to style forms with CSS. We won't get too deep into how information from a form is processed and handled on the back end of a website. Form processing is a deeper topic, outside the realm of this book; for now we'll stick to the creation and styling of forms.

## Initializing a Form

To add a [form](#) to a page, we'll use the `<form>` element. The `<form>` [element](#) identifies where on the page control elements will appear. Additionally, the `<form>` element will wrap all of the elements included within the form, much like a `<div>` element.

```
1      <form action="/login" method="post">
2          ...
3      </form>
```

A handful of different attributes can be applied to the `<form>` element, the most common of which are `action` and `method`. The `action` attribute contains the URL to which information included within the form will be sent for processing by the server. The `method` attribute is the HTTP method browsers should use to submit the form data. Both of these `<form>` attributes pertain to submitting and processing data.

## Text Fields & Textareas

When it comes to gathering text input from users, there are a few different elements available for obtaining data within forms. Specifically, text fields and textareas are used for collecting text- or string-based data. This data may include passages of text content, passwords, telephone numbers, and other information.

### Text Fields

One of the primary elements used to obtain text from users is the `<input>` element. The `<input>` [element](#) uses the `type` attribute to define what type of information is to be captured within the control. The most popular `type` attribute value is `text`, which denotes a single line of text input.

Along with setting a `type` attribute, it is best practice to give an `<input>` element a `name` attribute as well. The `name` attribute value is used as the name of the control and is submitted along with the input data to the server.

```
1      <input type="text" name="username">
```

### Text Fields Demo

HTML

RESULT



The `<input>` element is self-contained, meaning it uses only one tag and it does not wrap any other content. The value of the element is provided by its attributes and their corresponding values.

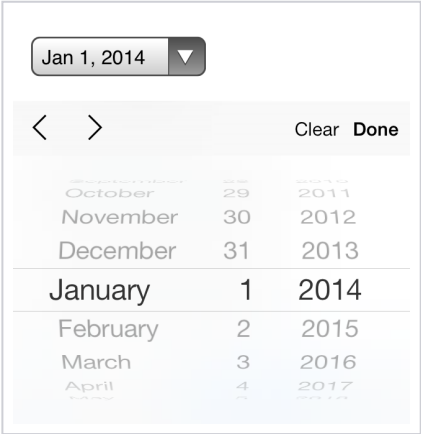
Originally, the only two text-based type attribute values were `text` and `password` (for password inputs); however, HTML5 brought along a handful of new type attribute values.

These values were added to provide clearer semantic meaning for inputs as well as to provide better controls for users. Should a browser not understand one of these HTML5 type attribute values, it will automatically fall back to the `text` attribute value. Below is a list of the new HTML5 input types.

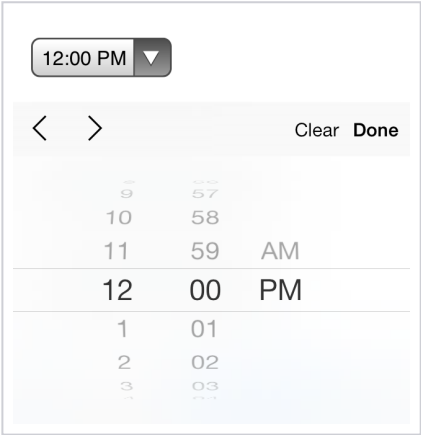
- `color`
- `date`
- `datetime`
- `email`
- `month`
- `number`
- `range`
- `search`
- `tel`
- `time`
- `url`
- `week`

The following `<input>` elements show a few of these HTML5 type attribute values in use; the following figures show how these unique values may look within iOS. Notice how the different values provide different controls, all of which make gathering input from users easier.

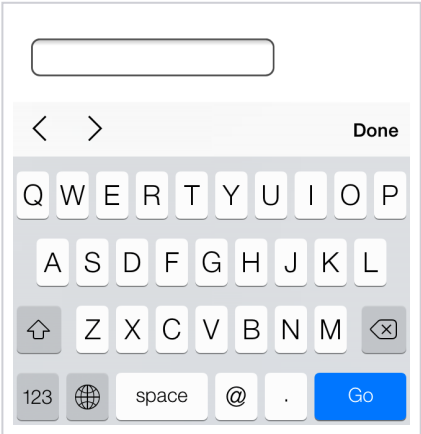
```
1      <input type="date" name="birthday">
2      <input type="time" name="game-time">
3      <input type="email" name="email-address">
4      <input type="url" name="website">
5      <input type="number" name="cost">
6      <input type="tel" name="phone-number">
```



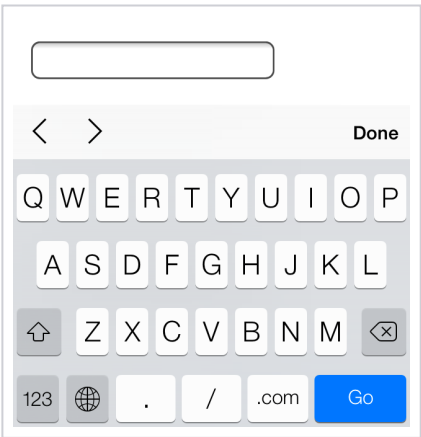
**Fig 10.01**  
iOS7 controls for an `<input>` element with a `type` attribute value of `date`



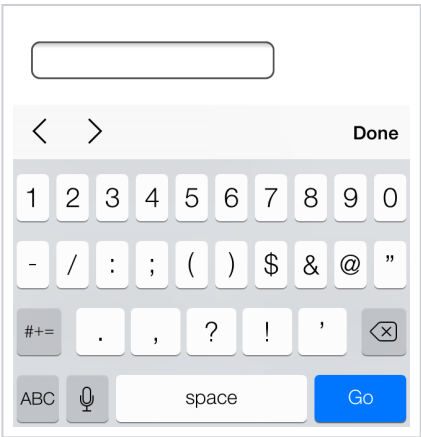
**Fig 10.02**  
iOS7 controls for an `<input>` element with a `type` attribute value of `time`



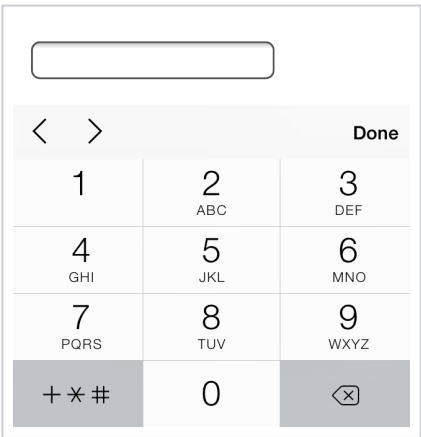
**Fig 10.03**  
iOS7 controls for an `<input>` element with a `type` attribute value of `email`



**Fig 10.04**  
iOS7 controls for an `<input>` element with a `type` attribute value of `url`



**Fig 10.05**  
iOS7 controls for an `<input>` element with a `type` attribute value of `number`



**Fig 10.06**  
iOS7 controls for an `<input>` element with a `type` attribute value of `tel`

Textarea


Another element that's used to capture text-based data is the `<textarea>` element. The `<textarea>` element differs from the `<input>` element in that it can accept larger passages of text spanning multiple lines. The `<textarea>` element also has start and end tags that can wrap plain text. Because the `<textarea>` element only accepts one type of value, the `type` attribute doesn't apply here, but the `name` attribute is still used.

```
1 <textarea name="comment">Add your comment here</textarea>
```

### Textarea Demo

HTML

RESULT



The `<textarea>` element has two sizing attributes: `cols` for width in terms of the average character width and `rows` for height in terms of the number of lines of visible text. The size of a textarea, however, is more commonly identified using the `width` and `height` properties within CSS.

## Multiple Choice Inputs & Menus

Apart from text-based input controls, HTML also allows users to select data using multiple choice and drop-down lists. There are a few different options and elements for these form controls, each of which has distinctive benefits.

### Radio Buttons

Radio buttons are an easy way to allow users to make a quick choice from a small list of options. Radio buttons permit users to select one option only, as opposed to multiple options.

To create a radio button, the `<input>` element is used with a `type` attribute value of `radio`. Each radio button element should have the same `name` attribute value so that all

of the buttons within a group correspond to one another.

With text-based inputs, the value of an input is determined by what a user types in; with radio buttons a user is making a multiple choice selection. Thus, we have to define the input value. Using the `value` attribute, we can set a specific value for each `<input>` element.

Additionally, to preselect a radio button for users we can use the Boolean attribute `checked`.

```
1      <input type="radio" name="day" value="Friday" checked> Friday
2      <input type="radio" name="day" value="Saturday"> Saturday
3      <input type="radio" name="day" value="Sunday"> Sunday
```

### Radio Buttons Demo

## Check Boxes

Check boxes are very similar to radio buttons. They use the same attributes and patterns, with the exception of checkbox as their `type` attribute value. The difference between the two is that check boxes allow users to select multiple values and tie them all to one control name, while radio buttons limit users to one value.

```
1      <input type="checkbox" name="day" value="Friday" checked> Friday
2      <input type="checkbox" name="day" value="Saturday"> Saturday
3      <input type="checkbox" name="day" value="Sunday"> Sunday
```

## Check Boxes Demo

## Drop-Down Lists

Drop-down lists are a perfect way to provide users with a long list of options in a practical manner. A long column of radio buttons next to a list of different options is not only visually unappealing, it's daunting and difficult for users to comprehend, especially those on a mobile device. Drop-down lists, on the other hand, provide the perfect format for a long list of choices.

To create a drop-down list we'll use the `<select>` and `<option>` elements. The `<select>` element wraps all of the menu options, and each menu option is marked up using the `<option>` element.

The `name` attribute resides on the `<select>` element, and the `value` attribute resides on the `<option>` elements that are nested within the `<select>` element. The `value` attribute on each `<option>` element then corresponds to the `name` attribute on the `<select>` element.

Each `<option>` element wraps the text (which is visible to users) of an individual option within the list.

Much like the checked Boolean attribute for radio buttons and check boxes, drop-down menus can use the `selected` Boolean attribute to preselect an option for users.

```
1      <select name="day">
2          <option value="Friday" selected>Friday</option>
3          <option value="Saturday">Saturday</option>
4      </select>
```



```
5      <option value="Sunday">Sunday</option>
      </select>
```

### Drop-Down Lists Demo

## Multiple Selections

The Boolean attribute `multiple`, when added to the `<select>` element for a standard drop-down list, allows a user to choose more than one option from the list at a time. Additionally, using the `selected` Boolean attribute on more than one `<option>` element within the menu will preselect multiple options.

The size of the `<select>` element can be controlled using CSS and should be adjusted appropriately to allow for multiple selections. It may be worthwhile to inform users that to choose multiple options they will need to hold down the Shift key while clicking to make their selections.

```
1      <select name="day" multiple>
2          <option value="Friday" selected>Friday</option>
3          <option value="Saturday">Saturday</option>
4          <option value="Sunday">Sunday</option>
5      </select>
```

### Multiple Selections Demo

## Form Buttons

After a user inputs the requested information, buttons allow the user to put that information into action. Most commonly, a submit input or submit button is used to process the data.

### Submit Input

Users click the submit button to process data after filling out a form. The submit button is created using the `<input>` element with a `type` attribute value of `submit`. The `value` attribute is used to specify the text that appears within the button.

```
1 <input type="submit" name="submit" value="Send">
```

### Submit Input Demo

## Submit Button

As an `<input>` element, the submit button is self-contained and cannot wrap any other content. If more control over the structure and design of the input is desired—along with the ability to wrap other elements—the `<button>` element may be used.

The `<button>` element performs the same way as the `<input>` element with the `type` attribute value of `submit`; however, it includes opening and closing tags, which may wrap other elements. By default, the `<button>` element acts as if it has a `type` attribute value of `submit`, so the `type` attribute and value may be omitted from the `<button>` element if you wish.

Rather than using the `value` attribute to control the text within the submit button, the text that appears between the opening and closing tags of the `<button>` element will appear.

```
1      <button name="submit">
2          <strong>Send Us</strong> a Message
3      </button>
```

### Submit Button Demo

## Other Inputs

Besides the applications we've just discussed, the `<input>` element has a few other use cases. These include passing hidden data and attaching files during form processing.

### Hidden Input

Hidden inputs provide a way to pass data to the server without displaying it to users. Hidden inputs are typically used for tracking codes, keys, or other information that is not pertinent to the user but is helpful when processing the form. This information is not displayed on the page; however, it can be found by viewing the source code of a page. It should therefore not be used for sensitive or secure information.

To create a hidden input, you use the `hidden` value for the `type` attribute. Additionally, include the appropriate `name` and `value` attribute values.

```
1      <input type="hidden" name="tracking-code" value="abc-123">
```

### File Input

To allow users to add a file to a form, much like attaching a file to an email, use the `file` value for the `type` attribute.

```
1      <input type="file" name="file">
```

### File Input Demo

Unfortunately, styling an `<input>` element that has a `type` attribute value of `file` is a tough task with CSS. Each browser has its own default input style, and none provide much control to override the default styling. JavaScript and other solutions can be employed to allow for file input, but they are slightly more difficult to construct.

## Organizing Form Elements

Knowing how to capture data with inputs is half the battle. Organizing form elements and controls in a usable manner is the other half. When interacting with forms, users need to understand what is being asked of them and how to provide the requested information.

By using labels, fieldsets, and legends, we can better organize forms and guide users to properly complete them.

### Label

Labels provide captions or headings for form controls, unambiguously tying them together and creating an accessible form for all users and assistive technologies. Created using the `<label>` element, labels should include text that describes the inputs or controls they pertain to.

Labels may include a `for` attribute. The value of the `for` attribute should be the same as the value of the `id` attribute on the form control the label corresponds to. Matching up the `for` and `id` attribute values ties the two elements together, allowing users to click on the `<label>` element to bring focus to the proper form control.

```
1      <label for="username">Username</label>
2      <input type="text" name="username" id="username">
```

## Label Demo

If desired, the `<label>` element may wrap form controls, such as radio buttons or check boxes. Doing so allows omission of the `for` and `id` attributes.

```
1      <label>
2          <input type="radio" name="day" value="Friday" checked> Friday
3      </label>
4      <label>
5          <input type="radio" name="day" value="Saturday"> Saturday
6      </label>
7      <label>
8          <input type="radio" name="day" value="Sunday"> Sunday
9      </label>
```

## Labels with Nested Inputs Demo

## Fieldset

Fieldsets group form controls and labels into organized sections. Much like a `<section>` or other structural element, the `<fieldset>` is a block-level element that wraps related elements, specifically within a `<form>` element, for better organization. Fieldsets, by default, also include a border outline, which can be modified using CSS.

```
1      <fieldset>
2          <label>
3              Username
4              <input type="text" name="username">
5          </label>
6          <label>
7              Password
8              <input type="text" name="password">
9          </label>
10     </fieldset>
```

### Fieldset Demo

## Legend

A legend provides a caption, or heading, for the `<fieldset>` element. The `<legend>` element wraps text describing the form controls that fall within the fieldset. The markup should include the `<legend>` element directly after the opening `<fieldset>` tag. On the page, the legend will appear within the top left part of the fieldset border.

```
1      <fieldset>
2          <legend>Login</legend>
3          <label>
4              Username
5              <input type="text" name="username">
6          </label>
7          <label>
8              Password
9              <input type="text" name="password">
10         </label>
11     </fieldset>
```

### Legend Demo



## Form & Input Attributes

To accommodate all of the different form, input, and control elements, there are a number of attributes and corresponding values. These attributes and values serve a handful of different functions, such as disabling controls and adding form validation. Described next are some of the more frequently used and helpful attributes.

### Disabled

The `disabled` Boolean attribute turns off an element or control so that it is not available for interaction or input. Elements that are disabled will not send any value to the server for form processing.

Applying the `disabled` Boolean attribute to a `<fieldset>` element will disable all of the form controls within the fieldset. If the `type` attribute has a `hidden` value, the `hidden` Boolean attribute is ignored.

```
1      <label>
2      Username
3      <input type="text" name="username" disabled>
4      </label>
```

### Disabled Demo

## Placeholder

The `placeholder` HTML5 attribute provides a hint or tip within the form control of an `<input>` or `<textarea>` element that disappears once the control is clicked in or gains focus. This is used to give users further information on how the form input should be filled in, for example, the email address format to use.

```
1      <label>
2          Email Address
3      <input type="email" name="email-address" placeholder="name@domain.co
4      </label>
```



### Placeholder Demo

The main difference between the `placeholder` and `value` attributes is that the `value` attribute value text stays in place when a control has focus unless a user manually deletes it. This is great for pre-populating data, such as personal information, for a user but not for providing suggestions.

## Required

The `required` HTML5 Boolean attribute enforces that an element or form control must contain a value upon being submitted to the server. Should an element or form control not have a value, an error message will be displayed requesting that the user complete the required field. Currently, error message styles are controlled by the browser and cannot be styled with CSS. Invalid elements and form controls, on the other hand, can be styled using the `:optional` and `:required` CSS pseudo-classes.

Validation also occurs specific to a control's type. For example, an `<input>` element with a `type` attribute value of `email` will require not only that a value exist within the control, but also that it is a valid email address.

```
1      <label>
2      Email Address
3      <input type="email" name="email-address" required>
4      </label>
```

### Required Demo

## Additional Attributes

Other form and form control attributes include, but are not limited to, the following. Please feel free to research these attributes as necessary.

- accept
- autocomplete
- autofocus
- formaction
- formenctype
- formmethod
- formnovalidate
- formtarget
- max
- maxlength
- min
- pattern
- readonly
- selectionDirection
- step

## Login Form Example

The following is an example of a complete login form that includes several different elements and attributes to illustrate what we've covered so far. These elements are then styled using CSS.

### HTML

```
1      <form>
2          <fieldset class="account-info">
3              <label>
4                  Username
5                  <input type="text" name="username">
6              </label>
7              <label>
8                  Password
9                  <input type="password" name="password">
10             </label>
11         </fieldset>
12         <fieldset class="account-action">
13             <input class="btn" type="submit" name="submit" value="Login">
14             <label>
15                 <input type="checkbox" name="remember"> Stay signed in
16             </label>
17         </fieldset>
18     </form>
```

### CSS

```
1      *,
2      *:before,
3      *:after {
4          box-sizing: border-box;
5      }
6      form {
7          border: 1px solid #c6c7cc;
8          border-radius: 5px;
9          font: 14px/1.4 "Helvetica Neue", Helvetica, Arial, sans-serif;
10         overflow: hidden;
11         width: 240px;
12     }
13     fieldset {
14         border: 0;
15         margin: 0;
16         padding: 0;
17     }
18     input {
19         border-radius: 5px;
20         font: 14px/1.4 "Helvetica Neue", Helvetica, Arial, sans-serif;
21         margin: 0;
22     }
23     .account-info {
24         padding: 20px 20px 0 20px;
25     }
26     .account-info label {
27         color: #395870;
28         display: block;
29         font-weight: bold;
30         margin-bottom: 20px;
31     }
32     .account-info input {
33         background: #fff;
34         border: 1px solid #c6c7cc;
35         box-shadow: inset 0 1px 1px rgba(0, 0, 0, .1);
36         color: #636466;
37         padding: 6px;
38         margin-top: 6px;
39         width: 100%;
40     }
41     .account-action {
42         background: #f0f0f2;
```

```
43     border-top: 1px solid #c6c7cc;
44     padding: 20px;
45 }
46 .account-action .btn {
47     background: linear-gradient(#49708f, #293f50);
48     border: 0;
49     color: #fff;
50     cursor: pointer;
51     font-weight: bold;
52     float: left;
53     padding: 8px 16px;
54 }
55 .account-action label {
56     color: #7c7c80;
57     font-size: 12px;
58     float: left;
59     margin: 10px 0 0 20px;
60 }
```

## Required Demo

## In Practice

With an understanding of how to build forms in place, let's create a registration page for our Styles Conference website so that we can begin to gather interest and sell tickets for the event.

- 1 Jumping into our `register.html` file, we'll begin by following the same layout pattern we used on our Speakers and Venue pages. This includes adding a `<section>` element with a class attribute value of `row` just below the registration lead-in section and nesting a `<div>` element with a class attribute value of `grid` directly inside the `<section>` element.

Our code just below the lead-in section for the Register page should look like this:

```
1      <section class="row">
2          <div class="grid">
3              ...
4          </div>
5      </section>
```

As a refresher, the class attribute value of `row` adds a white background and provides some vertical padding, while the class attribute value of `grid` centers our content in the middle of the page and provides some horizontal padding.

- 2 Inside the `<div>` element with a class attribute value of `grid` we're going to create two columns, one covering two-thirds of the page width and one covering one-third of the page width. The two-thirds column will be a `<section>` element on the left-hand side that tells users why they should register for our conference. The one-third column, then, will be a `<form>` element on the right-hand side providing a way for users to register for our conference.

We'll add these two elements, and their corresponding `col-2-3` and `col-1-3` classes, directly inside the `<div>` element with a class attribute value of `grid`. Since both of these elements will be inline-block elements, we need to open a comment directly after the two-thirds column closing tag and then close that comment directly before the one-third column opening tag.

In all, our code should look like this:

```
1      <section class="row">
2          <div class="grid">
3
4              <section class="col-2-3">
5                  ...
6              </section><!--
7
8              --><form class="col-1-3">
9                  ...
10             </form>
```

```

11
12     </div>
13 </section>

```

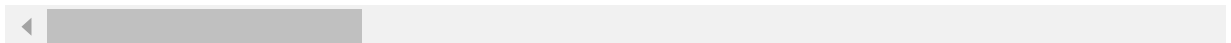
- 3 Now, inside our two-thirds column let's add some details about our event and why it's a good idea for aspiring designers and front-end developers to attend. We'll do so using a handful of different heading levels (along with their pre-established styles), a paragraph, and an unordered list.

In our `<section>` element with a class attribute value of `col-2-3`, the code should look like this:

```

1     <section class="col-2-3">
2
3     <h2>Purchase a Conference Pass</h2>
4     <h5>$99 per Pass</h5>
5
6     <p>Purchase your Styles Conference pass using the form to the right
7
8     <h4>Why Attend?</h4>
9
10    <ul>
11    <li>Over twenty world-class speakers</li>
12    <li>One full day of workshops and two full days of presentations<
13    <li>Hosted at The Chicago Theatre, a historical landmark</li>
14    <li>August in Chicago is simply amazing</li>
15    </ul>
16
17 </section>

```



- 4 Currently our unordered list doesn't have any list item markers. All of the browser default styles have been turned off by the CSS reset we added all the way back in Lesson 1. Let's create some custom styles specifically for this unordered list.

To do so, let's add a class attribute value of `why-attend` to the unordered list.

```

1     <ul class="why-attend">
2     ...

```



3 `</ul>`

With a class available to add styles to, let's create a new section for Register page styles at the bottom of our `main.css` file. Within this section let's use the class to select the unordered list and add a `list-style` of `square` and some bottom and left margins.

The new section at the bottom of our `main.css` file should look like this:

```

1      /*
2      =====
3      Register
4      =====
5      */
6
7      .why-attend {
8          list-style: square;
9          margin: 0 0 22px 30px;
10     }
```

- 5 The details section of our registration page is complete, so now it's time to address our registration form. We'll start by adding the `action` and `method` attributes to the `<form>` element. Since we haven't set up our form processing, these attributes will simply serve as placeholders and will need to be revisited.

The code for our `<form>` element should look like this:

```

1      <form class="col-1-3" action="#" method="post">
2          ...
3      </form>
```

- 6 Next, inside the `<form>` element we'll add a `<fieldset>` element. Inside the `<fieldset>` element we'll add a series of `<label>` elements that wrap a given form control.

We want to collect a user's name, email address, number of desired conference passes, and any potential comments. The name, email address, and number of conference passes are required fields, and we'll want to make sure we use the appropriate elements and attributes for each form control.

With a mix of different input types, select menus, textareas, and attributes, the code for our form should look like the following:

```
1      <form class="col-1-3" action="#" method="post">
2
3      <fieldset>
4
5          <label>
6              Name
7              <input type="text" name="name" placeholder="Full name" required
8          </label>
9
10         <label>
11             Email
12             <input type="email" name="email" placeholder="Email address" re
13         </label>
14
15         <label>
16             Number of Passes
17             <select name="quantity" required>
18                 <option value="1" selected>1</option>
19                 <option value="2">2</option>
20                 <option value="3">3</option>
21                 <option value="4">4</option>
22                 <option value="5">5</option>
23             </select>
24         </label>
25
26         <label>
27             Comments
28             <textarea name="comments"></textarea>
29         </label>
30
31     </fieldset>
32
33     <input type="submit" name="submit" value="Purchase">
34
35 </form>
```

Here we can see each form control nested within a `<label>` element. The Name form control uses an `<input>` element with a `type` attribute value of `text`, while

the Email form control uses an `<input>` element with a `type` attribute value of `email`.

Both the Name and Email form controls include the `required` Boolean attribute and a `placeholder` attribute.

The Number of Passes form control uses the `<select>` element and nested `<option>` elements. The `<select>` element itself includes the `required` Boolean attribute, and the first `<option>` element includes the `selected` Boolean attribute.

The Comments form control uses the `<textarea>` element without any special modifications. And lastly, outside of the `<fieldset>` element is the submit form control, which is formed by an `<input>` element with a `type` attribute value of `submit`.

- 7 With the form in place, it's time to add styles to it. We'll begin with a few default styles on the `<form>` element itself and on the `<input>`, `<select>`, and `<textarea>` elements.

Within the register section of our `main.css` file we'll want to add the following styles:

```
1  form {
2    margin-bottom: 22px;
3  }
4  input,
5  select,
6  textarea {
7    font: 300 16px/22px "Lato", "Open Sans", "Helvetica Neue", Helvetica
8  }
```

We'll start by placing a 22-pixel `margin` on the bottom of our form to help vertically space it apart from other elements. Then we'll add some standard `font`-based styles—including `weight`, `size`, `line-height`, and `family`—for all of the `<input>`, `<select>`, and `<textarea>` elements.

By default, every browser has its own interpretation of how the styles for form controls should appear. With this in mind, we have repeated the `font`-based styles from our `<body>` element to ensure that our styles remain consistent.

- 8 Let's add some styles to the elements within the `<fieldset>` element. Since we may add additional `<fieldset>` elements later on, let's add a class attribute value of `register-group` to our existing `<fieldset>` element, and from there we can apply unique styles to the elements nested within it.

```
1      <fieldset class="register-group">
2          ...
3      </fieldset>
```

Once the `register-group` class attribute value is in place, we'll add a few styles to the elements nested within the `<fieldset>` element. These styles will appear in our `main.css` file, below the existing form styles.

```
1      .register-group label {
2          color: #648880;
3          cursor: pointer;
4          font-weight: 400;
5      }
6      .register-group input,
7      .register-group select,
8      .register-group textarea {
9          border: 1px solid #c6c9cc;
10         border-radius: 5px;
11         color: #888;
12         display: block;
13         margin: 5px 0 27px 0;
14         padding: 5px 8px;
15     }
16     .register-group input,
17     .register-group textarea {
18         width: 100%;
19     }
20     .register-group select {
21         height: 34px;
22         width: 60px;
23     }
24     .register-group textarea {
25         height: 78px;
26     }
```

You'll notice that most of these properties and values revolve around the box model, which we covered in Lesson 4. We're primarily setting up the size of different form controls, ensuring that they are laid out appropriately. Aside from adding some box model styles, we're adjusting the `color` and `font-weight` of a few elements.

- 9 So far, so good: our form is coming together quite nicely. The only remaining element yet to be styled is the submit button. As it's a button, we actually have some existing styles we can apply here. If we think back to our home page, our hero section contained a button that received some styles by way of the `btn` class attribute value.

Let's add this `class` attribute value, `btn`, along with a new `class` attribute value of `btn-default` to our submit button. Specifically we'll use the class name of `btn-default` since this button is appearing on a white background and will be the default style for buttons moving forward.

```
1 <input class="btn btn-default" type="submit" name="submit" value="Purc
```



Now our submit button has some shared styles with the button on the home page. We'll use the `btn-default` class attribute value to then apply some new styles to our submit button specifically.

Going back to the buttons section of our `main.css` file, let's add the following:

```
1 .btn-default {  
2   border: 0;  
3   background: #648880;  
4   padding: 11px 30px;  
5   font-size: 14px;  
6 }  
7 .btn-default:hover {  
8   background: #77a198;  
9 }
```

These new styles, which define the size and background of our submit button, are then combined with the existing `btn` class styles to create the final presentation of our submit button.

Our Register page is finished, and attendees can now begin to reserve their tickets.

STYLES  
CONFERENCE

August 24–26th — Chicago, IL

HOME SPEAKERS SCHEDULE VENUE REGISTER

## Register

Every year we aim to have an unbelievable time, and this year we'd love it for you to join us. Conference passes only cost \$99, one of the best values you'll find.

### Purchase a Conference Pass

\$99 PER PASS

Purchase your Styles Conference pass using the form to the right. Multiple passes may be purchased within the same order, so feel free to bring a friend or two along. Once your order is finished we'll follow up and provide a receipt for your purchase. See you soon!

#### Why Attend?

- Over twenty world-class speakers
- One full day of workshops and two full days of presentations
- Hosted at The Chicago Theatre, a historical landmark
- August in Chicago is simply amazing

**Name**

**Email**

**Number of Passes**

**Comments**

**PURCHASE**

© Styles Conference

Home Speakers Schedule Venue Register

**Fig 10.07**

Our registration page, which includes a form

## Demo & Source Code

Below you may view the Styles Conference website in its current state, as well as download the source code for the website in its current state.

[View the Styles Conference Website](#) or [Download the Source Code](#) (Zip file)

## Summary

Forms play a large role in how users interact with, provide information to, and take action on websites. We've taken all the right steps to learn not only how to mark up

forms but also how to style them.

To quickly recap, within this lesson we discussed the following:

- How to initialize a form
- Ways to obtain text-based information from users
- Different elements and methods for creating multiple choice options and menus
- Which elements and attributes are best used to submit a form's data for processing
- How best to organize forms and give form controls structure and meaning
- A handful of attributes that help collect more qualified data

Our understanding of HTML and CSS is progressing quite nicely, and we only have one more component to learn: tables. In the next chapter, we'll take a look at how to organize and present data with tables.

## Resources & Links

- [Forms](#) via HTML Dog
- [Form Element](#) via Mozilla Developer Network
- [Input Element](#) via Mozilla Developer Network
- [A Form of Madness](#) via Dive Into HTML5

### Lesson 9

[Adding Media](#)

### Lesson 11

[Organizing Data with Tables](#)

## Learn More HTML & CSS or Study Other Topics

Learning how to code HTML & CSS and building successful websites can be challenging, and at times additional help and explanation can go a long way. Fortunately there are plenty of online schools, boot camps, workshops, and the alike, that can help.

Select your topic of interest below and I will recommend a course I believe will provide the best learning opportunity for you.

### Select Your Topic of Interest:

Design & Product
Front-end Development

Web Development

Mobile Development

Data & Machine Learning

Info & Cyber Security

## Your Course Recommendations:

*Select a topic above to view your course recommendations.*

## Learn to Code HTML & CSS the Book



*Learn to Code HTML & CSS* is an interactive beginner's guide with one express goal: **teach you how to develop and style websites** with HTML and CSS. Outlining the fundamentals, this book covers all of the common elements of front-end design and development.

**Buy Learn to Code HTML & CSS**

Also available at [Amazon](#) and [Barnes & Noble](#)

## Looking for Advanced HTML & CSS Lessons?



Checkout these [advanced HTML and CSS lessons](#) to take a deeper look at front-end design and development, perfect for any designer or front-end developer looking to round out their skills.

[View Advanced HTML & CSS Lessons](#)

## Join the Newsletter

To stay up to date and learn when new courses and lessons are posted, please sign up for the newsletter—spam free.

[Get Updates](#)

© Shay Howe

[Enjoy these lessons?](#)

[Follow @shayhowe](#)